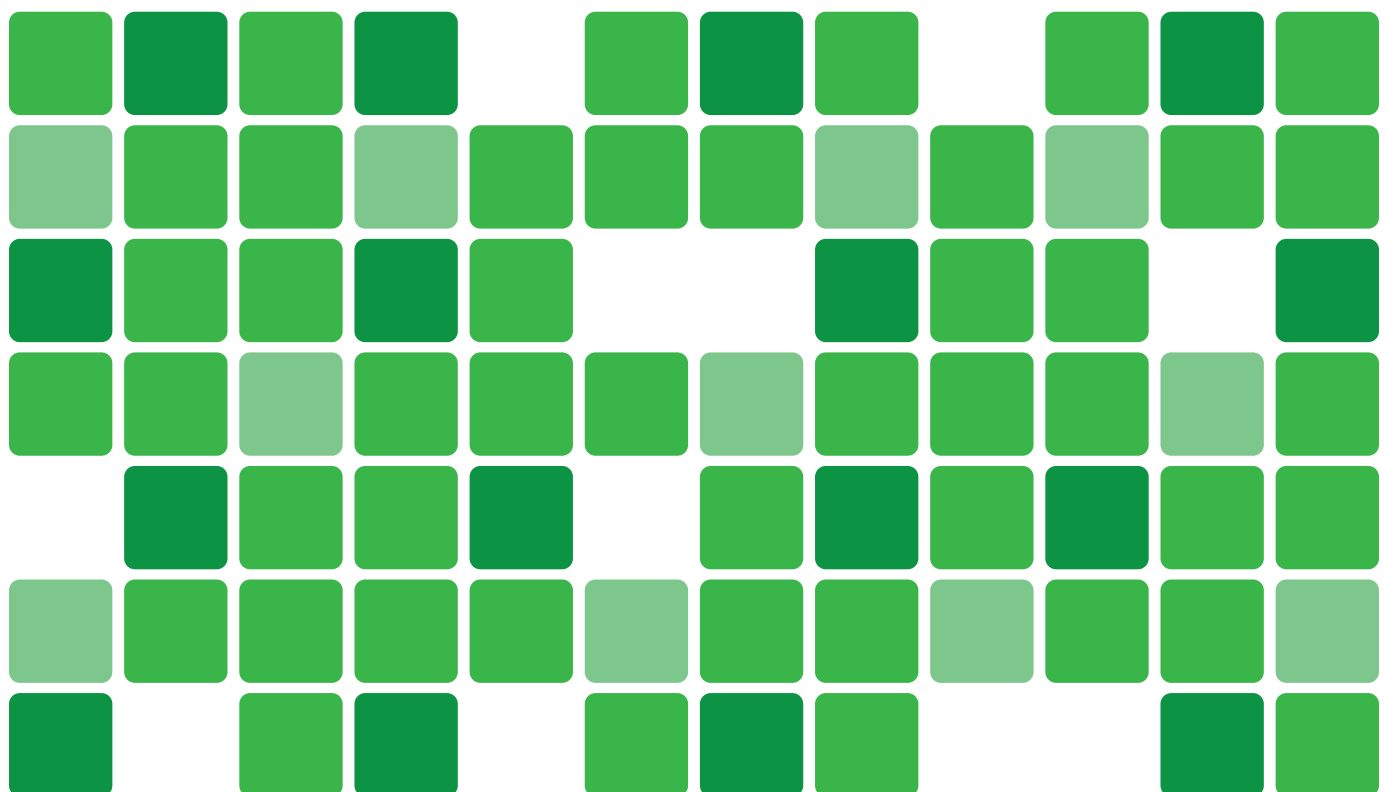




HTML5 e CSS3

Reprograme(se)

Prof. Ricardo Maroquio



Copyright © 2023 Ricardo Maroquio INSTITUTO FEDERAL DO ESPÍRITO SANTO CAMPUS CACHOEIRO

Disponível para download em: [HTTP://MAROQUIO.COM](http://maroquio.com)

Esta apostila está sob a licença *Creative Commons Attribution-NonCommercial 4.0*. Você só pode usar esse material se concordar com essa licença. Você pode acessar a licença em <https://creativecommons.org/licenses/by-nc-sa/4.0>. A menos que seja aplicável por lei ou acordado por escrito, material digital sob esta licença deve ser distribuído “COMO ESTÁ”, SEM GARANTIAS OU CONDIÇÕES DE NENHUM TIPO. Para maiores informações sobre o uso deste material, consulte a licença disponível no link supracitado.

Fevereiro de 2023



Sumário



I

Parte I: HTML

1	Introdução	18
1.1	O Funcionamento de um Site	18
1.2	Exercícios Propostos	28
1.3	Considerações Sobre o Capítulo	29
2	Documentos HTML e CSS	30
2.1	Estrutura de Uma Página HTML	30
2.2	Estrutura de Um Documento CSS	34
2.3	Exercícios Propostos	38
2.4	Considerações Sobre o Capítulo	39
3	Textos	40
3.1	Títulos e Subtítulos	40
3.2	Parágrafos e Quebras	43
3.3	Textos Pré-Formatados	43
3.4	Links	45
3.5	Formatação de Texto <i>Inline</i>	47
3.6	Citações	49
3.7	Exercícios Propostos	50
3.8	Considerações Sobre o Capítulo	51

4	Listas e Tabelas	52
4.1	Listas	52
4.2	Tabelas	56
4.3	Exercícios Propostos	62
4.4	Considerações Sobre o Capítulo	65
5	Imagens	66
5.1	Adicionando Imagens a Uma Página HTML	66
5.2	Formatos de Imagem	68
5.3	Figuras e Legendas	70
5.4	Imagens Responsivas	71
5.5	Mapas de Imagens	73
5.6	Imagens Como Links	76
5.7	Exercícios Propostos	77
5.8	Considerações Sobre o Capítulo	78
6	Áudio e Vídeo	79
6.1	Inserindo Áudio em Uma Página HTML	79
6.2	Inserindo Vídeo em Uma Página HTML	83
6.3	Incorporando Vídeos do YouTube em Uma Página HTML	88
6.4	Exercícios Propostos	89
6.5	Considerações Sobre o Capítulo	92
7	Formulários	93
7.1	Criando um Formulário Básico	93
7.2	Enviando Dados de Formulários	94
7.3	Campos de Formulários	95
7.4	Validação de Formulários com Atributos HTML	97
7.5	Tipos de Campos Avançados	99
7.6	Exercícios Propostos	105
7.7	Considerações Sobre o Capítulo	109

8	Introdução	111
8.1	Associação de CSS a Documentos HTML	112
8.2	Seletores Simples	114
8.3	Seletores Compostos	116
8.4	Exercícios Propostos	117
8.5	Considerações Sobre o Capítulo	119
9	Cores, Planos de Fundo, Bordas e Margens	120
9.1	Cores	121
9.2	Unidades de Medida	123
9.3	Planos de Fundo	126
9.4	Bordas	128
9.5	Margens Interna e Externa	131
9.6	Exercícios Propostos	133
9.7	Considerações Sobre o Capítulo	136
10	Fontes, Textos, Links e Listas	137
10.1	Fontes	137
10.2	Textos	140
10.3	Links	144
10.4	Listas	146
10.5	Exercícios Propostos	149
10.6	Considerações Sobre o Capítulo	151
11	Pseudo-Seletores CSS	152
11.1	Conteúdo Posterior e Anterior	152
11.2	Primeira Letra e Primeira Linha	153
11.3	Marcador de Itens de Lista	153
11.4	Valores de Atributos	154
11.5	Primeiro de Um Tipo	154

11.6	Campos de Formulário	155
11.7	Exercícios Propostos	158
11.8	Considerações Sobre o Capítulo	160
12	Exibição e Posicionamento de Elementos	161
12.1	Exibição de Elementos	161
12.2	Posicionamento de Elementos	163
12.3	Propriedades Complementares para Posicionamento	166
12.4	Exercícios Propostos	169
12.5	Considerações Sobre o Capítulo	172
13	Leiautes de Página com <i>Flexbox</i>	173
13.1	Introdução	173
13.2	Conceitos Básicos	174
13.3	Propriedades do Contêiner	174
13.4	Propriedades dos Itens	180
13.5	Exemplos Práticos	184
13.6	Exercícios Propostos	188
13.7	Considerações Sobre o Capítulo	190
14	Leiautes de Página com <i>CSS Grid</i>	191
14.1	Conceitos Básicos	191
14.2	Propriedades do Contêiner Grid	192
14.3	Propriedades dos Itens Grid	198
14.4	Exemplos Práticos	200
14.5	Exercícios Propostos	201
14.6	Considerações Sobre o Capítulo	202
15	Responsividade	203
15.1	Princípios Básicos da Responsividade	204
15.2	Responsividade no HTML	207
15.3	Implementando CSS Responsivo	209

15.4	Técnicas Avançadas de Responsividade	214
15.5	Testando e Otimizando a Responsividade	216
15.6	Exercícios Propostos	217
15.7	Considerações Sobre o Capítulo	220



Lista de Figuras



1.1	Sites conectados através de hiperlinks.	19
1.2	Os códigos que compõem uma página de hipertexto.	20
1.3	Páginas HTML estáticas e dinâmicas.	22
1.4	Diagrama de uma requisição de página pelo navegador.	23
1.5	Páginas de vídeos específicos do YouTube.	26
1.6	Páginas de vídeos específicos do YouTube com dados dinâmicos destacados.	26
1.7	Atualização parcial de página usando Javascript em segundo plano.	28
2.1	Renderização do código HTML do exemplo 2.2.	35
2.2	Renderização do código HTML do exemplo 2.2 com o CSS do exemplo 2.3 associado.	37
3.1	Renderização do código HTML do exemplo 3.1.	42
3.2	Renderização do código HTML do exemplo 3.2.	44
3.3	Renderização do texto pré-formatado do exemplo 3.3.	44
3.4	Renderização do texto de código-fonte do exemplo 3.4.	45
3.5	Renderização dos links dos exemplos desta seção.	46
3.6	Renderização dos elementos inline do código 3.9.	48
3.7	Renderização dos códigos de citação desta seção.	50
4.1	Renderização da lista não ordenada do exemplo 4.1.	53
4.2	Renderização da lista não ordenada com marcador quadrado do exemplo 4.2.	54
4.3	Renderização da lista ordenada do exemplo 4.3.	54
4.4	Renderização da lista de definições do exemplo 4.4.	55
4.5	Renderização da tabela do exemplo 4.5.	57
4.6	Renderização da tabela do exemplo 4.6.	57
4.7	Renderização da tabela do exemplo 4.7.	58

4.8	Renderização da tabela do exemplo 4.8.	60
4.9	Renderização da tabela do exemplo 4.9.	61
4.10	Renderização da tabela do exemplo 4.10.	62
5.1	Renderização da imagem do exemplo 5.1.	67
5.2	Renderização da imagem do exemplo 5.2.	68
5.3	Renderização da imagem com legenda do exemplo 5.3.	71
5.4	Imagem simples com 4 formas geométricas, com dimensões 690x142 pixels. (Disponível em https://imgbox.com/KAP90MBi)	73
13.1	Contêiner e itens <i>flexbox</i>	174
13.2	Possíveis configurações de direção e sentido do eixo <i>flexbox</i>	175
13.3	Possíveis configurações de alinhamento ao longo do eixo principal.	177
13.4	Configurações de alinhamento de itens no eixo <i>flexbox</i> transversal.	178
13.5	Possíveis alinhamentos de linhas em um contêiner <i>flexbox</i>	179
13.6	Configurações possíveis de <i>order</i>	180
13.7	Configurações possíveis de <i>flex-grow</i>	181
13.8	Possíveis configurações de alinhamento individual de itens <i>flexbox</i>	183
13.9	Centralizando um elemento horizontal e verticalmente em um contêiner com <i>flexbox</i>	185
13.10	Barra de navegação com <i>flexbox</i>	187
13.11	Galeria de imagens com <i>flexbox</i>	188



Lista de Exemplos de Código



1.1	Exemplo de cabeçalho de uma requisição HTTP.	25
2.1	O código de uma página HTML simples.	30
2.2	Página HTML simples com cabeçalho, corpo e rodapé.	34
2.3	Documento CSS simples.	35
2.4	Associação de documento CSS a um documento HTML.	36
3.1	Elementos textuais básicos.	41
3.2	Elementos textuais básicos.	43
3.3	Texto pré-formatado com elemento <code><pre></code>	44
3.4	Texto de código-fonte pré-formatado com elemento <code><code></code>	45
3.5	Link para interno apontando para uma página do próprio site.	45
3.6	Link externo apontando para a página do Google.	46
3.7	Link ancorado apontando para um elemento da própria página.	46
3.8	Link com atributo <code>target</code> para definição de local de abertura.	47
3.9	Elementos <i>inline</i> para formatação de texto.	48
3.10	Bloco de citação longa.	49
3.11	Trecho de citação curta.	50
4.1	Lista não ordenada básica.	53
4.2	Lista não ordenada com marcador quadrado.	54
4.3	Lista ordenada com algarismos romanos.	54
4.4	Lista de definições.	55
4.5	Tabela simples.	56
4.6	Tabela com bordas.	57
4.7	Tabela com <code><thead></code> , <code><tbody></code> e <code><tfoot></code>	59
4.8	Tabela com mesclagem de colunas.	60
4.9	Tabela com mesclagem de linhas.	61

4.10	Tabela com mesclagem de linhas e colunas.	62
5.1	Adicionando uma imagem ao documento	66
5.2	Imagem com atributo de largura.	67
5.3	Imagem com legenda.	71
5.4	Imagem responsiva com arquivos otimizados.	72
5.5	Código inicial do mapeamento de imagem.	74
5.6	Imagem e mapa de imagem.	74
5.7	Imagem e mapa de imagem com uma área clicável.	75
5.8	Imagem e mapa de imagem com uma área clicável.	75
5.9	Versão final do mapeamento da imagem de formas geométricas.	76
5.10	Imagem como link.	76
6.1	Inserindo um elemento de áudio.	80
6.2	Uso do elemento áudio com diferentes arquivos.	80
6.3	Uso do elemento áudio com atributo controls.	81
6.4	Uso do elemento áudio com atributo autoplay.	81
6.5	Uso do elemento áudio com atributos controlse loop.	82
6.6	Uso do elemento áudio com atributo muted.	82
6.7	Exemplo do atributo preload.	82
6.8	Uso do elemento video.	83
6.9	Uso do elemento videocom diferentes arquivos.	83
6.10	Uso do elemento videocom atributo controls.	84
6.11	Uso do elemento videocom o atributo muted.	84
6.12	Uso do elemento videocom o atributo autoplay.	85
6.13	Uso do elemento videocom o atributo loop.	85
6.14	Uso do elemento videocom o atributo poster.	85
6.15	Uso do elemento videocom todos os atributos.	86
6.16	Adicionando legenda a um vídeo com o elemento track.	86
6.17	Exemplo de um arquivo de legenda VTT.	87
6.18	Incorporando vídeo do YouTube.	89
6.19	Incorporando vídeo do YouTube com parâmetros.	89
7.1	Exemplo de formulário básico.	93
7.2	Exemplo de formulário com método POST.	94
7.3	Exemplo de formulário com método GET.	95
7.4	Exemplo de campo de texto longo.	95

7.5	Exemplo de campos de e-mail e senha.	96
7.6	Exemplos de <i>checkboxes</i>	96
7.7	Exemplo de campo de checagem exclusiva com elementos <i>radio buttons</i>	97
7.8	Exemplo de campo de seleção com elemento <i>select</i>	97
7.9	Exemplo de campo de texto com o atributo <i>required</i>	97
7.10	Exemplo de campo de número com os atributos <i>min</i> e <i>max</i>	98
7.11	Exemplo de campo de entrada com o atributo <i>pattern</i>	98
7.12	Exemplo de campo de texto com os atributos <i>maxlength</i> e <i>minlength</i>	99
7.13	Exemplo de campo do tipo botão.	99
7.14	Exemplo de campo do tipo cor.	99
7.15	Exemplo de campo do tipo data.	100
7.16	Exemplo de campo de data e hora.	100
7.17	Exemplo de campo de seleção de arquivo.	100
7.18	Exemplo de campo oculto.	101
7.19	Exemplo de botão de imagem.	101
7.20	Exemplo de campo de entrada de mês.	102
7.21	Exemplo de campo de entrada de número.	102
7.22	Exemplo de campo de entrada de intervalo.	103
7.23	Exemplo de botão de redefinição.	103
7.24	Exemplo de campo de busca.	104
7.25	Exemplo de campo de telefone.	104
7.26	Exemplo de campo de entrada do tipo <i>time</i>	104
7.27	Exemplo de campo de entrada do tipo <i>url</i>	105
7.28	Exemplo de campo de entrada de semana.	105
8.1	Exemplo de CSS via Atributo <i>style</i>	112
8.2	Exemplo de CSS via Elemento <i><style></i>	113
8.3	Arquivo CSS externo.	113
8.4	Exemplo de CSS via Arquivos Externos	114
8.5	Seletor CSS por tipo de elemento.	115
8.6	Seletor CSS por classe.	115
8.7	Exemplo de seletor por classe sendo usado.	115
8.8	Seletor CSS por ID.	116
8.9	Exemplo de seletor por ID sendo usado.	116
8.10	Seletor CSS por descendente.	117

8.11	Seletor CSS por classe e tipo de elemento.	117
8.12	Seletor CSS por classe e tipo de elemento com descendente.	117
9.1	Exemplo de formatos de cores em CSS.	122
9.2	Exemplo de gradiente em CSS.	123
9.3	Exemplo de uso da unidade pixels.	123
9.4	Exemplo de uso da unidade pontos.	124
9.5	Exemplo de uso das unidades centímetros e milímetros.	124
9.6	Exemplo de uso das unidades em rem.	124
9.7	Exemplo de uso da unidade porcentagem.	125
9.8	Exemplo de uso das unidades vwe vh.	125
9.9	Definindo a cor de fundo com background-color.	126
9.10	Definindo uma imagem de fundo com background-image.	126
9.11	Definindo a repetição da imagem de fundo com background-repeat.	127
9.12	Definindo a posição da imagem de fundo com background-position.	127
9.13	Definindo a fixação da imagem de fundo com background-attachment.	128
9.14	Definindo o tamanho da imagem de fundo com background-size	128
9.15	Exemplo de border-color.	129
9.16	Exemplo de uso da propriedade border-width.	129
9.17	Exemplo de uso da propriedade border-style.	129
9.18	Exemplo de uso da propriedade border-radius.	130
9.19	Exemplo de uso da propriedade border.	130
9.20	Exemplo de uso da propriedade border-left.	130
9.21	Exemplo de uso da propriedade outline.	131
9.22	Exemplo de uso da propriedade margin	132
9.23	Exemplo de margem interna com propriedade padding.	132
9.24	Exemplo de margincom valor auto.	133
10.1	Definindo o tamanho da fonte.	138
10.2	Definindo a família de fontes.	138
10.3	Definindo o estilo da fonte para itálico.	138
10.4	Definindo a espessura da fonte.	139
10.5	Definindo a variação da fonte.	139
10.6	Definindo propriedades de fonte de forma abreviada.	139
10.7	Exemplo de text-align.	140
10.8	Exemplo de uso das propriedades directione unicode-bidi.	141

10.9 Exemplo de uso da propriedade <code>text-decoration</code>	141
10.10 Exemplo de uso da propriedade <code>text-transform</code>	141
10.11 Exemplo de uso da propriedade <code>text-indent</code>	142
10.12 Modificando o espaçamento entre letras.	142
10.13 Exemplo de uso da propriedade <code>line-height</code>	142
10.14 Exemplo de uso da propriedade <code>word-spacing</code>	143
10.15 Exemplo de uso da propriedade <code>white-space</code>	143
10.16 Exemplo de uso da propriedade <code>text-shadow</code>	144
10.17 Formatação de links visitados.	144
10.18 Formatação de links com <code>hover</code> e <code>active</code>	145
10.19 Transformando um link em um botão.	146
10.20 Mudando o tipo de lista.	147
10.21 Usando uma imagem como marcador.	147
10.22 Mudando a posição do marcador na lista.	148
10.23 Ajustando margem, preenchimento e indentação da lista.	148
10.24 Exemplo completo de formatação de lista.	149
11.1 Adicionando uma seta após um link.	152
11.2 Adicionando uma seta antes de um link.	152
11.3 Aplicando estilo à primeira letra de um parágrafo.	153
11.4 Aplicando estilo à primeira linha de um parágrafo.	153
11.5 Estilizando o marcador de itens de lista.	153
11.6 Estilizando links específicos.	154
11.7 Estilizando o primeiro parágrafo de um artigo.	155
11.8 Estilizando campos de tipos específicos.	155
11.9 Estilizando campos em determinado estado de validação.	156
11.10 Estilizando campos de entrada de número com diferentes estados.	156
11.11 Estilizando campos de entrada de texto habilitados e desabilitados.	157
11.12 Estilizando campos de entrada de texto somente leitura e leitura e escrita.	157
11.13 Estilizando campos de entrada de texto requeridos e opcionais.	158
11.14 Estilizando campos de entrada de texto com foco.	158
12.1 Exemplo de utilização da propriedade <code>display</code>	162
12.2 Exemplo de utilização da propriedade <code>visibility</code>	163
12.3 Exemplo de utilização da propriedade <code>opacity</code>	163
12.4 Exemplo de utilização da propriedade <code>position</code>	164

12.5 Exemplo de utilização das propriedades left, right, top e bottom.	165
12.6 Exemplo de utilização da propriedade z-index.	166
12.7 Exemplo de uso da propriedade overflow.	167
12.8 Exemplo de uso da propriedade float.	168
12.9 Exemplo de uso da propriedade box-sizing.	169
12.10 Exemplo de uso da propriedade box-sizing em um seletor CSS universal.	169
13.1 Exemplo de contêiner <i>flexbox</i>	174
13.2 Alterando a direção do eixo <i>flexbox</i> com flex-direction.	175
13.3 Configurando o comportamento de quebra de linha em um contêiner <i>flexbox</i>	175
13.4 Exemplo de uso da propriedade flex-flow.	176
13.5 Exemplo de uso da propriedade justify-content.	176
13.6 Alinhando itens ao longo do eixo <i>flexbox</i> transversal.	178
13.7 Exemplo de uso da propriedade align-content.	179
13.8 Exemplo de uso da propriedade order.	180
13.9 Alterando o fator de crescimento de itens <i>flexbox</i>	181
13.10 Exemplo de uso do flex-shrink.	181
13.11 Configurando tamanhos iniciais de itens <i>flexbox</i>	182
13.12 Exemplo de uso da propriedade align-self.	183
13.13 Representações compactas de propriedades de itens <i>flexbox</i>	184
13.14 Alinhamento horizontal e vertical com <i>flexbox</i>	185
13.15 Barra de navegação com <i>flexbox</i>	186
13.16 Barra de navegação usando <i>flexbox</i>	186
13.17 CSS da galeria de imagens com <i>flexbox</i>	187
13.18 HTML da galeria de imagens com <i>flexbox</i>	188
14.1 Definindo um contêiner Grid.	192
14.2 Definindo colunas e linhas com grid-template-columns e grid-template-rows.	192
14.3 Exemplo de uso de grid-template-areas.	193
14.4 Exemplo de uso de grid-template.	194
14.5 Exemplo de uso de grid-gap, column-gap e row-gap.	195
14.6 Exemplo de uso de justify-items.	195
14.7 Exemplo de uso de align-items.	195
14.8 Exemplo de uso de place-items.	196
14.9 Exemplo de uso de justify-content.	196
14.10 Exemplo de uso de align-content.	197

14.11 Exemplo de uso de <code>place-content</code>	197
14.12 Definindo o tamanho das colunas e linhas automáticas com <code>grid-auto-columns</code> e <code>grid-auto-rows</code>	198
14.13 Definindo a direção de posicionamento automático dos itens <i>Grid</i> com <code>grid-auto-flow</code>	198
14.14 Posicionando um item <i>Grid</i> usando <code>grid-column-start</code> e <code>grid-column-end</code>	199
14.15 Posicionando um item <i>Grid</i> usando <code>grid-row-start</code> e <code>grid-row-end</code>	199
14.16 Posicionando um item <i>Grid</i> usando a propriedade <code>grid-area</code>	199
14.17 Leiaute de duas colunas.	200
14.18 Leiaute de cabeçalho, conteúdo e rodapé.	201
15.1 Exemplo de <i>media query</i>	203
15.2 Exemplo de viewport em HTML.	204
15.3 Exemplo de design fluido.	204
15.4 Exemplo de imagens flexíveis.	205
15.5 Exemplo de media query para telas pequenas e grandes.	205
15.6 Exemplo de unidades relativas.	206
15.7 Exemplo de metatag <code>viewport</code>	207
15.8 Exemplo de elementos e atributos HTML para responsividade.	208
15.9 Exemplo de uso do Bootstrap.	209
15.10 Exemplo de grid flexível.	210
15.11 Exemplo de imagem responsiva.	211
15.12 Exemplo de vídeo responsivo.	211
15.13 Exemplo de tipografia responsiva.	212
15.14 Exemplo de uso de media queries.	213
15.15 Exemplo de layout com Flexbox.	214
15.16 Exemplo de layout com CSS Grid.	215
15.17 Exemplo de uso de <code>srcset</code> e <code>picture</code>	215
15.18 Exemplo de carregamento condicional.	216
15.19 Código da página 1.	218
15.20 Código da página 2.	219
15.21 Código da página 3.	220

Parte I: HTML

1	Introdução	18
2	Documentos HTML e CSS	30
3	Textos	40
4	Listas e Tabelas	52
5	Imagens	66
6	Áudio e Vídeo	79
7	Formulários	93



1. Introdução

O mundo tem hoje mais de 4.7 bilhões de usuários de internet, sendo que cada usuário padrão passa cerca de 6 horas e meia por dia navegando. Conquistar um pouco da atenção desse usuário, seja através de um site ou de uma página em rede social, deixou de ser algo opcional.

Não é à toa que existem mais 1.8 bilhão de sites na web atualmente. Consequentemente, nunca houve tanta demanda por profissionais capazes de criar e administrar sites. Se você quer aproveitar essa oportunidade aprendendo a criar e a publicar sites, você está no lugar certo. Vale ressaltar que este material é totalmente voltado para iniciantes, isto é, para quem realmente não sabe nada sobre criação de sites.

1.1 O Funcionamento de um Site

Nesta seção, eu vou explicar os princípios básicos de funcionamento de um site, para que, antes de começar a codificar, você entenda como as coisas funcionam nos bastidores. Essa compreensão é imprescindível para que você consiga criar sites de qualidade de jeito certo.

Um site é composto por uma ou mais páginas de hipertexto. Diferentemente de um texto comum, uma página de hipertexto corresponde a uma página que pode conter textos, imagens e elementos multimídia de forma combinada para comunicar alguma ideia. Mas o principal diferencial de uma página de hipertexto é que ela pode conter ligações com outras páginas, de forma que o leitor possa navegar para essas outras páginas através dessas ligações, que são chamadas de hiperlinks, ou simplesmente *links*. A figura 1.1 apresenta exemplos fictícios de sites que possuem hiperlinks conectando-os.



Figura 1.1: Sites conectados através de hiperlinks.

Nos bastidores, para que o navegador consiga exibir o conteúdo da forma como você está acostumado a ver quando navega pela web, um texto contendo um conjunto de instruções deve indicar ao navegador **quais** elementos devem ser exibidos e **como** esses elementos devem ser exibidos. Essas instruções interpretadas pelo navegador são chamadas de **códigos**, que nada mais são do que textos escritos em um padrão que chamamos de **linguagem**. Toda página de um site, basicamente, é uma combinação de códigos escritos em três linguagens diferentes: HTML, CSS e Javascript, como mostra a figura 1.2.

A primeira linguagem, **HTML**, é uma sigla para *HyperText Markup Language*, traduzindo, Linguagem de Marcação de Hipertexto, ou seja, é uma linguagem de **marcação**. Em síntese, ela é a responsável por definir **quais** elementos serão exibidos em uma determinada página. **CSS** é outra sigla, que quer dizer *Cascading Style Sheets*, ou folhas de estilo em cascata, portanto, é uma linguagem de **folhas de estilo**, ou uma linguagem de definição de estilos. Ela é responsável por definir **como** os elementos de conteúdo serão mostrados.

É através da linguagem CSS que apresentamos um mesmo conteúdo de forma alegre, de forma séria, de forma suave, chamativa etc. Resumindo, as definições de estilos CSS é que nos permitem mudar a forma como um conteúdo é exibido, ou seja, o visual do conteúdo.

Diferentemente de HTML e CSS, **Javascript** é uma linguagem de **programação**. Ela pode ter diferentes propósitos em uma página web e, pelo fato de ser uma linguagem de programação, requer conhecimentos de lógica de programação. Uma abordagem profunda da linguagem Javascript está fora do escopo deste material. Veremos apenas qual é o papel da linguagem Javascript em uma página web.

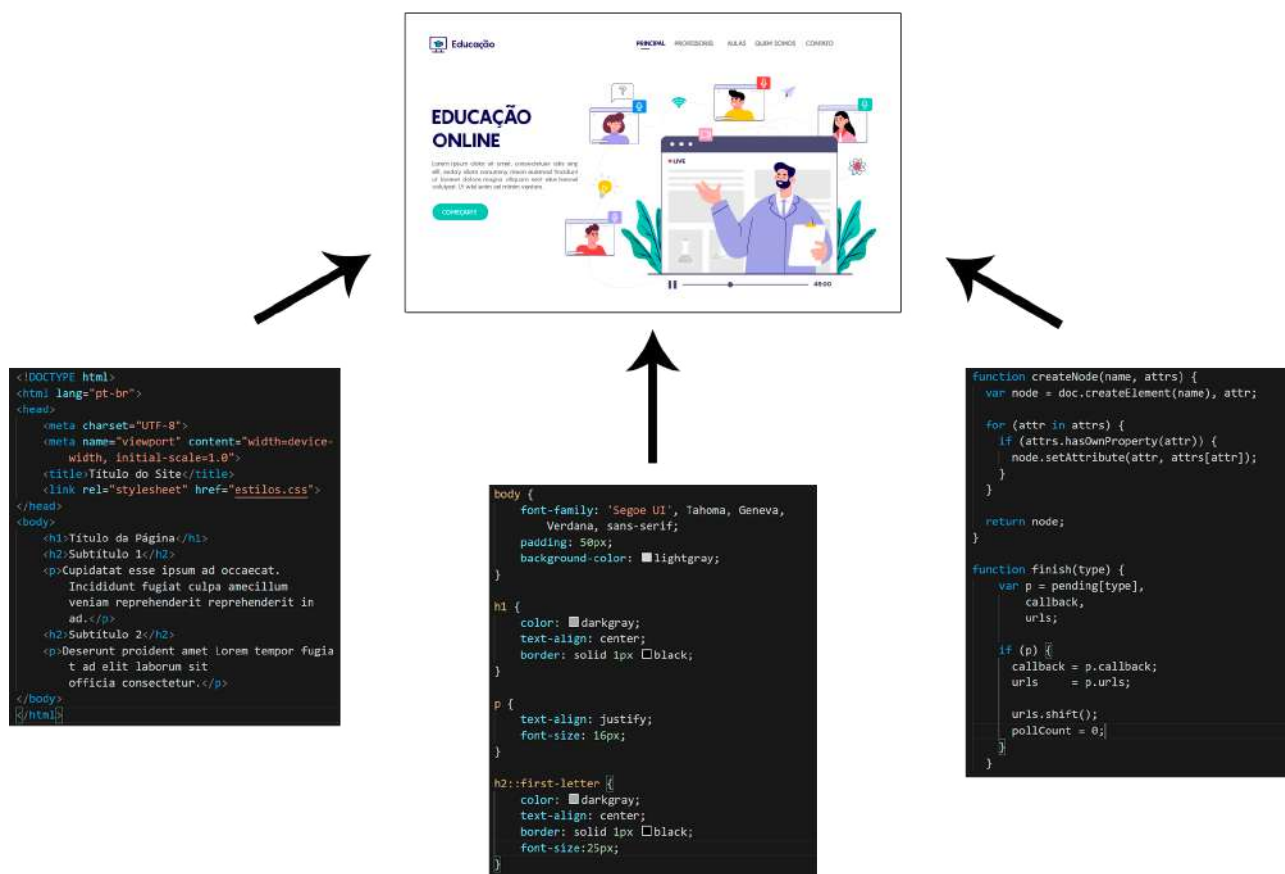


Figura 1.2: Os códigos que compõem uma página de hipertexto.

Primeiramente, é importante ressaltar que, para construir sites, não somos obrigados a usar Javascript. Na verdade, basicamente, somente sites que possuem interação com o usuário e/ou efeitos visuais avançados é que requerem o uso de Javascript. No final deste capítulo, eu voltarei a falar um pouco mais sobre Javascript, só para dizer como ele pode contribuir para o funcionamento de um site. Temos que ver algumas coisas antes para que você entenda melhor o papel do código Javascript.

Bem, essa combinação de HTML, CSS e Javascript forma o código-fonte de uma página web. Você consegue ver o código-fonte de qualquer página web clicando com o botão direito do mouse sobre a página em um navegador qualquer e selecionando a opção “Exibir código-fonte da página”, ou “Ver código-fonte da página”, ou alguma coisa parecida com isso (depende do navegador).

A princípio, o código-fonte que será exibido pode parecer algo meio assustador, mas em pouco tempo você já será capaz de compreender boa parte dele. O código-fonte de uma página web geralmente fica distribuído em mais de um arquivo. Na verdade, é uma boa prática separar em arquivos distintos os códigos HTML, CSS e Javascript. Pode acontecer, inclusive, de cada tipo de arquivo ser mantido por um programador diferente.

Normalmente, uma página contém apenas 1 arquivo HTML, mas pode estar associada a vários arquivos CSS e a vários arquivos Javascript. Os arquivos HTML podem ter a extensão “htm” ou “html”, enquanto os arquivos CSS possuem a extensão “css”, e os arquivos Javascript possuem a extensão “.js”.

Quanto ao código CSS, é comum ter um arquivo para estilizar botões, outro para estilizar o layout da página, outro para estilizar os textos, entre outros mais. Separar esses códigos CSS em arquivos diferentes pode ajudar a melhorar a organização do código-fonte de um site como um todo. Portanto você poderia criar um arquivo CSS para estilizar botões, outro para o layout, outro para a tipografia e assim por diante.

Outra observação importante é que as páginas de um mesmo site normalmente possuem um mesmo estilo. Geralmente usam as mesmas cores, a mesma tipografia, o mesmo layout, o mesmo estilo de cabeçalho e rodapé, entre outras coisas. Portanto, é comum que os mesmos arquivos CSS sejam compartilhados entre diferentes páginas web.

Assim, se você quiser mudar a tipografia usada em todas as páginas do site, basta ir ao arquivo CSS que contém essa formatação e mudar as propriedades de fonte desejadas. Assim, o site inteiro passará a ser mostrado com as novas definições de fonte. Pense em como isso facilita a manutenção de um site!

Quanto ao código Javascript, também é comum que as páginas compartilhem os mesmos códigos. É possível ter um código para mostrar uma janela com mensagem, para animar um elemento da página, para atualizar apenas uma parte da página, entre outros. Se determinado código Javascript é específico de uma página, recomenda-se separá-lo em um arquivo que será vinculado exclusivamente à página a que pertence. Isso evita que outras páginas carreguem códigos Javascript que não sejam necessários para ela. Assim como ocorre com CSS, é comum que um mesmo arquivo Javascript esteja vinculado a diferentes páginas, pois elas podem ter funcionalidades comportamentais em comum.

1.1.1 Páginas Estáticas e Dinâmicas

Os arquivos que compõem o código-fonte de uma página podem ter origem de duas fontes diferentes: podem ser arquivos prontos, que simplesmente serão baixados do servidor web que hospeda o site; ou podem ser gerados por um programa especial chamado de aplicação web, que roda no servidor web e que gera o código-fonte da página sempre que ela é solicitada por um usuário.

Quando as páginas de um site são totalmente compostas por arquivos prontos, dizemos que o site é estático. Quando as páginas (ao menos algumas delas) são geradas por uma aplicação web, dizemos que o site é dinâmico. É muito comum ter sites que possuem páginas estáticas e páginas dinâmicas. A figura 1.3 mostra como os sites estáticos e dinâmicos são servidos ao navegador do usuário.

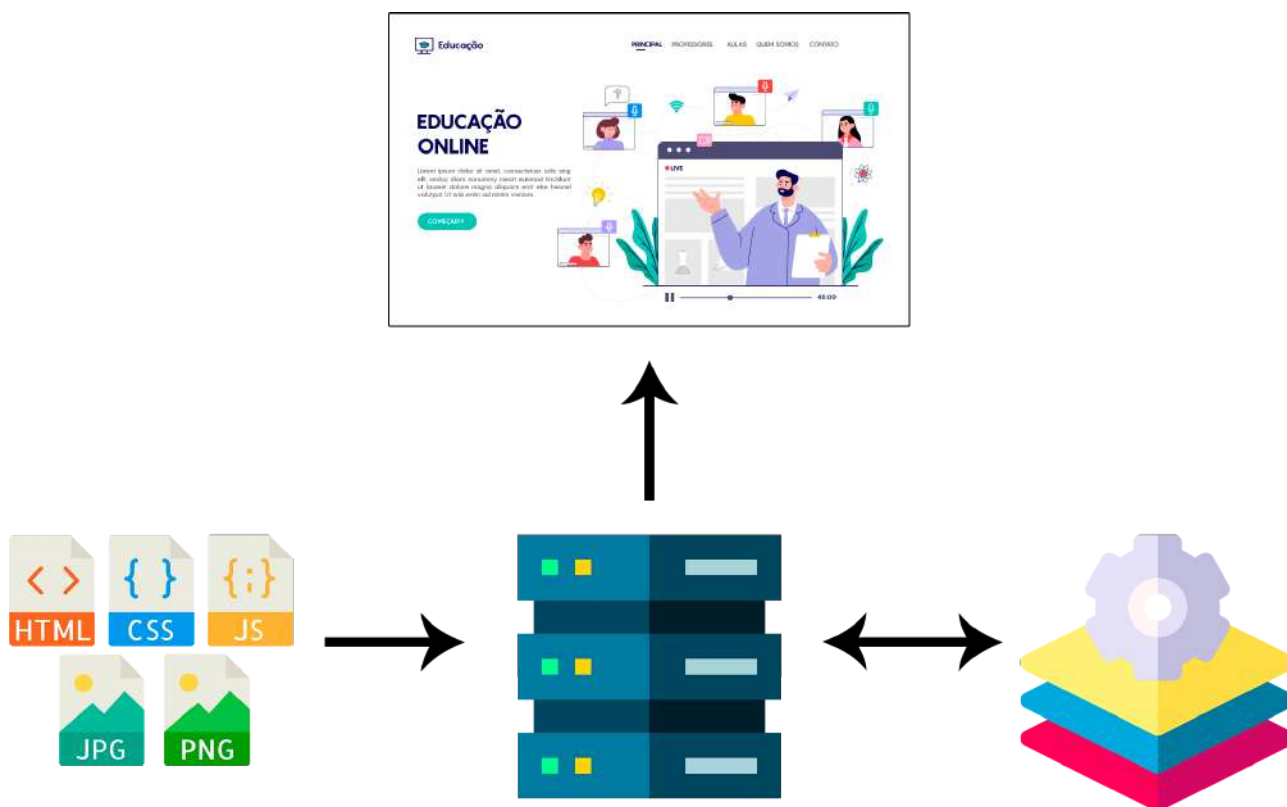


Figura 1.3: Páginas HTML estáticas e dinâmicas.

Normalmente, o código que é gerado dinamicamente por uma aplicação web é aquele relacionado ao conteúdo da página, ou seja, o código HTML. Os códigos CSS e Javascript raramente são gerados dinamicamente. Normalmente eles são criados de forma estática por um programador e permanecem intocados ao longo de toda a vida da aplicação web. Quando é necessário atualizar a aparência do site ou incluir novas funções Javascript para executar alguma tarefa adicional, os arquivos podem ser modificados por alguém, mas depois da modificação eles continuam estáticos no servidor, ou seja, eles **não serão gerados** pela aplicação web.

Um site estático tem a característica de não demandar atualizações com muita frequência. Um site que tenha o propósito de contar a história do Brasil, por exemplo, vai ter um conteúdo que dificilmente será alterado. Se houver necessidade de alteração, a frequência será tão baixa que basta fazer a alteração diretamente nos arquivos HTML. Não vale a pena criar uma aplicação web acessando um

banco de dados para gerar conteúdo se tal conteúdo muda somente uma vez por ano, por exemplo. Neste caso, vale a pena abrir o arquivo HTML em um editor e fazer a modificação diretamente no arquivo correspondente ao conteúdo da página.

1.1.2 O Conceito de Requisição de Página

Quando um endereço de um site é digitado na barra de endereços do navegador, a primeira coisa a ser baixada pelo navegador é o código HTML. Como já mencionado, esse código pode vir de um arquivo estático ou pode ser gerado por uma aplicação web que esteja rodando no servidor. Esse código HTML normalmente contém referências para os demais arquivos necessários para a página ser renderizada. Aí entram os arquivos com códigos CSS e Javascript, os arquivos de imagens e arquivos de outras mídias que façam parte da página. De posse do endereço desses arquivos associados, o navegador faz o download de cada um deles e renderiza a página.

A figura 1.4 apresenta um diagrama que mostra desde o momento em que o usuário digita o endereço de um site no navegador até o momento em que o navegador exibe a página para o usuário. Ressalto que esse diagrama é uma abstração do processo real. Isso quer dizer que algumas detalhes que acontecem nos bastidores foram suprimidos ou simplificados visando facilitar o entendimento do processo.

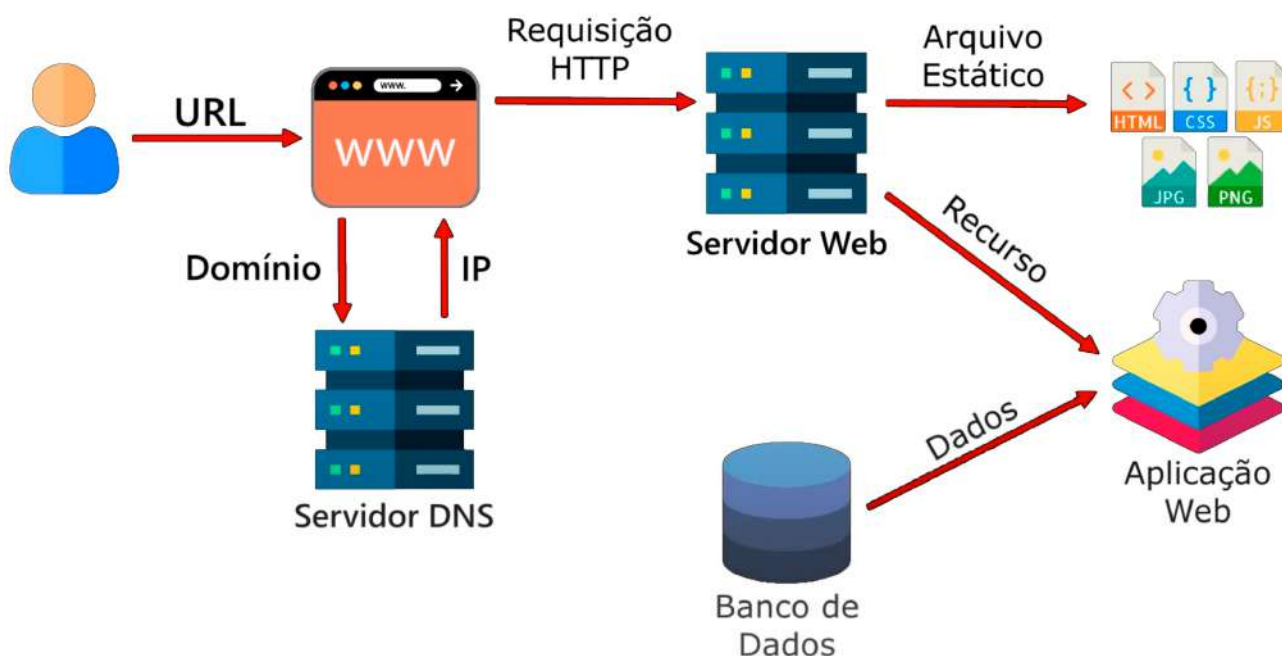


Figura 1.4: Diagrama de uma requisição de página pelo navegador.

Na figura 1.4, tudo começa com o usuário informando uma URL na barra de endereços do navegador. Acontece que, para acessar um servidor web, é necessário saber seu número IP. A sigla IP vem de *Internet Protocol*, ou “protocolo de internet”. O número IP de um servidor é mundialmente único, ou seja, ele é um endereço exclusivo do servidor em toda a internet.

Os nomes de domínio, como google.com.br, maroquio.com.br e youtube.com.br, são simplesmente apelidos para endereços IP de um servidor. O fato de usarmos nomes de domínio em vez de endereços IP numéricos é pelo fato dos nomes serem humanamente mais fáceis de se memorizar e mais fáceis de serem associados a entidades do mundo real. Em síntese, além de ser mais fácil memorizar um nome do que um número que pode ter até 12 dígitos, o nome do domínio pode ser composto pelo próprio nome da entidade que ele representa, facilitando ainda mais a memorização. O domínio microsoft.com é um exemplo disso.

Mas como o navegador faz para descobrir o endereço IP de um servidor web a partir de um nome de domínio? Na verdade, o navegador consulta primeiro um servidor DNS. DNS significa *Domain Name System*, ou “Sistema de Nomes de Domínios”. Esse servidor contém uma tabela que mapeia nomes de domínios para endereços IP. Portanto, quando ele recebe um pedido de consulta de um domínio, ele percorre essa tabela e retorna o IP do domínio consultado.

Existem vários servidores DNS espalhados pelo mundo, e essa tabela é atualizada com certa frequência para refletir a inclusão de novos domínios que tenham sido registrados, bem como a remoção de domínios que não estão mais ativos e a alteração de domínios que ocorre quando um site é mudado de um servidor web para outro.

Uma vez que o endereço IP do servidor web tenha sido obtido, uma requisição HTTP é feita ao servidor web correspondente ao IP em questão. Não se assuste com o termo “requisição HTTP”. Isso nada mais é do que o seu navegador fazendo um pedido de algum recurso para o servidor web usando uma linguagem que os dois conhecem bem, que é o protocolo **HTTP**.

A sigla HTTP, que você já deve ter visto no início do endereço de vários sites, quer dizer *Hyper Text Transfer Protocol*, ou, “Protocolo de Transferência de Hipertexto”. Lembra que eu falei que as páginas de um site são hipertextos? Pois é! Por isso o nome do protocolo diz respeito à transferência de hipertexto.

Existe uma forma correta de o navegador solicitar uma página de hipertexto a um servidor web. Essa forma é o que chamamos de “protocolo”, que neste caso é o HTTP. Uma requisição HTTP é composta por vários dados, entre eles, o idioma preferencial do usuário, o nome de domínio onde está o recurso

solicitado e o nome do recurso que está sendo solicitado. O recurso solicitado por ser um arquivo HTML, CSS, Javascript, ou pode ser um arquivo de imagem qualquer, ou, pode ser um nome de recurso que será gerado por uma aplicação web. O código 1.1 mostra um exemplo de cabeçalho HTTP para uma requisição ao servidor da Microsoft.

```
1 GET /produtos HTTP/1.1
2 Host: www.microsoft.com
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/89.0.4389.82 Safari/537.36
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
```

Exemplo de Código 1.1: Exemplo de cabeçalho de uma requisição HTTP.

Cada arquivo necessário para a renderização da página gera uma requisição HTTP. Veja que isso tudo acontece em poucos segundos. Lembre-se também de que, quanto menor é o tempo de carregamento de uma página, menos chance ela tem de ser abandonada pelo usuário. Um dos maiores desafios ao se construir um site é o de reduzir o tempo de carregamento das páginas. Ao longo deste material, veremos algumas técnicas que vão ajudar nessa questão.

Continuando no diagrama da figura 1.4, vamos imaginar que o recurso solicitado é uma página dinâmica. Nesse caso, para gerar a página, alguns dados a mais normalmente são necessários, e esses dados costumam ficar armazenados em um banco de dados. Vamos analisar um exemplo real rapidamente.

A figura 1.5 mostra as páginas com detalhes de 3 vídeos do meu canal no YouTube, cada uma com seu respectivo endereço. Veja que os endereços são bem parecidos. Todos começam com HTTPS, que nada mais é do que uma versão segura do protocolo HTTP. Em seguida vem o nome do domínio, nesse caso, *www.youtube.com*. Depois vem o nome do recurso que estamos acessando, que é a página *watch*; e, por fim, temos uma interrogação (?) seguida de um “V=XXXXXXX”, onde o X pode ser qualquer caractere alfanumérico.

Isso que vem depois da interrogação na URL do vídeo é um parâmetro. Esse parâmetro é o identificador do vídeo que está sendo exibido. É como se fosse o “CPF” do vídeo. Daqui a pouco vamos estudar mais sobre esse identificador. Veja que o estilo da página é o mesmo para os três vídeos. Isso caracteriza bem as páginas geradas dinamicamente. Elas têm um estilo praticamente idêntico, incluindo fonte de letra, cores, leiaute, botões etc., e alguns dados mudam de uma para outra.

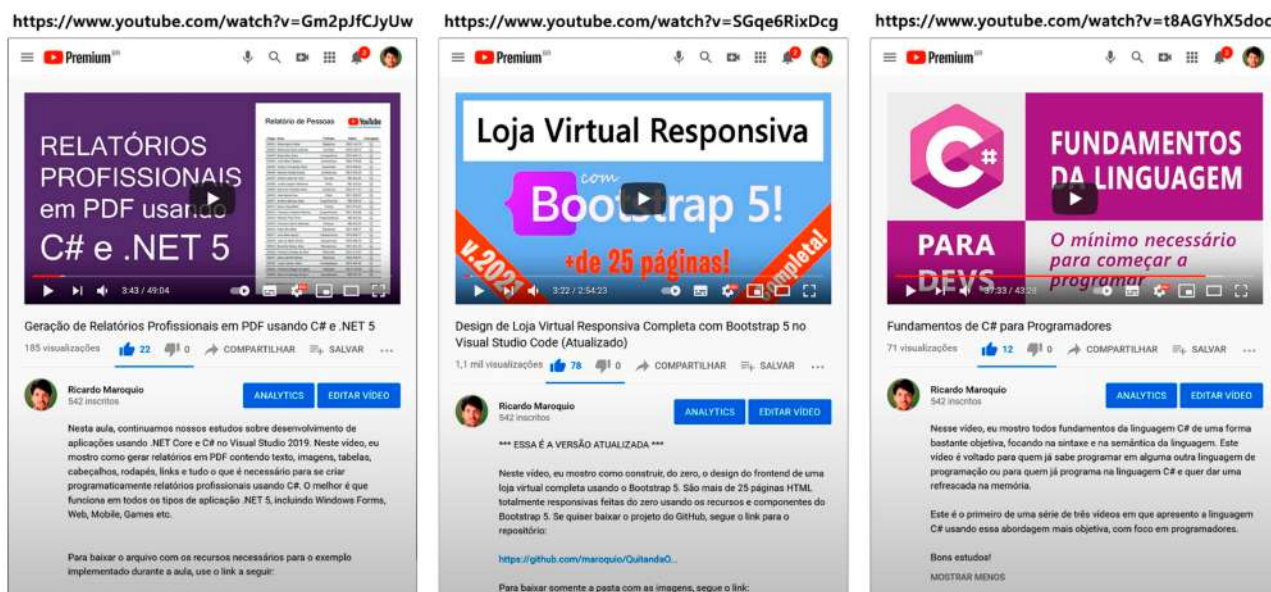


Figura 1.5: Páginas de vídeos específicos do YouTube.

Tente identificar, na figura 1.5, quais dados mudam de uma página para outra. Vamos lá. Temos a imagem de capa do vídeo, o título, a quantidade de visualizações, a data de publicação, a quantidade de likes e dislikes, e a descrição. Esses dados são obtidos de um banco de dados no servidor do YouTube. Se você nunca estudou banco de dados, não se preocupe, porque é algo desnecessário para esta fase do aprendizado. A figura 1.6 mostra os dados dinâmicos contornados por retângulos alaranjados.



Figura 1.6: Páginas de vídeos específicos do YouTube com dados dinâmicos destacados.

Tudo que vem depois de uma interrogação em uma URL é considerado parâmetro. Pode acontecer de uma mesma URL ter vários parâmetros separados pelo símbolo **&**. O valor desse parâmetro “V” funciona como um identificador. Com ele é possível encontrar um determinado vídeo no banco dados de vídeos do YouTube. Se você já estudou banco de dados, já deve ter feito a associação desse parâmetro “V” com a chave primária de uma tabela que guarda os dados dos vídeos.

A aplicação web do YouTube foi programada para usar esse parâmetro e consultar um banco de dados para buscar os dados específicos do vídeo em questão. Em seguida, a aplicação web combinou esses dados com um modelo genérico de página HTML criado por algum designer do YouTube, gerando o conteúdo e enviando de volta ao navegador do usuário através de uma resposta HTTP.

Esse modelo de página HTML já contém as referências para arquivos CSS, Javascript, imagens e o que mais for necessário para renderizar a página do vídeo por completo. Portanto, conforme já estudamos, logo que recebe esse conteúdo HTML, o navegador já faz novas requisições ao servidor do YouTube para buscar os outros arquivos necessários para a renderização da página. Depois de baixar todos os arquivos, a renderização da página é concluída.

1.1.3 Um Pouco Mais Sobre Javascript

Agora vamos voltar a falar do código Javascript que vimos rapidamente. Basicamente, ele desempenha dois papéis em uma página de hipertexto. O primeiro deles é a modificação de conteúdo mediante a ocorrência de algum evento. Por exemplo, se o usuário clicar em um botão, eu quero que determinado menu seja expandido ou que uma imagem sofra um zoom. Isso quer dizer que, com Javascript, você consegue manipular os elementos HTML que estão renderizados em uma página, incluindo seus estilos CSS. Isso pode deixar a página um pouco mais interativa.

O segundo papel desempenhado pelo Javascript é o de carregar conteúdo do servidor web em segundo plano, permitindo atualizar partes da página sem a necessidade de requisitar toda a página novamente. Como exemplo, imagine um site que mostre o placar de jogos de futebol em tempo real, como na figura 1.7. Para que a experiência do usuário seja a melhor possível, é interessante atualizar o placar no máximo a cada 1 segundo, senão o usuário escuta o grito de gol do vizinho e só 30 segundos depois o placar muda para ele, gerando uma péssima experiência.

Acontece que recarregar toda a página novamente a cada segundo é algo computacionalmente custoso, e milhares de usuários fazendo isso ao mesmo tempo sobrecarregaria o servidor. Neste caso, usando Javascript, é possível criar um temporizador que dispare uma função a cada 1 segundo, sendo que essa função faz uma consulta ao servidor web usando o endereço de um recurso que retorna



Figura 1.7: Atualização parcial de página usando Javascript em segundo plano.

somente o placar do jogo em questão. Em seguida, a função vai no elemento HTML que mostra o placar na página e atualiza seu conteúdo com o novo valor do placar recebido do servidor. Isso tudo é realizado em segundo plano pelo navegador e evita o recarregamento total da página.

Apesar de Javascript ser um recurso útil, principalmente para a criação de sites dinâmicos, o foco deste material é a criação de sites estáticos. Na verdade, independentemente do site ser estático ou dinâmico, o código que chega no navegador para ser renderizado tem exatamente as mesmas características, ou seja, tipicamente temos o código HTML para definir o conteúdo e o código CSS para definir a aparência do site. Eventualmente, podemos ter código Javascript para desempenhar algum dos papéis que mencionei. Isso quer dizer que, para criar sites dinâmicos, o primeiro passo é aprender a criar sites estáticos, que é o nosso propósito aqui.

1.2 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 1.1 O que é e para que serve um servidor web?

Exercício 1.2 O que é e para que serve um navegador web? ■

Exercício 1.3 O que é e para que serve o protocolo HTTP? ■

Exercício 1.4 Quais são os 3 tipos de códigos básicos de uma página de hipertexto? Explique para que serve cada um dos tipos. ■

Exercício 1.5 Quantos arquivos CSS e Javascript podem estar vinculados a uma mesma página de hipertexto? Justifique sua resposta. ■

Exercício 1.6 É possível diferentes páginas de hipertexto terem vínculo com os mesmos arquivos Javascript e CSS? Justifique sua resposta. ■

Exercício 1.7 Descreva a diferença entre uma página de hipertexto estática e uma dinâmica, pontuando as vantagens e desvantagens de cada um dos dois tipos. ■

Exercício 1.8 Qual é o caminho percorrido por uma requisição HTTP para obter uma página de hipertexto estática? ■

Exercício 1.9 Qual é o caminho percorrido por uma requisição HTTP para obter uma página de hipertexto dinâmica? ■

Exercício 1.10 Quais as vantagens da atualização parcial de conteúdo em segundo plano em uma página de hipertexto usando Javascript? ■

1.3 Considerações Sobre o Capítulo

Este capítulo apresentou os fundamentos do funcionamento de um site. Foram apresentados os papéis das linguagens HTML, CSS e Javascript, a diferença de páginas web estáticas e dinâmicas, o conceito e os detalhes básicos de uma requisição HTTP, e um exemplo útil de uso da linguagem Javascript para melhorar a experiência de visualização de um site com atualização parcial em tempo real. No capítulo seguinte, estudaremos a estrutura básica de documentos HTML e CSS.

2. Documentos HTML e CSS

Este capítulo aborda a estrutura básica dos documentos HTML e CSS. Veremos como é a hierarquia de elementos de um documento HTML, como se cria um novo elemento, o que são *tags*, atributos e valores de atributos. Veremos também como é o código de um documento CSS básico, com seletores e propriedades para formatação visual de elementos específicos.

2.1 Estrutura de Uma Página HTML

Uma página HTML também pode ser chamada de documento HTML. O exemplo 2.1 mostra o código de um documento HTML de uma página bem simples. Vale ressaltar que os números das linhas foram colocados só para facilitar a explicação. Eles não fazem parte do código do documento HTML.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3    <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Título do Site</title>
7    </head>
8    <body>
9      <h1>Título da Página</h1>
10     <p>Id culpa velit deserunt ullamco veniam. Occaecat aute voluptate
11       sint ut magna. Veniam consequat pariatur dolore eu sit ea.</p>
12   </body>
13 </html>
```

Exemplo de Código 2.1: O código de uma página HTML simples.

Agora nós vamos analisar detalhadamente os elementos que fazem parte desse documento HTML. Você vai precisar desse conhecimento para todo o restante de sua jornada como desenvolvedor web, portanto, preste bastante atenção.

Os documentos HTML possuem uma estrutura básica que está presente em todos eles. Um documento HTML é composto por elementos. Alguns tipos de elementos podem conter outros elementos. Quando possuem essa capacidade, esses elementos podem ser chamados de elementos contêiner.

Um elemento contêiner é delimitado por uma tag de abertura e uma tag de fechamento. Temos alguns elementos contêineres aqui no exemplo 2.1. Vamos ver quais são. Temos o elemento `html`, que inicia na linha 2 e termina na linha 13; o elemento `head`, que inicia na linha 3 e termina na linha 7; o elemento `body`, que começa na linha 8 e termina na linha 12; o elemento `title`, que começa e termina na linha 6; o elemento `h1`, com início e fim na linha 9; e o elemento `p`, que inicia na linha 10 e termina na linha 11.

Quando um elemento não pode conter outros elementos, ele é chamado de elemento simples e possui apenas a tag de abertura. Isso acontece com o elemento `DOCTYPE`, na linha 1, e com os elementos `meta`, nas linhas 4 e 5. Veja que são dois elementos distintos.

Uma tag, por sua vez, corresponde a uma ou mais palavras envolvidas pelos símbolos de menor e maior (`<tag>`). A tag de abertura pode conter mais de uma palavra, sendo que a primeira palavra corresponde ao tipo de elemento `html` dessa tag e as demais palavras nós veremos daqui a pouco.

Na tag de abertura da linha 4, por exemplo, a primeira palavra é `meta`, ou seja, esse é um elemento `html` do tipo `meta`. A tag de fechamento sempre terá somente o nome do tipo de elemento `html` precedido por uma barra (`</tag>`). Vamos identificar as tags de fechamento desse documento HTML.

No final da linha 6 temos o fechamento de um elemento do tipo `title`. No final da linha 7 temos o fechamento do elemento `head`. No final da linha 9 temos o fechamento do elemento do tipo `h1`. No final da linha 11 temos o fechamento de um elemento do tipo `p`. Na linha 12 temos o fechamento do elemento `body`, e na linha 13 temos o fechamento do elemento `html`.

Agora que já sabemos identificar as tags de abertura e de fechamento dos elementos, vamos analisar um pouco melhor os tipos de elemento presentes nessa página e suas tags de abertura. Na linha 1, temos o elemento `DOCTYPE`, que é um elemento simples, ou seja, ele tem somente a tag de abertura. O elemento `DOCTYPE` existe apenas para indicar ao navegador que esse é um documento HTML.

O elemento `DOCTYPE` sempre aparece no início do documento e seu uso é recomendado, apesar de sua ausência não causar nenhum erro de renderização de página nos navegadores atuais. Veja que dentro da tag `DOCTYPE` temos mais uma palavra, que é `html`. Essa palavra é um atributo. Portanto, elementos podem conter atributos em suas tags de abertura. Nunca teremos atributos dentro de uma tag de fechamento.

Esse ponto de exclamação que vem antes do nome do elemento `DOCTYPE` é um resquício histórico usado em versões antigas. Somente esse elemento inicia com esse ponto de exclamação. O elemento `DOCTYPE` vai aparecer desse jeito em qualquer documento HTML atual, porque esse é o padrão para indicar que se trata de um documento HTML 5.

Na linha 2, temos a tag de abertura de um elemento do tipo `html`. Observe que esse elemento é fechado na linha 13. Portanto, todos os elementos que estão entre as linhas 2 e 13 são filhos, netos, bisnetos ou algum outro descendente do elemento `html`.

Veja também que o elemento `html` possui o atributo `lang` em sua tag de abertura, mas esse atributo tem uma forma diferente do atributo `html` presente na tag da linha 1. Acontece que o atributo da linha 2 possui um valor, enquanto o atributo da linha 1 apenas existe.

Em algumas poucas situações, a simples existência de um atributo em uma tag é suficiente para se definir alguma configuração. Na maioria das situações, os atributos são acompanhados de um valor, como é o caso do atributo da linha 2. Nesse caso, o atributo `lang` tem o valor `pt-br`. O valor do atributo `lang` da tag `html` define o idioma predominante do conteúdo do documento HTML em questão.

Isso é importante para que o navegador exiba caracteres internacionais corretamente e também para que ferramentas de busca, como o Google, identifiquem mais facilmente que esse site tem mais relevância para pessoas que falam o idioma Português do Brasil, nesse caso.

Um documento HTML só pode ter 1 único elemento do tipo `html`. Ele é o elemento ancestral de todos os demais elementos da página, exceto do elemento `DOCTYPE`. Todos os elementos que estiverem entre as tags de abertura e de fechamento de um elemento contêiner, são considerados elementos filhos desse elemento contêiner. Elementos que são filhos de filhos são considerados elementos netos, e assim por diante.

Veja que a linha 3 tem um elemento contêiner do tipo `head`, que é fechado na linha 7. Assim como acontece com o elemento `html`, um documento HTML também só pode ter 1 elemento do tipo `head`. O elemento `head` contém metainformações relacionadas ao documento em questão, ou seja, os elementos dentro de `head` não correspondem a conteúdo, mas sim a informações sobre o próprio documento HTML, portanto, não são renderizados pelo navegador.

Alguns elementos de metainformações até podem ter influência no visual de elementos de conteúdo que são renderizados pelo navegador, mas eles próprios jamais serão renderizados pelo navegador. Dentro do elemento `head`, na linha 4, temos um elemento de metainformação que indica qual é o conjunto de caracteres que deve ser utilizado para mostrar textos nessa página. Nesse caso, está sendo usado o `utf-8`, que é o padrão para idiomas ocidentais.

O elemento de metainformação da linha 5, que também está dentro de `head`, parece meio assustador, mas não precisa se preocupar em entendê-lo plenamente nesse momento. A princípio, basta saber que ele define uma configuração para a exibição correta da página em dispositivos com diferentes tamanhos de tela.

Na linha 6 temos o elemento `title`. Esse elemento define o título da página mostrado na aba do navegador. É por isso que esse elemento fica dentro de `head`, porque ele não é considerado conteúdo. Vale observar que não faz sentido colocar dois ou mais elementos `title` em um documento HTML. Portanto, só temos 1 elemento `title` por documento.

Agora vamos ao elemento da linha 8. É dentro dele que tudo acontece. O elemento `body`, que é fechado na linha 12 nesse caso, envolve todos os elementos de conteúdo de um documento HTML. Assim como os elementos `html`, `head` e `title`, um documento HTML só tem um elemento `body`. Nesse caso nós temos simplesmente um elemento do tipo `h1`, que representa um título, e um elemento do tipo `p`, que representa um parágrafo.

Ao longo do curso, veremos todos os tipos de elementos HTML que podem ser colocados dentro do elemento `body`, incluindo vários níveis de título, parágrafos, tabelas, links, imagens, formulários, entre outros. Vamos lembrar alguns pontos importantes que aprendemos com esse documento HTML. Com exceção dos elementos entre as linhas 9 e 11, todos os demais elementos desse exemplo estão presentes em todas as páginas web.

Em síntese, um documento HTML é composto por um único elemento `html`, que por sua vez contém um elemento `head` e um elemento `body`. O elemento `head`, por sua vez, contém alguns elementos de metainformações e um elemento do tipo `title`. Ao longo do curso, veremos outros elementos que podem aparecer dentro do elemento `head`. Já o elemento `body` é o que contém todos os elementos correspondentes ao conteúdo da página, os quais também veremos em capítulos posteriores.

O exemplo 2.2 mostra o código de uma página HTML realmente simples, enquanto a figura 2.1 mostra como esse código é renderizado no navegador. O importante aqui é que você perceba que um documento HTML escrito puramente com código HTML, geralmente tem essa aparência sem graça. Se quisermos melhorar a aparência desse documento, temos que fazer uso de CSS. Apesar de ainda estarmos na parte de HTML, a próxima seção apresenta como é a estrutura básica de um documento CSS e como ele se relaciona com um documento HTML.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Título do Site</title>
7  </head>
8  <body>
9      <header>
10         <h1>Título da Página</h1>
11         <hr>
12     </header>
13     <main>
14         <h2>Subtítulo 1</h2>
15         <p>Cupidatat esse ipsum ad occaecat. Incididunt fugiat culpa ame
16             illum veniam reprehenderit reprehenderit in ad.</p>
17         <h2>Subtítulo 2</h2>
18         <p>Deserunt proident amet Lorem tempor fugiat ad elit laborum sit
19             officia consectetur.</p>
20     </main>
21     <footer>
22         <hr>
23         <p>
24             Copyright &copy; 2021 :: Todos os direitos reservados
25         </p>
26     </footer>
27 </body>
28 </html>
```

Exemplo de Código 2.2: Página HTML simples com cabeçalho, corpo e rodapé.

2.2 Estrutura de Um Documento CSS

O exemplo 2.3 apresenta um código CSS bem simples. Nós não nos aprofundaremos no estudo de CSS agora, porque o foco da parte I deste material é HTML. Entretanto, é importante você já saber como um documento CSS se relaciona com um documento HTML.

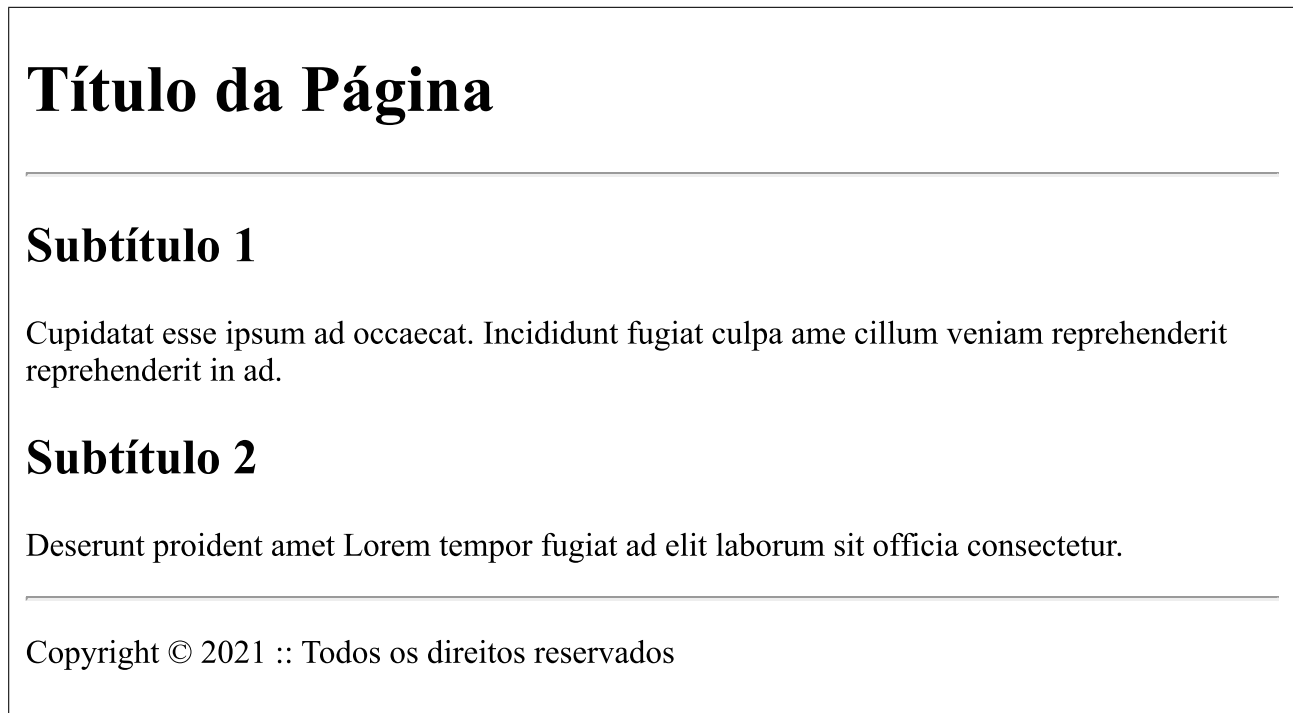


Figura 2.1: Renderização do código HTML do exemplo 2.2.

```
1  body {
2      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
3      padding: 50px;
4      background-color: lightgray;
5  }
6  h1 {
7      color: darkgray;
8      text-align: center;
9      border: solid 1px black;
10 }
11 p {
12     text-align: justify;
13     font-size: 16px;
14 }
```

Exemplo de Código 2.3: Documento CSS simples.

Um documento CSS contém uma série de seletores. Um **seletor** é composto por um **identificador** e por um bloco de **propriedades**, sendo que esse bloco é delimitado por abertura e fechamento de chaves. Esse documento CSS tem 3 seletores: o seletor `body`, que vai da linha 1 à linha 5; o seletor `h1`, da linha 6 a 10; e o seletor `p`, da linha 11 a 14.

O identificador de um seletor deve ser único no documento. Esse identificador pode ser criado de diferentes formas, sendo que cada forma atinge um ou vários elementos HTML. Nesse caso, estamos usando seletores por tipo, ou seja, os nomes dos seletores usados nesse exemplo coincidem com alguns tipos de elementos HTML que vimos na seção anterior.

Um seletor cujo identificador coincide com o nome de um tipo de elemento define o estilo do tipo de elemento HTML em questão. Existem outras formas de seletores, como o seletor por ID e o seletor por classes. Também existem seletores que são combinações desses 3 tipos. Mais adiante, teremos um tópico só para tratar de seletores, portanto, não se preocupe em tentar entender completamente o funcionamento dos seletores agora.

Nesse código CSS do exemplo 2.3, de acordo com as propriedades definidas pelo seletor `h1` que inicia na linha 6, o elemento do tipo `h1` terá cor de fonte cinza escura, alinhamento de texto centralizado e uma borda sólida de 1 pixel na cor preta.

Os outros seletores definem outras propriedades visuais para os elementos `body` e `p`. Algumas propriedades definidas para elementos do tipo contêiner podem afetar os elementos filhos, como é o caso da propriedade `font-family` do seletor `body`, na linha 2. Esse seletor define uma lista de fontes de letra para que o documento use a primeira que estiver disponível dessa lista. Como mencionei, existem várias formas de se construir seletores e várias propriedades que podem ser configuradas. Estudaremos mais sobre isso ao longo deste material.

2.2.1 Associando um Documento CSS a um Documento HTML

Esta seção mostra como associar o código CSS do exemplo 2.3 ao código HTML do exemplo 2.2. Vale ressaltar que esta não é a única forma de se aplicar definições CSS a um documento HTML, porém, é a forma mais usual. Outras formas de associação são mostradas na parte II deste material. A associação de um documento CSS a um documento HTML é algo relativamente simples de se fazer. O exemplo 2.4 mostra um trecho de código que contém a associação de um documento CSS chamado “estilos.css” a um documento HTML qualquer.

```
1  <head>
2      ...
3      <title>Título do Site</title>
4      <link rel="stylesheet" href="estilos.css">
5  </head>
```

Exemplo de Código 2.4: Associação de documento CSS a um documento HTML.

Como pode ser visto na linha 4, basta usar um elemento do tipo `link` dentro do elemento `head` da página, e definir o valor do atributo `rel` como `stylesheet` e o valor do atributo `href` com o nome do arquivo CSS que deseja vincular a esse documento HTML. Nesse caso, o nome arquivo é “estilos.css”. Vale ressaltar é possível ter vários documentos CSS associados a um único documento HTML, bem como um mesmo documento CSS associado a vários documentos HTML, e isso é bem comum.

Usar um único arquivo CSS (ou um conjunto de arquivos CSS) para todas as páginas de um site pode facilitar bastante a alteração do visual inteiro do site sem muito esforço, como veremos mais adiante na segunda parte deste material. Agora vamos ver na figura 2.2 o resultado da associação desse documento CSS ao documento HTML simples cujo código corresponde ao do exemplo 2.2.

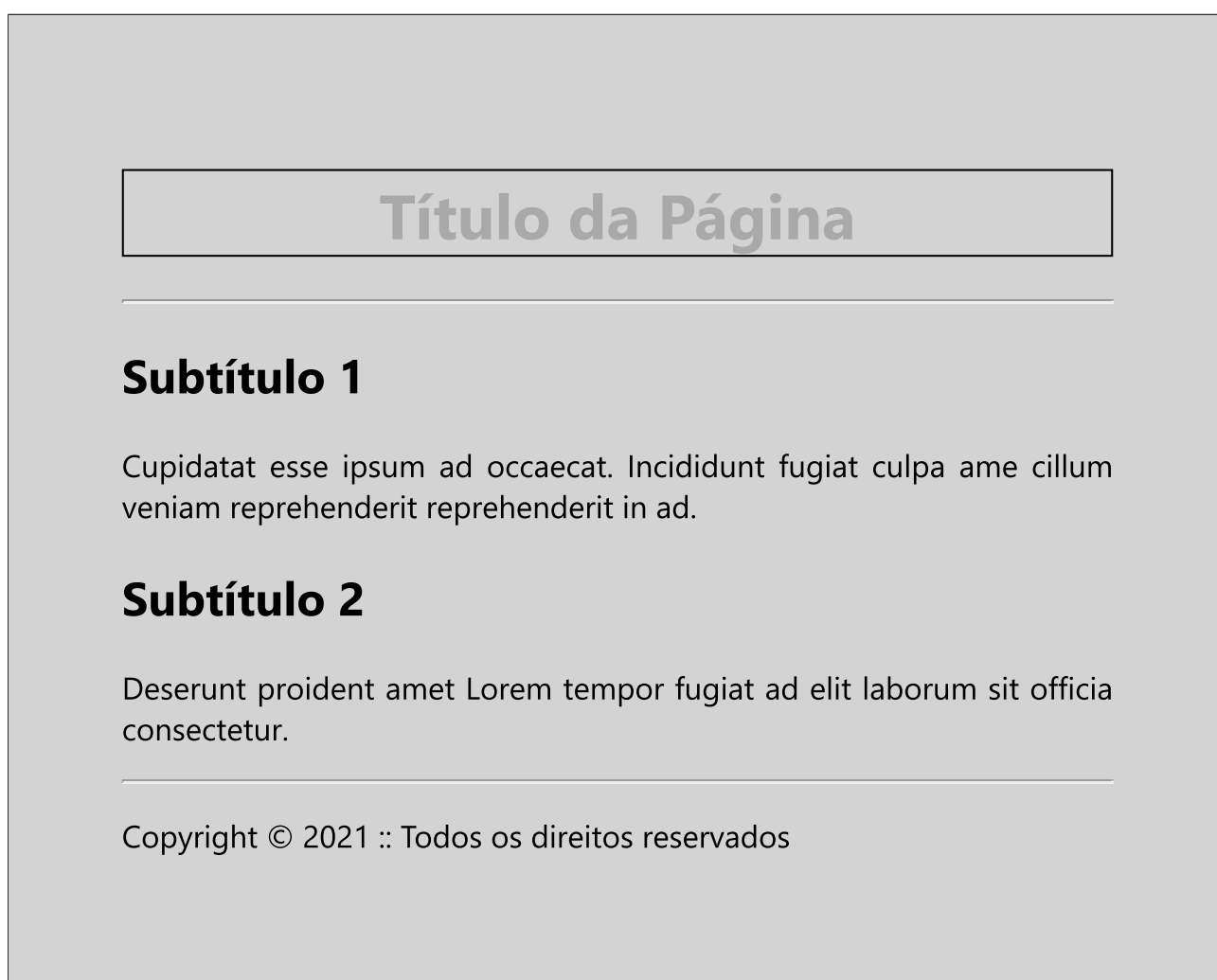


Figura 2.2: Renderização do código HTML do exemplo 2.2 com o CSS do exemplo 2.3 associado.

Certamente a renderização com a aplicação do CSS neste caso não melhorou muito, mas o objetivo aqui é só de mostrar que o CSS afetou o visual da página. Com certeza muita coisa poderia ser configurada via CSS neste documento HTML para deixá-lo com o visual bastante profissional. Veremos os recursos necessários para isso ao longo deste material.

2.3 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 2.1 Qual é a função da metainformação com atributo `charset="UTF-8"` em uma página HTML? ■

Exercício 2.2 Qual é a função da metainformação com atributo `name="viewport"` em uma página HTML? ■

Exercício 2.3 Qual é a função do elemento `<title>` em uma página HTML? ■

Exercício 2.4 Qual é a função do atributo `lang="pt-br"` do elemento HTML? ■

Exercício 2.5 Qual é o elemento HTML que contém todos os demais elementos de uma página? ■

Exercício 2.6 Qual é o elemento HTML que contém elementos com metainformações sobre uma página web? ■

Exercício 2.7 Qual é o elemento HTML que contém todos os elementos de conteúdo de uma página web? ■

Exercício 2.8 Qual é o elemento HTML que permite associar um documento CSS a uma página web na seção de metainformações? ■

Exercício 2.9 Indique se o trecho de código a seguir é HTML ou CSS:

```
<head>
<title>Meu site</title>
<link rel="stylesheet" href="estilos.css">
</head>
```

 ■

Exercício 2.10 Indique se o trecho de código a seguir é HTML ou CSS:

```
h1 {  
  color: darkgray;  
  text-align: center;  
  border: solid 1px black;  
}
```

2.4 Considerações Sobre o Capítulo

Este capítulo apresentou a estrutura básica dos documentos HTML e dos documentos CSS. Vimos quais são os principais elementos comumente presentes na hierarquia de elementos da maioria dos documentos HTML. Vimos também como criar e um documento CSS com propriedades que configuram elementos de acordo com seu tipo e como associar tal documento a um documento HTML qualquer. No capítulo seguinte, iniciaremos o estudo da linguagem HTML em si e seus elementos textuais básicos.



3. Textos

Neste capítulo nós veremos os elementos HTML textuais básicos. Além de aprender a usar vários tipos de elementos textuais básicos, vamos também conhecer o significado que cada tipo de elemento HTML carrega com ele, o que também é conhecido como **semântica** do elemento.

Existem elementos HTML específicos para títulos, parágrafos, códigos-fontes, citações, links e outras finalidades. Apesar disso, é possível (mas não recomendado) usar um elemento de parágrafo desempenhando o papel de um título, por exemplo, no que diz respeito ao visual. O problema é que o significado desse conteúdo não estará coerente, porque você estará mostrando um título em um elemento que deveria mostrar o texto de um parágrafo.

Esse tipo de incoerência pode impedir que um site seja compreendido e indexado corretamente por ferramentas de busca e pode impactar negativamente na acessibilidade de uma página. Portanto, é importante que se utilize os elementos HTML para os propósitos definidos por seus significados.

Como mencionei, essa questão está relacionada com a semântica dos elementos. Sempre que ouvir o termo **semântica**, lembre-se de que isso diz respeito ao significado transmitido por cada tipo de elemento HTML. Visto isso, vamos começar a estudar os elementos básicos da HTML para exibição de conteúdo textual.

3.1 Títulos e Subtítulos

Vamos começar pelos elementos responsáveis pela adição de títulos e subtítulos em uma página. Os elementos são o `h1`, `h2`, `h3`, `h4`, `h5` e `h6`. Quanto à semântica, todos esses elementos correspondem a títulos, mas os de nível menor são mais importantes do que os títulos de nível maior. Portanto, quanto maior o nível, menor sua importância semântica.

Nós vamos iniciar o estudo sobre o código dos elementos textuais básicos criando um documento HTML correspondente a um artigo fictício, contendo apenas alguns títulos e parágrafos e, posteriormente, vamos incrementar esse documento com outros elementos. O exemplo 3.1 mostra o código inicial desse documento HTML.

```
1  <!DOCTYPE>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Elementos Textuais Básicos</title>
7  </head>
8  <body>
9      <h1 id="titulo">Elementos Textuais Básicos</h1>
10     <hr>
11     <p>Cupidatat do veniam reprehenderit cillum sunt. Est quis mollit incididunt
12         voluptate incididunt eiusmod quis veniam. Non nulla officia voluptate.</p>
13     <p>Et magna id sunt commodo ea. Non occaecat ex duis officia laborum pariatur
14         proident laborum sit ullamco fugiat anim aliquip. Incidunt in amet ad.</p>
15     <!-- Este é um comentário -->
16     <p>Lorem velit laborum irure labore ad dolor aliquip minim elit laboris est.
17         Ex Lorem consequat culpa pariatur esse id laboris ex aute est. Nostrud</p>
18     <h2 id="subtitulo">Subtítulo Qualquer</h2>
19     <p>Cupidatat do veniam reprehenderit cillum sunt. Est quis mollit incididunt
20         voluptate incididunt eiusmod quis veniam. Non nulla officia voluptate.</p>
21 </body>
22 </html>
```

Exemplo de Código 3.1: Elementos textuais básicos.

Este código HTML do exemplo 3.1 corresponde a uma página web bem simples. Na linha 1, o elemento `<!DOCTYPE>` indica o tipo de documento que está sendo usado. O valor vazio (sem atributos) indica que este é um documento HTML5, que é a última versão da HTML. Na linha 2, o elemento `<html lang="pt-br">` é o elemento raiz de todo o documento HTML. O atributo `lang` especifica o idioma da página, que, neste caso, é o Português do Brasil. Na linha 3, o elemento `<head>` contém informações sobre a página, como o título da página e o tipo de codificação de caracteres. Na linha 4, o elemento `<meta charset="UTF-8">` especifica o tipo de codificação de caracteres que está sendo usado na página. "UTF-8" é uma codificação de caracteres amplamente utilizada e suportada que permite a exibição de caracteres de diferentes idiomas. A linha 5 apresenta o elemento `<meta name="viewport" content="width=device-width, initial-scale=1.0">`, que controla como a página é exibida

em dispositivos móveis. A largura da página será igual à largura do dispositivo e a escala inicial será igual a 1.0. Na linha 6, o elemento `<title>` define o título da página, que é exibido na guia do navegador.

Entre as linhas 8 e 21 estão os elementos de conteúdo da página, iniciado pelo elemento `<body>`. Na linha 9, o elemento `<h1 id="titulo">Elementos Textuais Básicos</h1>` é um elemento de cabeçalho de nível 1 que fornece um título para a página. O atributo `id` fornece um nome único para este elemento, que pode ser usado para referenciar este elemento em outros lugares na página, principalmente em links e códigos Javascript. Na linha 10, o elemento `<hr>` cria uma linha horizontal na página. Nas linhas 11, 13, 16 e 19 temos elementos `<p>`, que representam um parágrafo de texto. O código da linha 15, `<!-- Este é um comentário -->`, é um comentário HTML, ou seja, um texto que não será exibido na página. Comentários são usados para adicionar notas ou informações úteis sobre algo presente no código-fonte para facilitar o entendimento de outra pessoa que leia este código. Na linha 18, o elemento `<h2 id="subtitulo">Subtítulo Qualquer</h2>` é outro elemento de cabeçalho, porém, de nível 2, que fornece um subtítulo para a página. Ele é menos relevante e visualmente menor que o `h1` e é usado para se criar seções secundárias no documento. A figura 3.1 mostra o documento HTML do exemplo 3.1 renderizado.

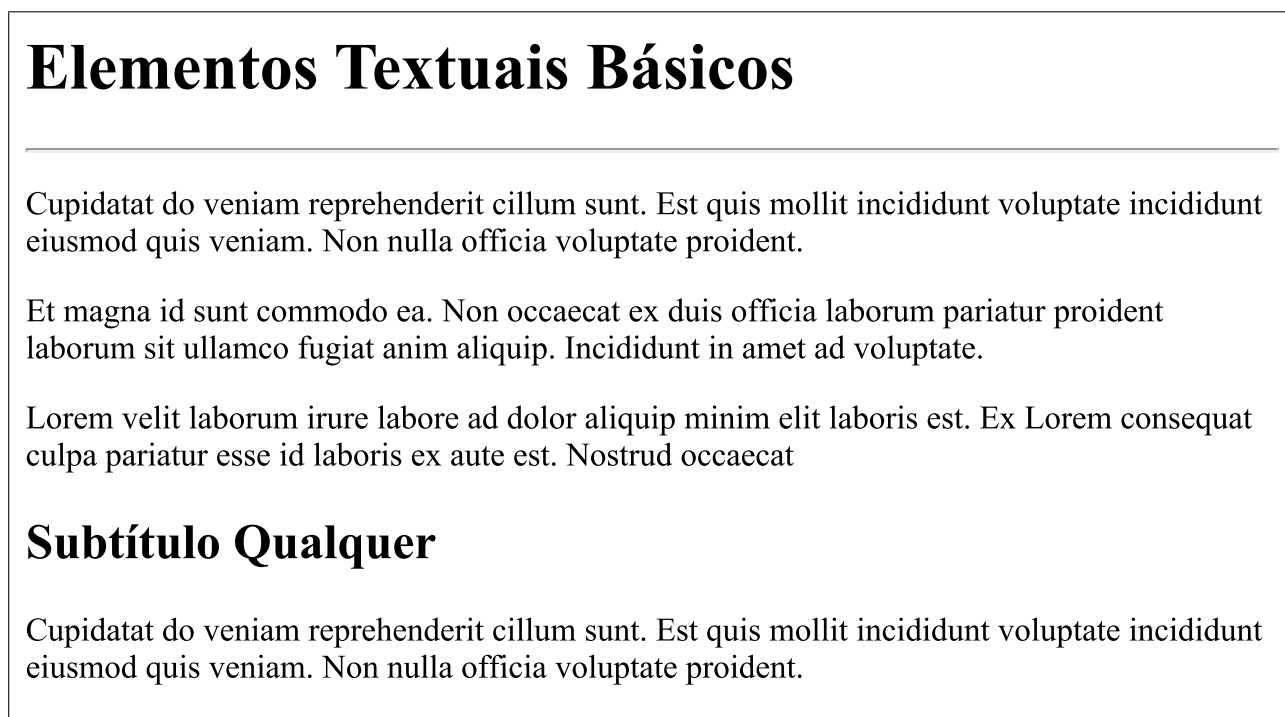


Figura 3.1: Renderização do código HTML do exemplo 3.1.

3.2 Parágrafos e Quebras

O elemento `<p>` é usado para definir um parágrafo em uma página HTML. O elemento `<p>` é um elemento de bloco e é usado para agrupar um bloco de texto em um único parágrafo. Elementos de bloco são aqueles que ocupam um bloco horizontal inteiro, não admitindo vizinhos laterais. O elemento `<p>` é importante porque ajuda a organizar e estruturar o conteúdo de uma página HTML. Ao usar o elemento `<p>`, você pode separar diferentes ideias e pensamentos em parágrafos individuais, o que torna o texto mais fácil de ler e entender.

Por padrão, espaços, tabulações e as quebras de linha inseridas pelo pressionamento da tecla *Enter* são condensados em um único caractere de espaço durante a renderização da página. Esse comportamento ocorre em qualquer conteúdo presente em um elemento HTML, exceto no elemento `<pre>`, apresentado mais adiante. Contudo, se você quiser forçar uma quebra de linha, você pode usar um elemento `
`. O elemento `<hr>` também insere uma quebra, porém, adiciona uma linha horizontal dividindo o conteúdo em duas partes. O elemento `<hr>` pode ser interessante para se criar separação de seções em um documento. O código 3.2 apresenta um exemplo de uso de parágrafos e quebras e a figura 3.2 mostra a renderização desse código.

```
1 <p>Este é um parágrafo que contém algum texto. Aqui está uma quebra de linha:<br>
2   Este é outro texto que aparece em uma nova linha. Aqui está outra quebra de linha:
3   <br><br> Este é outro texto que aparece duas linhas abaixo. O elemento <hr>
4   pode ser usado para adicionar uma linha horizontal:
5 </p>
6 <hr>
7 <p>Este é outro parágrafo que contém algum texto. Aqui está outra quebra de linha:
8   <br> Este é outro texto que aparece em uma nova linha. Aqui está outra quebra de
9   linha: <br><br> Este é outro texto que aparece duas linhas abaixo.
10 </p>
```

Exemplo de Código 3.2: Elementos textuais básicos.

3.3 Textos Pré-Formatados

Os elementos `<pre>` e `<code>` são usados em HTML para apresentar texto pré-formatado. Nesta seção, veremos como e quando usar cada um deles, levando em conta as características semânticas dos elementos, uma vez que o padrão visual de ambos é bastante semelhante.

Quando você usa o elemento `<pre>`, o navegador preserva a formatação original do texto, incluindo espaços em branco, quebras de linha e tabulações. Isso é útil quando você precisa apresentar texto que precisa ser exibido exatamente como está no código-fonte, sem ajustes automáticos de formatação.

Este é um parágrafo que contém algum texto. Aqui está uma quebra de linha:
Este é outro texto que aparece em uma nova linha. Aqui está outra quebra de linha:

Este é outro texto que aparece duas linhas abaixo. O elemento `<hr>` pode ser usado para adicionar uma linha horizontal:

Este é outro parágrafo que contém algum texto. Aqui está outra quebra de linha:
Este é outro texto que aparece em uma nova linha. Aqui está outra quebra de linha:

Este é outro texto que aparece duas linhas abaixo.

Figura 3.2: Renderização do código HTML do exemplo 3.2.

Além disso, por padrão, o elemento `<pre>` usa fonte monoespaçada, que são aquelas fontes em que todos os caracteres ocupam a mesma largura em pixels, como a fonte “Courier New”. O exemplo 3.3 mostra o código de um exemplo de texto pré-formatado usando o elemento `<pre>`, enquanto a figura 3.3 mostra a renderização desse exemplo.

```
1 <pre>
2     Este é um exemplo de texto
3     pré-formatado. Note como
4     os espaços em branco e as
5     quebras de linha são preservados.
6 </pre>
```

Exemplo de Código 3.3: Texto pré-formatado com elemento `<pre>`.

```
Este é um exemplo de texto
pré-formatado. Note como
os espaços em branco e as
quebras de linha são preservados.
```

Figura 3.3: Renderização do texto pré-formatado do exemplo 3.3.

O elemento `<code>` é usado em HTML para indicar um bloco de código ou texto que representa um comando, uma função ou outra representação de código-fonte. O navegador geralmente apresenta o texto dentro do elemento `<code>` em uma fonte monoespaçada, para destacá-lo como código. O exemplo 3.4 mostra o código HTML de um exemplo de texto de código-fonte usando o elemento `<code>` e a figura 3.4 mostra a renderização do exemplo.

A importância do elemento `<code>` é que ele permite que o autor da página identifique claramente um texto como código-fonte, o que ajuda a melhorar a legibilidade e acessibilidade do conteúdo. Além disso, ele permite que o navegador apresente o texto de forma diferenciada, o que pode ser útil para

```
1 | <p>Você pode usar o seguinte código em JavaScript para exibir uma mensagem:</p>
2 | <code>
3 |     alert("Olá, mundo!");
4 | </code>
```

Exemplo de Código 3.4: Texto de código-fonte pré-formatado com elemento `<code>`.

Você pode usar o seguinte código em JavaScript para exibir uma mensagem:
alert("Olá, mundo!");

Figura 3.4: Renderização do texto de código-fonte do exemplo 3.4.

destacar o código em relação ao texto normal na página. É possível usar o elemento `<code>` em combinação com outros elementos, como o `<pre>`, para facilitar a organização do código com suas indentações.

3.4 Links

O elemento `<a>` é usado em HTML para criar links, permitindo que o usuário navegue para outras páginas do próprio site, para outros recursos na web ou para elementos da própria página. É um dos elementos mais importantes e amplamente usados em HTML. Portanto, existem três tipos principais de links que você pode criar com o elemento `<a>` em HTML: links internos, links externos e links ancorados.

Links internos são links que apontam para outras páginas do mesmo site. Por exemplo, se você tem um site com uma página inicial e uma página de contato, pode criar um link na página inicial que leva o usuário para a página de contato. Para criar um link interno, você especifica o caminho relativo da página de destino no atributo `href`. O código 3.5 mostra um exemplo de link interno que aponta para uma página do próprio site chamada "contato.html".

```
1 | <p>Visite a <a href="contato.html">página de contato</a> para mais informações.</p>
```

Exemplo de Código 3.5: Link para interno apontando para uma página do próprio site.

Links externos são links que apontam para recursos fora do seu site. Por exemplo, você pode criar um link que leva o usuário para o site do Google. Para criar um link externo, você especifica o endereço completo da página de destino no atributo `href`. O código 3.6 mostra um exemplo de link externo que aponta para a página inicial do Google. Observe que o endereço completo do destino é passado, incluindo o nome de domínio, enquanto no link interno passamos somente o nome do recurso buscado.

```
1 | <p>Visite o site do <a href="https://www.google.com">Google</a>  
2 |   para mais informações.</p>
```

Exemplo de Código 3.6: Link externo apontando para a página do Google.

Links ancorados são links que apontam para elementos dentro da própria página. Por exemplo, você pode ter um link na parte inferior da página que leva o usuário para uma seção específica na parte superior da página. Para criar um link ancorado, você especifica o identificador (atributo `id`) do elemento de destino no atributo `href` com o prefixo `#`. O código 3.7 mostra um exemplo de link ancorado que aponta para um elemento da própria página cujo identificador é `secao1`.

```
1 | <h2 id="secao1">Seção 1</h2>  
2 | ...  
3 | <p>Volte para a <a href="#secao1">seção 1</a> para mais informações.</p>
```

Exemplo de Código 3.7: Link ancorado apontando para um elemento da própria página.

Os links internos, externos e ancorados são importantes porque permitem que o usuário navegue facilmente pelo seu site e pela web, conectando páginas e recursos. Além disso, eles são uma ótima maneira de tornar o conteúdo da página acessível para usuários com necessidades especiais, como usuários de leitor de tela. A figura 3.5 mostra a renderização dos links dos exemplos anteriores.

Visite a [página de contato](#) para mais informações.

Visite o site do [Google](#) para mais informações.

Seção 1

Lorem ipsum, dolor sit amet consectetur adipisicing elit. Labore non facere corporis, iste expedita dolorem in. Quidem pariatur eaque similique.

Volte para a [seção 1](#) para mais informações.

Figura 3.5: Renderização dos links dos exemplos desta seção.

3.4.1 Local de Abertura do Link

O atributo `target` é utilizado no elemento `<a>` para definir o local onde o recurso vinculado ao link deve ser aberto. Este atributo é útil, por exemplo, quando se deseja que um link seja aberto em uma janela ou guia diferente do navegador em que a página original está sendo exibida. O valor do atributo `target` pode ser um dos seguintes:

- a) `_self`: este é o valor padrão e indica que o link deve ser aberto na mesma janela ou guia do navegador em que a página atual está sendo exibida;
- b) `_blank`: este valor indica que o link deve ser aberto em uma nova janela ou guia do navegador;
- c) `_parent`: este valor indica que o link deve ser aberto na janela pai da janela atual. Esse valor é útil quando você está usando *frames* em sua página, algo pouco usual nos dias atuais;
- d) `_top`: Este valor indica que o link deve ser aberto na janela principal do navegador, substituindo todas as outras janelas ou guias abertas.

O exemplo 3.8 mostra como usar o atributo `target` para abrir um link em uma nova guia do navegador.

```
1 | <a href="https://www.exemplo.com" target="_blank">Exemplo.com</a>
```

Exemplo de Código 3.8: Link com atributo `target` para definição de local de abertura.

É importante lembrar que, ao usar o atributo `target`, você pode estar criando uma experiência de usuário diferente do esperado, já que o usuário pode ficar confuso ao abrir várias guias ou janelas do navegador sem saber como voltar à página anterior. Por isso, é recomendável usar esse atributo com moderação e fornecer ao usuário a opção de escolher se deseja ou não abrir um link em uma nova guia ou janela do navegador. Além disso, é importante lembrar que o uso do atributo `target` pode afetar a acessibilidade da página, e a melhor prática é sempre criar links que sejam fáceis de usar para todos os usuários.

3.5 Formatação de Texto *Inline*

Os elementos *inline* são elementos de formatação de texto em HTML que afetam apenas o texto dentro deles, sem afetar o layout da página. Eles são chamados de *inline* porque eles são exibidos na linha com o texto ao invés de criar um novo bloco. Alguns exemplos comuns de elementos *inline* de formatação de texto incluem:

- a) ``: destaca o texto como sendo importante e normalmente é exibido em negrito;
- b) ``: indica que o texto tem alguma ênfase e normalmente é exibido como itálico;
- c) `<mark>`: destaca o texto como sendo relevante e normalmente é exibido com fundo amarelo;
- d) `<abbr>`: usado para abreviaturas, com opção de exibir o significado completo ao passar o mouse sobre ele usando o atributo `title`;
- e) ``: é usado para marcar um texto que tenha sido excluído em uma versão anterior do documento e normalmente aparece riscado;
- f) `<ins>`: é usado para marcar um texto que tenha sido incluído em uma versão anterior do documento e normalmente aparece sublinhado;

- g) <sub>: é usado para exibir texto em índice inferior, ou seja, um texto menor que o tamanho normal que aparece abaixo da linha de base do texto;
- h) <sup>: é usado para exibir texto em índice superior, ou seja, um texto menor que o tamanho normal que aparece acima da linha de base do texto;
- i) <small>: é usado para exibir texto em tamanho menor que o tamanho normal do texto;

O exemplo 3.9 mostra o código de alguns dos elementos *inline* citados anteriormente e a figura 3.6 mostra a renderização desse código.

```
1 <p>Este é um texto <strong>importante</strong></p>
2 <p>Este é um texto <em>com ênfase</em></p>
3 <p>Este é um texto <mark>marcado</mark>.</p>
4 <p>Este o significado de <abbr title="Hypertext Markup Language">HTML</abbr></p>
5 <p>Este é um texto <del>excluído</del>.</p>
6 <p>Este é um texto <ins>inserido</ins>.</p>
7 <p>Este é um texto<sub>subscrito</sub>.</p>
8 <p>Este é um texto<sup>superescrito</sup>.</p>
9 <p>Este é um texto <small>menor que o tamanho normal</small>.</p>
```

Exemplo de Código 3.9: Elementos *inline* para formatação de texto.

Este é um texto **importante**

Este é um texto *com ênfase*

Este é um texto **marcado**.

Este o significado de **HTML**.

Este é um texto ~~excluído~~.

Este é um texto inserido.

Este é um texto_{subscrito}.

Este é um texto^{superescrito}.

Este é um texto menor que o tamanho normal.

Figura 3.6: Renderização dos elementos inline do código 3.9.

Os elementos inline de formatação de texto são importantes porque eles permitem que o autor da página destaque o texto de acordo com seu significado, melhorando a legibilidade e acessibilidade do conteúdo. Além disso, eles são uma ótima maneira de adicionar estilo e formatação ao texto sem afetar o layout da página.

Além dos elementos *inline* mencionados anteriormente, há outros elementos *inline* antigos em HTML que foram usados para formatação de texto, como ``, `<i>`, `<s>`, `<u>`, `<big>` e outros, mas eles tornaram obsoletos. Portanto, estes elementos são menos recomendados do que os elementos *inline* mencionados anteriormente, pois eles não têm um significado claro e acessível.

3.6 Citações

O elemento `<blockquote>` é usado para definir uma citação longa em um documento HTML, que, normalmente, são trechos de texto com mais de 3 linhas retirados de escritos de outros autores. Uma citação longa é um trecho de texto que é retirado de outro contexto e apresentado de forma destacada em um documento HTML. O elemento `<blockquote>` é útil para destacar citações importantes ou relevantes para o conteúdo da página. O exemplo 3.10 mostra o código de um texto que corresponde a uma citação.

```
1 <blockquote cite="https://pt.wikipedia.org/wiki/Herbert_Kelleher">
2   <p>O tempo é o melhor professor, o problema é que ele mata todos os seus alunos.</p>
3   <footer>Herbert Kelleher</footer>
4 </blockquote>
```

Exemplo de Código 3.10: Bloco de citação longa.

Neste exemplo, o elemento `<blockquote>` é usado para marcar uma citação longa, que é o trecho “O tempo é o melhor professor, o problema é que ele mata todos os seus alunos.”. O elemento `<footer>` é usado para fornecer informações sobre a fonte da citação, no caso, “Herbert Kelleher”. Além disso, opcionalmente, é possível usar o atributo `cite` do elemento `<blockquote>` para fornecer uma URL que aponte para a fonte original da citação. Neste caso, a fonte original é uma página da Wikipédia.

Em resumo, o elemento `<blockquote>` é uma ferramenta poderosa para destacar citações importantes ou relevantes em um documento HTML. Ao usar o elemento `<blockquote>` e o atributo `cite`, você pode fornecer informações sobre a fonte da citação e tornar seu conteúdo mais confiável e acessível.

O elemento `<blockquote>` resolve o problema das citações longas, mas temos também as citações curtas. As citações curtas são trechos de texto que são inseridos no meio de um parágrafo e não são destacados bloco. Para marcar citações curtas em HTML, você pode usar o elemento `<q>` e, opcionalmente, você também pode passar o atributo `cite` contendo a fonte da citação.

```
1 <p>O filósofo Jean-Paul Sartre afirmou que  
2   <q cite="https://pt.wikipedia.org/wiki/Jean-Paul_Sartre">  
3   O existencialismo é um humanismo</q>.</p>
```

Exemplo de Código 3.11: Trecho de citação curta.

Em resumo, o elemento `<q>` e o atributo `cite` são ferramentas úteis para marcar citações curtas em HTML. Assim como ocorre com a citação longa, ao usar esses recursos, você pode destacar o texto de uma citação curta, fornecer informações sobre a fonte da citação e tornar seu conteúdo mais claro e fácil de entender. A figura 3.7 mostra os exemplos desta seção renderizados em uma página web.

O tempo é o melhor professor, o problema é que ele mata todos os seus alunos.

Herbert Kelleher

O filósofo Jean-Paul Sartre afirmou que “O existencialismo é um humanismo”.

Figura 3.7: Renderização dos códigos de citação desta seção.

3.7 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 3.1 Crie uma página HTML com seis títulos de diferentes níveis (1 a 6) e, abaixo de cada título, um parágrafo de texto com 50 palavras (use lorem ipsum). ■

Exercício 3.2 Adicione à página criada no exercício 3.1 links ancorados em todos os parágrafos para voltarem ao título de nível 1. ■

Exercício 3.3 Adicione, ao fim da página criada no exercício 3.1, dentro de um novo parágrafo, um link externo apontando para a página da Wikipédia. ■

Exercício 3.4 Adicione, ao início da página criada no exercício 3.1, dentro de um novo parágrafo, um link interno apontando para uma página chamada “contato.html”. ■

Exercício 3.5 Crie um documento HTML que mostre o trecho de código-fonte apresentado no exemplo 2.3, combinando os elementos `<pre>` e `<code>` para apresentar o código da melhor forma possível. ■

Exercício 3.6 Adicione destaques ao texto do documento HTML anterior usando elementos *inline*, como ``, `` e `<mark>`. ■

Exercício 3.7 Crie um documento HTML contendo um título de nível 1, dois parágrafos com 20 palavras e uma citação longa (com ao menos 3 linhas). ■

Exercício 3.8 Adicione uma abreviação a um dos parágrafos do exercício 3.7. ■

Exercício 3.9 Marque um trecho de um dos parágrafos do exercício 3.7 usando o elemento de marcação da HTML. ■

Exercício 3.10 Crie duas páginas HTML - `index.html` e `contato.html` - com um título e um parágrafo cada. Ao fim da página `index.html`, adicione um link que navegue até a página `contato.html` e, ao fim de `contato.html`, adicione um link que volte para a página `index.html`. ■

3.8 Considerações Sobre o Capítulo

Este capítulo apresentou os elementos textuais básicos da linguagem HTML, incluindo títulos, parágrafos, quebras de linha, blocos de citação, texto pré-formatado, texto de código-fonte e links. No capítulo seguinte, veremos como apresentar dados usando listas e tabelas.



4. Listas e Tabelas

Este capítulo aborda a importância de se utilizar listas e tabelas no HTML para exibir dados de forma clara e organizada. O HTML é uma linguagem de marcação que permite a criação de diferentes estruturas dentro de uma página web, e listas e tabelas são elementos fundamentais para apresentar dados de maneira estruturada e fácil de compreender.

Listas permitem apresentar informações em uma série de itens relacionados, sejam eles numerados ou marcados. Já as tabelas são usadas para exibir informações em forma de linhas e colunas, facilitando a comparação e a análise de dados.

Neste capítulo, você aprenderá sobre os diferentes tipos de listas e tabelas que podem ser criados com HTML, bem como as tags e atributos necessários para sua implementação. Em resumo, ao final deste capítulo, você estará capacitado a criar listas e tabelas eficientes para suas páginas web, garantindo uma apresentação clara e organizada de dados.

4.1 Listas

Listas são elementos HTML que permitem organizar e apresentar informações de maneira estruturada e fácil de entender. Em HTML, existem três tipos principais de listas: listas não-ordenadas (*unordered lists*), listas ordenadas (*ordered lists*) e listas de definições (*definition lists*). Listas não-ordenadas são usadas para apresentar informações em que a ordem dos itens não é importante, como uma lista de ingredientes ou de itens de um cardápio. Listas ordenadas são usadas para apresentar informações em que a ordem dos itens é importante, como uma lista de instruções de montagem de um móvel ou uma lista cronológica de tarefas a fazer. Listas de definições, por sua vez, são usadas para apresentar termos e suas respectivas descrições, como em um dicionário, onde temos uma palavra e seu significado.

O propósito de usar listas na web é o de exibir dados em uma página de maneira clara e fácil de entender para o usuário, em uma estrutura sequencial ordenada ou não ordenada. Além disso, listas são uma ótima maneira de tornar o conteúdo da página acessível para usuários com necessidades especiais, como os usuários de leitor de tela, pois eles podem navegar facilmente pelos itens da lista um a um. Agora veremos como usar cada um dos tipos de lista mencionados.

4.1.1 Listas Não Ordenadas

As listas não ordenadas são usadas para apresentar uma série de itens em que a ordem não é importante. Como exemplos de uso, podemos citar ingredientes de uma receita, especificações técnicas de um produto etc. Em HTML, essas listas são criadas usando o elemento `` (*unordered lists*) e cada item da lista é criado usando o elemento `` (*list item*).

A lista não ordenada é apresentada com marcadores (bolinhas, quadrados etc.) antes de cada item da lista. O tipo de marcador pode ser alterado usando o atributo `type`. Por exemplo, o valor `square` apresenta quadrados como marcadores, `circle` apresenta um círculo e o valor `disc`, que é o padrão, apresenta bolinhas. O exemplo 4.1 mostra o código de uma lista não ordenada e a figura 4.1 mostra a renderização do exemplo.

```
1 <ul>
2   <li>Item 1</li>
3   <li>Item 2</li>
4   <li>Item 3</li>
5 </ul>
```

Exemplo de Código 4.1: Lista não ordenada básica.

- Item 1
- Item 2
- Item 3

Figura 4.1: Renderização da lista não ordenada do exemplo 4.1.

Vamos agora ver um exemplo de lista não ordenada com marcadores quadrados. O código do exemplo 4.2 mostra uma lista não ordenada com marcadores quadrados, enquanto a figura 4.2 mostra a renderização do código em questão. Veja na linha 1 do código que agora o atributo `type="square">` está presente na *tag* de abertura do elemento ``.

```
1 <ul type="square">
2   <li>Item 1</li>
3   <li>Item 2</li>
4   <li>Item 3</li>
5 </ul>
```

Exemplo de Código 4.2: Lista não ordenada com marcador quadrado.

- Item 1
- Item 2
- Item 3

Figura 4.2: Renderização da lista não ordenada com marcador quadrado do exemplo 4.2.

4.1.2 Listas Ordenadas

As listas ordenadas são usadas para apresentar uma série de itens em que a ordem é importante. Em HTML, essas listas são criadas usando o elemento `` (*ordered list*) e cada item da lista é criado usando o elemento `` (*list item*).

A lista ordenada é apresentada com números ou letras antes de cada item da lista, indicando a ordem dos itens. O tipo de número ou letra pode ser alterado usando o atributo `type`. Por exemplo, o valor `A` apresenta letras maiúsculas como marcadores, enquanto o valor `i` apresenta letras minúsculas romanas como marcadores. Se o atributo `type` não for passado, a lista é numerada com algarismos arábicos iniciando em 1. O exemplo 4.3 mostra o código de uma lista ordenada e a figura 4.3 mostra a renderização do exemplo. Veja que os marcadores são algarismos romanos em maiúsculas.

```
1 <ol type="I">
2   <li>HTML</li>
3   <li>CSS</li>
4   <li>Javascript</li>
5 </ol>
```

Exemplo de Código 4.3: Lista ordenada com algarismos romanos.

- I. HTML
- II. CSS
- III. Javascript

Figura 4.3: Renderização da lista ordenada do exemplo 4.3.

Outros dois atributos podem ser usados em listas ordenadas. O atributo `start` permite que você informe um valor inicial para a numeração dos itens. Por exemplo, se você passar `start="5"`, o primeiro elemento da lista iniciará com o valor 5. O outro atributo é o `reversed`, que não possui valor. A presença do atributo `reversed` faz com que a lista seja exibida na ordem inversa.

Assim como ocorre com listas não ordenadas, as listas ordenadas também são uma ótima maneira de tornar o conteúdo da página acessível para usuários com necessidades especiais, como usuários de leitor de tela, pois permitem que esses usuários naveguem item por item.

4.1.3 Listas de Definições

As listas de definições são usadas para apresentar termos e descrições, como em um dicionário. Em HTML, essas listas são criadas usando os elementos `<dl>` (*definition list*), `<dt>` (*definition term*) e `<dd>` (*definition description*). O elemento `<dt>` é usado para definir o termo, enquanto o elemento `<dd>` é usado para descrever o termo. Cada termo e sua descrição formam um par de definição. O exemplo 4.4 mostra o código de uma lista de definições e a figura 4.4 mostra a renderização desse código.

```
1 <dl>
2   <dt>HTML</dt>
3   <dd>Linguagem de Marcação Hipertexto</dd>
4   <dt>CSS</dt>
5   <dd>Folha de Estilo em Cascata</dd>
6   <dt>JavaScript</dt>
7   <dd>Linguagem de Programação de Script</dd>
8 </dl>
```

Exemplo de Código 4.4: Lista de definições.

HTML
Linguagem de Marcação Hipertexto
CSS
Folha de Estilo em Cascata
JavaScript
Linguagem de Programação de Script

Figura 4.4: Renderização da lista de definições do exemplo 4.4.

As listas de definições são úteis para apresentar informações de maneira clara e fácil de entender para o usuário, especialmente quando se trata de definir ou descrever termos técnicos ou complexos. Além disso, elas também são uma ótima maneira de tornar o conteúdo da página acessível para usuários com necessidades especiais, como usuários de leitor de tela.

4.2 Tabelas

As tabelas são um recurso fundamental na construção de páginas web, pois permitem a organização de informações em linhas e colunas, facilitando a visualização e comparação de dados. Na HTML, as tabelas são criadas com a utilização de *tags* específicas, permitindo que sejam criados diversos tipos de estruturas, como tabelas simples, tabelas com células mescladas, tabelas com cabeçalhos e rodapés, entre outras.

4.2.1 Tabelas Simples

As tabelas simples são o tipo mais comum de tabela HTML e consistem em uma grade retangular formada por **linhas** e **colunas**, em que cada **célula** pode conter texto, imagens ou outros elementos HTML. Elas são uma maneira eficiente de organizar e apresentar informações em formato tabular, facilitando a leitura e a compreensão de dados.

As tabelas simples podem ser utilizadas em diversas situações, como em listas de preços, horários, agenda de eventos, entre outros. Elas permitem que informações sejam apresentadas em colunas e linhas, com cabeçalhos e rodapés para cada seção, permitindo a comparação de dados e informações. O exemplo 4.5 mostra o código de uma tabela simples com a primeira linha como cabeçalho.

```
1 <table>
2   <tr>
3     <th>Produto</th>
4     <th>Preço</th>
5   </tr>
6   <tr>
7     <td>Camiseta</td>
8     <td>R$ 39,90</td>
9   </tr>
10  <tr>
11    <td>Calça Jeans</td>
12    <td>R$ 89,90</td>
13  </tr>
14  <tr>
15    <td>Tênis</td>
16    <td>R$ 129,90</td>
17  </tr>
18 </table>
```

Exemplo de Código 4.5: Tabela simples.

Neste exemplo 4.5, temos uma tabela com duas colunas, uma para o nome do produto e outra para o preço. A primeira linha da tabela é o cabeçalho, e é definida com o elemento `<tr>` para a linha que, por sua vez, contém elementos `<th>` para os títulos das colunas. Observe que, por padrão, o conteúdo textual das células de título é exibido em negrito. As linhas de dados são definidas com o elemento `<tr>` e as células de dados com o elemento `<td>`. A figura 4.5 mostra como fica a renderização dessa tabela.

Produto	Preço
Camiseta	R\$ 39,90
Calça Jeans	R\$ 89,90
Tênis	R\$ 129,90

Figura 4.5: Renderização da tabela do exemplo 4.5.

Apenas com a finalidade de melhorar a visualização das divisões entre as células das tabelas ao longo deste material, eu vou usar o atributo `border` com o valor igual a 1 para que bordas sejam adicionadas à tabela e às suas células. Esse atributo está obsoleto, portanto, seu uso não é recomendado, uma vez que pode-se obter resultados melhores usando CSS, como veremos na parte II deste material. Enquanto não aprendemos a fazer bordas com CSS, vamos usar o atributo `border`. O exemplo 4.6 mostra o código de uma tabela com o atributo `border` e a figura 4.6 mostra sua renderização.

```
1 <table border="1">
2   <tr>
3     <th>Nome</th>
4     <th>Idade</th>
5   </tr>
6   <tr>
7     <td>João</td>
8     <td>30</td>
9   </tr>
10  <tr>
11    <td>Maria</td>
12    <td>25</td>
13  </tr>
14 </table>
```

Exemplo de Código 4.6: Tabela com bordas.

Nome	Idade
João	30
Maria	25

Figura 4.6: Renderização da tabela do exemplo 4.6.

4.2.2 Cabeçalho, Corpo e Rodapé

Os elementos `<thead>`, `<tbody>` e `<tfoot>` são usados para separar diferentes partes de uma tabela HTML. Eles são opcionais, mas podem ajudar a organizar e estruturar melhor o conteúdo e aprimoram o aspecto semântico da tabela.

O elemento `<thead>` é usado para definir o cabeçalho da tabela. Ele deve ser usado apenas uma vez e deve ser colocado no início da tabela. O `<thead>` pode conter uma ou mais linhas (`<tr>`) e as células (`<th>`) representam os cabeçalhos das colunas. Ao utilizar o elemento `<thead>`, as células de cabeçalho podem ser diferenciadas visualmente do conteúdo da tabela, melhorando a legibilidade.

O elemento `<tbody>` é usado para agrupar o conteúdo da tabela em um ou mais blocos. Ele pode ser usado várias vezes dentro da tabela, mas deve ser colocado após o elemento `<thead>`, se este estiver presente. O `<tbody>` contém uma ou mais linhas (`<tr>`) e as células (`<td>`) representam os dados de cada coluna. Quando há várias linhas, é possível estilizar a tabela para diferenciar visualmente cada bloco de dados.

Por fim, o elemento `<tfoot>` é usado para definir o rodapé da tabela. Ele também deve ser usado apenas uma vez e deve ser colocado no final da tabela, após o último `<tbody>`. O `<tfoot>` pode conter uma ou mais linhas (`<tr>`) e as células (`<td>`) podem conter informações adicionais, como um total de valores, por exemplo. O exemplo 4.7 mostra uma tabela HTML que utiliza os elementos `<thead>`, `<tbody>` e `<tfoot>`.

A figura 4.7 mostra a renderização da tabela do exemplo 4.7. Observe que, por padrão, o cabeçalho é exibido em negrito, a região de conteúdo é exibida com fonte normal e o rodapé também é exibido com fonte normal. Outra vantagem de dividir a tabela em regiões semânticas é que a aparência de cada uma dessas regiões pode ser configurada via CSS, como veremos na parte II deste material.

Produto	Preço
Camiseta	R\$ 39,90
Calça Jeans	R\$ 89,90
Total	R\$ 129,80

Figura 4.7: Renderização da tabela do exemplo 4.7.

```
1 <table border="1">
2   <thead>
3     <tr>
4       <th>Produto</th>
5       <th>Preço</th>
6     </tr>
7   </thead>
8   <tbody>
9     <tr>
10      <td>Camiseta</td>
11      <td>R$ 39,90</td>
12    </tr>
13    <tr>
14      <td>Calça Jeans</td>
15      <td>R$ 89,90</td>
16    </tr>
17  </tbody>
18  <tfoot>
19    <tr>
20      <td>Total</td>
21      <td>R$ 129,80</td>
22    </tr>
23  </tfoot>
24 </table>
```

Exemplo de Código 4.7: Tabela com <thead>, <tbody> e <tfoot>.

4.2.3 Mesclagem de Linhas e Colunas

A mesclagem de linhas e colunas em tabelas HTML é uma técnica que permite combinar várias células em uma única célula, tanto na horizontal quanto na vertical. Isso é feito com os atributos `rowspan` (vertical) e `colspan` (horizontal) dos elementos `<th>` e `<td>`. Esses atributos devem ser valorados, respectivamente, com o número de linhas e de colunas que devem ser mescladas, ou seja, com o número de células abaixo ou à direita que devem ter seus espaços tomados pela célula em expansão.

A mesclagem de colunas (`colspan`) é usada quando se deseja combinar duas ou mais células adjacentes em uma única célula horizontal. O atributo `colspan` é definido na célula que será expandida e indica quantas colunas devem ser ocupadas. Por exemplo, se quisermos expandir uma célula por duas colunas adjacentes, podemos usar o atributo `colspan="2"`. O exemplo 4.8 mostra o código uma tabela HTML com mesclagem de colunas.

```
1 <table border="1">
2   <tr>
3     <th>Produto</th>
4     <th>Preço</th>
5     <th colspan="2">Disponibilidade</th>
6   </tr>
7   <tr>
8     <td>Camiseta</td>
9     <td>R$ 39,90</td>
10    <td>Disponível</td>
11    <td>Entrega em 24h</td>
12  </tr>
13  <tr>
14    <td>Calça Jeans</td>
15    <td>R$ 89,90</td>
16    <td>Indisponível</td>
17    <td>-</td>
18  </tr>
19 </table>
```

Exemplo de Código 4.8: Tabela com mesclagem de colunas.

Neste exemplo, a célula que contém a informação de disponibilidade (Disponível ou Indisponível) e a informação de entrega (Entrega em 24h ou -) é mesclada com a célula faltante da coluna à sua direita, utilizando o atributo `colspan="2"`. Isso permite que a tabela fique mais organizada e fácil de ler, como mostra a figura 4.8. A quantidade de colunas de uma tabela sempre será igual à quantidade máxima de células em uma linha qualquer. Por conta disso, no caso deste exemplo, a tabela possui 4 colunas, sendo que a última célula da primeira linha ocupa o espaço de duas colunas.

Produto	Preço	Disponibilidade	
Camiseta	R\$ 39,90	Disponível	Entrega em 24h
Calça Jeans	R\$ 89,90	Indisponível	-

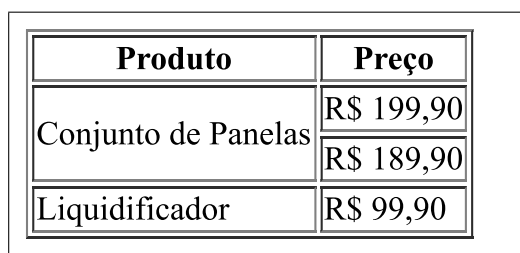
Figura 4.8: Renderização da tabela do exemplo 4.8.

A mesclagem de linhas (`rowspan`) é usada quando se deseja combinar duas ou mais células adjacentes em uma única célula vertical. O atributo `rowspan` é definido na célula que será mesclada e indica quantas linhas devem ser expandidas. Por exemplo, se quisermos expandir uma célula para que ela ocupe duas linhas adjacentes, podemos usar o atributo `rowspan="2"`. O exemplo 4.9 mostra o código uma tabela HTML com mesclagem de linhas.

```
1 <table border="1">
2   <tr>
3     <th>Produto</th>
4     <th>Preço</th>
5   </tr>
6   <tr>
7     <td rowspan="2">Conjunto de Panelas</td>
8     <td>R$ 199,90</td>
9   </tr>
10  <tr>
11    <td>R$ 189,90</td>
12  </tr>
13  <tr>
14    <td>Liquidificador</td>
15    <td>R$ 99,90</td>
16  </tr>
17 </table>
```

Exemplo de Código 4.9: Tabela com mesclagem de linhas.

Neste exemplo 4.9, a célula que contém o nome do produto (Conjunto de Panelas) é mesclada com a célula da linha abaixo, utilizando o atributo `rowspan="2"`. Isso permite que a tabela fique mais compacta e fácil de ler, como mostra a figura 4.9



Produto	Preço
Conjunto de Panelas	R\$ 199,90
	R\$ 189,90
Liquidificador	R\$ 99,90

Figura 4.9: Renderização da tabela do exemplo 4.9.

A mesclagem de linhas e colunas é uma técnica avançada de criação de tabelas HTML que requer planejamento cuidadoso e atenção aos detalhes. É importante lembrar que a mesclagem de linhas e colunas deve ser usada com moderação e apenas quando necessário, uma vez que pode tornar a tabela mais complexa e difícil de ler.

Além disso, a mesclagem de linhas e colunas pode afetar a acessibilidade da tabela, tornando-a menos legível para usuários de tecnologias assistivas, como usuários de leitores de tela. Uma boa prática é sempre testar a tabela em diferentes tamanhos de tela e verificar se ela está se comportando corretamente. Além disso, é importante usar os elementos de cabeçalho (`<th>`) para identificar os cabeçalhos de coluna e linha, tornando a tabela mais acessível e legível.

Em resumo, a mesclagem de linhas e colunas em tabelas HTML é uma técnica útil e poderosa para criar estruturas de tabela mais complexas. Combinada com outras técnicas de estilização e formatação, as tabelas podem ser uma maneira eficaz de organizar e apresentar informações em formato tabular para os usuários. Para concluir esta seção, o exemplo 4.10 mostra uma tabela com mesclagem de linhas e colunas, conforme ilustra a figura 4.10.

```
1 <table border="1">
2   <tr>
3     <th>Produto</th>
4     <th colspan="2">Preço</th>
5   </tr>
6   <tr>
7     <td rowspan="2">Conjunto de Panelas</td>
8     <td>À prazo</td>
9     <td>R$ 199,90</td>
10  </tr>
11  <tr>
12    <td>À vista</td>
13    <td>R$ 189,90</td>
14  </tr>
15  <tr>
16    <td>Liquidificador</td>
17    <td>Somente à vista</td>
18    <td>R$ 99,90</td>
19  </tr>
20 </table>
```

Exemplo de Código 4.10: Tabela com mesclagem de linhas e colunas.

Produto	Preço	
Conjunto de Panelas	À prazo	R\$ 199,90
	À vista	R\$ 189,90
Liquidificador	Somente à vista	R\$ 99,90

Figura 4.10: Renderização da tabela do exemplo 4.10.

4.3 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado. **Observação:** para os exercícios envolvendo tabelas, use o atributo para inserir borda.

Exercício 4.1 Crie uma lista de compras de supermercado com pelo menos 5 itens. ■

Exercício 4.2 Crie uma lista de características de um produto ou serviço oferecido por uma loja virtual qualquer. ■

Exercício 4.3 Crie uma lista com ao menos 5 redes sociais em que uma pessoa ou empresa possa estar presente. ■

Exercício 4.4 Crie uma lista numerada de passos para realizar uma receita de bolo de chocolate. ■

Exercício 4.5 Crie uma lista de rankings ou classificações como os melhores filmes que você já assistiu. ■

Exercício 4.6 Crie uma lista ordenada de tarefas que você executa do momento em que acorda até o momento em que deita para dormir. ■

Exercício 4.7 Crie uma lista com 5 termos e suas definições para um glossário de redes sociais. ■

Exercício 4.8 Crie uma lista com 5 adjetivos no idioma inglês e suas respectivas traduções para o idioma português. ■

Exercício 4.9 Crie uma lista de sinônimos e antônimos para 5 palavras-chave de um dicionário. ■

Exercício 4.10 Crie uma lista de abreviações e seus significados sobre astronomia. ■

Exercício 4.11 Crie uma tabela que exiba uma lista de 5 preços de produtos. Inclua uma coluna para o nome do produto, outra para o preço e uma terceira para uma breve descrição do produto. ■

Exercício 4.12 Crie uma tabela que exiba uma programação de 5 eventos para uma conferência. Inclua colunas para o nome do evento, o horário, a localização e a descrição. ■

Exercício 4.13 Crie uma tabela com três colunas e três linhas, e mescle a primeira célula das três linhas para criar uma célula que ocupe duas colunas. Adicione conteúdo a cada célula para exibir informações sobre um produto. ■

Exercício 4.14 Crie uma tabela que exiba informações de contato para uma empresa. Inclua colunas para o nome, o endereço, o telefone e o e-mail, sendo que o telefone e o e-mail devem estar sob o mesmo título (Contatos). ■

Exercício 4.15 Crie uma tabela que exiba informações sobre um time de futebol. Inclua colunas para o nome do jogador e para alguns dados técnicos, incluindo a posição, o número da camisa e as estatísticas de desempenho, sendo que essas 3 últimas informações devem ficar em 3 linhas diferentes, enquanto a coluna com o nome do jogador deve se expandir por essas 3 linhas. ■

Exercício 4.16 Crie uma tabela que exiba uma lista de 3 tarefas com datas de início e de conclusão. Mescle as 2 primeiras células da primeira linha para criar uma célula que ocupe 2 colunas, e mescle as 3 primeiras células da primeira coluna para criar uma célula que ocupe 3 linhas. Em síntese, a primeira célula da tabela vai ocupar 2 linhas e 3 colunas e deve ter o título “Tarefas”, enquanto as demais células devem armazenar as demais informações, ou seja, o título da tarefa, a data de início e a data de conclusão. ■

Exercício 4.17 Crie uma tabela que exiba informações sobre o desempenho de 3 atletas em diferentes eventos esportivos A, B e C. O primeiro atleta participou dos eventos A e B, o segundo participou de B e C, e o terceiro participou de A, B e C. A tabela deve ter 3 colunas. A primeira deve ter o nome do atleta expandido para a quantidade de linhas correspondente à quantidade de eventos que ele participou. A segunda coluna deve apresentar em linhas distintas os eventos que o atleta participou e a terceira coluna deve mostrar o tempo do atleta no respectivo evento. ■

Exercício 4.18 Faça uma tabela idêntica à apresentada abaixo. ■

Funcionário	Semestre	Gastos		Lucros	
		Líquido	Bruto	Líquido	Bruto
João	1º	1000	1200	200	300
	2º	1000	1200	200	300
Maria	1º	1000	1200	200	300
	2º	1000	1200	200	300

Exercício 4.19 Faça uma tabela idêntica à apresentada abaixo.

Brasil	
Estado	Time
São Paulo	Corinthians
	Palmeiras
	São Paulo
	Santos
Rio de Janeiro	Botafogo
	Flamengo
	Fluminense
	Vasco

Exercício 4.20 Faça uma tabela idêntica à apresentada abaixo.

Produto	Análise de Preço		
Caixa JBL Bump!	Submarino	Boleto	R\$ 300,00
		Cartão	R\$ 350,00
	Amazon	Boleto	R\$ 310,00
		Cartão	R\$ 340,00
Apple Watch SE	Submarino	Boleto	R\$ 2300,00
		Cartão	R\$ 2500,00
	Amazon	Boleto	R\$ 2350,00
		Cartão	R\$ 2350,00

4.4 Considerações Sobre o Capítulo

Este capítulo apresentou como criar listas e tabelas em HTML. Vimos as listas não ordenadas, ordenadas e de definições, bem como tabelas simples, com agrupamento de cabeçalho, conteúdo e rodapé e com linhas e colunas mescladas. No capítulo seguinte veremos como apresentar imagens em um documento HTML.

5. Imagens

As imagens são uma parte importante no design de uma página web. Elas podem ajudar a transmitir uma mensagem, destacar informações importantes e tornar uma página mais atraente visualmente. Além disso, as imagens também podem ser usadas para fornecer informações adicionais, como diagramas, gráficos e ilustrações.

No entanto, incluir imagens em uma página web é mais do que simplesmente fazer upload delas e inseri-las no código. É importante ter em mente questões como o formato de imagem, a otimização para a web e a responsividade.

Neste capítulo, vamos explorar as melhores práticas para incluir imagens em páginas web, incluindo como adicionar imagens a uma página HTML, escolher o formato de imagem correto e garantir que as imagens sejam exibidas corretamente em diferentes tamanhos de tela. Vamos também discutir como usar figuras, legendas e mapas de imagem, visando tornar as imagens mais significativas e esteticamente agradáveis.

5.1 Adicionando Imagens a Uma Página HTML

O elemento `` é usado para adicionar imagens a uma página HTML. É importante observar que o elemento `` não possui *tag* de fechamento, ou seja, não tem conteúdo. Em vez disso, ele termina no final da *tag* de abertura. O código 5.1 mostra um exemplo de como adicionar uma imagem a uma página HTML usando o elemento ``.

```
1 | <h1>Sala de Aula</h1>
2 | 
```

Exemplo de Código 5.1: Adicionando uma imagem ao documento

A primeira linha do exemplo de código 5.1 apresenta um título. Na segunda linha, temos o elemento ``, cujo atributo `src` especifica o caminho para a imagem e o atributo `alt` fornece uma descrição da imagem para usuários com deficiência visual ou para navegadores que não conseguem exibir a imagem. É importante incluir uma descrição precisa e informativa para o atributo `alt`, pois isso ajuda a garantir que o conteúdo da página seja acessível para todos. A figura 5.1 mostra o código 5.1 renderizado.

Sala de Aula



Figura 5.1: Renderização da imagem do exemplo 5.1.

Além do `src` e do `alt`, existem outros atributos que podem ser usados com o elemento ``. Por exemplo, o atributo `width` e o atributo `height` podem ser usados para especificar a largura e a altura da imagem, respectivamente. Vale ressaltar que, ao passar valores para ambos os atributos, devemos nos atentar para usar valores que mantenham as proporções da imagem. Caso contrário, a imagem ficará distorcida. Se você alterar o valor de um dos atributos apenas, o outro se ajusta automaticamente para manter a proporção. O código 5.2 mostra um exemplo de como adicionar uma imagem a uma página HTML usando o elemento `` com uma largura predefinida. Neste caso, a altura será ajustada automaticamente para manter a proporção.

```
1 | <h1>Aluno Usando Computador</h1>
2 | 
```

Exemplo de Código 5.2: Imagem com atributo de largura.

No exemplo 5.2, a largura da imagem é especificada como 200 pixels. Em síntese, o elemento `` é uma ferramenta poderosa para adicionar imagens a uma página HTML. É importante lembrar de incluir descrições precisas e informativas nas imagens, além de usar outros atributos, como `width` e `height`, para controlar o tamanho da imagem. A figura 5.2 mostra o código 5.2 renderizado.



Figura 5.2: Renderização da imagem do exemplo 5.2.

5.2 Formatos de Imagem

Quando se trata de incluir imagens em uma página web, é importante escolher o formato de imagem correto para garantir a qualidade da imagem, o tempo de carregamento rápido da página e a compatibilidade com diferentes navegadores.

Existem muitos formatos de imagem disponíveis, mas alguns são mais comuns na web do que outros. Nesta seção, vamos explorar os formatos de imagem mais usados na web, a saber, PNG, JPEG, GIF e SVG. Vamos discutir suas vantagens, desvantagens e indicações para uso.

Com essa compreensão, você poderá tomar decisões embasadas sobre o formato de imagem a ser usado para cada imagem na sua página web, ajudando a garantir que suas páginas web sejam exibidas com qualidade e rapidamente para seus usuários. Agora, vamos conhecer um pouco melhor cada um dos formatos mais populares na web.

5.2.1 PNG (*Portable Network Graphics*)

O formato PNG é uma opção popular para imagens que exigem transparência, como ícones e gráficos. Ele suporta transparência sem fundo e é capaz de armazenar múltiplas camadas de informações, o que o torna uma boa opção para imagens com áreas transparentes. Além disso, o PNG também suporta compressão sem perda de qualidade, o que significa que a qualidade da imagem não é afetada mesmo após múltiplos salvamentos.

Desvantagens: o PNG pode ter um tamanho de arquivo maior do que outros formatos de imagem, como JPEG, o que pode afetar negativamente o tempo de carregamento da página.

Indicações: o PNG é uma boa opção para imagens que precisam de transparência, como ícones, gráficos e logos.

5.2.2 JPEG (*Joint Photographic Experts Group*)

O formato JPEG é uma opção popular para imagens fotográficas, pois ele suporta uma ampla gama de cores e permite uma alta taxa de compressão sem perda de qualidade. Isso significa que as imagens JPEG podem ser comprimidas significativamente sem perda de qualidade, o que é útil para reduzir o tamanho do arquivo e melhorar o tempo de carregamento da página.

Desvantagens: o JPEG não suporta transparência e pode apresentar pixels artificiais em áreas com cores uniformes ou gradientes suaves. Além disso, a compressão excessiva pode resultar em perda de qualidade da imagem.

Indicações: o JPEG é uma boa opção para imagens fotográficas, principalmente de elementos naturais, pois suporta uma ampla gama de cores e permite uma alta taxa de compressão sem perda de qualidade.

5.2.3 GIF (*Graphics Interchange Format*)

O formato GIF é uma opção popular para imagens animadas, pois ele suporta animações simples e transparência limitada. Ele é útil para criar pequenos cliques animados e gráficos animados, como ícones animados e banners.

Desvantagens: o GIF tem uma paleta de cores limitada, o que significa que ele não é uma boa opção para imagens fotográficas ou gráficos com muitas cores. Além disso, as animações GIF podem ser grandes e demorar muito para carregar.

Indicações: o GIF é uma boa opção para imagens animadas simples, como ícones animados, gráficos animados e banners.

5.2.4 SVG (*Scalable Vector Graphics*)

O formato SVG é um formato de gráfico vetorial baseado em XML, o que significa que ele é escalável sem perda de qualidade. Ele é útil para criar gráficos e ilustrações vetoriais que podem ser redimensionados sem perda de qualidade. Além disso, o SVG pode ser animado e estilizado com CSS, o que o torna uma opção versátil para a criação de gráficos interativos.

Desvantagens: o SVG pode ser mais complexo de criar e editar do que outros formatos de imagem, como JPEG ou PNG. Além disso, alguns navegadores antigos podem ter dificuldades para exibir imagens SVG corretamente.

Indicações: o SVG é uma boa opção para gráficos vetoriais escaláveis, como logotipos, ícones e ilustrações, além de gráficos interativos animados.

5.3 Figuras e Legendas

As figuras e legendas são uma maneira eficaz de apresentar imagens de forma clara e organizada em uma página web. Uma figura é composta pela imagem em si e pela legenda, que é um texto que descreve ou complementa a imagem. Vale ressaltar que a legenda e o atributo `alt` são coisas distintas, apesar de poderem possuir o mesmo texto descritivo e de ambos serem úteis para acessibilidade.

Ao usar figuras com legendas, você pode fornecer informações adicionais sobre a imagem e torná-la mais significativa para o usuário. Além disso, as legendas também podem ser úteis para descrever imagens para usuários com deficiência visual, que podem usar tecnologias de leitura de tela para ler as legendas.

Para incluir uma figura com legenda em uma página web, você pode usar o elemento `<figure>` contendo um elemento `` e um elemento `<figcaption>`. O elemento `<figure>` é usado para envolver a imagem e a legenda, enquanto o elemento `<figcaption>` é usado para fornecer a legenda em si. O exemplo 5.3 mostra como usar o elemento `<figure>` e o elemento `<figcaption>` para incluir uma figura e uma legenda em uma página web.

No exemplo 5.3, a imagem é incluída usando o elemento ``, como discutido anteriormente. A legenda é incluída usando o elemento `<figcaption>`. Veja na figura 5.3 a renderização da imagem com legenda do exemplo 5.3.

```
1 <figure>
2   
3   <figcaption>Sala de aula com alunos no computador.</figcaption>
4 </figure>
```

Exemplo de Código 5.3: Imagem com legenda.



Sala de aula com alunos no computador.

Figura 5.3: Renderização da imagem com legenda do exemplo 5.3.

Em resumo, as figuras e legendas são uma ferramenta útil para apresentar imagens de forma clara e organizada em uma página web. Ao usar o elemento `<figure>` e o elemento `<figcaption>`, você pode fornecer informações adicionais sobre a imagem e torná-la mais significativa para o usuário.

5.4 Imagens Responsivas

Para usar imagens de forma responsiva em uma página web, isto é, cujo tamanho se adeque à resolução de tela do dispositivo, você pode usar o recurso de largura e altura do elemento `img` (`width` e `height`). Porém, os valores passados devem estar em percentual. Assim, se a página for aberta em diferentes tamanhos de tela ou se o usuário redimensionar o navegador, a imagem será redimensionada para o percentual relativo à área de visualização disponível.

O redimensionamento proporcional ao tamanho da área de visualização promove a responsividade, porém, essa não é a melhor solução, uma vez que uma mesma imagem será usada para diferentes resoluções de tela. Isso pode desperdiçar banda em dispositivos de tela pequena e gerar imagens ruins em dispositivos de tela grande, pois um único arquivo contendo a imagem com dimensões médias será usado.

O elemento `<picture>` é uma maneira eficaz de fornecer imagens responsivas em uma página web. Ele permite que você forneça diferentes versões (arquivos) de uma imagem, cada uma delas otimizada para determinado tamanho de tela. O exemplo 5.4 mostra como usar o elemento `<picture>` para fornecer imagens responsivas.

```
1 <picture>
2   <source media="(max-width: 767px)" srcset="imagem-pequena.png">
3   <source media="(min-width: 768px)" srcset="imagem-grande.png">
4   
5 </picture>
```

Exemplo de Código 5.4: Imagem responsiva com arquivos otimizados.

Neste exemplo 5.4, o elemento `<source>` é usado para especificar diferentes versões da imagem. A primeira versão da imagem é fornecida com o atributo `srcset="imagem-pequena.png"` e o atributo `media="(max-width: 767px)"`, o que significa que o arquivo "imagem-pequena.png" será usado em telas com largura máxima de 767 pixels. A segunda versão da imagem é fornecida com o atributo `srcset="imagem-grande.png"` e o atributo `media="(min-width: 768px)"`, o que significa que o arquivo "imagem-grande.png" será usado em telas com largura mínima de 768 pixels.

A imagem padrão é fornecida no próprio atributo `src` do elemento ``, e será usada em caso de falha ou em navegadores que não suportam o elemento `<picture>`. Neste caso, foram fornecidas 3 versões da imagem, mas é possível combinar os valores de `min-width` e `max-width` para definir imagens compatíveis com telas em uma faixa de tamanho mais específica. Isso abre possibilidades para se fornecer infinitas versões da imagem para os mais diversos tamanhos de tela. Vale ressaltar que o navegador buscará no servidor somente a imagem compatível com a sua resolução de tela.

Em síntese, o elemento `<picture>` permite que você forneça imagens responsivas, oferecendo diferentes versões da imagem para diferentes tamanhos de tela. Ao usar o elemento `<source>` e os atributos `media` e `srcset`, você pode especificar quando cada versão da imagem deve ser usada, garantindo que a imagem correta seja carregada e exibida para cada tamanho de tela.

5.5 Mapas de Imagens

Os mapas de imagem são uma técnica popular para criar links em regiões dentro de imagens em HTML. Eles permitem que diferentes áreas da imagem sejam associadas a links diferentes, criando uma maneira interativa e atraente de navegar em um site. Um mapa de imagem pode ser utilizado para diversos fins, desde criar links para diferentes áreas de um produto em uma loja virtual até adicionar interatividade em uma página de informações turísticas a partir de um mapa geográfico do local.

Em HTML, um mapa de imagem é criado utilizando o elemento `<map>` juntamente com elementos do tipo `<area>`. As áreas da imagem são definidas utilizando o atributo `coords`, que define as coordenadas dos pontos que formam a área clicável da imagem, e o atributo `shape`, que define a forma da área da imagem (por exemplo, retangular, circular ou poligonal).

Ao clicar em uma área definida no mapa de imagem, o usuário é redirecionado para destino específico, funcionando como um link. Os mapas de imagem também podem ser utilizados para criar efeitos de destaque, como a exibição de informações adicionais sobre uma área específica da imagem quando o usuário passa o mouse sobre ela.

Nesta seção, exploraremos como criar e usar mapas de imagem em HTML. Vamos começar entendendo os conceitos básicos de como as áreas são definidas e como criar um mapa de imagem simples. Em seguida, discutiremos como adicionar diferentes formas de áreas, como polígonos, retângulos e círculos, e como adicionar links.

5.5.1 Um Mapa de Imagem Simples

Para criar um mapa de imagem simples em HTML, vamos ter que seguir alguns passos. O primeiro deles é identificar quais são as regiões da imagem que você quer tornar clicável. Para verificarmos isso, vamos considerar a imagem da figura 5.4.

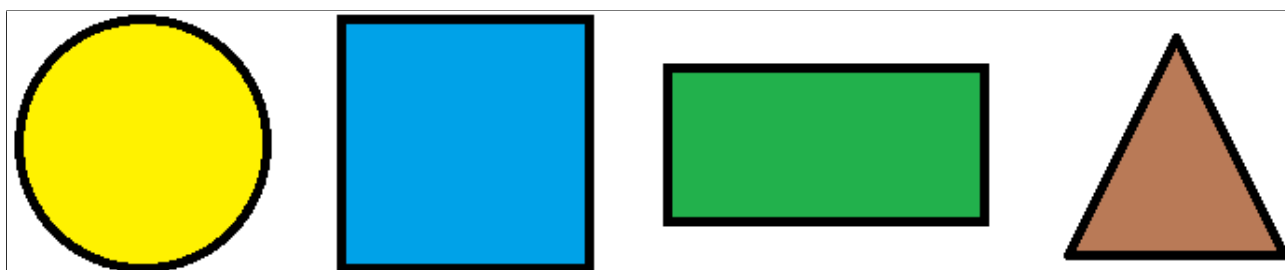


Figura 5.4: Imagem simples com 4 formas geométricas, com dimensões 690x142 pixels.
(Disponível em <https://imgbox.com/KAP90MBi>)

Imagine que precisemos tornar cada forma geométrica clicável, obedecendo precisamente seus limites. Vamos começar adicionando a imagem à página web utilizando o elemento ``. Certifique-se de apontar o atributo `src` para o caminho correto da imagem e de definir o atributo `alt` para fornecer uma descrição alternativa da imagem, que é importante para fins de acessibilidade. O código inicial é mostrado no exemplo 5.5.

```
1 | 
```

Exemplo de Código 5.5: Código inicial do mapeamento de imagem.

Agora que temos a imagem, temos que criar um elemento `<map>` para definirmos o mapeamento. É importante também fornecermos um nome para este elemento `<map>` através de seu atributo `name`. O exemplo 5.6 mostra o código até este momento.

```
1 |   
2 | <map name="mapaFormas">  
3 | </map>
```

Exemplo de Código 5.6: Imagem e mapa de imagem.

Agora, dentro do elemento `<map>`, temos que adicionar um elemento `<area>` para cada região que queremos mapear, sendo que cada elemento `<area>` deve ter um atributo `shape`, que define a forma da área, e um atributo `coords`, que define as coordenadas da área em relação à imagem. Vamos começar mapeando o círculo amarelo. Primeiramente, temos que descobrir as coordenadas do centro do círculo. Vale ressaltar que a coordenada (0,0) está localizada no canto superior esquerdo da imagem, portanto, quanto mais à direita, mais próximo de 690 fica o valor da coordenada X, e quanto mais para baixo, mais próximo de 142 fica o valor da coordenada Y.

Para descobrir as coordenadas do centro desse círculo, eu abri essa imagem no *Paint*, editor de imagens básico do *Windows*, posicionei o cursor do mouse no centro do círculo, e olhei na barra de status na parte inferior as coordenadas do cursor do mouse. O valor mostrado, neste caso, foi de (73,69), portanto, essas são as coordenadas do centro do círculo amarelo. Agora, eu preciso saber o valor do raio do círculo. Para isso, eu posicionei o cursor do mouse na extremidade direita do círculo, vejo o valor da coordenada X, e subtraio da coordenada X do centro. Neste caso, a coordenada X da extremidade vale 142, portanto, o raio vale 69. Agora temos que criar um elemento `<area>` do tipo “círculo” e passar esses valores. O exemplo 5.7 mostra como está ficando o código até aqui.

```
1 
2 <map name="mapaFormas">
3   <area shape="circle" coords="73,69,69" href="circulo.html" alt="Círculo amarelo">
4 </map>
```

Exemplo de Código 5.7: Imagem e mapa de imagem com uma área clicável.

Na linha 3 do exemplo 5.7, temos o elemento `<area>`. Veja o atributo `shape="circle"`, indicando que essa área é um círculo. Isso é importante para que o elemento interprete corretamente as coordenadas que estão no atributo `coords`. Sempre que `shape="circle"`, as coordenadas passadas devem corresponder ao centro do círculo e ao raio do círculo. Por isso o valor do atributo `coords` é igual a 73, 69, 69, pois os dois primeiros valores correspondem às coordenadas X,Y do centro do círculo e o terceiro valor corresponde ao raio. O atributo `href` contém o endereço do link correspondente à região, ou seja, para onde o usuário será redirecionado quando clicar no círculo. Por fim, o atributo `alt` promove a acessibilidade através de uma descrição da imagem.

Vamos prosseguir criando uma região clicável para o quadrado azul. Neste caso, temos que usar o atributo `shape="rect"` e passar como coordenadas os valores X,Y do canto superior esquerdo e os valores X,Y do canto inferior direito. Você pode obter esses valores usando a mesma técnica utilizada para o círculo amarelo, com o *Paint* ou outro editor de imagem. Eu obtive as coordenadas (177,2) para o canto superior esquerdo e (314,139) para o canto inferior direito do quadrado azul. Aproveitando que a próxima forma é um retângulo verde e, conseqüentemente, usa o mesmo tipo de `shape` do quadrado, eu analisei as coordenadas dos cantos do retângulo verde e cheguei aos valores (352,28) e (526,114). Portanto, eu vou agora complementar o código com novas áreas para essas formas. O exemplo 5.8 mostra como ficou o código até aqui.

```
1 
2 <map name="mapaFormas">
3   <area shape="circle" coords="73,69,69" href="circulo.html" alt="Círculo amarelo">
4   <area shape="rect" coords="177,2,314,139" href="quadrado.html" alt="Quadrado azul">
5   <area shape="rect" coords="352,28,526,114" href="retangulo.html" alt="Retângulo verde">
6 </map>
```

Exemplo de Código 5.8: Imagem e mapa de imagem com uma área clicável.

Se você tentar clicar nas regiões com o código como está, verá que ainda não funciona. Isso acontece porque, apesar de nosso mapa já definir 3 regiões, nós ainda não informamos para a imagem que ela deve usar o mapa em questão. Para fazer isso, temos que ir até o elemento `` e passar o atributo `usemap="#mapaFormas"`. Faça isso e teste. Eu vou deixar para mostrar esse trecho junto com o próximo código, portanto, vamos a ele.

Agora vamos adicionar a região do triângulo vermelho. Acontece que essa região não é circular e nem retangular, portanto, não podemos usar os tipos `circle` e `rect` no atributo `shape`. Para polígonos quaisquer, devemos usar o tipo `poly` passando as coordenadas de cada vértice do polígono em ordem. Eu vou adotar o sentido horário, portanto, vou ver qual é a posição do vértice superior do triângulo, do direito e do esquerdo, nesta ordem. Os valores que obtive foram (627,12), (688,132) e (566,132), respectivamente. O exemplo 5.9 mostra a versão final do mapa incluindo as 4 áreas e o atributo `usemap` no elemento `img`.

```
1 
2 <map name="mapaFormas">
3   <area shape="circle" coords="73,69,69" href="circulo.html" alt="Círculo amarelo">
4   <area shape="rect" coords="177,2,314,139" href="quadrado.html" alt="Quadrado azul">
5   <area shape="rect" coords="352,28,526,114" href="retangulo.html" alt="Retângulo verde">
6   <area shape="poly" coords="627,12,688,132,566,132" href="triangulo.html" alt="Triângulo vermelho">
7 </map>
```

Exemplo de Código 5.9: Versão final do mapeamento da imagem de formas geométricas.

Obviamente, se você tiver que mapear polígonos complexos, a tarefa não será muito fácil. Imagine, por exemplo, se você tiver que mapear cada Estado em um mapa do Brasil. Serão muitos vértices! Bem, neste caso, eu aconselho o uso de alguma ferramenta auxiliar. O site <https://www.image-map.net> permite que você faça o *upload* de uma imagem e crie o mapeamento de regiões de forma bem mais conveniente do que a que mostrei aqui. Sugiro que explore essa ferramenta caso tenha que fazer mapeamentos mais complexos.

5.6 Imagens Como Links

Em HTML, você pode usar imagens como links envolvendo um elemento `` correspondente à imagem com um elemento `<a>` correspondente ao link. Quando o usuário clicar na imagem, ele será redirecionado para a página vinculada ao link. O exemplo 5.10 mostra o código de uma imagem sendo usada como link.

```
1 <a href="contato.html">
2   
3 </a>
```

Exemplo de Código 5.10: Imagem como link.

5.7 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 5.1 Crie uma imagem com o atributo `src` apontando para o arquivo “imagem.jpg” e com um texto alternativo “Imagem de um gato”.

Exercício 5.2 Crie uma imagem para o arquivo local “cachorro.jpg” com uma largura de 400 pixels e uma altura de 300 pixels.

Exercício 5.3 Crie uma imagem com largura de 300 pixels contendo uma foto do planeta Saturno, sendo que essa foto deve estar em um endereço externo obtido via buscador.

Exercício 5.4 Crie uma imagem para o arquivo “logo.png” e com texto alternativo “Logotipo da empresa”, que seja um link para a página externa “https://exemplo.com”.

Exercício 5.5 Crie um elemento `<picture>` com duas resoluções diferentes de imagens, uma para telas de até 720px de altura e outra para telas com altura maior que 720px. Use nomes de arquivos fictícios.

Exercício 5.6 Crie um elemento `<picture>` com imagens diferentes para telas em formato “retrato” e telas em formato “paisagem”. Use nomes de arquivos fictícios. (pesquise como configurar o atributo `media` para essa situação).

Exercício 5.7 Crie um mapeamento para a imagem disponível em <https://tinyurl.com/3j558uax>, de forma que cada logotipo aponte para o site da respectiva empresa. Mostre a imagem com 800px de largura.

Exercício 5.8 Crie um mapeamento para a imagem disponível em <https://tinyurl.com/3bu9ekz3>, de forma que cada região do Brasil aponte para um arquivo HTML com seu nome (norte.html, nordeste.html etc.). Mostre a imagem com 800px de largura.

5.8 Considerações Sobre o Capítulo

Este capítulo apresentou como exibir imagens usando HTML. Vimos como mostrar imagens simples, imagens responsivas que se ajustam ao tamanho a tela, como carregar imagens de forma otimizada e responsiva e como mapear imagens criando regiões clicáveis. No capítulo seguinte veremos como apresentar áudio e vídeo em um documento HTML.



6. Áudio e Vídeo

A incorporação de elementos de áudio e vídeo em uma página web pode ser uma ótima maneira de melhorar a experiência do usuário e tornar o conteúdo mais interessante e interativo. O HTML, a partir da versão 5, oferece elementos específicos para inserir áudio e vídeo em uma página web, permitindo que os desenvolvedores personalizem a apresentação e a interação com conteúdo multimídia.

Ao inserir áudio em uma página HTML, o desenvolvedor pode escolher entre vários formatos de arquivo, cada um com suas próprias vantagens e desvantagens. Além disso, a HTML oferece vários **atributos** que permitem personalizar a apresentação e a interação com o áudio, como `controls` para adicionar controles de reprodução padrão, `autoplay` para iniciar a reprodução automaticamente e `muted` para silenciar o áudio.

Da mesma forma, ao inserir vídeo em uma página HTML, é possível personalizar a apresentação e a interação com o conteúdo multimídia. O HTML também oferece vários atributos que permitem personalizar a reprodução do vídeo, como `controls` para adicionar controles de reprodução padrão, `autoplay` para iniciar a reprodução automaticamente e `loop` para repetir a reprodução. Além disso, é possível incorporar vídeos do YouTube em uma página HTML usando um *iframe* e personalizar a reprodução com parâmetros de URL.

Em resumo, este capítulo aborda tudo o que você precisa saber para incorporar elementos de áudio e vídeo em uma página web, desde a inserção de arquivos de áudio e vídeo com o elemento `<source>` até a personalização da reprodução com diferentes atributos e parâmetros de URL.

6.1 Inserindo Áudio em Uma Página HTML

A inclusão de áudio em uma página HTML é muito parecida com a inclusão de imagens ou vídeos. O HTML oferece o elemento `<audio>`, que permite inserir arquivos de áudio em uma página web.

6.1.1 Elemento HTML para Áudio

O elemento `<audio>` é um contêiner que define o áudio a ser incluído em uma página HTML. O código 6.1 mostra o HTML básico para incluir um arquivo de áudio em uma página.

```
1 | <audio src="caminho/do/arquivo.mp3"></audio>
```

Exemplo de Código 6.1: Inserindo um elemento de áudio.

Nesse exemplo 6.1, o atributo `src` aponta para o arquivo de áudio a ser incluído na página. Note que, assim como no elemento `` para imagens, não é necessário fechar a tag do elemento `<audio>` (ou seja, o código acima é suficiente para incluir o áudio na página).

6.1.2 Inserindo Arquivos de Áudio em Diferentes Formatos

Para garantir a compatibilidade do arquivo de áudio com diferentes navegadores, é importante incluir diferentes formatos de arquivo usando o elemento `<source>`. O elemento `<source>` é usado para especificar diferentes versões do arquivo de áudio, cada uma em um formato diferente. O navegador irá escolher o formato suportado mais apropriado. O código 6.2 mostra o HTML necessário para incluir um arquivo de áudio com diferentes formatos em uma página.

```
1 | <audio controls>
2 |   <source src="caminho/do/arquivo.mp3" type="audio/mpeg">
3 |   <source src="caminho/do/arquivo.ogg" type="audio/ogg">
4 |   <source src="caminho/do/arquivo.wav" type="audio/wav">
5 | </audio>
```

Exemplo de Código 6.2: Uso do elemento áudio com diferentes arquivos.

Nesse exemplo 6.2, incluímos três versões do arquivo de áudio, cada uma em um formato diferente: MP3, Ogg e WAV. O navegador irá escolher a versão mais apropriada suportada pelo dispositivo em que a página está sendo visualizada.

6.1.3 Formatos de Arquivos Suportados Pelos Navegadores

É importante saber que nem todos os navegadores suportam todos os formatos de arquivo de áudio. Por exemplo, o Internet Explorer só suporta o formato MP3. Portanto, é recomendável incluir pelo menos a versão MP3 do arquivo de áudio para garantir a compatibilidade com esse navegador.

Os formatos mais comuns de arquivo de áudio suportados pelos navegadores são MP3, Ogg e WAV. No entanto, outros formatos também podem ser suportados, dependendo do navegador e do sistema operacional em que a página está sendo visualizada. Vale ressaltar que o formato WAV não tem compactação, portanto, seu carregamento pode ser mais demorado que os demais formatos.

6.1.4 Atributos de Configuração

O elemento `<audio>` oferece vários atributos que permitem personalizar a reprodução do áudio na página. A seguir, alguns dos atributos mais comuns.

6.1.4.1 Exibição de Controles de Reprodução

O atributo `controls` adiciona os controles padrão de reprodução, como *play*, *pause*, *volume* e *progresso*, ao elemento `<audio>`. O código 6.3 mostra o HTML para adicionar os controles padrão de reprodução.

```
1 <audio controls>
2   <source src="caminho/do/arquivo.mp3">
3 </audio>
```

Exemplo de Código 6.3: Uso do elemento áudio com atributo `controls`.

Nesse exemplo 6.3, o atributo `controls` adiciona os controles padrão de reprodução ao elemento `<audio>`. Isso inclui botões de *play*, *pause*, *volume* e *progresso*, que permitem ao usuário controlar a reprodução do áudio.

6.1.4.2 Reprodução Automática

O atributo `autoplay` inicia a reprodução do áudio automaticamente quando a página é carregada. Vale ressaltar que, por questões de privacidade, o navegador nem sempre aceita iniciar um áudio ao carregar uma página. Portanto, não se pode contar com o funcionamento pleno desse atributo. O código 6.4 mostra o HTML para reproduzir automaticamente um arquivo de áudio.

```
1 <audio autoplay>
2   <source src="caminho/do/arquivo.mp3">
3 </audio>
```

Exemplo de Código 6.4: Uso do elemento áudio com atributo `autoplay`.

Nesse exemplo 6.4, o atributo `autoplay` inicia a reprodução do áudio assim que a página é carregada.

6.1.4.3 Repetição da Reprodução

O atributo `loop` permite que o áudio seja reproduzido continuamente, repetindo a reprodução quando ela termina. O código 6.5 mostra o HTML para repetir a reprodução de um arquivo de áudio.

```
1 | <audio controls loop>
2 |     <source src="caminho/do/arquivo.mp3">
3 | </audio>
```

Exemplo de Código 6.5: Uso do elemento áudio com atributos `controls` e `loop`.

Nesse exemplo 6.5, o atributo `loop` permite que o áudio seja reproduzido continuamente, repetindo a reprodução quando ela termina.

6.1.4.4 Silenciando o Áudio

O atributo `muted` silencia o áudio do elemento `<audio>`. O código 6.6 mostra o HTML para incluir um arquivo de áudio com o áudio silenciado.

```
1 | <audio controls muted>
2 |     <source src="caminho/do/arquivo.mp3">
3 | </audio>
```

Exemplo de Código 6.6: Uso do elemento áudio com atributo `muted`.

Nesse exemplo 6.6, o atributo `muted` silencia o áudio do elemento `<audio>`.

6.1.4.5 Pré-carregamento do Áudio

O atributo `preload` especifica se o navegador deve pré-carregar o vídeo quando a página é carregada. O valor `none` indica que o navegador não deve pré-carregar o vídeo. O valor `metadata` indica que o navegador deve pré-carregar apenas os metadados do vídeo, como a duração, resolução e taxa de bits. O valor `auto` indica que o navegador deve pré-carregar o vídeo inteiro.

O uso do atributo `preload` pode afetar o desempenho do site, já que o pré-carregamento de um vídeo grande pode atrasar o tempo de carregamento da página. É recomendado utilizar o valor `none` para vídeos grandes e o valor `auto` para vídeos curtos. O código 6.7 um exemplo de como utilizar o atributo `preload`.

```
1 | <video src="video.mp4" preload="none"></video>
```

Exemplo de Código 6.7: Exemplo do atributo `preload`.

Neste exemplo 6.7, o valor `none` é utilizado para evitar o pré-carregamento do vídeo `video.mp4`.

6.2 Inserindo Vídeo em Uma Página HTML

É possível exibir vídeos em uma página HTML usando o elemento `<video>`. O HTML oferece várias maneiras de personalizar a apresentação e a interação com o vídeo. As subseções seguintes mostram como configurar a exibição de um vídeo em uma página web.

6.2.1 Elemento HTML para Exibição de Vídeo

O elemento `<video>` é um contêiner que define o vídeo a ser incluído em uma página HTML. O código 6.8 mostra o HTML básico para incluir um vídeo em uma página web.

```
1 | <video src="caminho/do/arquivo.mp4"></video>
```

Exemplo de Código 6.8: Uso do elemento video.

Nesse exemplo 6.8, o atributo `src` aponta para o arquivo de vídeo a ser incluído na página. Note que, assim como no elemento `` para imagens e `<audio>` para áudio, não é necessário fechar a `tag` do elemento `<video>`.

6.2.2 Inserindo Arquivos de Vídeo em Diferentes Formatos

Para vídeos, também é importante incluir diferentes formatos de arquivo usando o elemento `<source>` para garantir a compatibilidade do vídeo com diferentes navegadores. O elemento `<source>` é usado para especificar diferentes versões do arquivo de vídeo, cada uma em um formato de vídeo diferente. O navegador escolhe aquele formato que é mais apropriado para execução. Atualmente, o formato MP4 é nativamente suportado pelos navegadores mais populares. Você também pode definir um formato como padrão, como veremos mais adiante. O código 6.9 mostra o HTML para incluir um arquivo de vídeo com diferentes formatos.

```
1 | <video controls>
2 |   <source src="caminho/do/arquivo.mp4" type="video/mp4">
3 |   <source src="caminho/do/arquivo.webm" type="video/webm">
4 |   <source src="caminho/do/arquivo.ogv" type="video/ogg">
5 | </video>
```

Exemplo de Código 6.9: Uso do elemento video com diferentes arquivos.

Nesse exemplo 6.9, incluímos três versões do arquivo de vídeo, cada uma em um formato diferente: MP4, WebM e Ogg. O navegador irá escolher a versão mais apropriada suportada pelo dispositivo em que a página está sendo visualizada.

6.2.3 Formatos de Arquivos Suportados Pelos Navegadores

Assim como com o elemento `<audio>`, nem todos os navegadores suportam todos os formatos de arquivo de vídeo. Os formatos mais comuns de arquivo de vídeo suportados pelos navegadores são MP4, WebM e Ogg. No entanto, outros formatos também podem ser suportados, dependendo do navegador e do sistema operacional em que a página está sendo visualizada.

6.2.4 Atributos de Configuração

O elemento `<video>` oferece vários atributos que permitem personalizar a reprodução do vídeo na página. A seguir, alguns dos atributos mais comuns.

6.2.4.1 Controles de Reprodução

O atributo `controls` adiciona os controles padrão de reprodução, como play, pause, volume e progresso, ao elemento `<video>`. O código 6.10 mostra o HTML para adicionar os controles padrão de reprodução.

```
1 <video controls>
2   <source src="caminho/do/arquivo.mp4">
3 </video>
```

Exemplo de Código 6.10: Uso do elemento `video` com atributo `controls`.

Nesse exemplo, o atributo `controls` adiciona os controles padrão de reprodução ao elemento `<video>`. Isso inclui botões de *play*, *pause*, *volume* e progresso, que permitem ao usuário controlar a reprodução do vídeo.

6.2.4.2 Selenciando o Vídeo

O atributo `muted` silencia o áudio do elemento `<video>`. O código 6.11 mostra o HTML para incluir um arquivo de vídeo com o áudio silenciado.

```
1 <video controls muted>
2   <source src="caminho/do/arquivo.mp4">
3 </video>
```

Exemplo de Código 6.11: Uso do elemento `video` com o atributo `muted`.

Nesse exemplo 6.11, o atributo `muted` silencia o áudio do elemento `<video>`.

6.2.4.3 Reprodução Automática

O atributo `autoplay` inicia a reprodução do vídeo automaticamente quando a página é carregada. Vale ressaltar que, se o áudio do vídeo não estiver mutado, por questões de privacidade, este atributo nem sempre vai funcionar. O código 6.12 mostra o HTML para reproduzir automaticamente um arquivo de vídeo.

```
1 <video autoplay muted>
2   <source src="caminho/do/arquivo.mp4">
3 </video>
```

Exemplo de Código 6.12: Uso do elemento `video` com o atributo `autoplay`.

Nesse exemplo 6.12, o atributo `autoplay` inicia a reprodução do vídeo assim que a página é carregada.

6.2.4.4 Repetição da Reprodução

O atributo `loop` permite que o vídeo seja reproduzido continuamente, repetindo a reprodução quando ela termina. O código 6.13 mostra o HTML para repetir a reprodução de um vídeo.

```
1 <video controls loop>
2   <source src="caminho/do/arquivo.mp4">
3 </video>
```

Exemplo de Código 6.13: Uso do elemento `video` com o atributo `loop`.

Nesse exemplo 6.13, o atributo `loop` permite que o vídeo seja reproduzido continuamente, repetindo a reprodução quando ela termina.

6.2.4.5 Imagem de Capa

O atributo `poster` especifica uma imagem de visualização para o vídeo antes de ele ser reproduzido. O código 6.14 mostra o HTML para adicionar uma imagem de visualização.

```
1 <video controls poster="caminho/do/imagem.jpg">
2   <source src="caminho/do/arquivo.mp4">
3 </video>
```

Exemplo de Código 6.14: Uso do elemento `video` com o atributo `poster`.

Nesse exemplo, o atributo `poster` especifica o caminho para a imagem de visualização. Quando o usuário clica no botão de *play*, a imagem de visualização é substituída pelo vídeo.

Neste exemplo 6.15, adicionamos os atributos `controls`, `autoplay`, `loop`, `muted` e `poster` ao elemento `<video>`. Além disso, definimos um arquivo de vídeo em formato MP4.

```
1 <video controls autoplay loop muted poster="exemplo.jpg">
2   <source src="exemplo.mp4" type="video/mp4">
3 </video>
```

Exemplo de Código 6.15: Uso do elemento `video` com todos os atributos.

6.2.5 Adição de Legendas

As legendas são uma ótima forma de tornar seu vídeo mais acessível para pessoas com deficiência auditiva, além de ajudar a compreender melhor o conteúdo do vídeo para aqueles que têm dificuldades com o idioma falado. O elemento HTML `<video>` possui suporte nativo para legendas, e o processo de adicioná-las é bem simples.

O formato VTT (WebVTT) é um formato de legenda de vídeo baseado em texto que é suportado pelos navegadores modernos. O VTT é um formato simples e fácil de usar, que permite adicionar legendas a um vídeo em um arquivo separado, o que é ideal para garantir que o conteúdo do vídeo seja acessível a um público mais amplo.

O formato VTT suporta uma ampla gama de recursos, como a exibição de várias linhas de texto ao mesmo tempo, exibição de legendas em diferentes cores e tamanhos de fonte, sincronização precisa com o áudio do vídeo e exibição de marcações de tempo para cenas específicas do vídeo. O código 6.16 mostra um exemplo de como adicionar legendas VTT a um elemento de vídeo.

```
1 <video controls>
2   <source src="video.mp4" type="video/mp4">
3   <track label="Português" kind="subtitles" srclang="pt-br"
4     src="legenda.pt-br.vtt" default>
5   <track label="English" kind="subtitles" srclang="en-us"
6     src="legenda.en-us.vtt">
7 </video>
```

Exemplo de Código 6.16: Adicionando legenda a um vídeo com o elemento `track`.

Nesse exemplo 6.16, estamos adicionando um arquivo de legenda em VTT ao nosso elemento de vídeo usando a tag `<track>`. O atributo `label` é usado para definir o rótulo da legenda exibido no menu de seleção de legendas do player de vídeo. O atributo `srclang` é usado para especificar o idioma da legenda e o atributo `kind` define o tipo de legenda, no caso, `subtitles`.

O atributo `src` aponta para o arquivo de legenda em VTT. Note que o atributo `default` é usado para definir uma legenda como padrão, caso haja mais de uma legenda e nenhuma outra legenda seja selecionada. Dessa forma, é possível adicionar legendas VTT aos seus vídeos em HTML e torná-los mais acessíveis a um público mais amplo.

6.2.5.1 O Arquivo de Legenda VTT

O formato de legendas VTT é suportado pelos principais navegadores web e é fácil de ser implementado em páginas HTML. Além disso, o VTT oferece diversas opções de configuração, permitindo que sejam definidas fontes, tamanhos, cores e posicionamento das legendas, aprimorando a experiência do usuário com o conteúdo multimídia. O código 6.17 mostra um trecho de um arquivo VTT. A listagem em seguida mostra uma explicação sobre cada um dos elementos que compõem o trecho do arquivo de legenda VTT deste exemplo de código.

```
1 WEBVTT
2
3 STYLE
4 ::cue {
5   font-family: Arial, sans-serif;
6   font-size: 1.2em;
7   color: white;
8   background-color: black;
9 }
10
11 00:00:05.000 --> 00:00:08.000
12 Bem-vindo ao curso de HTML5 e CSS3!
13
14 00:00:10.000 --> 00:00:13.000
15 Neste curso, você vai aprender a criar páginas web
16
17 00:00:14.000 --> 00:00:17.000
18 com as tecnologias mais modernas da atualidade.
19
20 00:00:20.000 --> 00:00:24.000
21 Não é necessário ter conhecimento prévio em programação.
22
23 00:00:25.000 --> 00:00:29.000
24 Você também poderá usar editores na nuvem.
25
26 00:00:30.000 --> 00:00:34.000
27 Então, vamos começar!
```

Exemplo de Código 6.17: Exemplo de um arquivo de legenda VTT.

- a) **WEBVTT**: esta é a linha de identificação do arquivo de legenda VTT. Ele é usado para indicar que o arquivo é um arquivo de legenda no formato WebVTT.
- b) **00:00:05.000 → 00:00:08.000**: este é um marcador de tempo que indica quando a legenda deve aparecer e desaparecer. O primeiro marcador de tempo indica quando a legenda deve começar a ser exibida e o segundo indica quando ela deve ser removida. O formato é “hora:minuto:segundo.milissegundo”.
- c) **Bem-vindo ao curso de HTML5 e CSS3!**: este é o texto da legenda que será exibido.
- d) As linhas seguintes seguem o mesmo formato, com marcadores de tempo e texto das legendas correspondentes.

O bloco de código logo abaixo da palavra **STYLE** corresponde a um seletor CSS para configurar a fonte, cor e outras propriedades da legenda no arquivo VTT. Nesse caso, a fonte está definida como Arial ou uma fonte sans-serif caso Arial não esteja disponível, o tamanho da fonte é 1.2em, a cor do texto é branca e o fundo da legenda é preto.

As propriedades de estilo que podem ser definidas no arquivo VTT incluem `font-family`, `font-style`, `font-weight`, `font-size`, `color`, `background-color`, `text-align`, `direction`, `writing-mode` e outras. É importante ressaltar que nem todos os navegadores suportam todas as propriedades de estilo. Vale ressaltar também que esse é um trecho de código opcional.

Concluindo, os arquivos de legenda VTT podem ser criados com qualquer editor de texto, como o Bloco de Notas do Windows ou o TextEdit do Mac. Basta salvar o arquivo com a extensão “.vtt” e incluir o arquivo no elemento `<track>` do vídeo HTML.

6.3 Incorporando Vídeos do YouTube em Uma Página HTML

O YouTube é uma das maiores plataformas de compartilhamento de vídeo do mundo. É possível incorporar vídeos do YouTube em uma página HTML usando um *iframe*. Além disso, o YouTube oferece parâmetros de URL para personalizar a reprodução do vídeo incorporado, incluindo o tempo de início do vídeo, a opção de iniciar o vídeo automaticamente, entre outras. As subseções a seguir mostram como incorporar vídeos do YouTube em uma página HTML.

6.3.1 Usando *iframe* para Incorporar Vídeo do YouTube

O YouTube oferece um código de incorporação que pode ser copiado e colado em uma página HTML para exibir o vídeo desejado. O código 6.18 mostra HTML e usa um *iframe* para incorporar um vídeo do YouTube.


```
1 <iframe width="560" height="315" src="https://www.youtube.com/embed/VIDEO_ID"  
2 frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope;  
3 picture-in-picture" allowfullscreen></iframe>
```

Exemplo de Código 6.18: Incorporando vídeo do YouTube.

Nesse exemplo, substitua VIDEO_ID pelo ID do vídeo do YouTube que você deseja incorporar. O atributo width define a largura do vídeo, e o atributo height define a altura. Os outros atributos são usados para permitir que o vídeo seja reproduzido automaticamente e em tela cheia.

6.3.2 Personalizando a Reprodução de Vídeo do YouTube com Parâmetros de URL

O YouTube oferece parâmetros de URL para personalizar a reprodução do vídeo incorporado. Esses parâmetros são adicionados ao final da URL do vídeo e são separados por &. Os parâmetros mais comuns incluem controls, autoplay, mute e loop. O código 6.19 mostra o HTML para incorporar um vídeo do YouTube com parâmetros de URL.

```
1 <iframe width="560" height="315" src="https://www.youtube.com/embed/YOUR_VIDEO_ID?  
2 controls=0&autoplay=1&mute=1&loop=1" frameborder="0" allow="accelerometer;  
3 encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
```

Exemplo de Código 6.19: Incorporando vídeo do YouTube com parâmetros.

Nesse exemplo, substitua VIDEO_ID pelo ID do vídeo do YouTube que você deseja incorporar. Os parâmetros de URL personalizam a reprodução do vídeo, removendo os controles padrão, iniciando a reprodução automaticamente, silenciando o áudio e repetindo a reprodução.

6.4 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 6.1 Crie um elemento de áudio com o atributo controls e adicione dois elementos <source> com diferentes formatos de áudio para que o navegador possa escolher qual usar com base na compatibilidade. ■

Exercício 6.2 Adicione um elemento de áudio com uma música embutida, dois elementos <source> com diferentes formatos de áudio e um elemento <track> com a letra da música. ■

Exercício 6.3 Crie um elemento de áudio que comece a tocar automaticamente quando a página é carregada e que tenha controles visíveis. Adicione dois elementos `<source>` com diferentes formatos de áudio. ■

Exercício 6.4 Adicione um elemento de áudio com um atributo `loop` para que o áudio seja reproduzido continuamente. Adicione 3 elementos `<source>` com formatos distintos, tornando o MP3 como padrão. ■

Exercício 6.5 Crie um elemento de áudio com um atributo `preload` definido como “none” e adicione dois elementos `<source>` com diferentes formatos de áudio para que o navegador não carregue o áudio previamente. ■

Exercício 6.6 Adicione um áudio de um poema com início automático e com legendas em português e inglês. ■

Exercício 6.7 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, definindo a URL do vídeo e os atributos de largura e altura. ■

Exercício 6.8 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando múltiplas fontes de vídeo usando as tags `<source>` e especificando o tipo MIME de cada uma. ■

Exercício 6.9 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando múltiplas fontes de vídeo usando as tags `<source>` e definindo a ordem de preferência de cada uma. ■

Exercício 6.10 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando uma legenda usando a tag `<track>` e definindo o tipo e o URL do arquivo de legenda. ■

Exercício 6.11 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando múltiplas faixas de legenda usando as tags `<track>` e definindo o tipo e o URL de cada arquivo de legenda. ■

Exercício 6.12 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando um controle de volume usando o atributo `controls` e definindo o valor inicial do controle de volume. ■

Exercício 6.13 Crie um arquivo HTML que exiba um vídeo usando a tag `<video>`, adicionando um controle de progresso usando o atributo `controls` e definindo o valor inicial do controle de progresso. ■

Exercício 6.14 Crie um arquivo HTML que exiba um vídeo do YouTube incorporado em uma página, usando a tag `<iframe>` e os parâmetros de URL para definir o tempo inicial e final do vídeo. ■

Exercício 6.15 Crie um arquivo HTML que exiba uma playlist do YouTube incorporada em uma página, usando a tag `<iframe>` e os parâmetros de URL para definir o índice do vídeo inicial e o modo de reprodução. ■

Exercício 6.16 Crie um arquivo HTML que exiba um vídeo do YouTube incorporado em uma página, usando a tag `<iframe>` e os parâmetros de URL para definir o tamanho da tela do player e ocultar a barra de progresso. ■

Exercício 6.17 Crie um arquivo HTML que exiba uma playlist do YouTube incorporada em uma página, usando a tag `<iframe>` e os parâmetros de URL para definir o tamanho da tela do player e ocultar a barra de progresso. ■

Exercício 6.18 Crie um arquivo HTML que exiba um vídeo do YouTube incorporado em uma página, usando a tag `<iframe>` e os parâmetros de URL para reproduzir o vídeo em loop e exibir os controles de volume. ■

Exercício 6.19 Crie um arquivo HTML que exiba uma playlist do YouTube incorporada em uma página, usando a tag `<iframe>` e os parâmetros de URL para reproduzir a playlist em loop e exibir os controles de volume. ■

Exercício 6.20 Crie um arquivo HTML que exiba um vídeo do YouTube incorporado em uma página, usando a tag `<iframe>` e os parâmetros de URL para definir a qualidade do vídeo e exibir a legenda em português (se disponível). ■

6.5 Considerações Sobre o Capítulo

Neste capítulo, discutimos como incorporar áudio e vídeo em páginas HTML. Vimos como usar a *tag* `<audio>` para incorporar arquivos de áudio e a *tag* `<video>` para incorporar arquivos de vídeo. Além disso, aprendemos como incluir legendas usando a *tag* `<track>` e como incorporar vídeos do YouTube com *iframes* usando a API do YouTube. No próximo capítulo, veremos como usar formulários HTML para criar páginas interativas.

7. Formulários

Formulários são elementos fundamentais em páginas web que permitem aos usuários inserir informações e interagir com o conteúdo do site. Essas informações podem ser coletadas e processadas pelo servidor, permitindo ao desenvolvedor personalizar a experiência do usuário. Neste capítulo, vamos aprender os conceitos básicos sobre formulários e sua importância na criação de páginas web.

7.1 Criando um Formulário Básico

Para criar um formulário em HTML, precisamos utilizar a tag `<form>`. Esta tag permite que os campos de entrada sejam agrupados em um formulário, possibilitando o envio das informações do usuário para um servidor. O exemplo 7.1 apresenta a estrutura básica de um formulário.

```
1 <form action="processarForm.php" method="post">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <label for="email">E-mail:</label>
5   <input type="email" id="email" name="email"><br><br>
6   <input type="submit" value="Enviar">
7 </form>
```

Exemplo de Código 7.1: Exemplo de formulário básico.

No exemplo acima, temos um formulário que envia os dados para a página “processarForm.php” usando o método POST. Esse formulário possui dois campos de entrada de texto, um para o nome e outro para o e-mail, seguidos por um botão de envio. O atributo `for` do `<label>` está associado ao atributo `id` do `<input>`, permitindo ao usuário clicar no rótulo para selecionar o campo de entrada correspondente.

7.2 Enviando Dados de Formulários

Quando o usuário preenche um formulário e clica em “Enviar”, as informações inseridas são enviadas para o servidor, onde podem ser processadas e armazenadas por um programa que roda no servidor. Esse programa é criado usando-se uma linguagem de programação de aplicações web, como C#, Python, Java, PHP, Javascript, entre outras. Para enviar os dados do formulário, é preciso especificar o método de envio e o endereço de destino. Isso pode ser feito usando os atributos `method` e `action` da tag `<form>`.

O atributo `method` especifica o método HTTP que será usado para enviar os dados do formulário. Existem dois métodos comuns: GET e POST. O método GET envia os dados do formulário como parte da URL, enquanto o método POST envia os dados no corpo da requisição. O método POST é geralmente usado quando os dados enviados contêm informações confidenciais, como senhas ou informações financeiras.

O atributo `action` especifica a página de destino para onde os dados do formulário serão enviados. Essa página pode ser uma página do próprio site ou de um site externo. Quando o usuário clica no botão “Enviar”, o navegador redireciona o usuário para a página de destino especificada no atributo `action`. O exemplo 7.2 mostra um formulário que envia dados usando o método POST, tendo como destino um *script* em linguagem PHP cujo endereço é “processarForm.php”. Vale ressaltar que existem várias linguagens para criação de programas servidor (aplicações web), sendo que a PHP foi usado como exemplo por ser uma das mais populares.

```
1 <form action="processarForm.php" method="post">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <label for="email">E-mail:</label>
5   <input type="email" id="email" name="email"><br><br>
6   <input type="submit" value="Enviar">
7 </form>
```

Exemplo de Código 7.2: Exemplo de formulário com método POST.

No exemplo 7.2, o método POST é especificado no atributo `method` da tag `form` e os dados do formulários são enviados através do corpo da requisição quando o usuário clica no botão “Enviar”. O processamento do formulário, neste caso, vai ocorrer no *script* “processarForm.php” definido no atributo `action`. No exemplo 7.3, a seguir, temos um formulário que envia dados usando o método GET para serem processados pelo *script* “processarForm.php”.

```
1 <form action="processarForm.php" method="get">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <label for="email">E-mail:</label>
5   <input type="email" id="email" name="email"><br><br>
6   <input type="submit" value="Enviar">
7 </form>
```

Exemplo de Código 7.3: Exemplo de formulário com método GET.

No exemplo 7.3, o método GET é especificado no atributo `method` da tag `<form>`, e os dados do formulário são enviados como parte da URL quando o usuário clica no botão “Enviar”. Observe que o endereço de destino é o mesmo para ambos os exemplos, que é “processarForm.php”.

7.3 Campos de Formulários

Os campos de formulários permitem aos usuários inserir informações em um formulário. Existem diversos tipos de campos de formulários disponíveis em HTML, incluindo campos de texto, e-mails, senhas, *checkboxes*, *radio buttons* e *selects*. Esta seção apresenta uma explicação sobre cada um desses tipos de campos.

7.3.1 Campos de Texto Longo

Os campos de texto longo permitem que os usuários insiram uma ou mais linhas de texto em um formulário. Eles são criados usando a tag `<input>` com o atributo `type` definido como `text` ou `textarea`. O exemplo 7.4 mostra como criar um campo de texto em HTML.

```
1 <label for="comentario">Deixe um comentário:</label><br>
2 <textarea id="comentario" name="comentario" rows="5" cols="50"></textarea>
```

Exemplo de Código 7.4: Exemplo de campo de texto longo.

Neste exemplo 7.4, a tag `<textarea>` é usada para criar um campo de texto longo com 5 linhas e 50 colunas. O atributo `id` é usado para associar o rótulo do campo de texto e o atributo `name` é usado para identificar o campo de texto no formulário.

7.3.2 Campos de E-mail e Senha

Os campos de e-mail e senha são semelhantes aos campos de texto, mas possuem recursos adicionais. O campo de e-mail é usado para capturar endereços de e-mail e o campo de senha é usado para capturar senhas. O código 7.5 mostra um exemplo de uso.

```
1 <label for="email">E-mail:</label>
2 <input type="email" id="email" name="email"><br><br>
3
4 <label for="senha">Senha:</label>
5 <input type="password" id="senha" name="senha"><br><br>
```

Exemplo de Código 7.5: Exemplo de campos de e-mail e senha.

Neste exemplo 7.5, o campo de e-mail é criado usando a *tag* `<input>` com o atributo `type` definido como `email`. O campo de senha é criado usando a *tag* `<input>` com o atributo `type` definido como `password`.

7.3.3 Checkboxes

Os *checkboxes* são usados para permitir que os usuários selecionem uma ou mais opções em um formulário. Eles são criados usando a *tag* `<input>` com o atributo `type` definido como `checkbox`. O código 7.6 mostra um exemplo de como criar um *checkbox*.

```
1 <input type="checkbox" id="checkbox1" name="checkbox1" value="1">
2 <label for="checkbox1">Opção 1</label><br>
3 <input type="checkbox" id="checkbox2" name="checkbox2" value="2">
4 <label for="checkbox2">Opção 2</label><br>
5 <input type="checkbox" id="checkbox3" name="checkbox3" value="3">
6 <label for="checkbox3">Opção 3</label><br>
```

Exemplo de Código 7.6: Exemplos de *checkboxes*.

Neste exemplo 7.6, três *checkboxes* são criados usando a *tag* `<input>` com o atributo `type` definido como `checkbox`. O atributo `id` é usado para associar o rótulo (`label`) ao campo (`input`). É importante ressaltar que essa associação de um rótulo a um campo tem funções semântica e interativa. A função semântica vai proporcionar recursos de acessibilidade, dando um título ao campo, enquanto a função interativa faz com que, ao clicar no rótulo, o campo associado receba o foco de digitação.

7.3.4 Radio Buttons

Os *radio buttons* são usados para permitir que os usuários selecionem uma opção em um conjunto de opções mutuamente exclusivas. Eles são criados usando a *tag* `<input>` com o atributo `type` definido como `radio`. O código 7.7 mostra um exemplo de como criar *radio buttons* em HTML.

Neste exemplo 7.7, três *radio buttons* são criados usando a *tag* `<input>` com o atributo `type` definido como `radio`. O atributo `name` é usado para agrupar as opções mutuamente exclusivas. Quando o usuário seleciona uma opção, todas as outras opções no mesmo grupo são desmarcadas.


```
1 <label for="opcao1">Opção 1</label>
2 <input type="radio" id="opcao1" name="opcao" value="1"><br>
3 <label for="opcao2">Opção 2</label>
4 <input type="radio" id="opcao2" name="opcao" value="2"><br>
5 <label for="opcao3">Opção 3</label>
6 <input type="radio" id="opcao3" name="opcao" value="3"><br>
```

Exemplo de Código 7.7: Exemplo de campo de checagem exclusiva com elementos *radio buttons*.

7.3.5 Selects

Os selects são usados para permitir que os usuários selecionem uma opção em um conjunto de opções. Eles são criados usando a tag `<select>` com a tag `<option>` dentro dela. O código 7.8 um exemplo de como criar um select.

```
1 <label for="esporte">Esporte favorito:</label>
2 <select id="esporte" name="esporte">
3   <option value="futebol">Futebol</option>
4   <option value="basquete">Basquete</option>
5   <option value="volei">Vôlei</option>
6 </select>
```

Exemplo de Código 7.8: Exemplo de campo de seleção com elemento select.

Neste exemplo 7.8, um campo de seleção é criado usando a tag `<select>` e três opções são adicionadas usando a tag `<option>`. O atributo `value` é usado para definir o valor da opção que será enviado para o servidor quando o usuário seleciona uma opção.

7.4 Validação de Formulários com Atributos HTML

Os atributos HTML fornecem uma maneira fácil e rápida de validar os dados inseridos pelo usuário em um formulário. O uso desses atributos complementa a validação via JavaScript. Esta seção apresenta alguns dos atributos de validação disponíveis em HTML.

7.4.1 Campos Obrigatórios

O atributo `required` é usado para garantir que um campo de entrada seja preenchido antes que o formulário possa ser enviado. O código 7.9 mostra um exemplo de uso.

```
1 <label for="nome">Nome:</label>
2 <input type="text" id="nome" name="nome" required>
```

Exemplo de Código 7.9: Exemplo de campo de texto com o atributo `required`.

Neste exemplo 7.9, o atributo `required` é adicionado ao campo de texto para garantir que o usuário insira um nome antes que o formulário seja enviado.

7.4.2 Valores Mínimos e Máximos

O atributo `min` é usado para definir o valor mínimo que pode ser inserido em um campo de entrada, enquanto o atributo `max` é usado para definir o valor máximo. Esses atributos são úteis para garantir que os usuários insiram valores válidos em campos numéricos, datas ou horários. O código 7.10 mostra um exemplo de como usar os atributos `min` e `max` em um campo numérico.

```
1 <label for="idade">Idade:</label>
2 <input type="number" id="idade" name="idade" min="18" max="65">
```

Exemplo de Código 7.10: Exemplo de campo de número com os atributos `min` e `max`.

Neste exemplo, os atributos `min` e `max` são adicionados ao campo de número para garantir que o usuário insira uma idade entre 18 e 65 anos.

7.4.3 Valores Baseados em Expressões Regulares

O atributo `pattern` é usado para especificar uma expressão regular que deve corresponder ao valor inserido em um campo de entrada. Isso é útil para garantir que os usuários insiram valores válidos em campos de entrada que requerem um formato específico, como um endereço de e-mail ou um número de telefone. O código 7.11 mostra um exemplo de como usar o atributo `pattern` em um campo de entrada de texto comum.

```
1 <label for="email">E-mail:</label>
2 <input type="email" id="email" name="email" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$">
```

Exemplo de Código 7.11: Exemplo de campo de entrada com o atributo `pattern`.

Neste exemplo, o atributo `pattern` é adicionado ao campo de entrada de e-mail para garantir que o valor inserido corresponda a um endereço de e-mail válido. A expressão regular especificada no atributo `pattern` é usada para validar o valor inserido.

7.4.4 Tamanhos Mínimo e Máximo

O atributo `maxlength` é usado para definir o número máximo de caracteres que podem ser inseridos em um campo de entrada, enquanto o atributo `minlength` é usado para definir o número mínimo de caracteres. Esses atributos são úteis para limitar o tamanho do texto inserido em campos de texto. O código 7.12 mostra um exemplo de como usar os atributos `maxlength` e `minlength` em um campo de texto.

```
1 | <label for="comentario">Deixe um comentário:</label><br>
2 | <textarea id="comentario" name="comentario" minlength="10" maxlength="500"></textarea>
```

Exemplo de Código 7.12: Exemplo de campo de texto com os atributos `maxlength` e `minlength`.

Neste exemplo, os atributos `maxlength` e `minlength` são adicionados ao campo de texto para garantir que o usuário insira um comentário com pelo menos 10 caracteres e no máximo 500 caracteres.

7.5 Tipos de Campos Avançados

Nesta seção, serão apresentados os tipos de campos avançados disponíveis em HTML, bem como seus atributos e exemplos de uso.

7.5.1 Botão

O campo do tipo botão é utilizado para criar botões que executam ações específicas em um formulário. Esses botões não enviam dados do formulário para o servidor, apenas realizam alguma ação definida pelo programador. Para criar um botão, utiliza-se a tag `<button>` com o atributo `type` definido como `button`. O exemplo 7.13 a seguir mostra um exemplo de código HTML para criar um botão.

```
1 | <button type="button">Clique aqui</button>
```

Exemplo de Código 7.13: Exemplo de campo do tipo botão.

O código 7.13 a seguir mostra um exemplo de código HTML para criar um botão. Ao clicar no botão, nada será enviado ao servidor, mas é possível executar uma ação, como redirecionar o usuário para outra página.

7.5.2 Cor

O campo do tipo cor permite que o usuário escolha uma cor a partir de uma paleta de cores. Para criar um campo de seleção de cor, utiliza-se a tag `<input>` com o atributo `type` definido como `color`. O exemplo 7.14 a seguir mostra um exemplo de código HTML para criar um campo de seleção de cor.

```
1 | <label for="cor">Escolha uma cor:</label>
2 | <input type="color" id="cor" name="cor">
```

Exemplo de Código 7.14: Exemplo de campo do tipo cor.

No código 7.14, o campo de seleção de cor é definido pelo atributo `type="color"`. Quando o usuário clica no campo, uma paleta de cores é exibida e o usuário pode escolher uma cor. O valor selecionado é enviado para o servidor juntamente com os outros dados do formulário.

7.5.3 Data

O campo do tipo data é utilizado para permitir que o usuário insira uma data. Para criar um campo de data, utiliza-se a tag `<input>` com o atributo `type` definido como `date`. O exemplo 7.15 a seguir mostra um exemplo de código HTML para criar um campo de data.

```
1 <label for="data">Insira uma data:</label>
2 <input type="date" id="data" name="data">
```

Exemplo de Código 7.15: Exemplo de campo do tipo data.

No exemplo 7.15, o campo de data é definido pelo atributo `type="date"`. Quando o usuário clica no campo, é exibido um calendário para seleção da data. O valor selecionado é enviado para o servidor juntamente com os outros dados do formulário.

7.5.4 Data e Hora

O campo de data e hora permite que o usuário selecione uma data e hora usando um calendário integrado. Esse campo é criado usando a tag `<input>` com o atributo `type` definido como `"datetime-local"`. O formato da data e hora selecionados depende do formato especificado no atributo `value`. O exemplo 7.16 mostra um campo de data e hora.

```
1 <label for="data-hora">Data e Hora:</label>
2 <input type="datetime-local" id="data-hora" name="data-hora" value="2022-04-05T14:30"><br><br>
```

Exemplo de Código 7.16: Exemplo de campo de data e hora.

No exemplo acima, o atributo `value` especifica a data e hora inicial do campo no formato `"AAAA-MM-DDTHH:MM"`, onde `"AAAA"` é o ano, `"MM"` é o mês, `"DD"` é o dia, `"HH"` é a hora e `"MM"` é o minuto. O campo de data e hora exibido para o usuário depende do navegador utilizado e pode variar entre diferentes navegadores.

7.5.5 Arquivo

O campo de seleção de arquivo permite que o usuário selecione um ou mais arquivos do seu computador para serem enviados para o servidor. Esse campo é criado usando a tag `<input>` com o atributo `type` definido como `"file"`. O exemplo 7.17 mostra um campo de seleção de arquivo.

```
1 <label for="arquivo">Selecione um arquivo:</label>
2 <input type="file" id="arquivo" name="arquivo"><br><br>
```

Exemplo de Código 7.17: Exemplo de campo de seleção de arquivo.

No exemplo acima, o usuário pode clicar no botão “Procurar” para selecionar um arquivo no seu computador. Quando o usuário envia o formulário, o arquivo selecionado é enviado para o servidor junto com os outros dados do formulário.

7.5.6 Campo Oculto

O tipo de campo `hidden` é usado para enviar informações ocultas do formulário para o servidor. Esse tipo de campo é útil para enviar informações que não precisam ser exibidas ao usuário, como por exemplo, informações de identificação. O código 7.18 a seguir mostra um exemplo de como criar um campo oculto.

```
1 <form action="processarForm.php" method="post">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <input type="hidden" id="usuarioId" name="usuarioId" value="12345">
5   <input type="submit" value="Enviar">
6 </form>
```

Exemplo de Código 7.18: Exemplo de campo oculto.

No exemplo acima, temos um campo oculto com o nome `usuarioId` e valor `12345`. Esse valor é enviado junto com o formulário, mas não é exibido para o usuário. O campo oculto pode ser acessado e processado no lado do servidor pelo nome `usuarioId`.

7.5.7 Imagem

O tipo de campo `image` é usado para criar botões com imagens clicáveis dentro de um formulário. O código 7.19 a seguir mostra um exemplo de como criar um botão de imagem.

```
1 <form action="processarForm.php" method="post">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <input type="image" src="botao.png" alt="Enviar">
5 </form>
```

Exemplo de Código 7.19: Exemplo de botão de imagem.

No exemplo acima, temos um botão de imagem com a imagem `botao.png` e o texto alternativo `Enviar`. Quando o usuário clica na imagem, o formulário é enviado para o servidor.

7.5.8 Mês

O tipo de campo `month` é usado para permitir que o usuário selecione um mês e um ano em um campo de entrada. O código 7.20 a seguir mostra um exemplo de como criar um campo de entrada de mês.

```
1 <form action="processarForm.php" method="post">
2   <label for="data">Selecione um mês:</label>
3   <input type="month" id="data" name="data"><br><br>
4   <input type="submit" value="Enviar">
5 </form>
```

Exemplo de Código 7.20: Exemplo de campo de entrada de mês.

No exemplo 7.20, temos um campo de entrada de mês com o nome `data`. Quando o usuário seleciona um mês e um ano, essas informações são enviadas para o servidor.

7.5.9 Número

O campo `<input>` com o atributo `type` definido como `"number"` permite que o usuário insira um número. O navegador exibe uma interface específica para entrada de números que inclui controles de aumento e diminuição. O valor inserido é validado automaticamente pelo navegador para garantir que seja um número. É possível definir os valores máximo e mínimo permitidos pelo campo usando os atributos `max` e `min`, respectivamente. Além disso, o atributo `step` pode ser usado para definir o incremento ou decremento entre os valores permitidos. O exemplo 7.21 apresenta um código HTML para um campo de entrada de número.

```
1 <label for="idade">Idade:</label>
2 <input type="number" id="idade" name="idade" min="18" max="100" step="1">
```

Exemplo de Código 7.21: Exemplo de campo de entrada de número.

Nesse exemplo 7.21, o campo de entrada `<input>` com o atributo `type` definido como `"number"` é usado para permitir a entrada da idade do usuário. O valor mínimo permitido é definido como 18 pelo atributo `min`, enquanto o valor máximo permitido é definido como 100 pelo atributo `max`. O atributo `step` define o incremento/decremento entre os valores permitidos, neste caso, 1.

7.5.10 Intervalo

O campo `<input>` com o atributo `type` definido como `"range"` permite que o usuário selecione um valor em um intervalo de valores predefinidos. O navegador exibe uma interface específica para seleção do valor, que pode ser uma barra deslizante, uma caixa de seleção ou outros controles, dependendo do navegador e do sistema operacional. O valor selecionado é retornado como um número. É possível definir os valores mínimo e máximo permitidos pelo campo usando os atributos `min` e `max`, respectivamente. Além disso, o atributo `step` pode ser usado para definir o incremento ou decremento entre os valores permitidos. O exemplo 7.22 apresenta um código HTML para um campo de entrada de intervalo.

```
1 <label for="nota">Nota:</label>
2 <input type="range" id="nota" name="nota" min="0" max="10" step="0.1">
```

Exemplo de Código 7.22: Exemplo de campo de entrada de intervalo.

Nesse exemplo 7.22, o campo de entrada `<input>` com o atributo `type` definido como `"range"` é usado para permitir a seleção da nota de um usuário. O valor mínimo permitido é definido como 0 pelo atributo `min`, enquanto o valor máximo permitido é definido como 10 pelo atributo `max`. O atributo `step` define o incremento/decremento entre os valores permitidos, neste caso, 0.1.

7.5.11 Botão de Redefinição

O botão de redefinição é um tipo de botão que permite que o usuário redefina os valores de todos os campos do formulário para seus valores iniciais. Quando o usuário clica em um botão de redefinição, todos os valores inseridos em campos do formulário são apagados e retornam para seus valores padrão. O código 7.23 a seguir mostra um exemplo de código para um botão de redefinição.

```
1 <form action="processarForm.php" method="post">
2   <label for="nome">Nome:</label>
3   <input type="text" id="nome" name="nome"><br><br>
4   <label for="email">E-mail:</label>
5   <input type="email" id="email" name="email"><br><br>
6   <input type="reset" value="Limpar">
7   <input type="submit" value="Enviar">
8 </form>
```

Exemplo de Código 7.23: Exemplo de botão de redefinição.

No exemplo 7.23, temos um formulário que contém um botão de redefinição e um botão de envio. Quando o usuário clica no botão de redefinição, todos os campos do formulário são limpos e retornam aos seus valores padrão. O botão de redefinição é criado usando o tipo de campo `reset`, como mostrado na linha 6 do código.

7.5.12 Busca

O campo de busca é um tipo de campo de entrada de texto que permite que o usuário insira termos de pesquisa para buscar informações em um site ou aplicativo. Quando o usuário digita um termo de pesquisa e pressiona Enter, a busca é processada no servidor e página é atualizada para exibir os resultados da pesquisa. O código 7.24 a seguir mostra um exemplo de código para um campo de busca.

```
1 <form>
2   <label for="busca">Buscar:</label>
3   <input type="search" id="busca" name="busca">
4   <input type="submit" value="Buscar">
5 </form>
```

Exemplo de Código 7.24: Exemplo de campo de busca.

No exemplo 7.24, temos um campo de busca que permite que o usuário digite um termo de pesquisa e clique no botão "Buscar" para iniciar a busca. O campo de busca é criado usando o tipo de campo `search`, como mostrado na linha 3 do código.

7.5.13 Telefone

O campo de telefone é um tipo de campo de entrada de texto que permite que o usuário insira um número de telefone. Quando o usuário clica em um campo de telefone em um dispositivo móvel, o teclado exibe automaticamente as teclas numéricas para facilitar a entrada do número de telefone. O código 7.25 a seguir mostra um exemplo de código para um campo de telefone.

```
1 <form>
2   <label for="telefone">Telefone:</label>
3   <input type="tel" id="telefone" name="telefone">
4 </form>
```

Exemplo de Código 7.25: Exemplo de campo de telefone.

No exemplo 7.25, temos um campo de telefone que permite que o usuário insira um número de telefone. O campo de telefone é criado usando o tipo de campo `tel`, como mostrado na linha 3 do código.

7.5.14 Horário

O campo de entrada do tipo `time` permite ao usuário selecionar um horário. Esse campo é representado por uma caixa de seleção que permite selecionar horas, minutos e segundos. O exemplo 7.26 mostra como criar um campo de entrada do tipo `time`.

```
1 <label for="horario">Horário:</label>
2 <input type="time" id="horario" name="horario">
```

Exemplo de Código 7.26: Exemplo de campo de entrada do tipo `time`.

Nesse exemplo 7.26, o atributo `id` do `<input>` deve ser definido e o atributo `name` é opcional. O valor do campo é enviado como uma *string* no formato “hh:mm:ss”, onde “hh” representa as horas (00 a 23), “mm” representa os minutos (00 a 59) e “ss” representa os segundos (00 a 59). Se o usuário não selecionar um valor para os segundos, o valor padrão será “00”. Se o usuário não selecionar um valor para as horas ou minutos, o valor padrão será “00”.

7.5.15 URL

O campo de entrada do tipo `url` permite ao usuário inserir uma URL válida. Esse campo é semelhante ao campo de entrada do tipo `text`, mas com a validação adicional para garantir que a entrada seja uma URL com sintaxe válida. O exemplo 7.27 mostra como criar um campo de entrada do tipo `url`.

```
1 <label for="site">Site:</label>
2 <input type="url" id="site" name="site">
```

Exemplo de Código 7.27: Exemplo de campo de entrada do tipo `url`.

Nesse exemplo 7.27, o valor do campo de entrada deve ser uma URL válida. Se o usuário inserir uma URL inválida, o navegador pode exibir uma mensagem de erro.

7.5.16 Semana

O campo de entrada de semana permite que o usuário selecione uma semana e é representado visualmente como um controle com um calendário. O atributo `type` é definido como `week` para criar um campo de entrada de semana. O valor do campo é uma sequência no formato “AAAA-Sx”, onde “AAAA” é o ano e “x” é o número da semana. O código 7.28 mostra um exemplo a seguir mostra como criar um campo de entrada de semana.

```
1 <label for="semana">Selecione uma semana:</label>
2 <input type="week" id="semana" name="semana">
```

Exemplo de Código 7.28: Exemplo de campo de entrada de semana.

No exemplo 7.28, o valor do campo de entrada deve ser uma semana válida. Se o usuário inserir uma semana inválida, o navegador pode exibir uma mensagem de erro.

7.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 7.1 Crie um formulário HTML com um campo de texto para o atributo nome e um botão de envio. Utilize o atributo `action` do elemento `<form>` para enviar o formulário para uma página PHP ou outro script do lado do servidor. ■

Exercício 7.2 Crie um formulário HTML com um campo de texto para o atributo e-mail e um campo de senha para a senha do usuário. Adicione um botão de envio e utilize o atributo `method` do elemento `<form>` para definir o método HTTP como POST. ■

Exercício 7.3 Crie um formulário HTML com uma caixa de seleção para escolher uma cor (vermelho, verde ou azul) e um botão de envio. ■

Exercício 7.4 Crie um formulário HTML com um campo de texto para receber a idade do usuário e um campo de seleção para escolher o gênero (masculino, feminino ou outro). Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar os campos obrigatórios. ■

Exercício 7.5 Crie um formulário HTML com um campo de texto para o endereço de e-mail e um campo de seleção para escolher um assunto (dúvida, sugestão ou reclamação). Adicione um campo de texto longo para a mensagem e um botão de envio. Utilize o atributo `mailto` do elemento `<form>` para enviar o formulário por e-mail. ■

Exercício 7.6 Crie um formulário HTML com um campo de texto para a busca de um termo no Google e um botão de envio. Utilize o atributo `action` do elemento `<form>` para enviar o formulário para o mecanismo de busca do Google. Quando o formulário for enviado, exiba os resultados da busca em uma nova página. ■

Exercício 7.7 Crie um formulário HTML com dois campos de seleção para escolher o dia e o mês de nascimento do usuário. Adicione um botão de envio e utilize o atributo `method` do elemento `<form>` para definir o método HTTP como GET. ■

Exercício 7.8 Crie um formulário HTML com uma caixa de seleção para escolher um país e um campo de texto para inserir o endereço. Utilize o elemento `<textarea>` para o campo de texto e o atributo `required` do elemento `<select>` para tornar a escolha do país obrigatória. Adicione um botão de envio e utilize o atributo `method` do elemento `<form>` para definir o método HTTP como POST. ■

Exercício 7.9 Crie um formulário HTML com dois campos de seleção de horário para escolher o horário de início e de término de uma reserva em um hotel. Adicione um campo de texto para inserir o nome do hóspede e um botão de envio. Utilize o atributo `method` do elemento `<form>` para definir o método HTTP como POST e o atributo `action` para enviar o formulário para um script PHP que irá processar este formulário. ■

Exercício 7.10 Crie um formulário HTML com um campo do tipo `date` para inserir uma data de nascimento. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.11 Crie um formulário HTML com um campo do tipo `datetime-local` para inserir a data e hora de um evento. Adicione um botão de envio e utilize o atributo `min` do elemento `<input>` para definir a data mínima como a data atual. ■

Exercício 7.12 Crie um formulário HTML com um campo do tipo `time` para inserir o horário de um compromisso. Adicione um botão de envio e utilize o atributo `step` do elemento `<input>` para definir incrementos de 15 minutos no horário. ■

Exercício 7.13 Crie um formulário HTML com um campo do tipo `range` para selecionar um número entre 1 e 10. Adicione um botão de envio e utilize o atributo `value` do elemento `<input>` para definir o valor inicial como 5. ■

Exercício 7.14 Crie um formulário HTML com um campo do tipo `color` para selecionar uma cor. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.15 Crie um formulário HTML com um campo do tipo `week` para selecionar uma semana do ano. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.16 Crie um formulário HTML com um campo do tipo `month` para selecionar um mês do ano. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.17 Crie um formulário HTML com um campo do tipo `url` para inserir uma URL. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.18 Crie um formulário HTML com um campo do tipo `search` para inserir um termo de busca. Adicione um botão de envio e utilize o atributo `placeholder` do elemento `<input>` para mostrar um texto de exemplo dentro do campo. ■

Exercício 7.19 Crie um formulário HTML com um campo receber um e-mail e um `input` do tipo `image` para permitir que o usuário clique em uma imagem para enviar o formulário. Utilize o atributo `src` do elemento para definir a imagem do botão. ■

Exercício 7.20 Crie um formulário HTML com um campo do tipo `number` para inserir um número entre 1 e 100. Adicione um botão de envio e utilize o atributo `required` do elemento `<input>` para tornar o campo obrigatório. ■

Exercício 7.21 Crie um formulário HTML com um campo para inserir um endereço de e-mail. Adicione um botão de envio e utilize o atributo `type` do elemento `<input>` definido como `"email"` para aplicar a validação de e-mail. ■

Exercício 7.22 Crie um formulário HTML com um campo do tipo `password` para inserir uma senha. Adicione um botão de envio e utilize o atributo `minlength` do elemento `<input>` para definir um tamanho mínimo de 8 caracteres. ■

Exercício 7.23 Crie um formulário HTML com um campo do tipo `number` para inserir a idade do usuário. Adicione um botão de envio e utilize o atributo `min` do elemento `<input>` para definir a idade mínima como 18 anos e a máxima como 120 anos. ■

Exercício 7.24 Crie um formulário HTML com um campo do tipo `url` para inserir uma URL. Adicione um botão de envio e utilize o atributo `pattern` do elemento `<input>` para definir um padrão que permita somente URLs que comecem com `"https://"` ou `"http://"`. ■

Exercício 7.25 Crie um formulário HTML com um campo do tipo `tel` para inserir um número de telefone. Adicione um botão de envio e utilize o atributo `pattern` do elemento `<input>` para definir um padrão que permita somente números de telefone no formato "(xx) xxxx-xxxx" ou "(xx) xxxxx-xxxx". ■

7.7 Considerações Sobre o Capítulo

Neste capítulo, discutimos como criar formulários em HTML e como usar diferentes tipos de campos de formulários. Vimos como os atributos HTML podem ser usados para validar dados de formulários. É importante lembrar que a validação de formulários é uma parte crucial da criação de formulários para garantir que os usuários insiram dados válidos e precisos. Concluindo, lembre-se de que, ao criar formulários em HTML, é importante manter o design do formulário simples e fácil de entender para os usuários. Os rótulos devem ser claros e informativos, e os campos devem ser organizados de forma lógica. Além disso, os formulários devem ser testados em vários navegadores e dispositivos para garantir que funcionem corretamente para todos os usuários.



Parte II: CSS

8	Introdução	111
9	Cores, Planos de Fundo, Bordas e Margens	120
10	Fontes, Textos, Links e Listas	137
11	Pseudo-Seletores CSS	152
12	Exibição e Posicionamento de Elementos	161
13	Leiautes de Página com <i>Flexbox</i>	173
14	Leiautes de Página com <i>CSS Grid</i>	191
15	Responsividade	203



8. Introdução

O CSS (*Cascading Style Sheets*) é uma linguagem utilizada para estilizar páginas web. Ela é responsável pela aparência visual das páginas e é fundamental para garantir uma boa experiência do usuário.

Através do CSS, é possível definir a cor, fonte, tamanho, margem, espaçamento e outros atributos dos elementos HTML. Dessa forma, os desenvolvedores podem criar páginas com um design agradável e coerente, que transmite informações de forma clara e objetiva.

Além disso, o CSS permite criar layouts responsivos, que se adaptam a diferentes tamanhos de tela. Isso é essencial para garantir que o conteúdo da página seja exibido de forma adequada em dispositivos móveis, que têm telas menores e diferentes proporções em relação aos computadores desktop.

Outra funcionalidade do CSS é a possibilidade de adicionar animações e transições aos elementos HTML. Esses recursos permitem criar uma experiência mais interativa e envolvente para o usuário, tornando a navegação mais agradável e aumentando o engajamento com a página.

Em resumo, o CSS é uma ferramenta essencial para os desenvolvedores de páginas web, pois permite criar páginas visualmente atraentes, coerentes e adaptáveis a diferentes dispositivos. Ao longo desta seção, serão abordados conceitos fundamentais do CSS, desde os seletores básicos até técnicas avançadas de layout e animação.

Ao longo deste capítulo, veremos como associar códigos CSS a um documento HTML, incluindo a inserção pelo atributo `style`, pelo elemento `style` e por arquivo CSS externo. Veremos também como criar seletores CSS básicos baseados em *tags*, classes e IDs, além de seletores compostos pela combinação desses.

8.1 Associação de CSS a Documentos HTML

A associação de estilos CSS a documentos HTML é uma tarefa fundamental para o desenvolvimento de páginas web. Por meio do CSS, é possível definir estilos para elementos HTML, como fontes, cores, tamanhos, margens, posicionamento, entre outras propriedades. Basicamente, o visual de uma página HTML depende do CSS aplicado a ela.

Existem três maneiras principais de associar estilos CSS a documentos HTML: via atributo `style`, via elemento `<style>` e via arquivos externos. Cada uma dessas formas tem suas vantagens e desvantagens e é de suma importância conhecê-las para escolher a melhor opção em cada situação.

8.1.1 CSS via Atributo de Estilo

A forma mais básica de associar um estilo CSS a um elemento HTML é por meio do atributo `style`. Esse atributo pode ser utilizado em qualquer elemento HTML e permite definir estilos diretamente no próprio elemento. O código 8.1 mostra um exemplo de como utilizar o atributo `style` em um elemento HTML para definir estilos CSS, uma vez que você terá que utilizar o atributo `style` em todos os elementos HTML que desejar estilizar.

```
1 | <p style="color: red; font-size: 24px;">Este é um exemplo  
2 | de texto com estilo definido pelo atributo style.</p>
```

Exemplo de Código 8.1: Exemplo de CSS via Atributo `style`.

Neste exemplo 8.1, o atributo `style` é utilizado no elemento `<p>` para definir a cor do texto como vermelho e o tamanho da fonte como 24 pixels. Uma das principais vantagens de utilizar o atributo `style` é a sua simplicidade, já que não é necessário criar um arquivo CSS separado para definir os estilos. No entanto, essa forma de associar estilos pode tornar o código HTML mais extenso e difícil de manter em projetos maiores.

8.1.2 CSS via Elemento de Estilo

Outra forma de associar estilos CSS a documentos HTML é por meio do elemento `<style>`. Esse elemento pode ser utilizado dentro da seção `<head>` do documento HTML e permite definir estilos para todos os elementos do documento. O código 8.2 mostra um exemplo de como utilizar o elemento `<style>` para definir estilos CSS para todos os elementos `<p>` de um documento HTML.

Neste exemplo 8.2, o elemento `<style>` é utilizado para definir que todos os elementos `<p>` do documento terão a cor vermelha no texto e o tamanho de 24 pixels na fonte. Uma das principais vantagens de utilizar o elemento `<style>` é a sua capacidade de definir estilos para todos os elementos


```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Exemplo de CSS via Elemento <style></title>
5          <style>
6              p {
7                  color: red;
8                  font-size: 24px;
9              }
10         </style>
11     </head>
12     <body>
13         <p>Este é um exemplo de texto com estilo definido pelo elemento style.</p>
14         <p>Este é outro exemplo de texto com estilo definido pelo elemento style.</p>
15     </body>
16 </html>
```

Exemplo de Código 8.2: Exemplo de CSS via Elemento `<style>`.

de um documento HTML de uma só vez. Além disso, essa forma de associar estilos permite separar a definição dos estilos da estrutura do documento HTML, o que torna o código mais organizado e fácil de manter em projetos maiores. Porém, essa abordagem não permite compartilhar o CSS entre páginas distintas.

8.1.3 CSS via Arquivos Externos

A forma mais comum e recomendada de associar estilos CSS a documentos HTML é por meio de arquivos externos. Nessa abordagem, os estilos são definidos em um arquivo CSS separado e associados ao documento HTML por meio de uma tag `<link>` na seção `<head>` do documento HTML. O código 8.3 mostra um exemplo de um arquivo CSS externo que define estilos para todos os elementos `<p>` de um documento HTML.

```
1  p {
2      color: red;
3      font-size: 24px;
4      text-transform: uppercase;
5  }
```

Exemplo de Código 8.3: Arquivo CSS externo.

O código 8.3 mostra um exemplo de como criar um arquivo CSS externo para definir estilos CSS para todos os elementos `<p>` de um documento HTML, formatando a fonte do parágrafo com a cor vermelha, com 24 pixels de tamanho e com todas as letras em maiúsculas. Considere que esse

arquivo CSS externo chama-se “estilos.css” e esteja na mesma pasta do documento HTML. Neste caso, o código 8.4 mostra como associar o arquivo CSS externo ao documento HTML por meio da tag `<link>` na seção `<head>`.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Exemplo de CSS via Arquivos Externos</title>
5          <link rel="stylesheet" type="text/css" href="estilos.css">
6      </head>
7      <body>
8          <h1>Exemplo de CSS via Arquivos Externos</h1>
9          <p>Este é um exemplo de texto com estilo definido por um
10         arquivo CSS externo.</p>
11         <p>Este é outro exemplo de texto com estilo definido por um
12         arquivo CSS externo.</p>
13         <p><a href="temp.html">Este é um exemplo de link com estilo definido por um
14         arquivo CSS externo.</a></p>
15     </body>
16 </html>
```

Exemplo de Código 8.4: Exemplo de CSS via Arquivos Externos

No exemplo 8.4, os estilos são definidos em um arquivo CSS externo chamado “estilos.css”, que é associado ao documento HTML por meio da tag `<link>`. Veremos mais adiante que até o elemento `<a>` será afetado pelas configurações CSS para o elemento `<p>`. Por agora, perceba que a principal vantagem de utilizar arquivos CSS externos é a capacidade de reutilizar os mesmos estilos em vários documentos HTML. Além disso, essa abordagem permite separar completamente a definição dos estilos da estrutura e conteúdo do documento HTML, o que torna o código mais organizado, fácil de manter e otimizado para SEO (*Search Engine Optimization*).

8.2 Seletores Simples

Os seletores CSS são usados para identificar quais elementos HTML receberão os estilos CSS de um determinado bloco. Existem vários tipos de seletores CSS. Nesta seção, apresentaremos os seletores básicos, a saber, seletores por tipo de elemento, seletores por classe e seletores por ID.

8.2.1 Seletor por Tipo de Elemento

O seletor por tipo de elemento é usado para selecionar todos os elementos HTML de um determinado tipo. Por exemplo, para aplicar um estilo a todos os parágrafos de um documento, podemos usar o seletor de tipo de elemento `p`. O código 8.5 mostra um exemplo de seletor por tipo de elemento que aplica uma cor de fundo a todos os parágrafos de um documento.

```
1 | p {  
2 |     background-color: yellow;  
3 | }
```

Exemplo de Código 8.5: Seletor CSS por tipo de elemento.

Neste exemplo 8.5, o seletor `p` seleciona todos os elementos HTML do tipo `p`, e a propriedade `background-color` define a cor de fundo como amarelo.

8.2.2 Seletor por Classe

O seletor por classe é usado para selecionar elementos HTML com uma determinada classe. As classes são definidas no HTML usando o atributo `class`. Podemos usar o seletor por classe para aplicar estilos a vários elementos HTML com a mesma classe. O código 8.6 mostra um exemplo de seletor por classe que aplica uma cor de fundo a todos os elementos com a classe `destaque`.

```
1 | .destaque {  
2 |     background-color: yellow;  
3 | }
```

Exemplo de Código 8.6: Seletor CSS por classe.

Neste exemplo 8.6, o seletor `.destaque` seleciona todos os elementos HTML com a classe `destaque`, e a propriedade `background-color` define a cor de fundo como amarelo. Para usar a classe `destaque`, basta adicionar o atributo `class="destaque"` ao elemento HTML em questão. Por exemplo, o código 8.7 mostra como aplicar a classe `destaque` a um elemento `<p>`.

```
1 | ...  
2 | <p class="destaque">Este é um exemplo de texto com estilo definido por uma  
3 | classe CSS.</p>  
4 | ...
```

Exemplo de Código 8.7: Exemplo de seletor por classe sendo usado.

8.2.3 Seletor por ID

O seletor por ID é usado para selecionar um elemento HTML com um ID específico. Os IDs são definidos no HTML usando o atributo `id`. Cada ID deve ser exclusivo em um documento HTML. O código 8.8 mostra um exemplo de seletor CSS por ID, que aplica uma cor de fundo a um elemento cujo ID é igual a `cabecalho`.

```
1 | #cabecalho {  
2 |     background-color: yellow;  
3 | }
```

Exemplo de Código 8.8: Seletor CSS por ID.

Neste exemplo 8.8, o seletor `#cabecalho` seleciona o elemento HTML cujo ID seja igual a `cabecalho` e a propriedade `background-color` define a cor de fundo como amarelo. O exemplo 8.9 mostra um elemento `<h1>` cujo ID é igual a `cabecalho`, o que quer dizer que ele será afetado pelo seletor CSS em questão.

```
1 | ...  
2 | <h1 id="cabecalho">Cabecalho com estilo definido por uma classe CSS.</h1>  
3 | ...
```

Exemplo de Código 8.9: Exemplo de seletor por ID sendo usado.

8.3 Seletores Compostos

Os seletores compostos são uma combinação de dois ou mais seletores que são usados para selecionar elementos HTML de forma mais específica. Existem vários tipos de seletores compostos e, nesta subseção, apresentaremos os mais comuns.

8.3.1 Seletor por Descendente

O seletor por descendente é usado para selecionar elementos HTML que são filhos de outro elemento HTML. Por exemplo, para aplicar um estilo a todos os parágrafos dentro de um elemento qualquer da classe `container`, podemos usar o seletor por descendente `.container p`. O código 8.10 mostra um exemplo de seletor por descendente que aplica uma cor de fundo a todos os parágrafos dentro de um elemento da classe `container`.

Neste exemplo 8.10, o seletor `.container p` seleciona todos os parágrafos que são descendentes de um elemento da classe `container`, e a propriedade `background-color` define a cor de fundo como amarelo. Observe que o caractere de espaço que precede o `p` já define a relação de descendência de `p` em relação à classe `.container`.

```
1 | .container p {  
2 |     background-color: yellow;  
3 | }
```

Exemplo de Código 8.10: Seletor CSS por descendente.

8.3.2 Seletor de Classe e Tipo de Elemento

O seletor de classe e tipo de elemento é usado para selecionar elementos HTML que são de um determinado tipo e têm uma determinada classe. Por exemplo, para aplicar um estilo a todos os botões da classe `botao`, podemos usar o seletor de classe e tipo de elemento `button.botao`. O código 8.11 mostra um exemplo de seletor de classe e tipo de elemento que aplica uma cor de fundo a todos os botões com a classe `botao`.

```
1 | button.botao {  
2 |     background-color: yellow;  
3 | }
```

Exemplo de Código 8.11: Seletor CSS por classe e tipo de elemento.

Neste exemplo 8.11, o seletor `button.botao` seleciona todos os botões com a classe `botao`, e a propriedade `backgroundcolor` define a cor de fundo como amarelo. Vale mencionar que eu poderia usar um seletor desse tipo combinado com um seletor por descendente, conforme o código 8.12.

```
1 | button.botao span {  
2 |     font-size: 16px;  
3 | }
```

Exemplo de Código 8.12: Seletor CSS por classe e tipo de elemento com descendente.

O código 8.12 define um estilo para um elemento HTML `` que está contido dentro de um botão da classe `botao`, definindo o tamanho da fonte como 16 pixels.

8.4 Exercícios Propostos

Esta série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios, sem distrações por perto. Assim, você absorverá de forma muito mais eficiente o conteúdo estudado neste capítulo e abordado nos exercícios.

Exercício 8.1 Associe a um documento HTML qualquer o código CSS para que todos os parágrafos tenham fonte de tamanho 18px utilizando as três formas de associação de código CSS a documentos HTML: inline (atributo `style`), interno (elemento `<style>`) e externo (arquivo CSS). ■

Exercício 8.2 Crie um seletor CSS para o elemento HTML `<h1>` utilizando o seletor de tipo de elemento. ■

Exercício 8.3 Crie um seletor CSS para o elemento HTML com o ID `titulo` utilizando o seletor por ID. ■

Exercício 8.4 Crie um seletor CSS para todos os elementos HTML da classe `destaque` utilizando o seletor por classe. ■

Exercício 8.5 Crie um seletor CSS para os elementos HTML do tipo `<p>` dentro de um elemento da classe `container` utilizando o seletor de descendência. ■

Exercício 8.6 Crie um seletor CSS para todos os elementos HTML do tipo `<a>` da classe `link` utilizando o seletor de combinação `tipo.classe`. ■

Exercício 8.7 Crie um seletor CSS para todos os elementos HTML do tipo `` da classe `lista` utilizando o seletor de combinação `tipo.classe`. ■

Exercício 8.8 Crie um seletor CSS para todos os elementos HTML do tipo `` que estão dentro de um elemento com a classe `lista` utilizando o seletor de descendência. ■

Exercício 8.9 Defina uma cor de fundo verde para todos os elementos `<p>` do documento. ■

Exercício 8.10 Defina uma fonte em negrito para o elemento `<h2>` com o ID `titulo`. ■

Exercício 8.11 Defina uma cor de fundo amarela para todos os elementos com a classe `destaque`. ■

Exercício 8.12 Defina uma margem de 10 pixels para todos os elementos `<div>` descendentes de um elemento da classe `container`. ■

Exercício 8.13 Defina uma cor de texto azul para todos os elementos `<a>` da classe `link`. ■

Exercício 8.14 Defina uma cor de fundo cinza para todos os elementos `` da classe `lista`. ■

Exercício 8.15 Defina um tamanho de fonte de 14 pixels para todos os elementos `` que estão dentro de um elemento da classe `lista`. ■

Exercício 8.16 Selecione todos os elementos `<h1>` e defina um tamanho de fonte de 24 pixels para eles. ■

Exercício 8.17 Selecione todos os elementos com a classe `negrito` e defina um estilo de fonte em negrito para eles. ■

Exercício 8.18 Selecione todos os elementos `<p>` que estão dentro de um elemento com o ID `conteudo` e defina uma cor de texto vermelha para eles. ■

Exercício 8.19 Selecione todos os elementos `` que estão dentro de um elemento com a classe `galeria` e defina uma largura de 200 pixels para eles. ■

8.5 Considerações Sobre o Capítulo

Este capítulo introduziu o uso de CSS em documentos HTML, mostrando como associar código CSS a documentos HTML e como criar seletores CSS simples e compostos para definir estilos de elementos HTML específicos. No próximo capítulo, veremos como formatar cores, planos de fundo, bordas e margens de elementos HTML.



9. Cores, Planos de Fundo, Bordas e Margens

CSS (*Cascading Style Sheets*) é uma das linguagens mais utilizadas no desenvolvimento de sites e aplicações web, sendo fundamental para a aparência visual e estilização dos elementos da página. Um dos aspectos mais importantes do CSS é a habilidade de definir cores, fundos, bordas e margens para cada elemento, permitindo uma personalização completa da interface do usuário.

As cores em CSS são uma das maneiras mais eficientes de chamar a atenção do usuário e definir a identidade visual de um site ou aplicativo. As cores podem ser usadas em diversos elementos da página, desde o texto e fundo até as bordas e margens. Com a capacidade de definir cores em diferentes formatos, como RGB, HSL e hexadecimais, o CSS permite uma ampla gama de escolhas para personalização da interface.

Além das cores, o CSS também permite definir o plano de fundo dos elementos, que pode ser transparente, sólido ou gradiente (degradê). As bordas e margens são outras propriedades que ajudam a definir a aparência e o layout dos elementos. As bordas podem ser personalizadas em termos de espessura, estilo e cor, enquanto as margens definem a distância entre os elementos e as bordas da página. A habilidade de personalizar essas propriedades é fundamental para criar interfaces atraentes e funcionais em sites e aplicativos da web.

Este capítulo mostra como utilizar as propriedades de cores, planos de fundo, bordas e margens em CSS para estilizar elementos em uma página web. Veremos como utilizar diferentes formatos de cores, bem como gradientes e transparências, para personalizar a aparência dos elementos. Além disso, veremos as unidades de medida usadas em CSS e exploraremos as diferentes propriedades de configuração de planos de fundo, bordas e margens, e como ajustá-las para atender às necessidades de design. Com essas habilidades, você poderá criar interfaces atraentes e personalizadas em seus projetos de desenvolvimento web.

9.1 Cores

Cores são fundamentais em CSS para criar uma identidade visual atraente para um site ou aplicativo da web. Nesta seção, exploraremos como usar as cores em CSS e como formatá-las corretamente em seus códigos.

9.1.1 Introdução às Cores em CSS

As cores em CSS são definidas usando valores hexadecimais, RGB, HSL ou nomes de cores pré-definidos. Para especificar uma cor, você pode usar a propriedade `color` para textos e a propriedade `background-color` para o plano de fundo. É importante lembrar que as cores também podem ser usadas em outros lugares, como nas bordas, nos gradientes e em transparências.

9.1.2 Formatos de Cores em CSS

Existem diferentes formatos de cores em CSS, cada um com sua própria sintaxe e uso. Aqui estão os formatos mais comuns:

- a) **Nomeadas:** São cores pré-definidas como `red`, `green`, `blue` etc. Você pode usar essas cores diretamente em seus códigos.
- b) **RGB e RGBA:** São valores numéricos que representam a intensidade de vermelho, verde e azul. O formato RGB tem três valores, enquanto o formato RGBA tem quatro valores, sendo o último a opacidade (transparência), que varia entre 0 e 1.
- c) **HSL e HSLA:** São valores numéricos que representam a matiz, saturação e luminosidade. O formato HSL tem três valores, enquanto o formato HSLA tem quatro valores, sendo o último a opacidade (transparência), que varia entre 0 e 1.
- d) **Hexadecimal:** É o formato mais comum para especificar cores em CSS. O valor hexa é representado por uma combinação de seis ou três valores hexadecimais (0-9, A-F), onde os dois primeiros dígitos representam o vermelho, os dois do meio representam o verde e os dois últimos representam o azul.

No código 9.1, o primeiro seletor usa uma cor nomeada `red` para o texto de um cabeçalho (`h1`). O segundo usa a sintaxe RGB para definir a cor do texto de um parágrafo (`p`) como vermelho. O terceiro seletor usa a sintaxe RGBA para definir um fundo verde com 50% de transparência para uma `div`. O quarto usa a sintaxe HSL para definir um plano de fundo com uma matiz verde-azulada para um `span`. O quinto seletor usa a sintaxe HSLA para definir um plano de fundo com uma matiz azul-esverdeada para um link (`a`) com 30% de transparência. Por fim, o sexto seletor usa a sintaxe hexadecimal para definir a cor do texto de um item de lista (`li`) como azul.

```
1  /* Cores nomeadas */
2  h1 {
3      color: red;
4  }
5
6  /* RGB */
7  p {
8      color: rgb(255, 0, 0);
9  }
10
11 /* RGBA */
12 div {
13     background-color: rgba(0, 255, 0, 0.5);
14 }
15
16 /* HSL */
17 span {
18     background-color: hsl(120, 100%, 50%);
19 }
20
21 /* HSLA */
22 a {
23     background-color: hsla(240, 100%, 50%, 0.3);
24 }
25
26 /* Hexadecimal */
27 li {
28     color: #0000ff;
29 }
```

Exemplo de Código 9.1: Exemplo de formatos de cores em CSS.

9.1.3 Gradientes

Os gradientes são recursos importantes para adicionar profundidade e dimensão visual aos elementos da página. O gradiente pode ser linear ou radial, com uma transição suave entre duas ou mais cores. O código 9.2 mostra um exemplo de como usar gradiente em CSS.

Neste código 9.2, o primeiro exemplo usa um gradiente linear para o plano de fundo seletor CSS para a classe `header` com uma transição suave entre as cores `#f8f8f8` e `#ffffff`. O segundo exemplo usa um gradiente radial para o plano de fundo de elementos do tipo `button` com uma transição suave entre as cores `#ffffff` e `#008000`.

```
1  /* Gradiente linear */
2  .header {
3      background-image: linear-gradient(to bottom, #f8f8f8, #ffffff);
4  }
5
6  /* Gradiente radial */
7  button {
8      background-image: radial-gradient(circle, #ffffff, #008000);
9  }
```

Exemplo de Código 9.2: Exemplo de gradiente em CSS.

9.2 Unidades de Medida

As unidades de medida são uma parte essencial do CSS, pois elas definem as dimensões dos elementos na página. Existem diversas unidades de medida disponíveis no CSS, sendo que cada uma tem suas próprias características e é adequada para diferentes situações.

9.2.1 Pixels (px)

A unidade de medida px (pixels) é a mais comum e representa a menor unidade de medida em um monitor. O tamanho de um pixel é definido pelo próprio monitor e pode variar de acordo com a resolução da tela. O código 9.3 mostra um exemplo de como definir o tamanho de um elemento em pixels.

```
1  .exemplo {
2      width: 200px;
3      height: 100px;
4  }
```

Exemplo de Código 9.3: Exemplo de uso da unidade pixels.

Neste exemplo 9.3, estamos definindo o tamanho de um elemento usando a unidade de medida px. O elemento terá uma largura de 200 pixels e uma altura de 100 pixels.

9.2.2 Pontos (pt)

A unidade de medida pt (pontos) é geralmente usada para definir o tamanho de fontes em páginas da web. Um ponto é equivalente a 1/72 polegadas. O código 9.4 mostra um exemplo de como definir o tamanho de uma fonte em pontos.

Neste exemplo 9.4, estamos definindo o tamanho de uma fonte usando a unidade de medida pt. A fonte terá um tamanho de 12 pontos.

```
1 | .exemplo {  
2 |     font-size: 12pt;  
3 | }
```

Exemplo de Código 9.4: Exemplo de uso da unidade pontos.

9.2.3 Centímetros (cm) e Milímetros (mm)

As unidades de medida `cm` (centímetros) e `mm` (milímetros) são usadas para definir tamanhos absolutos em relação ao sistema métrico. Um centímetro é equivalente a 10 milímetros. O código 9.5 mostra um exemplo de como definir o tamanho de um elemento em centímetros e em milímetros.

```
1 | .exemplo {  
2 |     width: 5cm;  
3 |     height: 20mm;  
4 | }
```

Exemplo de Código 9.5: Exemplo de uso das unidades centímetros e milímetros.

Neste exemplo 9.5, estamos definindo o tamanho de um elemento usando as unidades de medida `cm` e `mm`. O elemento terá uma largura de 5 centímetros e uma altura de 20 milímetros.

9.2.4 EM e REM

As unidades de medida `em` e `rem` são usadas para definir tamanhos relativos. A unidade `em` é baseada no tamanho da fonte do elemento pai, enquanto a unidade `rem` é baseada no tamanho da fonte do elemento raiz (normalmente o elemento `html`). O código 9.6 mostra um exemplo de como definir o tamanho de uma fonte em `em` e `rem`.

```
1 | .exemplo {  
2 |     font-size: 1.2em; /* 1.2 vezes o tamanho da fonte do elemento pai */  
3 | }  
4 |  
5 | html {  
6 |     font-size: 16px; /* tamanho da fonte do elemento raiz */  
7 | }  
8 |  
9 | body {  
10 |     font-size: 1rem; /* tamanho da fonte do elemento raiz */  
11 | }
```

Exemplo de Código 9.6: Exemplo de uso das unidades `em` e `rem`.

Neste exemplo 9.6, estamos definindo o tamanho de uma fonte usando as unidades de medida `em` e `rem`. A fonte do elemento da classe `exemplo` terá um tamanho 1.2 vezes maior do que a fonte do elemento pai, enquanto a fonte do elemento `body` terá o tamanho padrão definido pela unidade `rem`.

9.2.5 Porcentagem (%)

A unidade de medida `%` (porcentagem) é usada para definir o tamanho de um elemento em relação ao tamanho do elemento pai. Por exemplo, se definirmos a largura de um elemento como `50%`, ele terá metade da largura do elemento pai. O código 9.7 mostra um exemplo de como definir o tamanho de um elemento em porcentagem.

```
1  .porcentagem {  
2      width: 50%;  
3      height: 50%;  
4  }
```

Exemplo de Código 9.7: Exemplo de uso da unidade porcentagem.

Neste exemplo 9.7, estamos definindo a largura e altura de um elemento como `50%` em relação ao tamanho do elemento pai. Isso significa que o elemento será exibido com metade da largura e metade da altura do elemento pai.

A unidade de medida porcentagem é muito útil para criar leiautes responsivos, em que o tamanho dos elementos se adapta automaticamente à tela do usuário. Por exemplo, podemos definir a largura de uma imagem como `100%` para que ela ocupe todo o espaço disponível na tela, independentemente do tamanho da tela do usuário.

9.2.6 Outras unidades de medida

Além das unidades de medida mencionadas acima, existem outras unidades de medida disponíveis no CSS, como `vw` (*viewport width*), `vh` (*viewport height*), `vmin` (*viewport minimum*) e `vmax` (*viewport maximum*). Essas unidades de medida são baseadas no tamanho da janela do navegador (*viewport*) e são úteis para criar leiautes responsivos. O código 9.8 mostra um exemplo de como definir o tamanho de um elemento usando as unidades de medida `vw` e `vh`.

```
1  .exemplo {  
2      width: 50vw; /* 50% da largura da janela do navegador */  
3      height: 30vh; /* 30% da altura da janela do navegador */  
4  }
```

Exemplo de Código 9.8: Exemplo de uso das unidades `vw` e `vh`.

Neste exemplo 9.8, estamos definindo o tamanho de um elemento usando as unidades de medida `vw` e `vh`. O elemento terá uma largura de 50% da largura da janela do navegador e uma altura de 30% da altura da janela do navegador.

Em resumo, as unidades de medida são uma parte importante do CSS e são usadas para definir as dimensões dos elementos na página. É importante entender as características de cada unidade de medida para escolher a mais adequada para cada situação.

9.3 Planos de Fundo

O plano de fundo é uma parte importante do design de uma página web. Com CSS, podemos adicionar cor, imagens e texturas de fundo à nossa página, tornando-a mais atraente. Nesta seção, vamos explorar as propriedades CSS que controlam o plano de fundo de um elemento.

9.3.1 Cor de Fundo

A propriedade `background-color` define a cor de fundo de um elemento. Podemos definir a cor usando um dos formatos de cores apresentados anteriormente neste capítulo. O código 9.9 mostra um exemplo de como definir a cor de fundo de um elemento.

```
1 | .exemplo {  
2 |     background-color: #ffffff;  
3 | }
```

Exemplo de Código 9.9: Definindo a cor de fundo com `background-color`.

Neste exemplo 9.9, a classe `.exemplo` tem a cor de fundo definida como branco. O valor hexadecimal `#ffffff` representa a cor branca em RGB.

9.3.2 Imagem de Fundo

A propriedade `background-image` define uma imagem de fundo para um elemento. Podemos definir a imagem usando uma URL ou um caminho relativo. É importante escolher uma imagem que tenha um tamanho adequado para o elemento. O código 9.10 mostra um exemplo de como definir uma imagem de fundo para um elemento.

```
1 | .exemplo {  
2 |     background-image: url("imagem.jpg");  
3 | }
```

Exemplo de Código 9.10: Definindo uma imagem de fundo com `background-image`.

Neste exemplo 9.10, a classe `.exemplo` tem a imagem de fundo definida como `imagem.jpg`. É importante lembrar que a imagem deve estar no mesmo diretório do arquivo HTML ou ter um caminho relativo correto.

9.3.3 Padrão de Repetição da Imagem de Fundo

A propriedade `background-repeat` define como a imagem de fundo deve ser repetida dentro do elemento. Existem quatro valores possíveis para essa propriedade: `repeat` (repete horizontalmente e verticalmente), `repeat-x` (repete apenas horizontalmente), `repeat-y` (repete apenas verticalmente) e `no-repeat` (não repete). O código 9.11 mostra um exemplo de como definir a repetição da imagem de fundo para um elemento.

```
1  .exemplo {  
2      background-image: url("imagem.jpg");  
3      background-repeat: repeat-x;  
4  }
```

Exemplo de Código 9.11: Definindo a repetição da imagem de fundo com `background-repeat`.

Neste exemplo 9.11, a classe `.exemplo` tem a imagem de fundo definida como `imagem.jpg` e a repetição definida como `repeat-x`, ou seja, a imagem será repetida apenas horizontalmente.

9.3.4 Posição da Imagem de Fundo

A propriedade `background-position` define a posição inicial da imagem de fundo dentro do elemento. Podemos definir a posição usando palavras-chave (`top`, `bottom`, `left`, `right`, `center`) ou valores em porcentagem ou pixels. O código 9.12 mostra um exemplo de como definir a posição da imagem de fundo para um elemento.

```
1  .exemplo {  
2      background-image: url("imagem.jpg");  
3      background-position: center;  
4  }
```

Exemplo de Código 9.12: Definindo a posição da imagem de fundo com `background-position`.

Neste exemplo 9.12, a classe `.exemplo` tem a imagem de fundo definida como `imagem.jpg` e a posição definida como `center`, ou seja, a imagem será centralizada dentro do elemento.

9.3.5 Fixação da Imagem de Fundo

A propriedade `background-attachment` define se a imagem de fundo deve se mover com o restante do conteúdo da página ou ficar fixa no lugar enquanto o conteúdo se move. Existem dois valores possíveis para essa propriedade: `scroll` (a imagem se move com o restante do conteúdo) e `fixed` (a imagem fica fixa no lugar enquanto o conteúdo se move). O código 9.13 mostra um exemplo de como definir a fixação da imagem de fundo para um elemento.

```
1  .exemplo {  
2      background-image: url("imagem.jpg");  
3      background-attachment: fixed;  
4  }
```

Exemplo de Código 9.13: Definindo a fixação da imagem de fundo com `background-attachment`.

Neste exemplo 9.13, a classe `.exemplo` tem a imagem de fundo definida como `imagem.jpg` e a fixação definida como `fixed`, ou seja, a imagem ficará fixa no lugar enquanto a página rola.

9.3.6 Tamanho da Imagem de Fundo

A propriedade `background-size` define o tamanho da imagem de fundo em relação ao elemento. Podemos definir o tamanho usando valores em porcentagem, pixels, palavras-chave (`cover` e `contain`) ou uma combinação de valores para largura e altura. O código 9.14 mostra um exemplo de como definir o tamanho da imagem de fundo para um elemento.

```
1  .exemplo {  
2      background-image: url("imagem.jpg");  
3      background-size: cover;  
4  }
```

Exemplo de Código 9.14: Definindo o tamanho da imagem de fundo com `background-size`

Neste exemplo 9.14, a classe `.exemplo` tem a imagem de fundo definida como `imagem.jpg` e o tamanho definido como `cover`, ou seja, a imagem será redimensionada para cobrir todo o elemento, mantendo a proporção.

9.4 Bordas

As bordas são importantes para definir o layout e o estilo de um elemento HTML. Em CSS, existem diversas propriedades que permitem personalizar a aparência das bordas, como `border-color`, `border-width`, `border-style`, `border-radius`, e as propriedades de bordas individuais, como `border-left` e `border-right`.

9.4.1 Cor da Borda

A propriedade `border-color` define a cor da borda de um elemento. Ela pode ser definida de várias formas, como por um nome de cor, um valor hexadecimal ou RGB. O código 9.15 mostra um exemplo de como definir a cor da borda para um elemento.

```
1 | p {  
2 |     border-color: red;  
3 | }
```

Exemplo de Código 9.15: Exemplo de `border-color`.

Neste exemplo 9.15, a borda de um parágrafo é definida com a cor vermelha. É possível também definir a cor das quatro bordas individualmente, usando as propriedades `border-top-color`, `border-right-color`, `border-bottom-color` e `border-left-color`.

9.4.2 Largura da Borda

A propriedade `border-width` define a largura da borda de um elemento. Ela pode ser definida em valores absolutos, como em pixels, ou em valores relativos, como em porcentagem. O código 9.16 mostra um exemplo de como definir a largura da borda para um elemento.

```
1 | p {  
2 |     border-width: 2px;  
3 | }
```

Exemplo de Código 9.16: Exemplo de uso da propriedade `border-width`.

Neste exemplo 9.16, a borda de um parágrafo é definida com a largura de 2 pixels. É possível também definir a largura das quatro bordas individualmente, usando as propriedades `border-top-width`, `border-right-width`, `border-bottom-width` e `border-left-width`.

9.4.3 Estilo da Borda

A propriedade `border-style` define o estilo da borda de um elemento. Existem diversos estilos de bordas, como `solid`, `dashed`, `dotted`, `double`, `groove`, `ridge`, `inset` e `outset`. O código 9.17 mostra um exemplo de como definir o estilo da borda para um elemento.

```
1 | p {  
2 |     border-style: dotted;  
3 | }
```

Exemplo de Código 9.17: Exemplo de uso da propriedade `border-style`.

Neste exemplo 9.17, a borda de um parágrafo é definida com o estilo pontilhado. É possível também definir o estilo das quatro bordas individualmente, usando as propriedades `border-top-style`, `border-right-style`, `border-bottom-style` e `border-left-style`.

9.4.4 Arredondamento dos Cantos

A propriedade `border-radius` define o raio dos cantos de uma borda. Ela pode ser definida em valores absolutos, como em pixels, ou em valores relativos, como em porcentagem. O código 9.18 mostra um exemplo de como definir o raio dos cantos da borda para um elemento.

```
1 | p {  
2 |     border-radius: 10px;  
3 | }
```

Exemplo de Código 9.18: Exemplo de uso da propriedade `border-radius`.

Neste exemplo 9.18, os cantos da borda de um parágrafo são definidos com um raio de 10 pixels. É possível também definir o raio dos cantos individualmente, usando as propriedades `border-top-left-radius`, `border-top-right-radius`, `border-bottom-left-radius` e `border-bottom-right-radius`.

9.4.5 Propriedade Abreviada

Além das propriedades de bordas que aceitam valores individuais, é possível usar a propriedade `border` para definir todas as propriedades de borda de um elemento de uma só vez. O código 9.19 mostra um exemplo de como definir a borda para um elemento.

```
1 | p {  
2 |     border: 2px dashed red;  
3 | }
```

Exemplo de Código 9.19: Exemplo de uso da propriedade `border`.

Neste exemplo 9.19, a borda de um parágrafo é definida com a largura de 2 pixels, o estilo tracejado e a cor vermelha, tudo em uma única propriedade `border`. Também é possível usar a propriedade `border-left` para definir todas as propriedades de borda da borda esquerda de um elemento de uma só vez. O código 9.20 mostra um exemplo de como definir a borda esquerda para um elemento.

```
1 | p {  
2 |     border-left: 2px dashed red;  
3 | }
```

Exemplo de Código 9.20: Exemplo de uso da propriedade `border-left`.

Neste exemplo 9.20, a borda esquerda de um parágrafo é definida com a largura de 2 pixels, o estilo tracejado e a cor vermelha, tudo em uma única propriedade `border-left`. O uso dessas propriedades pode tornar o código CSS mais simples e legível, especialmente quando se deseja aplicar as mesmas propriedades para todas as bordas ou apenas para uma borda específica de um elemento.

9.4.6 Bordas do Tipo *Outline*

Além das bordas normais, que são definidas pelas propriedades `border`, é possível criar bordas do tipo *outline* em CSS. As bordas do tipo *outline* são semelhantes às bordas normais, mas não afetam o tamanho ou o layout do elemento HTML e podem ter um afastamento. As bordas do tipo *outline* são definidas pela propriedade `outline`, que pode receber valores como largura, estilo e cor. O código 9.21 mostra um exemplo de como definir a borda do tipo *outline* para um elemento.

```
1  p {  
2      outline: 2px dashed blue;  
3      outline-offset: 10px; /* afastamento da borda */  
4  }
```

Exemplo de Código 9.21: Exemplo de uso da propriedade `outline`.

Neste exemplo 9.21, uma borda do tipo *outline* é criada para um parágrafo com a largura de 2 pixels, o estilo tracejado e a cor azul.

A principal diferença entre as bordas normais e as bordas do tipo *outline* é que as bordas do tipo *outline* não afetam o tamanho ou o layout do elemento HTML, ou seja, a área do elemento não é afetada pela largura da borda do tipo *outline*. Além disso, as bordas do tipo *outline* são desenhadas acima das bordas normais e podem ter um afastamento, o que significa que elas podem ser usadas em conjunto para criar estilos mais complexos.

9.5 Margens Interna e Externa

As propriedades de configuração de margens são `margin` (externa) e `padding` (interna). Elas são essenciais para definir a distância entre os elementos HTML em uma página web. É importante entender a diferença entre elas e os possíveis valores que podem ser usados.

A propriedade `margin` define a margem em torno do elemento. Os valores podem ser definidos para as margens superior, inferior, esquerda e direita usando as propriedades `margin-top`, `margin-bottom`, `margin-left` e `margin-right`, respectivamente.

Os valores da margem podem ser especificados em `px` (pixels), `em` (relativos à fonte), `rem` (relativos ao tamanho da fonte do elemento pai), `%` (porcentagem) ou `auto` (para centralizar o elemento na tela). Por exemplo, o código 9.22 mostra como definir uma margem de 10 pixels em torno de um elemento.

```
1 | div {  
2 |     margin: 10px;  
3 | }
```

Exemplo de Código 9.22: Exemplo de uso da propriedade `margin`

Neste exemplo 9.22, a propriedade `margin` é definida com um valor de 10 pixels, o que significa que a margem em torno do elemento `div` será de 10 pixels em todas as direções.

A propriedade `padding`, por sua vez, define a margem interna do elemento. Os valores podem ser definidos para a margem superior, inferior, esquerda e direita usando as propriedades `padding-top`, `padding-bottom`, `padding-left` e `padding-right`, respectivamente. Os valores podem ser especificados da mesma forma que a propriedade `margin`. Por exemplo, o código 9.23 mostra como definir uma margem interna de 10 pixels dentro de um elemento.

```
1 | div {  
2 |     padding: 10px;  
3 | }
```

Exemplo de Código 9.23: Exemplo de margem interna com propriedade `padding`.

Neste exemplo 9.23, a propriedade `padding` é definida com um valor de 10 pixels, o que significa que a margem interna do elemento `div` será de 10 pixels em todas as direções. É importante notar que as margens interna e externa podem afetar o leiaute da página, especialmente quando usados em conjunto com outros elementos HTML.

Em resumo, as propriedades `margin` e `padding` são importantes para definir a distância entre os elementos HTML em uma página web. Os valores podem ser definidos usando uma das unidades de medida ou o valor `auto`, dependendo da necessidade. Ao usar essas propriedades, é importante considerar como elas afetam o leiaute da página e usá-las com moderação.

9.5.1 Valores Automáticos para Margens

O valor `auto` da propriedade `margin` é usado para centralizar um elemento na tela. Quando a margem é definida como `auto`, o navegador define automaticamente um valor para a margem, a fim de centralizar o elemento na tela. Por exemplo, o código 9.24 mostra como centralizar um elemento `div` na tela usando a propriedade `margin` com o valor `auto`.

```
1  div {  
2      margin: auto;  
3      width: 50%;  
4      border: 1px solid black;  
5      padding: 10px;  
6  }
```

Exemplo de Código 9.24: Exemplo de `margin` com valor `auto`.

Neste exemplo 9.24, a propriedade `margin` é definida como `auto`, o que significa que o navegador definirá automaticamente o valor da margem para centralizar **horizontalmente** o elemento `div` na tela. Além disso, a largura do elemento é definida como 50%, a borda é definida como 1 pixel, estilo sólido e cor preta. A margem interna é definida como 10 pixels.

É importante notar que o valor `auto` da propriedade `margin` só funciona em elementos com uma largura definida, como um elemento com a propriedade `width` definida. Caso contrário, o navegador não terá como calcular a margem necessária para centralizar o elemento na tela.

Em resumo, o valor `auto` da propriedade `margin` é usado para centralizar um elemento na tela. É importante definir uma largura para o elemento para que o navegador possa calcular a margem necessária para centralizá-lo corretamente.

9.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 9.1 Crie uma classe CSS que defina a cor vermelha (`red`) para o fundo de um elemento usando a propriedade `background-color`. ■

Exercício 9.2 Crie um seletor CSS que defina nominalmente a cor azul (`blue`) para o texto de todos os elementos `<a>` que estejam dentro de um parágrafo. ■

Exercício 9.3 Crie uma classe CSS que defina uma cor RGB para o fundo de um elemento. A cor deve ser um tom de verde com os valores RGB de 50, 205 e 50. ■

Exercício 9.4 Crie um seletor CSS que defina uma cor RGBA para o texto de um elemento ``. A cor deve ser um tom de cinza com os valores RGB de 128, 128, 128 e uma transparência de 50%. ■

Exercício 9.5 Crie uma classe CSS que defina uma cor HSL para o fundo de um elemento, sendo que a cor deve ser um tom de rosa com uma matiz (hue) de 330 graus, uma saturação (saturation) de 75% e uma luminosidade (lightness) de 70%. ■

Exercício 9.6 Crie uma classe CSS que defina uma cor hexadecimal (`#FFA500`) para o fundo de um elemento usando a propriedade `background-color`. ■

Exercício 9.7 Crie uma classe CSS que defina uma cor nomeada (`green`) para o texto de um elemento usando a propriedade `color`. ■

Exercício 9.8 Crie um seletor CSS que defina uma cor HSLA para o fundo de um elemento `<dl>`, sendo que a cor deve ser um tom de azul com uma matiz (hue) de 200 graus, uma saturação (saturation) de 50%, uma luminosidade (lightness) de 50% e uma transparência de 75%. ■

Exercício 9.9 Crie uma classe CSS que defina uma cor RGBA para o texto de um elemento usando a propriedade `color`. A cor deve ser um tom de roxo com os valores RGB de 128, 0, 128 e uma transparência de 25%. ■

Exercício 9.10 Crie uma classe CSS que defina uma imagem de fundo para um elemento. A imagem deve ter a URL "imagem.jpg" e repetir-se horizontalmente. ■

Exercício 9.11 Crie uma classe CSS que defina uma cor de fundo para um elemento. A cor deve ser um tom de verde claro (`#90ee90`). ■

Exercício 9.12 Crie uma classe CSS que defina um gradiente linear como fundo de um elemento. O gradiente deve começar com a cor vermelha (`red`) e terminar com a cor branca (`white`). ■

Exercício 9.13 Crie uma classe CSS que defina um gradiente radial como fundo de um elemento. O gradiente deve começar com a cor amarela (`yellow`) no centro e terminar com a cor laranja (`orange`) nas bordas. ■

Exercício 9.14 Crie uma classe CSS que defina uma imagem de fundo para um elemento. A imagem deve ter a URL “imagem.jpg” e ser exibida somente uma vez (no-repeat). ■

Exercício 9.15 Crie uma classe CSS que defina uma cor de fundo para um elemento `<blockquote>`, sendo que a cor deve ser um tom de cinza escuro. ■

Exercício 9.16 Crie uma classe CSS que defina um gradiente linear como fundo de um elemento. O gradiente deve começar com a cor azul (blue) e terminar com a cor verde (green), em um ângulo de 45 graus. ■

Exercício 9.17 Crie uma classe CSS que defina uma borda sólida de 2 pixels de espessura na cor preta para um elemento. ■

Exercício 9.18 Crie uma classe CSS que defina uma borda pontilhada de 1 pixel de espessura na cor vermelha para um elemento. ■

Exercício 9.19 Crie uma classe CSS que defina uma borda com estilo de dupla linha de 4 pixels de espessura na cor verde para um elemento. ■

Exercício 9.20 Crie uma classe CSS que defina bordas laterais com estilo de traço de 3 pixels de espessura na cor azul e com bordas superior e inferior com estilo pontilhado de 2 pixels de espessura na cor amarela. ■

Exercício 9.21 Crie uma classe CSS que defina uma borda de contorno (outline) de 2 pixels de espessura (outline-width) e de estilo sólido (outline-style), na cor vermelha (outline-color) para um elemento usando a propriedade outline. ■

Exercício 9.22 Crie uma classe CSS que defina uma borda de contorno (outline) de 4 pixels de espessura (outline-width) e de estilo pontilhado (outline-style), na cor verde (outline-color) para um elemento usando a propriedade outline. ■

Exercício 9.23 Crie uma classe CSS que defina uma borda de contorno (outline) de 3 pixels de espessura (outline-width) e de estilo tracejado (outline-style), na cor azul (outline-color) para um elemento usando a propriedade outline. ■

Exercício 9.24 Crie uma classe CSS que defina uma margem externa de 10 pixels para todos os lados de um elemento. ■

Exercício 9.25 Crie uma classe CSS que defina uma margem interna de 20 pixels para todos os lados de um elemento. ■

Exercício 9.26 Crie uma classe CSS que defina uma margem externa de 5 pixels para o topo e para a esquerda de um elemento. ■

Exercício 9.27 Crie uma classe CSS que defina uma margem interna de 30 pixels para o topo e para a direita de um elemento. ■

Exercício 9.28 Crie uma classe CSS que defina uma margem externa de 0 pixels para o topo e para a direita e 10 pixels para a base e para a esquerda de um elemento. ■

Exercício 9.29 Crie uma classe CSS que defina uma margem interna de 5 pixels para o topo, 10 pixels para a direita, 15 pixels para a base e 20 pixels para a esquerda de um elemento. ■

9.7 Considerações Sobre o Capítulo

Nesta seção, aprendemos sobre as cores, unidades de medida, planos de fundo, bordas, outlines, margens e paddings em CSS. Através de exemplos práticos, vimos como é possível personalizar o estilo de elementos HTML usando essas propriedades. Saber usar corretamente esses recursos é essencial para criar páginas web visualmente atraentes e funcionais. No próximo capítulo, veremos como formatar fontes, textos, links e listas.



10. Fontes, Textos, Links e Listas

A formatação de fontes, textos, links e listas é essencial para a apresentação de conteúdo em uma página web. Com o uso de CSS, podemos definir a aparência visual desses elementos de forma precisa e personalizada, tornando a página mais atraente e legível para o usuário.

Ao entender as propriedades relacionadas à formatação de fontes, podemos escolher a fonte e o tamanho ideal para o texto, garantindo que ele seja fácil de ler. Além disso, o uso correto de propriedades como `letter-spacing` e `line-height` pode ajudar a tornar o texto mais confortável para o leitor.

A formatação de links e listas também é importante, pois permite que o usuário identifique facilmente esses elementos na página. Com a personalização desses elementos, é possível criar uma hierarquia visual e destacar informações importantes. Em suma, o conhecimento sobre a formatação de fontes, textos, links e listas em CSS é fundamental para o desenvolvimento de um site atraente e funcional. As seções a seguir abordam esses tópicos detalhadamente, mostrando exemplos de uso de cada um deles.

10.1 Fontes

CSS permite que você personalize as fontes de texto de várias maneiras. As fontes são uma parte crítica de um site e podem afetar sua legibilidade, estética e até mesmo o desempenho da página. Nesta seção, vamos abordar as principais propriedades que controlam as fontes em CSS, incluindo o tamanho, a família, o estilo, a espessura (ou peso) e a variação. Veremos também como usar um seletor abreviado para compactar essa configuração.

10.1.1 Tamanho da Fonte

A propriedade `font-size` define o tamanho da fonte em pixels, em pontos ou em outras unidades. É importante lembrar que o tamanho da fonte pode afetar a legibilidade e a aparência da página. É importante usar tamanhos de fonte adequados para diferentes partes da página, como cabeçalhos e parágrafos. O código 10.1 mostra um exemplo de como definir o tamanho da fonte.

```
1 | p {  
2 |     font-size: 16px;  
3 | }
```

Exemplo de Código 10.1: Definindo o tamanho da fonte.

Neste exemplo 10.1, definimos o tamanho da fonte para 16 pixels para todos os elementos `<p>` da página.

10.1.2 Família de Fontes

A propriedade `font-family` define a família de fontes usada no texto. É possível definir várias fontes alternativas para garantir que o texto seja exibido corretamente em diferentes navegadores e sistemas operacionais. O código 10.2 mostra um exemplo de como definir a família de fontes.

```
1 | p {  
2 |     font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;  
3 | }
```

Exemplo de Código 10.2: Definindo a família de fontes.

Neste exemplo 10.2, definimos a fonte *Helvetica Neue* como a fonte preferida, mas se ela não estiver disponível, usaremos *Helvetica* ou *Arial*, ou qualquer fonte *sans-serif* disponível.

10.1.3 Estilo da Fonte

A propriedade `font-style` define o estilo da fonte como normal, itálico ou oblíqua. A fonte em itálico pode ser usada para enfatizar palavras-chaves ou frases. O código 10.3 mostra um exemplo de como definir o estilo da fonte para itálico.

```
1 | p {  
2 |     font-style: italic;  
3 | }
```

Exemplo de Código 10.3: Definindo o estilo da fonte para itálico.

Neste exemplo 10.3, definimos o estilo itálico para todos os elementos `<p>` da página.

10.1.4 Espessura (Peso) da Fonte

A propriedade `font-weight` define a espessura da fonte como `normal`, `negrito` ou `mais leve`. É uma maneira fácil de enfatizar palavras-chaves ou frases. O código 10.4 mostra um exemplo de como definir a espessura da fonte.

```
1 | a {  
2 |     font-weight: bold;  
3 | }
```

Exemplo de Código 10.4: Definindo a espessura da fonte.

Neste exemplo 10.4, definimos a fonte em negrito para todos os elementos `<a>` da página.

10.1.5 Variação da Fonte

A propriedade `font-variant` define a variação da fonte como `normal` ou `small-caps`. A variação `small-caps` é útil para enfatizar palavras ou frases, tornando-as em letras maiúsculas menores do que as maiúsculas normais. O código 10.5 mostra um exemplo de como definir a variação da fonte.

```
1 | p {  
2 |     font-variant: small-caps;  
3 | }
```

Exemplo de Código 10.5: Definindo a variação da fonte.

Neste exemplo 10.5, definimos a variação `small-caps` para todos os elementos `<p>` da página.

10.1.6 Propriedade Abreviada

A propriedade `font` permite definir de forma abreviada várias propriedades de fontes em uma única linha de código. Ela inclui a `font-size`, `font-family`, `font-style`, `font-weight` e `font-variant`. O código 10.6 mostra um exemplo de como usar a propriedade `font`.

```
1 | h1 {  
2 |     font: italic small-caps bold 24px/1.5  
3 |         "Helvetica Neue", Helvetica, Arial, sans-serif;  
4 | }
```

Exemplo de Código 10.6: Definindo propriedades de fonte de forma abreviada.

Neste exemplo 10.6, definimos várias propriedades de fontes para o elemento `<h1>`, incluindo `font-style` em itálico, `font-variant` em `small-caps`, `font-weight` em negrito, `font-size` em 24 pixels com uma altura de linha de 1,5 e `font-family` com a preferência para *Helvetica Neue*, seguida por *Helvetica*, *Arial* ou qualquer fonte *sans-serif* disponível.

10.2 Textos

Nesta seção, vamos explorar várias propriedades CSS que afetam o estilo de texto em um documento web. Veremos como essas propriedades afetam a aparência do texto e como elas podem ser usadas para criar efeitos visuais interessantes.

10.2.1 Alinhamento Horizontal

A propriedade CSS `text-align` é usada para definir o alinhamento horizontal do texto em um elemento. Ela pode ter os valores `left`, `right`, `center` e `justify` que, respectivamente, alinham o texto à esquerda, à direita, no centro ou distribui o texto uniformemente na largura disponível, criando espaçamento adicional entre as palavras, se necessário. O código 10.7 mostra um exemplo de como usar a propriedade `text-align` para alinhar o texto à esquerda.

```
1 | p {  
2 |     text-align: left;  
3 | }
```

Exemplo de Código 10.7: Exemplo de `text-align`.

Neste exemplo 10.7, a propriedade `text-align` é aplicada ao elemento `<p>`, fazendo com que todo o texto dentro do elemento seja alinhado à esquerda.

10.2.2 Direção da Escrita

As propriedades CSS `direction` e `unicode-bidi` são usadas para controlar a direção do texto em um elemento. A propriedade `direction` pode ter os valores `ltr` (esquerda para a direita) ou `rtl` (direita para a esquerda).

A propriedade `unicode-bidi` é usada para controlar a direção do texto em elementos que contêm texto com diferentes direções de fluxo, como texto em árabe ou hebraico. Ela pode ter os valores `normal` para texto esquerda-para-direita, ou `bidi-override` para texto direita-para-esquerda. O código 10.8 mostra um exemplo de como usar as propriedades `direction` e `unicode-bidi` para controlar a direção do texto em um elemento.

```
1 | h1 {  
2 |     direction: rtl;  
3 |     unicode-bidi: bidi-override;  
4 | }
```

Exemplo de Código 10.8: Exemplo de uso das propriedades `direction` e `unicode-bidi`.

Neste exemplo 10.8, as propriedades `direction` e `unicode-bidi` são aplicadas ao elemento `h1`, fazendo com que todo o texto dentro do elemento seja exibido da direita para a esquerda.

10.2.3 Decorações

A propriedade CSS `text-decoration` é usada para adicionar ou remover decorações de texto, como sublinhado, riscado ou sobrelinhado. Ela pode ter um os valores `none`, `underline`, `overline` e `line-through`. O código 10.9 mostra um exemplo de uso.

```
1 | a {  
2 |     text-decoration: line-through;  
3 | }
```

Exemplo de Código 10.9: Exemplo de uso da propriedade `text-decoration`.

Neste exemplo 10.9, a propriedade `text-decoration` é aplicada ao elemento `<a>`, fazendo com que todos os links no documento tenham uma linha riscando o texto.

10.2.4 Transformações

A propriedade CSS `text-transform` é usada para transformar textos. Ela pode ter os valores `none`, `uppercase`, `lowercase` e `capitalize` que, respectivamente, exclui qualquer transformação do texto, converte em letras maiúsculas, converte em letras minúsculas e converte a primeira letra de cada palavra em letra maiúscula. O código 10.10 mostra um exemplo de como usar a propriedade `text-transform` para transformar o texto em maiúsculas.

```
1 | h2 {  
2 |     text-transform: uppercase;  
3 | }
```

Exemplo de Código 10.10: Exemplo de uso da propriedade `text-transform`.

Neste exemplo 10.10, a propriedade `text-transform` é aplicada ao elemento `h2`, fazendo com que todo o texto dentro do elemento seja convertido para letras maiúsculas.

10.2.5 Indentação

A propriedade CSS `text-indent` é usada para definir a indentação da primeira linha de um bloco de texto. O código 10.11 mostra um exemplo de como usar a propriedade `text-indent` para criar uma indentação na primeira linha de um parágrafo.

```
1 | p {  
2 |     text-indent: 20px;  
3 | }
```

Exemplo de Código 10.11: Exemplo de uso da propriedade `text-indent`.

Neste exemplo 10.11, a propriedade `text-indent` é aplicada ao elemento `<p>`, fazendo com que a primeira linha do parágrafo seja indentada em 20 pixels.

10.2.6 Espaçamento Entre Letras

A propriedade CSS `letter-spacing` é usada para ajustar o espaçamento entre letras em um texto. Ela pode ter um valor positivo ou negativo para aumentar ou diminuir o espaçamento entre as letras. O código 10.12 mostra um exemplo de como usar a propriedade `letter-spacing` para aumentar o espaçamento entre letras do texto de um elemento.

```
1 | h3 {  
2 |     letter-spacing: 2px;  
3 | }
```

Exemplo de Código 10.12: Modificando o espaçamento entre letras.

Neste exemplo 10.12, a propriedade `letter-spacing` é aplicada ao elemento `h3`, aumentando o espaçamento entre as letras em 2 pixels.

10.2.7 Altura da Linha

A propriedade CSS `line-height` é usada para definir a altura da linha de um bloco de texto. O valor pode ser especificado em pixels, *em* ou porcentagem. O código 10.13 mostra um exemplo de como usar a propriedade `line-height` para aumentar a altura da linha em um elemento.

```
1 | p {  
2 |     line-height: 1.5em;  
3 | }
```

Exemplo de Código 10.13: Exemplo de uso da propriedade `line-height`.

Neste exemplo 10.13, a propriedade `line-height` é aplicada ao elemento `<p>`, aumentando a altura da linha para 1,5 vezes o tamanho da fonte.

10.2.8 Espaçamento Entre Palavras

A propriedade CSS `word-spacing` é usada para ajustar o espaçamento entre palavras em um texto. Ela pode ter um valor positivo ou negativo para aumentar ou diminuir o espaçamento entre as palavras. O código 10.14 mostra um exemplo de como usar a propriedade `word-spacing` para aumentar o espaçamento entre as palavras em um texto.

```
1 | h4 {  
2 |     word-spacing: 4px;  
3 | }
```

Exemplo de Código 10.14: Exemplo de uso da propriedade `word-spacing`.

Neste exemplo 10.14, a propriedade `word-spacing` é aplicada ao elemento `h4`, aumentando o espaçamento entre as palavras para 4 pixels.

10.2.9 Quebras no Espaço em Branco

A propriedade CSS `white-space` é usada para definir como o espaço em branco é tratado em um elemento. Ele pode assumir os valores `normal`, `nowrap`, `pre` e `pre-wrap`. O valor `normal` trata vários espaços em branco como um único espaço, e quebra linhas automaticamente quando o texto atinge a largura máxima do elemento. O valor `nowrap` impede que o texto quebre em várias linhas. O valor `pre` preserva os espaços em branco e quebras de linha no texto exatamente como eles são no código-fonte. O valor `pre-wrap` preserva os espaços em branco e quebras de linha, mas permite que o texto seja quebrado em várias linhas. O código 10.15 mostra um exemplo de como usar a propriedade `white-space` para preservar os espaços em branco e quebras de linha em um elemento.

```
1 | p {  
2 |     white-space: pre-wrap;  
3 | }
```

Exemplo de Código 10.15: Exemplo de uso da propriedade `white-space`.

Neste exemplo 10.15, a propriedade `white-space` é aplicada ao elemento `<p>`, preservando os espaços em branco e quebras de linha no texto.

10.2.10 Sombras

A propriedade CSS `text-shadow` é usada para adicionar sombras ao texto em um elemento. Ela pode ter um ou mais valores, separados por vírgulas, que definem a posição da sombra, a cor da sombra e o tamanho da sombra. O código 10.16 mostra um exemplo de como usar a propriedade `text-shadow` para adicionar uma sombra ao texto em um elemento.

```
1 | h5 {  
2 |     text-shadow: 2px 2px #000000;  
3 | }
```

Exemplo de Código 10.16: Exemplo de uso da propriedade `text-shadow`.

Neste exemplo 10.16, a propriedade `text-shadow` é aplicada ao elemento `h5`, adicionando uma sombra preta de 2 pixels de tamanho à direita e abaixo do texto.

10.3 Links

Os links são elementos importantes em uma página web, pois permitem que o usuário navegue pelo conteúdo. Porém, é possível ir além da formatação padrão dos links e criar efeitos visuais e de interatividade com CSS. Nesta seção, vamos explorar algumas técnicas para estilizar links usando CSS.

10.3.1 Links Visitados e Não Visitados

Os pseudo-seletores `a:link` e `a:visited` são usados para definir a aparência dos links que ainda não foram visitados e dos links que já foram visitados, respectivamente. É possível personalizar a cor do texto, a cor do fundo, a borda, entre outras propriedades. O código 10.17 mostra um exemplo de formatação de links visitados.

```
1 | a:visited {  
2 |     color: #999;  
3 |     text-decoration: line-through;  
4 | }
```

Exemplo de Código 10.17: Formatação de links visitados.

Neste exemplo 10.17, definimos que os links visitados terão a cor do texto cinza escuro (`#999`) e o texto riscado (`text-decoration: line-through`).

10.3.2 Links Ativos e Sob o Mouse

Os pseudo-seletores `a:hover` e `a:active` são usados para definir a aparência dos links quando o usuário passa o mouse sobre eles e quando eles estão sendo clicados, respectivamente. É possível personalizar a cor do texto, a cor do fundo, a borda, entre outras propriedades. O código 10.18 mostra um exemplo de formatação de links quando o usuário passa o mouse sobre eles e quando eles estão sendo clicados.

```
1  a:hover {  
2      color: #000;  
3      background-color: #ff0;  
4  }  
5  
6  a:active {  
7      color: #fff;  
8      background-color: #f00;  
9  }
```

Exemplo de Código 10.18: Formatação de links com hover e active.

Neste exemplo 10.18, definimos que quando o usuário passar o mouse sobre o link, a cor do texto será preta (`#000`) e o fundo será amarelo (`#ff0`). Quando o link estiver sendo clicado, a cor do texto será branca (`#fff`) e o fundo será vermelho (`#f00`).

Vale mencionar que os pseudo-seletores `a:hover` e `a:active` não funcionam em dispositivos móveis, pois não há eventos de mouse em dispositivos móveis. Ainda, os pseudo-seletores `:hover` e `:active` podem ser usados em outros tipos de elementos HTML, como botões e imagens.

10.3.3 Transformando um Link em Botão

Usando as técnicas abordadas até aqui, é possível transformar um link em um botão, adicionando bordas, fundo e espaçamento. O código 10.19 mostra um exemplo de transformação de um link em botão.

Neste exemplo 10.19, definimos uma classe `button` para o link e adicionamos propriedades para exibir o link como um botão, incluindo margem interna (`padding`), alinhamento de texto (`text-align`), cor de fundo (`background-color`), cor do texto (`color`), remoção de sublinhado (`text-decoration:none`), cantos arredondados (`border-radius`) e sombra (`box-shadow`).

```
1  a.button {
2      display: inline-block;
3      padding: 0.5em 1em;
4      text-align: center;
5      background-color: #f00;
6      color: #fff;
7      text-decoration: none;
8      border-radius: 0.5em;
9      box-shadow: 0 0.2em 0.4em rgba(0, 0, 0, 0.4);
10 }
11
12 a.button:hover {
13     background-color: rgb(201, 0, 0);
14 }
15
16 a.button:active {
17     position: relative;
18     top: 0.1em;
19     left: 0.1em;
20     box-shadow: none;
21 }
```

Exemplo de Código 10.19: Transformando um link em um botão.

Essas técnicas de formatação de links em CSS permitem que você crie uma experiência mais agradável e interativa para os usuários da sua página web. Ao usar os pseudo-seletores `a:link`, `a:visited`, `a:hover` e `a:active` e transformar um link em botão, você pode personalizar a aparência dos links de acordo com o design da sua página.

10.4 Listas

Listas são elementos comuns em documentos HTML. Com CSS, podemos modificar a aparência das listas, tornando-as mais atraentes e personalizadas. Esta seção aborda as técnicas básicas para a formatação de listas usando CSS.

10.4.1 Tipo de Lista

A primeira coisa que podemos modificar é o tipo de lista. O tipo padrão de lista é uma lista com marcadores (*bullets*), mas também podemos ter listas numeradas. Para mudar o tipo de lista, usamos a propriedade `list-style-type`. Os valores possíveis incluem:

- a) `disc`: o marcador padrão, que é um círculo sólido;
- b) `circle`: um círculo aberto;

- c) square: um quadrado sólido;
- d) decimal: uma lista numerada com números arábicos;
- e) lower-alpha: uma lista numerada com letras minúsculas;
- f) upper-alpha: uma lista numerada com letras maiúsculas;
- g) lower-roman: uma lista numerada com números romanos em minúsculas;
- h) upper-roman: uma lista numerada com números romanos em maiúsculas.

O código 10.20 mostra um exemplo de como mudar o tipo de um lista.

```
1 | ul {  
2 |     list-style-type: square;  
3 | }
```

Exemplo de Código 10.20: Mudando o tipo de lista.

Neste exemplo 10.20, a propriedade `list-style-type` é aplicada a um elemento ``, alterando o tipo de lista para quadrados sólidos.

10.4.2 Imagem do Marcador

Outra forma de personalizar uma lista é usando imagens como marcadores. Para fazer isso, usamos a propriedade `list-style-image`. O valor desta propriedade é a URL da imagem que será usada como marcador. É importante notar que a imagem é usada em substituição ao marcador padrão. Portanto, se a imagem não carregar corretamente, nenhum marcador será exibido. O código 10.21 mostra um exemplo de como usar uma imagem como marcador.

```
1 | ul {  
2 |     list-style-image: url('marcador.png');  
3 | }
```

Exemplo de Código 10.21: Usando uma imagem como marcador.

Neste exemplo 10.21, a propriedade `list-style-image` é aplicada a um elemento `ul`, usando a imagem `marcador.png` como marcador.

10.4.3 Posição do Marcador

A propriedade `list-style-position` define a posição do marcador em relação ao conteúdo do item da lista. Por padrão, o marcador é exibido à esquerda do conteúdo do item, mas com essa propriedade é possível movê-lo para a direita ou para dentro do conteúdo do item. Os valores possíveis para a

propriedade `list-style-position` são `inside`, onde o marcador é colocado dentro do conteúdo do item da lista, e `outside`, onde o marcador é colocado à esquerda do conteúdo do item da lista. O código 10.22 mostra um exemplo de como mudar a posição do marcador na lista.

```
1 | ul {  
2 |     list-style-position: inside;  
3 | }
```

Exemplo de Código 10.22: Mudando a posição do marcador na lista.

Neste exemplo 10.22, a propriedade `list-style-position` é aplicada a um elemento `ul`, movendo o marcador para dentro do conteúdo do item da lista. Essa propriedade é especialmente útil quando usamos imagens como marcadores, pois podemos posicioná-las de forma mais precisa em relação ao conteúdo do item da lista. Por exemplo, se a imagem do marcador tiver um tamanho maior do que o padrão, podemos usar a propriedade `list-style-position: inside` para garantir que o conteúdo do item não fique muito afastado do marcador.

10.4.4 Margem, Preenchimento e Indentação

Por fim, podemos ajustar a margem, o preenchimento e a indentação das listas para deixá-las mais atraentes. Para isso, usamos as propriedades `margin`, `padding` e `text-indent`. A propriedade `margin` define a margem da lista, enquanto a propriedade `padding` define o espaço interno da lista. A propriedade `text-indent` define o recuo da primeira linha do texto da lista. O código 10.23 mostra um exemplo de formatação de margens e indentação de uma lista.

```
1 | ul {  
2 |     margin: 0;  
3 |     padding: 0;  
4 |     text-indent: 10px;  
5 | }
```

Exemplo de Código 10.23: Ajustando margem, preenchimento e indentação da lista.

Neste exemplo 10.23, as propriedades `margin`, `padding` e `text-indent` são aplicadas a um elemento ``, definindo margem, preenchimento e recuo do texto da lista. O código 10.24 mostra um exemplo completo de como formatar uma lista usando as técnicas apresentadas.

Neste exemplo 10.24, todas as técnicas apresentadas são aplicadas a um elemento ``, criando uma lista personalizada com uma imagem como marcador, margens externa e interna zeradas, e um recuo de 10 pixels na primeira linha do texto.

```
1  ul {  
2      list-style-image: url('marcador.png');  
3      margin: 0;  
4      padding: 0;  
5      text-indent: 10px;  
6  }
```

Exemplo de Código 10.24: Exemplo completo de formatação de lista.

10.5 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 10.1 Crie um seletor CSS para modificar a cor do texto de todos os parágrafos de uma página HTML. Escolha a cor desejada em formato hexadecimal. ■

Exercício 10.2 Considere um texto de um site que utiliza a fonte “Arial”, mas você deseja aplicar a fonte “Roboto” como uma alternativa. Crie um seletor CSS que defina “Roboto” como a fonte principal e “Arial” como a segunda opção caso “Roboto” não esteja disponível. ■

Exercício 10.3 Imagine que você queira alterar o tamanho da fonte de todos os títulos de nível 3 de uma página HTML para 2rem. Escreva um seletor CSS para realizar essa tarefa. ■

Exercício 10.4 Suponha que em um site você queira enfatizar determinadas palavras ou frases aplicando um estilo de fonte em negrito. Crie um seletor CSS que aplique o estilo desejado a um elemento da classe destaque. ■

Exercício 10.5 Em um site, você deseja aplicar um estilo de fonte em itálico para citações textuais contidas em elementos <blockquote>. Crie um seletor CSS que atinja esse objetivo. ■

Exercício 10.6 Aplique a propriedade CSS adequada em todos os títulos de nível 2 para deixá-los em caixa alta. ■

Exercício 10.7 Utilize a propriedade CSS adequada para aumentar o espaçamento entre linhas dos parágrafos em 1.5 vezes. ■

Exercício 10.8 Aplique a propriedade CSS ideal para aumentar o espaçamento entre as palavras dos títulos de nível 3 em 0.2em. ■

Exercício 10.9 Utilize a propriedade CSS correta para evitar a quebra de linha nos elementos `` com a classe `nowrap`. ■

Exercício 10.10 Crie uma regra CSS que altere a cor dos links usando a propriedade `color` com um valor hexadecimal. ■

Exercício 10.11 Crie um seletor com a propriedade CSS para remover o sublinhado de todos os links. ■

Exercício 10.12 Utilize a propriedade `font-weight` para tornar os links em negrito quando o mouse estiver sobre eles. ■

Exercício 10.13 Aplique a propriedade `background-color` para adicionar uma cor de fundo aos links quando forem clicados. ■

Exercício 10.14 Crie uma regra CSS que modifique a cor dos links visitados usando a propriedade `color` e o pseudo-seletor `:visited`. ■

Exercício 10.15 Utilize a propriedade `border` para adicionar uma borda aos links quando o mouse estiver sobre eles. ■

Exercício 10.16 Aplique a propriedade `padding` para aumentar o espaçamento interno em todos os links em 5px. ■

Exercício 10.17 Crie uma regra CSS que utilize a propriedade `list-style-type` para criar listas ordenadas com números romanos. ■

Exercício 10.18 Utilize a propriedade `list-style-image` para substituir o marcador padrão de listas não ordenadas por uma imagem. ■

Exercício 10.19 Aplique a propriedade `list-style-position` para posicionar os marcadores de listas não ordenadas fora do conteúdo. ■

Exercício 10.20 Utilize a propriedade `padding-left` para aumentar o recuo à esquerda das listas ordenadas em 30px. ■

Exercício 10.21 Aplique a propriedade `margin-bottom` para adicionar uma margem inferior de 20px entre os itens das listas. ■

10.6 Considerações Sobre o Capítulo

Neste capítulo, estudamos as propriedades necessárias para se formatar fontes, textos, links e listas usando CSS, incluindo alinhamentos, alturas, tamanhos, espaçamentos e marcadores. Com os conhecimentos adquiridos neste capítulo, é possível criar textos muito mais atraentes e adequados para os mais diversos contextos de um site. No capítulo seguinte, veremos como usar pseudo-seletores CSS para criarmos seletores mais precisos e eficientes.



11. Pseudo-Seletores CSS

Pseudo-seletores são utilizados para aplicar estilos a elementos com base em um estado ou característica específica, sem precisar modificar o HTML. Nesta seção, vamos explorar alguns pseudo-seletores comuns do CSS e como utilizá-los.

11.1 Conteúdo Posterior e Anterior

Usando CSS, é possível adicionar conteúdo antes ou depois de um elemento HTML. O pseudo-seletor `:after` é utilizado para adicionar conteúdo após o conteúdo de um elemento. É comumente usado para adicionar elementos decorativos, como ícones e setas, a elementos de navegação. O código 11.1 mostra como adicionar uma seta após um link.

```
1 | a:after {  
2 |     content: " >>";  
3 | }
```

Exemplo de Código 11.1: Adicionando uma seta após um link.

Neste exemplo 11.1, o conteúdo de `:after` é definido como uma seta usando a propriedade `content`. A seta é adicionada após o conteúdo do link.

O pseudo-seletor `:before` é utilizado para adicionar conteúdo antes do conteúdo de um elemento. É comumente usado para adicionar elementos decorativos, como ícones e setas, a elementos de navegação. O código 11.2 mostra como adicionar uma seta antes de um link.

```
1 | a:before {  
2 |     content: ">> ";  
3 | }
```

Exemplo de Código 11.2: Adicionando uma seta antes de um link.

Neste exemplo 11.2, o conteúdo de `:before` é definido como uma seta usando a propriedade `content`. A seta é adicionada antes do conteúdo do link.

11.2 Primeira Letra e Primeira Linha

O pseudo-seletor `:first-letter` é utilizado para aplicar estilos apenas à primeira letra de um elemento. É comumente usado para adicionar destaque a letras iniciais em textos. O código 11.3 mostra como aplicar um estilo de cor diferente à primeira letra de um parágrafo.

```
1 | p:first-letter {  
2 |     color: red;  
3 | }
```

Exemplo de Código 11.3: Aplicando estilo à primeira letra de um parágrafo.

Neste exemplo 11.3, o pseudo-seletor `:first-letter` é usado para aplicar a cor vermelha apenas à primeira letra do parágrafo.

O pseudo-seletor `:first-line` é utilizado para aplicar estilos apenas à primeira linha de um elemento. É comumente usado para definir estilos para o início de um parágrafo ou título. O código 11.4 mostra como aplicar um estilo de cor diferente à primeira linha de um parágrafo.

```
1 | p:first-line {  
2 |     color: blue;  
3 | }
```

Exemplo de Código 11.4: Aplicando estilo à primeira linha de um parágrafo.

Neste exemplo 11.4, o pseudo-seletor `:first-line` é usado para aplicar a cor azul apenas à primeira linha do parágrafo.

11.3 Marcador de Itens de Lista

O pseudo-seletor `li::marker` é utilizado para selecionar o marcador de itens de lista ``. O marcador é a parte do item de lista que aparece antes do conteúdo do elemento. O código 11.5 mostra como estilizar o marcador de um item de lista com o *emoji* de um hambúrguer.

```
1 | li::marker {  
2 |     content: "\01F354";  
3 | }
```

Exemplo de Código 11.5: Estilizando o marcador de itens de lista.

Neste exemplo 11.5, o pseudo-seletor `li::marker` é usado para selecionar o marcador de itens de lista alterar seu conteúdo para o *emoji* de um hambúrguer. A propriedade `content` é usada para remover o conteúdo padrão do marcador e substituir pelo conteúdo à sua escolha.

11.4 Valores de Atributos

Os operadores `^=` e `$=` são utilizados para selecionar elementos cujos atributos começam com algo e que terminam com algo, respectivamente. O código 11.6 mostra como estilizar links que começam com `http` em negrito e links que terminam com `.pdf` em vermelho.

```
1  a[href^="http"] {  
2      font-weight: bold;  
3  }  
4  
5  a[href$=".pdf"] {  
6      color: red;  
7  }  
8  
9  a[href="#topo"] {  
10     color: blue;  
11 }
```

Exemplo de Código 11.6: Estilizando links específicos.

Neste exemplo 11.6, os pseudo-seletores `[href^="http"]` e `[href$=".pdf"]` são usados para selecionar links com endereço que começam com `http` e que terminam com `.pdf`, respectivamente, enquanto `[href="#topo"]` seleciona endereços iguais a `#topo`.

11.5 Primeiro de Um Tipo

Os pseudo-seletores `:first-of-type` e `:not(:first-of-type)` são utilizados para selecionar, respectivamente, o primeiro elemento de um determinado tipo e todos os elementos desse tipo com exceção do primeiro. O operador `:not` desempenha o papel de selecionar o contrário de seu conteúdo. O código 11.7 mostra como estilizar o primeiro parágrafo de um artigo em negrito e todos os outros parágrafos em itálico.

Neste exemplo 11.7, o pseudo-seletor `:first-of-type` é usado para selecionar o primeiro parágrafo do artigo e aplicar um negrito a ele. Em seguida, o pseudo-seletor `:not(:first-of-type)` é usado para selecionar todos os outros parágrafos do artigo e aplicar um estilo de itálico a eles.

```
1 article p:first-of-type {  
2     font-weight: bold;  
3 }  
4  
5 article p:not(:first-of-type) {  
6     font-style: italic;  
7 }
```

Exemplo de Código 11.7: Estilizando o primeiro parágrafo de um artigo.

11.6 Campos de Formulário

Esta seção apresenta pseudo-seletores que podem ser utilizados para selecionar campos de formulário baseando-se em seus atributos, tipos, valores e estados de validação.

11.6.1 Campos de Determinado Tipo

O pseudo-seletor `input[type="tipo"]` é utilizado para selecionar elementos `<input>` de um tipo específico. O código 11.8 mostra como estilizar um campo de entrada de texto e um campo de entrada de senha.

```
1 input[type="text"] {  
2     border-color: blue;  
3 }  
4 input[type="password"] {  
5     border: 1px solid black;  
6 }
```

Exemplo de Código 11.8: Estilizando campos de tipos específicos.

Neste exemplo 11.8, o pseudo-seletor `input[type="text"]` é usado para aplicar cor de borda azul a campos do tipo texto. Em seguida, o pseudo-seletor `input[type="password"]` é usado para aplicar uma borda preta a campos do tipo senha.

11.6.2 Campos Válidos e Inválidos

Os pseudo-seletores `input:valid` e `input:invalid` são utilizados para selecionar elementos `<input>` cujos valores sejam válidos e inválidos, respectivamente. O código 11.9 mostra como estilizar campos de entrada de texto válido e inválido.

Neste exemplo 11.9, o pseudo-seletor `input:valid` é usado para aplicar um estilo de borda verde a elementos válidos, enquanto o pseudo-seletor `input:invalid` é usado para selecionar campos de entrada de texto que sejam inválidos, aplicando um estilo de borda vermelha a eles.

```
1  input:valid {  
2      border-color: green;  
3  }  
4  input:invalid {  
5      border-color: red;  
6  }
```

Exemplo de Código 11.9: Estilizando campos em determinado estado de validação.

11.6.3 Campos Limitados e Marcados

Os pseudo-seletores `input:in-range`, `input:out-of-range` e `input:checked` também são utilizados para selecionar elementos `<input>` que estejam em determinados estados de validação, a saber, “dentro dos limites”, “fora dos limites” e “checado”, respectivamente. O código 11.10 mostra como estilizar campos numéricos, de data e de checagem com esses pseudo-seletores.

```
1  input[type="number"]:in-range {  
2      border-color: green;  
3  }  
4  
5  input[type="date"]:out-of-range {  
6      border-color: red;  
7  }  
8  
9  input[type="checkbox"]:checked {  
10     background-color: blue;  
11 }
```

Exemplo de Código 11.10: Estilizando campos de entrada de número com diferentes estados.

Neste exemplo 11.10, o pseudo-seletor `input[type="number"]:in-range` é usado para selecionar um campo de entrada de número que está dentro do intervalo e aplicar um estilo de borda verde a ele. O pseudo-seletor `input[type="date"]:out-of-range` é usado para selecionar um campo de entrada de número que está fora do intervalo e aplicar um estilo de borda vermelha a ele. O pseudo-seletor `input[type="checkbox"]:checked` é usado para selecionar um campo de entrada de número que está marcado e aplicar um estilo de fundo azul a ele.

11.6.4 Campos Habilitados e Desabilitados

Os pseudo-seletores `:enabled` e `:disabled` são utilizados para selecionar elementos `<input>` que estejam habilitados e desabilitados, respectivamente. O código 11.11 mostra como estilizar campos de entrada de texto habilitados e desabilitados.

```
1  input:enabled {  
2      border-color: green;  
3  }  
4  
5  input:disabled {  
6      border-color: red;  
7  }
```

Exemplo de Código 11.11: Estilizando campos de entrada de texto habilitados e desabilitados.

Neste exemplo 11.11, o pseudo-seletor `input:enabled` é usado para selecionar um campo de entrada de texto que esteja habilitado e aplicar um estilo de borda verde a ele. O pseudo-seletor `input:disabled` é usado para selecionar um campo de entrada de texto que esteja desabilitado e aplicar um estilo de borda vermelha a ele.

11.6.5 Campos Somente-Leitura e Leitura-Escrita

Os pseudo-seletores `:read-only` e `:read-write` também são utilizados para selecionar elementos `<input>` que estejam em determinados estados alternativos, a saber, “somente leitura” e “leitura e escrita”, respectivamente. O código 11.12 mostra como estilizar campos de entrada de texto somente leitura e leitura e escrita.

```
1  input:read-only {  
2      border-color: green;  
3  }  
4  
5  input:read-write {  
6      border-color: red;  
7  }
```

Exemplo de Código 11.12: Estilizando campos de entrada de texto somente leitura e leitura e escrita.

Neste exemplo 11.12, o pseudo-seletor `input:read-only` é usado para selecionar um campo de entrada de texto que seja somente leitura e aplicar um estilo de borda verde a ele. O pseudo-seletor `input:read-write` é usado para selecionar um campo de entrada de texto que seja de leitura e escrita e aplicar um estilo de borda vermelha a ele.

11.6.6 Campos Obrigatórios e Opcionais

Os pseudo-seletores `:required` e `:optional` também são utilizados para selecionar elementos `<input>` que estejam em determinados estados alternativos, a saber, “obrigatório” e “opcional”, respectivamente. O código 11.13 mostra como estilizar campos de entrada de texto requeridos e opcionais.

```
1  input:required {
2      border-color: green;
3  }
4
5  input:optional {
6      border-color: red;
7  }
```

Exemplo de Código 11.13: Estilizando campos de entrada de texto requeridos e opcionais.

Neste exemplo 11.13, o pseudo-seletor `input:required` é usado para selecionar um campo de entrada de texto que seja obrigatório e aplicar um estilo de borda verde a ele. O pseudo-seletor `input:optional` é usado para selecionar um campo de entrada de texto que seja opcional e aplicar um estilo de borda vermelha a ele.

11.6.7 Campos com Foco

O pseudo-seletor `:focus` é utilizado para selecionar elementos `<input>` que estejam com o foco. O código 11.14 mostra como estilizar campos de entrada de texto com foco.

```
1  input[type="text"]:focus {
2      border-color: green;
3  }
```

Exemplo de Código 11.14: Estilizando campos de entrada de texto com foco.

Neste exemplo 11.14, o pseudo-seletor `input:focus` é usado para selecionar um campo de entrada de texto que esteja com o foco e aplicar um estilo de borda verde a ele.

11.7 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 11.1 Utilize o pseudoseletor `:before` para adicionar um asterisco (*) antes dos títulos de nível 1. ■

Exercício 11.2 Aplique o pseudoseletor `:after` para adicionar uma linha horizontal após todos os parágrafos. ■

Exercício 11.3 Utilize o pseudoseletores `::first-letter` para aumentar a fonte da primeira letra dos parágrafos. ■

Exercício 11.4 Aplique o pseudoseletores `::first-line` para alterar a cor da primeira linha de todos os parágrafos. ■

Exercício 11.5 Use o pseudoseletores `::marker` para alterar a cor dos marcadores dos itens de lista para verde. ■

Exercício 11.6 Crie um seletor CSS que utilize o pseudoseletores `[value="10"]` para aplicar uma cor de fundo aos elementos `input` com o valor 10. ■

Exercício 11.7 Utilize o pseudoseletores `:first-of-type` para alterar a cor do texto do primeiro elemento de título `h2` em uma página. ■

Exercício 11.8 Aplique o pseudoseletores `:last-of-type` para adicionar uma margem inferior no último elemento de parágrafo de uma página. ■

Exercício 11.9 Utilize o pseudoseletores `:nth-of-type(3)` para aplicar um estilo de fonte em negrito no terceiro item de uma lista não ordenada. ■

Exercício 11.10 Aplique o pseudoseletores `:nth-child(even)` para alterar a cor de fundo dos itens de lista em posições pares. ■

Exercício 11.11 Utilize o pseudoseletores `:disabled` para alterar a cor de fundo dos campos de formulário desabilitados. ■

Exercício 11.12 Aplique o pseudoseletores `:enabled` para adicionar uma borda aos campos de formulário habilitados. ■

Exercício 11.13 Utilize o pseudoseletores `:checked` para alterar a cor dos campos do tipo `checkbox` selecionados. ■

Exercício 11.14 Aplique o pseudoseletor `:required` para adicionar uma borda vermelha aos campos de formulário obrigatórios. ■

Exercício 11.15 Utilize o pseudoseletor `:valid` para aplicar uma borda verde nos campos de formulário com entrada válida. ■

Exercício 11.16 Aplique o pseudoseletor `:invalid` para adicionar uma borda vermelha nos campos de formulário com entrada inválida. ■

Exercício 11.17 Utilize o pseudoseletor `input[type="email"]` para alterar a cor do texto dos campos de e-mail. ■

Exercício 11.18 Aplique o pseudoseletor `input[type="password"]` para adicionar uma margem inferior nos campos de senha. ■

Exercício 11.19 Utilize o pseudoseletor `:focus` para aplicar uma borda azul aos campos de formulário quando estiverem em foco. ■

Exercício 11.20 Aplique o pseudoseletor `input[type="submit"]` para alterar a cor de fundo do botão de envio ao passar o mouse. ■

11.8 Considerações Sobre o Capítulo

Este capítulo apresentou alguns dos pseudo-seletores mais comuns em CSS, com exemplos de código e situações de uso para cada um deles. Aprender a utilizar os pseudo-seletores é uma habilidade importante para quem deseja se tornar um desenvolvedor web profissional. No capítulo seguinte, veremos como utilizar CSS para configurar a exibição e o posicionamento de elementos HTML em uma página.



12. Exibição e Posicionamento de Elementos

O posicionamento e a exibição de elementos são aspectos cruciais no design de uma página web. Eles determinam como os elementos HTML serão organizados na tela, criando uma estrutura visual que facilita a navegação e a compreensão do conteúdo pelos usuários.

Utilizar CSS para controlar a exibição e o posicionamento de elementos permite criar layouts flexíveis e responsivos, adaptáveis a diferentes dispositivos e tamanhos de tela. Além disso, o uso adequado das propriedades CSS relacionadas ao posicionamento e exibição permite melhorar a acessibilidade e a experiência do usuário, independentemente de suas preferências de navegação ou limitações.

Neste capítulo, exploraremos as propriedades e técnicas de CSS essenciais para criar layouts eficientes, garantindo que seu site ou aplicação web seja visualmente atraente e funcional.

12.1 Exibição de Elementos

Nesta seção, discutiremos como controlar a exibição de elementos HTML usando CSS. As propriedades `display`, `visibility` e `opacity` permitem alterar a maneira como os elementos são exibidos na página, o que pode ser útil em diferentes situações, como ocultar elementos, alterar o fluxo do layout e ajustar a transparência. Exploraremos cada uma dessas propriedades e seus valores nas subseções a seguir.

12.1.1 Controle de Exibição

A propriedade `display` é fundamental para controlar como os elementos HTML são exibidos na página. Ela possui diversos valores, mas abordaremos os mais comuns: `none`, `block`, `inline` e `inline-block`. O código 12.1 mostra um exemplo de como utilizar a propriedade `display`.

```
1  /* Ocultar um elemento */
2  .oculto {
3      display: none;
4  }
5
6  /* Elemento de bloco */
7  .bloco {
8      display: block;
9  }
10
11 /* Elemento inline */
12 .em-linha {
13     display: inline;
14 }
15
16 /* Elemento inline-block */
17 .bloco-em-linha {
18     display: inline-block;
19 }
```

Exemplo de Código 12.1: Exemplo de utilização da propriedade `display`.

Neste exemplo 12.1, a classe `.oculto` oculta um elemento ao atribuir o valor `none` à propriedade `display`. A classe `.bloco` define um elemento como bloco, ou seja, ele ocupará toda a largura disponível e quebrará a linha antes e depois do elemento. A classe `.em-linha` define um elemento como `inline`, fazendo com que ele ocupe apenas a largura necessária e seja exibido na mesma linha que seus elementos adjacentes. Por fim, a classe `.bloco-em-linha` define um elemento como `inline-block`, o que permite que ele seja exibido na mesma linha que outros elementos e ainda mantenha as características de um elemento de bloco, como largura e altura definidas.

12.1.2 Controle de Visibilidade

A propriedade `visibility` controla a visibilidade dos elementos, podendo ocultá-los ou torná-los visíveis. Ela possui dois valores principais: `hidden` e `visible`. O código 12.2 mostra um exemplo de utilização da propriedade `visibility`.

Neste exemplo 12.2, a classe `.oculto` atribui o valor `hidden` à propriedade `visibility`, tornando o elemento invisível. No entanto, diferentemente do `display: none`, o elemento ainda ocupará espaço no layout da página. A classe `.visivel` atribui o valor `visible` à propriedade `visibility`, garantindo que o elemento seja visível. Vale ressaltar que o valor padrão da propriedade `visibility` é `visible`, ou seja, os elementos são visíveis por padrão.

```
1  /* Elemento oculto */
2  .oculto {
3      visibility: hidden;
4  }
5
6  /* Elemento visível */
7  .visivel {
8      visibility: visible;
9  }
```

Exemplo de Código 12.2: Exemplo de utilização da propriedade `visibility`.

12.1.3 Controle de Opacidade

A propriedade `opacity` permite controlar a transparência (ou opacidade) de um elemento e seus conteúdos, incluindo imagens e texto. Seu valor pode variar de 0 a 1, onde 0 significa totalmente transparente e 1, totalmente opaco. O código 12.3 mostra um exemplo de utilização da propriedade `opacity`.

```
1  /* Elemento com transparência de 50% */
2  .transparencia {
3      opacity: 0.5;
4  }
```

Exemplo de Código 12.3: Exemplo de utilização da propriedade `opacity`.

Neste exemplo 12.3, a classe `.transparencia` atribui o valor 0.5 à propriedade `opacity`, tornando o elemento e seus conteúdos parcialmente transparentes, neste caso, com 50% de transparência.

12.2 Posicionamento de Elementos

Nesta seção, discutiremos como controlar o posicionamento dos elementos HTML usando CSS. As propriedades `position`, `left`, `right`, `top`, `bottom` e `z-index` permitem alterar o posicionamento e a ordem de empilhamento dos elementos na página. Exploraremos cada uma dessas propriedades e seus valores nas subseções a seguir.

12.2.1 Modos de Posicionamento

A propriedade `position` define como um elemento é posicionado na página e aceita os valores `static`, `relative`, `absolute`, `fixed` e `sticky`. O código 12.4 mostra um exemplo de utilização dos diferentes valores da propriedade `position`.

```
1  /* Posicionamento estático */
2  .estático {
3      position: static;
4  }
5
6  /* Posicionamento relativo */
7  .relativo {
8      position: relative;
9      left: 20px;
10     top: 10px;
11 }
12
13 /* Posicionamento absoluto */
14 .absoluto {
15     position: absolute;
16     right: 10px;
17     bottom: 20px;
18 }
19
20 /* Posicionamento fixo */
21 .fixo {
22     position: fixed;
23     top: 0;
24     right: 0;
25 }
26
27 /* Posicionamento com afastamento */
28 .sticky {
29     position: sticky;
30     top: 10px;
31 }
```

Exemplo de Código 12.4: Exemplo de utilização da propriedade position.

Neste exemplo 12.4, a propriedade position é utilizada em diferentes seletores por classe, com diferentes valores. Quando o valor é static (estático), o elemento é posicionado de acordo com o fluxo normal de renderização da página. No caso do valor relative (relativo), o elemento é posicionado em relação à posição que ele teria se fosse estático, deslocando-se 20 pixels à direita e 10 pixels abaixo. O valor absolute (absoluto) posiciona o elemento em relação ao primeiro ancestral com posicionamento não estático (relativo ou absoluto), neste caso, 10 pixels à esquerda e 20 pixels acima. Com o valor fixed (fixo), o elemento é posicionado em relação à janela do navegador, tendo com referência o topo à esquerda. Por fim, o valor sticky (pegajoso) faz com que o elemento role com a página até um determinado limite baseado na janela de visualização do navegador, estando sempre visível.

12.2.2 Afastamentos

As propriedades `left`, `right`, `top` e `bottom` são utilizadas em conjunto com a propriedade `position` para determinar o posicionamento de um elemento em relação a um dos lados de seu elemento pai, sendo que ela funciona somente quando o próprio elemento não é estático, conforme explicado na subseção 12.2.1. Em síntese, o referencial (0,0) dos afastamentos vai depender do tipo de posicionamento do próprio elemento e do elemento pai. O código 12.5 mostra um exemplo de utilização dessas propriedades.

```
1  /* Elemento com posicionamento absoluto */
2  .absoluto {
3      position: absolute;
4      left: 20px;
5      top: 10px;
6  }
7
8  /* Elemento com posicionamento fixo */
9  .fixo {
10     position: fixed;
11     right: 30px;
12     bottom: 15px;
13 }
```

Exemplo de Código 12.5: Exemplo de utilização das propriedades `left`, `right`, `top` e `bottom`.

Neste exemplo 12.5, as propriedades `left`, `right`, `top` e `bottom` são utilizadas em conjunto com a propriedade `position` para posicionar os elementos. No primeiro caso, o elemento com `position: absolute` é posicionado 20 pixels à direita e 10 pixels abaixo do seu ancestral posicionado. No segundo caso, o elemento com `position: fixed` é posicionado 30 pixels à esquerda e 15 pixels acima em relação à janela do navegador. Um detalhe interessante a mencionar é que, quando todos os elementos ancestrais são estáticos, o referencial (0,0) é a janela do navegador. Outro detalhe interessante é que, se você definir um valor para `left` e `right` e o elemento não possuir uma largura definida, o elemento será esticado para ocupar todo o espaço disponível entre os dois afastamentos laterais. O mesmo vale para `top` e `bottom`.

12.2.3 Ordem de Empilhamento

A propriedade `z-index` é usada para controlar a ordem de empilhamento dos elementos na página. Um elemento com um maior valor de `z-index` aparecerá na frente de um elemento com um valor menor. O valor padrão é zero. Valores positivos fazem os elementos se aproximarem de você, enquanto valores negativos fazem os elementos se afastarem de você. O código 12.6 mostra um exemplo de utilização da propriedade `z-index`.

```
1  .elemento1 {
2      position: absolute;
3      z-index: 1;
4  }
5
6  .elemento2 {
7      position: absolute;
8      z-index: 2;
9  }
10
11 .elemento3 {
12     position: absolute;
13     z-index: 3;
14 }
```

Exemplo de Código 12.6: Exemplo de utilização da propriedade `z-index`.

Neste exemplo 12.6, três elementos são posicionados de forma absoluta, mas com diferentes valores de `z-index`. O elemento com o maior valor de `z-index` (`elemento3`) aparecerá na frente dos outros dois elementos, enquanto o elemento com o menor valor (`elemento1`) ficará atrás dos outros.

12.3 Propriedades Complementares para Posicionamento

Nesta seção, abordaremos propriedades CSS adicionais relacionadas ao posicionamento de elementos, como `overflow`, `float`, `clear` e `box-sizing`. Essas propriedades são úteis para ajustar a aparência e o comportamento dos elementos em um leiaute de página. Dividiremos a seção em três subseções, cada uma focando em um conjunto específico de propriedades: transbordo de conteúdo, flutuação de elementos e modos de dimensionamento.

12.3.1 Transbordo de Conteúdo

A propriedade `overflow` controla o que acontece quando o conteúdo de um elemento excede seu tamanho especificado, seja em largura ou altura. Ela pode ser dividida em duas propriedades específicas: `overflow-x` e `overflow-y`, que controlam, respectivamente, o transbordo horizontal e vertical. Os possíveis valores para essas propriedades são: `visible`, `hidden`, `scroll` e `auto`. O código 12.7 mostra um exemplo de uso da propriedade `overflow`.

```
1  div {  
2      width: 200px;  
3      height: 200px;  
4      border: 1px solid black;  
5      overflow: scroll;  
6  }
```

Exemplo de Código 12.7: Exemplo de uso da propriedade `overflow`.

Neste exemplo 12.7, temos uma `div` com largura e altura fixas de 200 pixels. A propriedade `overflow` é definida como `scroll`, o que significa que, caso o conteúdo exceda o tamanho da `div`, barras de rolagem serão habilitadas, permitindo que o usuário visualize todo o conteúdo. Vale ressaltar que, nesse modo, as barras de rolagem ficam sempre visíveis e são habilitadas sob demanda. Vamos analisar os outros valores possíveis:

- a) `visible`: É o valor padrão. Se o conteúdo exceder o tamanho do elemento, ele será exibido normalmente, podendo invadir o espaço de outros elementos adjacentes;
- b) `hidden`: Caso o conteúdo exceda o tamanho do elemento, ele será ocultado e não haverá barras de rolagem;
- c) `auto`: Semelhante ao `scroll`, mas as barras de rolagem só serão exibidas se o conteúdo realmente exceder o tamanho do elemento.

12.3.2 Flutuação de Elementos

A propriedade `float` é usada para posicionar um elemento de tal forma que outros elementos do documento fluam ao seu redor. Os valores possíveis para essa propriedade são: `left`, `right` e `none` (padrão). Com o `float`, os elementos são movidos para a esquerda ou direita, permitindo que o conteúdo adjacente flua ao seu redor.

```
1  img {  
2      float: left;  
3      margin-right: 10px;  
4      margin-bottom: 10px;  
5  }
```

Exemplo de Código 12.8: Exemplo de uso da propriedade `float`.

Neste exemplo 12.8, temos uma imagem que utiliza a propriedade `float` com o valor `left`. Isso faz com que a imagem seja alinhada à esquerda e o texto ao seu redor flua ao longo de sua borda direita. Além disso, adicionamos margens à direita e à parte inferior da imagem para criar espaço entre ela e o texto adjacente.

A propriedade `clear`, por sua vez, é usada em conjunto com o `float` para controlar o comportamento dos elementos adjacentes. Ela determina de qual lado de um elemento flutuante os elementos subsequentes não devem ser posicionados. Os valores possíveis para `clear` são: `left`, `right`, `both` e `none` (padrão).

No exemplo 12.8, se a imagem estivesse inserida dentro de um parágrafo e a propriedade `clear` fosse aplicada ao parágrafo com o valor `both`, isso faria com que o parágrafo não fosse posicionado ao lado de elementos flutuantes à esquerda ou à direita, ou seja, o parágrafo começaria em uma nova linha abaixo dos elementos flutuantes.

12.3.3 Modos de Dimensionamento

A propriedade `box-sizing` define como o navegador calcula o tamanho total de um elemento, incluindo bordas, margens internas e externas. Os valores possíveis para essa propriedade são `content-box` (padrão) e `border-box`.

Quando o valor é `content-box`, a largura e altura especificadas se aplicam apenas ao conteúdo do elemento, sem incluir bordas, margens internas e externas. Isso significa que o tamanho total do elemento será maior do que o valor especificado para largura e altura, devido às bordas e margens adicionados.

Por outro lado, quando o valor é `border-box`, a largura e altura especificadas incluem o conteúdo, bordas e margens internas, mas não as margens externas. Dessa forma, o tamanho total do elemento será exatamente igual aos valores de largura e altura especificados, independentemente das espessuras de borda e de margens internas. O código 12.9 mostra um exemplo de uso da propriedade `box-sizing`.


```
1  div {  
2    width: 300px;  
3    height: 200px;  
4    padding: 10px;  
5    border: 5px solid black;  
6    box-sizing: border-box;  
7  }
```

Exemplo de Código 12.9: Exemplo de uso da propriedade `box-sizing`.

Neste exemplo 12.9, temos uma `div` com largura e altura fixas de 300 e 200 pixels, respectivamente. A propriedade `box-sizing` é definida como `border-box`, o que significa que a largura e altura especificadas incluem o conteúdo, as bordas e a margem interna (`padding`). Portanto, o tamanho total do elemento será exatamente 300 pixels de largura e 200 pixels de altura, mesmo com o preenchimento de 10 pixels e a borda de 5 pixels.

Se utilizássemos o valor padrão `content-box` no exemplo 12.9, o tamanho total do elemento seria calculado como a soma da largura e altura do conteúdo (300x200 pixels), mais o preenchimento (10 pixels em cada lado) e mais a borda (5 pixels em cada lado). Nesse caso, o elemento teria um tamanho total de 330x230 pixels.

É muito comum usar a propriedade `border-box: box-sizing;` em um seletor CSS universal (*) para que todos os elementos de uma página sejam dimensionados dessa forma. Isso é especialmente útil quando se está trabalhando com elementos que possuem largura e altura fixas, como imagens e vídeos. Essa configuração universal também costuma incluir a atribuição de zero para valores de margens externas e internas. O código 12.10 mostra um exemplo de uso da propriedade `box-sizing` em um seletor CSS universal.

```
1  * {  
2    margin: 0;  
3    padding: 0;  
4    box-sizing: border-box;  
5  }
```

Exemplo de Código 12.10: Exemplo de uso da propriedade `box-sizing` em um seletor CSS universal.

12.4 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 12.1 Crie um seletor CSS que aplique a propriedade `display` com valor `inline` nos elementos `<h1>` para exibi-los na mesma linha. ■

Exercício 12.2 Crie um seletor CSS que utilize a propriedade `display` com valor `block` nos elementos `` para exibi-los como blocos independentes. ■

Exercício 12.3 Crie um seletor CSS que aplique a propriedade `display` com valor `inline-block` nos elementos `` para permitir ajustes de margem e preenchimento. ■

Exercício 12.4 Crie um seletor CSS que utilize a propriedade `display` com valor `none` nos elementos com a classe "ocultar" para remover `<div>` da exibição. ■

Exercício 12.5 Crie um seletor CSS que aplique a propriedade `visibility` com valor `hidden` nos elementos com a classe "invisível" para esconder `<p>`. ■

Exercício 12.6 Crie um seletor CSS que utilize a propriedade `visibility` com valor `visible` nos elementos com a classe "visível" para garantir a exibição de `<p>`. ■

Exercício 12.7 Crie um seletor CSS que aplique a propriedade `opacity` com valor `0.5` nos elementos `` para deixá-los semitransparentes. ■

Exercício 12.8 Crie um seletor CSS que utilize a propriedade `opacity` com valor `1` nos elementos `` ao passar o mouse sobre eles para torná-los totalmente opacos. ■

Exercício 12.9 Crie um seletor CSS que aplique a propriedade `display` com valor `none` nos elementos `<h2>` com atributo `data-hide="true"`. ■

Exercício 12.10 Crie um seletor CSS que aplique a propriedade `visibility` com valor `hidden` nos elementos `<p>` com atributo `data-visibility="false"`. ■

Exercício 12.11 Crie um seletor CSS que aplique a propriedade `position` com valor `static` no elemento `<div>` para posicioná-lo na ordem normal do fluxo. ■

Exercício 12.12 Crie um seletor CSS que utilize a propriedade `position` com valor `relative` no elemento `` para movê-lo em relação à sua posição original. ■

Exercício 12.13 Crie um seletor CSS que aplique a propriedade `position` com valor `absolute` no elemento `<p>` para posicioná-lo em relação ao seu ancestral mais próximo posicionado. ■

Exercício 12.14 Crie um seletor CSS que utilize a propriedade `position` com valor `fixed` no elemento `<nav>` para fixá-lo na janela do navegador. ■

Exercício 12.15 Crie um seletor CSS que aplique a propriedade `position` com valor `sticky` no elemento `<header>` para mantê-lo fixo durante a rolagem. ■

Exercício 12.16 Crie um seletor CSS que utilize as propriedades `top` e `left` com valores em pixels para posicionar o elemento `<div>` com `position: relative`. ■

Exercício 12.17 Crie um seletor CSS que aplique as propriedades `bottom` e `right` com valores em porcentagem para posicionar o elemento `<p>` com `position: absolute`. ■

Exercício 12.18 Crie um seletor CSS que utilize a propriedade `z-index` para controlar a ordem de empilhamento do elemento `` em relação aos elementos adjacentes. ■

Exercício 12.19 Crie um seletor CSS que aplique a propriedade `z-index` com valor maior nos elementos `<button>` para mantê-los acima dos outros elementos. ■

Exercício 12.20 Crie um seletor CSS que utilize a propriedade `z-index` com valor negativo no elemento `<div>` para posicioná-lo abaixo dos elementos com valor `z-index` padrão. ■

Exercício 12.21 Crie um elemento `div` com o conteúdo transbordando verticalmente. Utilize a propriedade `overflow-y` para adicionar uma barra de rolagem vertical. ■

Exercício 12.22 Explique a diferença entre os valores `visible`, `hidden`, `scroll` e `auto` para a propriedade `overflow`. ■

Exercício 12.23 Descreva um cenário prático em que seria útil aplicar a propriedade `float` com o valor `left`. ■

Exercício 12.24 Crie uma galeria de imagens utilizando a propriedade `float` para organizar as imagens lado a lado, e aplique o valor `clear` apropriado para evitar sobreposição com outros elementos da página. ■

Exercício 12.25 Explique a diferença entre os valores `left`, `right`, `both` e `none` para a propriedade `clear`, e como eles afetam elementos adjacentes a elementos flutuantes. ■

Exercício 12.26 Crie dois elementos `div` com largura, altura, margem interna e borda definidos. Aplique a propriedade `box-sizing` com o valor `border-box` em um deles e `content-box` no outro. Descreva a diferença no tamanho total dos elementos. ■

Exercício 12.27 Explique como a propriedade `overflow` afeta a aparência de um elemento com conteúdo transbordando, e como isso pode ser útil ao criar leiautes responsivos. ■

Exercício 12.28 Utilize a propriedade `float` para criar um leiaute de duas colunas, com uma coluna à esquerda para o menu de navegação e outra à direita para o conteúdo principal. ■

Exercício 12.29 Crie um exemplo que demonstre o uso de `overflow-x` e `overflow-y` em conjunto para controlar o transbordo de conteúdo em ambas as direções (horizontal e vertical). ■

Exercício 12.30 Descreva um cenário em que seria útil usar a propriedade `box-sizing` com o valor `border-box` para criar um leiaute flexível e adaptável. ■

12.5 Considerações Sobre o Capítulo

Ao longo deste capítulo, exploramos as principais propriedades CSS relacionadas à exibição e ao posicionamento de elementos em uma página web. Aprendemos como controlar a visibilidade e a exibição de elementos usando as propriedades `display`, `visibility` e `opacity`. Além disso, discutimos o posicionamento de elementos com as propriedades `position`, `left`, `right`, `top`, `bottom` e `z-index`. No capítulo seguinte, veremos como usar a tecnologia CSS *flexbox* para criar leiautes modernos e sofisticados.



13. Leiautes de Página com *Flexbox*


Neste capítulo, abordaremos a criação de leiautes de página com CSS *flexbox*, uma tecnologia moderna para a criação de leiautes sofisticados. Veremos cada uma das propriedades juntamente com exemplos práticos. Veremos também alguns exemplos práticos de uso do *flexbox*. Ao final do capítulo, você será capaz de criar leiautes de página sofisticados e responsivos para seus projetos web.

13.1 Introdução

Desde o início da web, os desenvolvedores enfrentam o desafio de criar leiautes eficientes e adaptáveis para suas páginas. Ao longo dos anos, a linguagem CSS evoluiu, incorporando novas propriedades e técnicas para facilitar a criação de leiautes.

No início da web, os leiautes eram baseados principalmente em tabelas HTML. No entanto, essa abordagem tinha várias limitações, além de misturar conteúdo e apresentação. Com o tempo, o CSS começou a ganhar recursos específicos para criação de leiautes, como o `position`, o `float` e o `clear`. Ainda assim, criar leiautes complexos e flexíveis era um desafio. Felizmente, as tecnologias avançaram e, atualmente, dispomos de soluções mais robustas e eficientes, como o *flexbox*.

O *Flexbox* é um modelo de layout do CSS que oferece uma maneira eficiente de alinhar, distribuir e dimensionar itens dentro de um contêiner, independentemente do tamanho do contêiner. Ele é especialmente útil para criar leiautes responsivos e adaptáveis para diferentes dispositivos e tamanhos de tela. Nas seções a seguir, abordaremos os conceitos básicos do *flexbox*, as propriedades do contêiner e dos itens *flexbox*, e exemplos práticos de como usar o *Flexbox* em projetos web.



13.2 Conceitos Básicos

O *flexbox* é baseado em uma estrutura de contêiner e itens. O contêiner *flexbox* é o elemento pai que envolve os itens *flexbox*, que são os elementos filhos. Ao aplicar a propriedade `display: flex;` ao contêiner, ele se torna um contêiner *flexbox* e seus **filhos diretos** se tornam itens *flexbox*. O código 13.1 mostra um exemplo de como criar um contêiner *flexbox* e seus itens *flexbox*.

```
1 | .container {  
2 |     display: flex;  
3 | }
```

Exemplo de Código 13.1: Exemplo de contêiner *flexbox*.

Neste exemplo 13.1, aplicamos a propriedade `display` com o valor *flexbox* ao elemento com a classe `container`. Isso transforma o elemento em um contêiner *flexbox* e seus filhos diretos se tornam itens *flexbox*. A figura 13.1 mostra um contêiner e alguns itens *flexbox*.



Figura 13.1: Contêiner e itens *flexbox*.

O *flexbox* possui dois eixos direcionais: o eixo principal (*main axis*) e o eixo transversal (*cross axis*). O eixo principal é definido pela propriedade `flex-direction`, enquanto o eixo transversal é sempre perpendicular ao eixo principal. A direção e a orientação desses eixos determinam como os itens *flexbox* são organizados dentro do contêiner. Veremos isso e mais propriedades *flexbox* nas seções seguintes.

13.3 Propriedades do Contêiner

As propriedades do contêiner *flexbox* afetam a organização, o alinhamento e o espaçamento dos itens *flexbox* dentro do contêiner. Vamos explorar algumas das principais propriedades e seus possíveis valores.

13.3.1 Direção dos Eixos

A propriedade `flex-direction` define o eixo principal e a direção na qual os itens *flexbox* são organizados. Os possíveis valores são: `row` (padrão), `row-reverse`, `column` e `column-reverse`. O código 13.2 mostra um exemplo de uso do `flex-direction`.

```
1  .container {  
2      display: flex;  
3      flex-direction: column;  
4  }
```

Exemplo de Código 13.2: Alterando a direção do eixo *flexbox* com *flex-direction*.

Neste exemplo 13.2, definimos a direção do eixo principal como *column*, o que organiza os itens *flexbox* em uma coluna, um embaixo do outro. A figura 13.2 mostra as configurações para os possíveis valores dessa propriedade. Neste caso, a configuração em coluna com seta para baixo corresponde ao exemplo 13.2.

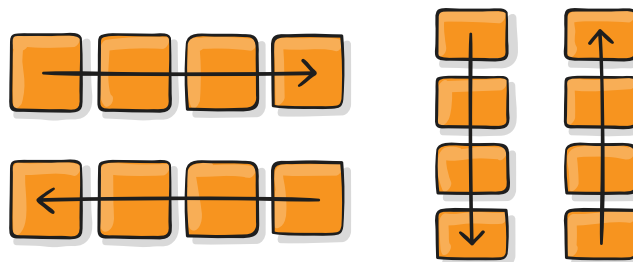


Figura 13.2: Possíveis configurações de direção e sentido do eixo *flexbox*.

13.3.2 Quebras de Linha

A propriedade *flex-wrap* controla se os itens *flexbox* devem ser ajustados em várias linhas ou colunas quando não há espaço suficiente no contêiner. Os possíveis valores são: *nowrap* (padrão), *wrap* e *wrap-reverse*. O código 13.3 mostra um exemplo de uso do *flex-wrap*.

```
1  .container {  
2      display: flex;  
3      flex-wrap: wrap;  
4  }
```

Exemplo de Código 13.3: Configurando o comportamento de quebra de linha em um contêiner *flexbox*.

Neste exemplo 13.3, definimos a propriedade *flex-wrap* como *wrap*, permitindo que os itens *flexbox* se ajustem em várias linhas, se necessário, de acordo com o tamanho do contêiner.

13.3.3 Propriedade Abreviada para Direção e Quebra de Linha

A propriedade *flex-flow* do CSS é uma propriedade abreviada que define simultaneamente as propriedades *flex-direction* e *flex-wrap* de um contêiner *flexbox*. Ao usar *flex-flow*, você pode definir a direção e o comportamento de quebra de linha dos itens *flexbox* em uma única declaração. O código 13.4 mostra um exemplo de uso da propriedade *flex-flow*.

```
1  .container {  
2      display: flex;  
3      flex-flow: row wrap;  
4  }
```

Exemplo de Código 13.4: Exemplo de uso da propriedade `flex-flow`.

Neste exemplo 13.4, temos um contêiner *flexbox* com a propriedade `flex-flow` configurada para `row wrap`. Isso significa que os itens *flexbox* serão posicionados ao longo do eixo horizontal (eixo principal) e, se não houver espaço suficiente, ocorrerá uma quebra de linha e os itens serão posicionados na linha seguinte.

13.3.4 Alinhamento ao Longo do Eixo Principal

A propriedade `justify-content` do CSS é usada para alinhar os itens *flexbox* ao longo do eixo principal de um contêiner *flexbox*. A propriedade `justify-content` tem várias opções para controlar o espaçamento entre os itens *flexbox*. Os possíveis valores para a propriedade `justify-content` são:

- a) `flex-start`: Alinha os itens *flexbox* ao início do contêiner *flexbox*.
- b) `flex-end`: Alinha os itens *flexbox* ao final do contêiner *flexbox*.
- c) `center`: Alinha os itens *flexbox* no centro do contêiner *flexbox*.
- d) `space-between`: Distribui os itens *flexbox* uniformemente no contêiner *flexbox*, com o primeiro item alinhado ao início e o último item alinhado ao final do contêiner.
- e) `space-around`: Distribui os itens *flexbox* uniformemente no contêiner *flexbox*, com espaço igual ao redor de cada item.
- f) `space-evenly`: Distribui os itens *flexbox* uniformemente no contêiner *flexbox*, garantindo que o espaço entre qualquer par de itens adjacentes seja igual.

O código 13.5 mostra um exemplo de uso da propriedade `justify-content`.

```
1  .container {  
2      display: flex;  
3      justify-content: space-around;  
4  }
```

Exemplo de Código 13.5: Exemplo de uso da propriedade `justify-content`.

Neste exemplo 13.5, temos um contêiner *flexbox* com a propriedade *justify-content* configurada para *space-around*. Isso significa que os itens *flexbox* serão distribuídos uniformemente no contêiner *flexbox*, com espaço igual ao redor de cada item. A figura 13.3 mostra as possíveis configurações para a propriedade *justify-content*.

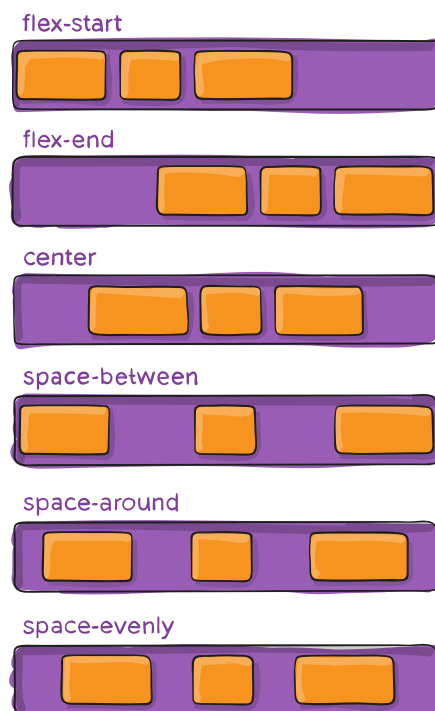


Figura 13.3: Possíveis configurações de alinhamento ao longo do eixo principal.

13.3.5 Alinhamento ao Longo do Eixo Transversal

A propriedade *align-items* do CSS é utilizada para alinhar os itens *flexbox* ao longo do eixo transversal de um contêiner *flexbox*. Essa propriedade tem várias opções para controlar o alinhamento vertical dos itens *flexbox* quando o eixo principal é horizontal, ou o alinhamento horizontal dos itens *flexbox* quando o eixo principal é vertical. Os possíveis valores para a propriedade *align-items* são:

- a) *flex-start*: Alinha os itens *flexbox* ao início do contêiner *flexbox* no eixo transversal.
- b) *flex-end*: Alinha os itens *flexbox* ao final do contêiner *flexbox* no eixo transversal.
- c) *center*: Alinha os itens *flexbox* no centro do contêiner *flexbox* no eixo transversal.
- d) *baseline*: Alinha os itens *flexbox* de acordo com a linha-base do texto de cada item.
- e) *stretch*: Estende os itens *flexbox* para preencher todo o espaço disponível do contêiner *flexbox* no eixo transversal, desde que o item não possua uma altura (quando o eixo principal é horizontal) ou largura (quando o eixo principal é vertical) fixa.

O código 13.6 mostra um exemplo de uso da propriedade *align-items*.

```
1  .container {  
2      display: flex;  
3      align-items: center;  
4      height: 300px;  
5  }
```

Exemplo de Código 13.6: Alinhando itens ao longo do eixo *flexbox* transversal.

Neste exemplo 13.6, temos um contêiner *flexbox* com a propriedade `align-items` configurada para `center`. Isso significa que os itens *flexbox* serão centralizados no eixo transversal do contêiner *flexbox*. Neste caso, o contêiner *flexbox* tem uma altura maior que a dos itens *flexbox*. A figura 13.4 mostra as configurações para os possíveis valores da propriedade `align-items`.

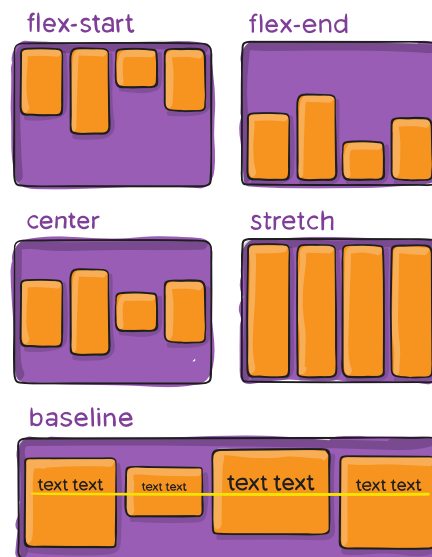


Figura 13.4: Configurações de alinhamento de itens no eixo *flexbox* transversal.

13.3.6 Alinhamento de Linhas *Flexbox*

A propriedade `align-content` do CSS é utilizada para alinhar as linhas *flexbox* no contêiner *flexbox* quando há espaço extra disponível no eixo transversal. Essa propriedade não possui efeito quando há apenas uma linha de itens *flexbox*. A propriedade `align-content` controla o espaçamento entre as linhas *flexbox*, e é aplicável apenas ao contêiner *flexbox*, não aos itens individuais. Os possíveis valores para a propriedade `align-content` são:

- a) `normal`: O espaço extra é distribuído igualmente entre as linhas.
- b) `flex-start`: As linhas são agrupadas no início do contêiner *flexbox*.
- c) `flex-end`: As linhas são agrupadas no final do contêiner *flexbox*.
- d) `center`: As linhas são agrupadas no centro do contêiner *flexbox*.

- e) `space-between`: O espaço extra é distribuído igualmente entre as linhas, com a primeira e a última linhas alinhadas ao início e ao fim do contêiner, respectivamente.
- f) `space-around`: O espaço extra é distribuído igualmente entre as linhas, com metade do espaço aplicado antes e depois de cada linha.
- g) `space-evenly`: O espaço extra é distribuído igualmente entre as linhas, garantindo que o espaço entre todas as linhas seja igual, incluindo o espaço antes da primeira linha e após a última linha.

O código 13.7 mostra um exemplo de uso da propriedade `align-content`.

```
1  .container {  
2      display: flex;  
3      flex-wrap: wrap;  
4      align-content: space-around;  
5  }
```

Exemplo de Código 13.7: Exemplo de uso da propriedade `align-content`.

Neste exemplo 13.7, temos um contêiner *flexbox* com a propriedade `align-content` configurada para `space-around`. Isso significa que o espaço extra no eixo transversal será distribuído igualmente entre as linhas, com metade do espaço aplicado antes e depois de cada linha. A figura 13.5 mostra as possíveis configurações para a propriedade `align-content`.

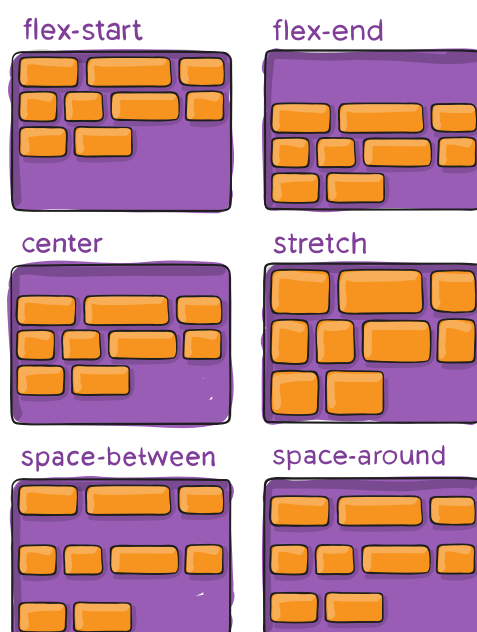


Figura 13.5: Possíveis alinhamentos de linhas em um contêiner *flexbox*.

13.4 Propriedades dos Itens

Nesta subseção, discutiremos as propriedades dos itens *flexbox* que podem ser aplicadas aos elementos filhos de um contêiner *flexbox* para controlar seu comportamento dentro do contêiner. As propriedades abordadas incluem `order`, `flex-grow`, `flex-shrink`, `flex-basis`, `align-self` e `flex`.

13.4.1 Ordenação

A propriedade `order` é utilizada para controlar a ordem dos itens *flexbox* dentro do contêiner *flexbox*. O código 13.8 mostra um exemplo de uso da propriedade `order`.

```
1  .item1 {  
2      order: 2;  
3  }  
4  
5  .item2 {  
6      order: 1;  
7  }
```

Exemplo de Código 13.8: Exemplo de uso da propriedade `order`.

Neste exemplo 13.8, o item *flexbox* da classe `item1` será exibido após o item *flexbox* da classe `item2` no contêiner *flexbox*, pois o valor da propriedade `order` de `item1` é maior que o de `item2`. Os possíveis valores para a propriedade `order` são números inteiros, que podem ser positivos, negativos ou zero. Os itens são ordenados de acordo com o valor da propriedade `order` em ordem crescente. A figura 13.6 mostra alguns possíveis valores para a propriedade `order` e como os itens *flexbox* ficariam ordenados.

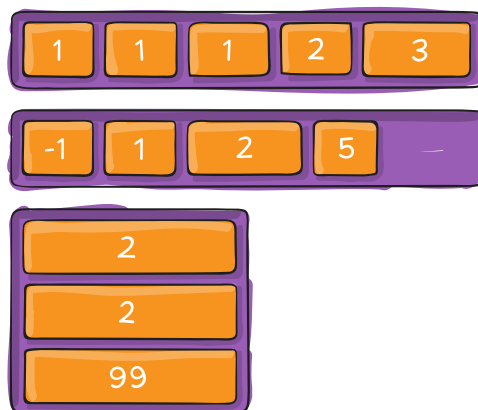


Figura 13.6: Configurações possíveis de `order`.

13.4.2 Fator de Crescimento

A propriedade `flex-grow` define o fator de crescimento de um item *flexbox*, ou seja, como ele deve crescer em relação aos outros itens *flexbox* dentro do contêiner *flexbox* quando há espaço disponível.

O código 13.9 mostra um exemplo de uso da propriedade `flex-grow`.

```
1  .item1 {  
2      flex-grow: 1;  
3  }  
4  
5  .item2 {  
6      flex-grow: 2;  
7  }
```

Exemplo de Código 13.9: Alterando o fator de crescimento de itens *flexbox*.

Neste exemplo 13.9, o item *flexbox* da classe `item2` crescerá duas vezes mais rápido que o item da classe `item1` quando houver espaço disponível no contêiner *flexbox*. Os possíveis valores para a propriedade `flex-grow` são números não negativos. Um valor zero indica que o item não deve crescer além do tamanho necessário para acomodar seu conteúdo. A figura 13.7 mostra configurações para alguns possíveis valores da propriedade `flex-grow`.

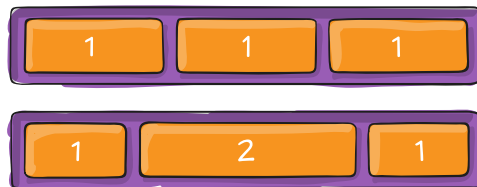


Figura 13.7: Configurações possíveis de `flex-grow`.

13.4.3 Fator de Encolhimento

A propriedade `flex-shrink` define o fator de encolhimento do item *flexbox* em relação aos outros itens *flexbox* do contêiner quando não há espaço suficiente no contêiner. O código 13.10 mostra um exemplo do uso de `flex-shrink`.

```
1  .item1 {  
2      flex-shrink: 1;  
3  }  
4  
5  .item2 {  
6      flex-shrink: 2;  
7  }
```

Exemplo de Código 13.10: Exemplo de uso do `flex-shrink`.

Neste exemplo 13.10, o item *flexbox* da classe *item1* encolherá com a metade da velocidade do item *flexbox* da classe *item2* quando o espaço no contêiner for insuficiente. Os possíveis valores para *flex-shrink* são números positivos, que determinam o fator de encolhimento em relação aos outros itens *flexbox*. Um valor de zero indica que o item não deve encolher.

13.4.4 Tamanho Base

A propriedade *flex-basis* define o tamanho inicial do item *flexbox* antes de ser distribuído o espaço disponível no contêiner. O código 13.11 mostra um exemplo de uso do *flex-basis*.

```
1  .item1 {  
2      flex-basis: 200px;  
3  }  
4  .item2 {  
5      flex-basis: 50%;  
6  }
```

Exemplo de Código 13.11: Configurando tamanhos iniciais de itens *flexbox*.

Neste exemplo 13.11, o item *flexbox* da classe *item1* tem um tamanho inicial de 200 pixels, enquanto o item *flexbox* da classe *item2* tem um tamanho inicial de 50% da largura do contêiner. Os possíveis valores para *flex-basis* são valores de largura (em px, vw, % etc.), que definem o tamanho inicial do item com base em um valor de largura específico. O valor *auto* também é aceito e define o tamanho inicial do item com base no conteúdo do item.

13.4.5 Alinhamento Individual de Itens

A propriedade *align-self* permite que você ajuste individualmente o alinhamento de um item *flexbox* ao longo do eixo cruzado, anulando o alinhamento definido pela propriedade *align-items* no contêiner *flexbox*. Os possíveis valores para essa propriedade são:

- a) *auto*: herda o alinhamento do contêiner *flexbox* (definido por *align-items*).
- b) *flex-start*: alinha o item *flexbox* ao início do eixo cruzado.
- c) *flex-end*: alinha o item *flexbox* ao final do eixo cruzado.
- d) *center*: centraliza o item *flexbox* no eixo cruzado.
- e) *baseline*: alinha o item *flexbox* pela linha base do texto.
- f) *stretch*: estica o item *flexbox* para preencher o eixo cruzado, se não houver altura ou largura definida.

O código 13.12 mostra um exemplo de uso da propriedade *align-self*.

```
1  .container {  
2      display: flex;  
3      align-items: center;  
4      height: 200px;  
5  }  
6  
7  .item-1 {  
8      align-self: flex-start;  
9  }  
10  
11 .item-2 {  
12     align-self: flex-end;  
13 }
```

Exemplo de Código 13.12: Exemplo de uso da propriedade `align-self`.

Neste exemplo 13.12, os itens *flexbox* estão alinhados ao centro no eixo transversal devido à propriedade `align-items` no contêiner. Porém, o item da classe `item1` está alinhado ao início do eixo transversal, graças à propriedade `align-self`, e o item *flexbox* da classe `item2` está alinhado ao final do eixo transversal pelo mesmo motivo. A figura 13.8 mostra algumas configurações para possíveis valores da propriedade `align-self`.

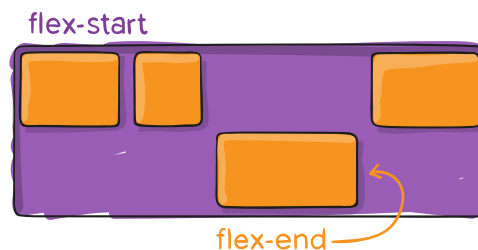


Figura 13.8: Possíveis configurações de alinhamento individual de itens *flexbox*.

13.4.6 Representação Compacta de Propriedades

A propriedade `flex` é uma forma compacta de definir as propriedades `flex-grow`, `flex-shrink` e `flex-basis` em um único comando. Os valores devem ser definidos na ordem mencionada. Se apenas um valor for fornecido, ele será interpretado como `flex-grow`. Se dois valores forem fornecidos, eles serão interpretados como `flex-grow` e `flex-basis`, respectivamente. O código 13.13 mostra um exemplo de utilização da propriedade abreviada `flex`.

Neste exemplo 13.13, temos um contêiner *flexbox* com dois itens *flexbox*. O item da classe `item1` tem um fator de crescimento de 2, um fator de encolhimento de 1 e uma base *flexbox* de 100px. O item da classe `item2` tem um fator de crescimento de 1, um fator de encolhimento de 1 e uma base *flexbox* de

```
1  .container {  
2      display: flex;  
3  }  
4  
5  .item1 {  
6      flex: 2 1 100px;  
7  }  
8  
9  .item2 {  
10     flex: 1 1 200px;  
11 }
```

Exemplo de Código 13.13: Representações compactas de propriedades de itens *flexbox*.

200px. Dessa forma, o item da classe `item1` crescerá duas vezes mais rápido que o item da classe `item2` quando houver espaço disponível no contêiner, e ambos encolherão no mesmo ritmo quando o espaço do contêiner for reduzido.

13.5 Exemplos Práticos

Nesta seção, apresentaremos exemplos práticos de uso do *flexbox* para criar diferentes tipos de leiautes. Esses exemplos demonstram o poder e a flexibilidade do *flexbox* na construção de leiautes modernos e sofisticados.

13.5.1 Alinhamento Horizontal e Vertical

Um dos casos de uso mais comuns do *flexbox* é alinhar itens horizontal e verticalmente dentro de um contêiner, tarefa que não é tão simples de se fazer usando outras técnicas. No código 13.14, mostramos um exemplo de como fazer isso.

Neste exemplo 13.14, temos um contêiner *flexbox* com altura de 100vh (100% da altura da janela de visualização) com fundo cinza e um item *flexbox* dentro dele. Utilizamos as propriedades `justify-content` e `align-items` para alinhar o item no centro horizontal e vertical do contêiner. O item é estilizado com uma cor de fundo e bordas arredondadas. O código 13.14 produz o resultado mostrado na figura 13.9.

13.5.2 Criação de Uma Barra de Navegação

Outro exemplo prático de uso do *flexbox* é a criação de uma barra de navegação. O código 13.15 ilustra como criar uma barra de navegação simples.


```
1 <head>
2   <style>
3     .container {
4       display: flex;
5       justify-content: center;
6       align-items: center;
7       height: 100vh;
8       background-color: lightgray;
9     }
10
11    .item {
12      background-color: tomato;
13      border-radius: 4px;
14      height: 100px;
15      width: 100px;
16    }
17  </style>
18 <head>
19 <body>
20   <div class="container">
21     <div class="item">Item</div>
22   </div>
23 </body>
```

Exemplo de Código 13.14: Alinhamento horizontal e vertical com *flexbox*.

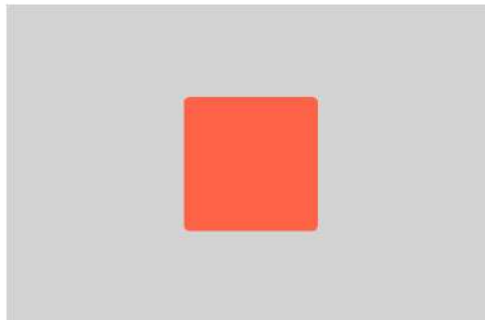


Figura 13.9: Centralizando um elemento horizontal e verticalmente em um contêiner com *flexbox*.

Neste exemplo 13.15, utilizamos o Flexbox para criar uma barra de navegação com um logotipo e um menu. A propriedade `justify-content` com o valor `space-between` garante que o logotipo e o menu sejam posicionados nos extremos opostos da barra de navegação. O menu também é um contêiner *flexbox*, e os itens do menu são estilizados com cores, espaçamento e sem sublinhado nos links. O código 13.16 mostra o código HTML para o exemplo 13.15.

```
1  .navbar {
2      display: flex;
3      justify-content: space-between;
4      background-color: #333;
5      padding: 1em;
6  }
7
8  .navbar .logo {
9      color: white;
10     font-size: 1.5em;
11     padding: .5em;
12 }
13
14 .navbar .menu {
15     display: flex;
16 }
17
18 .menu a {
19     color: white;
20     text-decoration: none;
21     padding: 1em;
22 }
```

Exemplo de Código 13.15: Barra de navegação com *flexbox*.

```
1  ...
2  <nav class="navbar">
3      <div class="logo">Logotipo</div>
4      <div class="menu">
5          <a href="#">Menu 1</a>
6          <a href="#">Menu 2</a>
7          <a href="#">Menu 3</a>
8          <a href="#">Menu 4</a>
9          <a href="#">Menu 5</a>
10     </div>
11 </nav>
12  ...
```

Exemplo de Código 13.16: Barra de navegação usando *flexbox*

A barra de navegação é responsiva e se ajusta automaticamente ao tamanho da janela de visualização. Quando a janela é pequena, o menu é colocado abaixo do logotipo. Quando a janela é grande, o menu é colocado ao lado do logotipo. O código 13.15 produz o resultado mostrado na figura 13.10.



Figura 13.10: Barra de navegação com *flexbox*.

13.5.3 Criação de Um Leiaute de Galeria de Imagens

O *flexbox* também pode ser usado para criar leiautes de galerias de imagens. No código 13.17, apresentamos o CSS de um exemplo de galeria responsiva usando o *flexbox*.

```
1  body {
2      margin: 0;
3      padding: 0;
4      box-sizing: border-box;
5  }
6
7  .galeria {
8      display: flex;
9      flex-wrap: wrap;
10     gap: 10px;
11     padding: 10px;
12 }
13
14 .imagem {
15     flex: 1 1 calc(33.333% - 20px);
16     background-position: center;
17     background-repeat: no-repeat;
18     background-size: cover;
19     min-height: 200px;
20     border: solid 1px gray;
21 }
```

Exemplo de Código 13.17: CSS da galeria de imagens com *flexbox*.

Neste exemplo 13.17, criamos o CSS de um leiaute de galeria de imagens responsivo usando *flexbox*. A propriedade *flex-wrap* com o valor *wrap* permite que os itens *flexbox* quebrem para a próxima linha quando não houver espaço suficiente na linha atual. A propriedade *gap* com o valor 10px adiciona espaçamento vertical e horizontal de 10px entre os itens *flexbox*. Cada item da galeria tem uma largura flexível, calculada com a função *calc()* e um espaçamento entre eles. A imagem de fundo é dimensionada e posicionada no centro de cada item. O código 13.18 mostra o código HTML para o exemplo 13.17.

```
1 ...  
2 <div class="galeria">  
3   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
4   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
5   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
6   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
7   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
8   <div class="imagem" style="background-image: url('ifes.svg');"></div>  
9 </div>  
10 ...
```

Exemplo de Código 13.18: HTML da galeria de imagens com *flexbox*.

A galeria é responsiva e se ajusta automaticamente ao tamanho da janela de visualização. Quando a janela é pequena, as imagens são exibidas em uma coluna. Quando a janela é grande, as imagens são exibidas em três colunas. O código 13.17 produz o resultado mostrado na figura 13.11.



Figura 13.11: Galeria de imagens com *flexbox*.

Ao trabalhar com *flexbox*, é essencial compreender as propriedades e como elas interagem entre si. Os exemplos apresentados nesta seção demonstram como o *flexbox* pode ser usado para criar leiautes responsivos e adaptáveis com facilidade. O domínio desses conceitos e propriedades permitirá criar interfaces modernas e funcionais para diversos projetos.

13.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 13.1 Crie um contêiner *flexbox* simples usando a propriedade `display`. ■

Exercício 13.2 Adicione três itens *flexbox* dentro de um contêiner *flexbox* e defina a propriedade `flex-grow` para 1 em cada item, garantindo que todos os itens ocupem igualmente o espaço disponível. ■

Exercício 13.3 Dentro de um contêiner *flexbox*, coloque cinco itens *flexbox*. Faça com que o terceiro item cresça duas vezes mais que os outros itens, usando a propriedade `flex-grow`. ■

Exercício 13.4 Utilize a propriedade `justify-content` em um contêiner *flexbox* para alinhar seus itens *flexbox* à direita. ■

Exercício 13.5 Alinhe verticalmente os itens de um contêiner *flexbox* no centro, usando a propriedade `align-items`. ■

Exercício 13.6 Crie um contêiner *flexbox* com três itens *flexbox* e organize-os em colunas, usando a propriedade `flex-direction`. ■

Exercício 13.7 Em um contêiner *flexbox* com cinco itens *flexbox*, utilize a propriedade `flex-wrap` para permitir que os itens sejam dispostos em várias linhas, conforme necessário. ■

Exercício 13.8 Em um contêiner *flexbox* com três itens *flexbox*, utilize a propriedade abreviada `flex` para definir `flex-grow`, `flex-shrink` e `flex-basis` em um único comando. ■

Exercício 13.9 Crie um contêiner *flexbox* com seis itens e faça com que a ordem de apresentação dos itens seja invertida, usando a propriedade `flex-direction`. ■

Exercício 13.10 Em um contêiner *flexbox* com três itens, defina o alinhamento individual de cada item usando a propriedade `align-self`. ■

Exercício 13.11 Crie um contêiner *flexbox* com quatro itens *flexbox* e defina um espaço entre os itens, usando a propriedade `gap`. ■

Exercício 13.12 Desenvolva um exemplo de um leiaute de cabeçalho com logotipo à esquerda e menu de navegação à direita, utilizando *flexbox*. ■

Exercício 13.13 Utilizando *flexbox*, crie um leiaute de três colunas com uma barra lateral à esquerda, uma área de conteúdo principal no centro e uma barra lateral à direita. ■

Exercício 13.14 Desenvolva um exemplo de um leiaute de rodapé com três seções, utilizando *flexbox* para distribuir igualmente o espaço entre elas. ■

Exercício 13.15 Crie um leiaute responsivo de *cards* de produtos usando *flexbox*, onde os *cards* se ajustam automaticamente com base no tamanho da tela. ■

Exercício 13.16 Utilizando *flexbox*, crie um exemplo de um formulário responsivo com campos alinhados verticalmente e um botão de envio alinhado à direita. ■

Exercício 13.17 Crie um exemplo de uma galeria de imagens responsiva utilizando *flexbox*, onde as imagens se ajustam automaticamente com base no tamanho da tela. ■

Exercício 13.18 Utilizando *flexbox*, desenvolva um leiaute de uma página de blog responsiva com uma área de conteúdo principal e uma barra lateral de *widgets*. ■

Exercício 13.19 Crie um menu de navegação horizontal responsivo com ícones e textos, utilizando *flexbox*. ■

Exercício 13.20 Desenvolva um exemplo de um leiaute de uma seção de depoimentos com imagens e textos, utilizando *flexbox* para ajustar o posicionamento e o espaçamento dos elementos. ■

13.7 Considerações Sobre o Capítulo

Neste capítulo, foram apresentados os conceitos básicos de *flexbox* e como ele pode ser usado para criar leiautes responsivos e adaptáveis. O *flexbox* é uma ferramenta poderosa e versátil que permite criar leiautes modernos e funcionais com facilidade. No capítulo seguinte, serão apresentados os conceitos básicos de leiaute baseado em *grid* CSS e como ele pode ser usado para criar leiautes complexos e responsivos.



14. Leiautes de Página com CSS *Grid*

O leiaute baseado em *Grid* é uma poderosa ferramenta CSS que permite a criação de leiautes de página complexos e responsivos de forma simples e eficiente. Com o aumento da diversidade de dispositivos e tamanhos de tela, é essencial que os desenvolvedores web possam criar páginas que se adaptem a essas variações, proporcionando uma experiência de usuário consistente e agradável. Neste capítulo, abordaremos os principais conceitos, propriedades e técnicas relacionadas ao leiaute baseado em CSS *Grid Layout*, permitindo ao leitor aprimorar suas habilidades no desenvolvimento de leiautes modernos e eficazes.

14.1 Conceitos Básicos

Antes de nos aprofundarmos nas propriedades específicas do *Grid*, é importante compreender seus conceitos fundamentais. O *Grid* é um sistema de leiaute bidimensional, no qual os elementos HTML são posicionados em linhas e colunas, facilitando a organização do conteúdo e proporcionando maior controle sobre o design responsivo.

Nesta seção, apresentaremos os conceitos básicos do *Grid*, incluindo a definição de um contêiner *Grid*, a manipulação de itens *Grid*, e como configurar linhas e colunas. Esses conceitos formam a base para o uso eficaz das propriedades e técnicas avançadas discutidas nas seções posteriores deste capítulo.

14.1.1 Definindo um Contêiner *Grid*

Para começar a utilizar o *Grid*, é necessário primeiro definir um contêiner *Grid*. O contêiner *Grid* é um elemento HTML que conterá todos os itens *Grid*, que por sua vez, serão posicionados nas linhas e colunas criadas pelo contêiner.

Para transformar um elemento HTML em um contêiner *Grid*, basta aplicar a propriedade `display` com o valor `grid`. O código 14.1 mostra um exemplo de como definir um contêiner *Grid*.

```
1 | .container {  
2 |     display: grid;  
3 | }
```

Exemplo de Código 14.1: Definindo um contêiner Grid.

Neste exemplo 14.1, a classe `.container` pode ser aplicada a um elemento HTML para transformá-lo em um contêiner *Grid*. A partir desse momento, qualquer elemento filho direto do contêiner *Grid* será tratado como um item *Grid* e poderá ser posicionado nas linhas e colunas criadas pelo contêiner.

Com a compreensão dos conceitos básicos do *Grid* e a definição de um contêiner *Grid*, estamos prontos para explorar as diversas propriedades que nos permitem criar leiautes complexos e responsivos, abordadas nas próximas seções deste capítulo.

14.2 Propriedades do Contêiner Grid

Nesta seção, abordaremos as propriedades específicas do contêiner *Grid*. Essas propriedades controlam a estrutura básica do leiaute em termos de linhas, colunas e espaçamento. Além disso, veremos como definir o comportamento de posicionamento automático dos itens *Grid*. Ao longo desta seção, apresentaremos exemplos práticos para ilustrar a aplicação de cada propriedade no desenvolvimento de leiautes responsivos.

14.2.1 Definindo as Linhas e Colunas

As propriedades `grid-template-columns` e `grid-template-rows` são usadas para definir o tamanho e a quantidade de colunas e linhas no contêiner *Grid*, respectivamente. Os valores podem ser definidos em unidades fixas (como `px`, `em`, `rem`) ou flexíveis (como `fr`, `%`, `auto`). O código 14.2 mostra um exemplo de como definir colunas e linhas em um contêiner *Grid*.

```
1 | .container {  
2 |     display: grid;  
3 |     grid-template-columns: 1fr 2fr 1fr;  
4 |     grid-template-rows: 100px auto 100px;  
5 | }
```

Exemplo de Código 14.2: Definindo colunas e linhas com `grid-template-columns` e `grid-template-rows`.

Neste exemplo 14.2, o contêiner *Grid* terá três colunas, sendo a primeira e a última com tamanho proporcional de 1 (1fr) e a coluna do meio com tamanho proporcional de 2 (2fr). Além disso, o contêiner terá três linhas, sendo a primeira e a última com altura fixa de 100px e a linha do meio com altura automática.

14.2.2 Definindo Áreas Nomeadas

A propriedade `grid-template-areas` é utilizada para nomear áreas específicas do layout *Grid*, facilitando a organização e posicionamento dos itens. O código 14.3 mostra um exemplo de como utilizar `grid-template-areas`.

```
1  .container {
2      display: grid;
3      grid-template-columns: repeat(3, 1fr);
4      grid-template-rows: auto;
5      grid-template-areas:
6          "header header header"
7          "sidebar main main"
8          "footer footer footer";
9  }
10
11  .header {
12      grid-area: header;
13  }
14
15  .sidebar {
16      grid-area: sidebar;
17  }
18
19  .main {
20      grid-area: main;
21  }
22
23  .footer {
24      grid-area: footer;
25  }
```

Exemplo de Código 14.3: Exemplo de uso de `grid-template-areas`.

Neste exemplo 14.3, definimos áreas nomeadas no contêiner *Grid* utilizando a propriedade `grid-template-areas`. Em seguida, atribuímos as áreas nomeadas aos itens do *Grid* utilizando a propriedade `grid-area`.

14.2.3 Propriedade Abreviada para Definição e Grid

A propriedade `grid-template` é uma propriedade abreviada que permite definir `grid-template-rows`, `grid-template-columns` e `grid-template-areas` em uma única declaração. O código 14.4 mostra um exemplo de uso dessa propriedade.

```
1  .container {
2      display: grid;
3      grid-template:
4          "header header header" 50px
5          "sidebar main main" 1fr
6          "footer footer footer" 30px
7      / 1fr 1fr 1fr;
8  }
9
10 .header {
11     grid-area: header;
12 }
13
14 .sidebar {
15     grid-area: sidebar;
16 }
17
18 .main {
19     grid-area: main;
20 }
21
22 .footer {
23     grid-area: footer;
24 }
```

Exemplo de Código 14.4: Exemplo de uso de `grid-template`.

Neste exemplo 14.4, definimos as áreas, linhas e colunas utilizando a propriedade `grid-template`. A declaração das áreas é seguida pelas dimensões das linhas e, após a barra "/", são definidas as dimensões das colunas.

14.2.4 Espaçamento Entre Linhas e Colunas

Nesta subseção, abordaremos as propriedades `grid-gap`, `column-gap` e `row-gap`, que são usadas para definir o espaço entre linhas e colunas no leiaute *Grid*. O código 14.5 mostra um exemplo de como utilizar essas propriedades.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      grid-gap: 10px;  
5      column-gap: 20px;  
6      row-gap: 30px;  
7  }
```

Exemplo de Código 14.5: Exemplo de uso de `grid-gap`, `column-gap` e `row-gap`.

Neste exemplo 14.5, a propriedade `grid-gap` define um espaço de 10 pixels entre linhas e colunas. As propriedades `column-gap` e `row-gap` têm prioridade sobre `grid-gap` e, no caso desse exemplo, definem um espaço de 20 pixels entre as colunas e 30 pixels entre as linhas.

14.2.5 Alinhamento Horizontal de Itens

A propriedade `justify-items` é utilizada para alinhar os itens do *Grid* horizontalmente dentro de suas respectivas células. O código 14.6 mostra um exemplo de como utilizar `justify-items`.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      justify-items: center;  
5  }
```

Exemplo de Código 14.6: Exemplo de uso de `justify-items`.

Neste exemplo 14.6, a propriedade `justify-items` é definida com o valor `center`, alinhando horizontalmente os itens do *Grid* no centro de suas células.

14.2.6 Alinhamento Vertical de Itens

A propriedade `align-items` é utilizada para alinhar os itens do *Grid* verticalmente dentro de suas respectivas células. O código 14.7 mostra um exemplo de como utilizar `align-items`.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      align-items: center;  
5  }
```

Exemplo de Código 14.7: Exemplo de uso de `align-items`.

Neste exemplo 14.7, a propriedade `align-items` é definida com o valor `center`, alinhando verticalmente os itens do *Grid* no centro de suas células.

14.2.7 Propriedade Abreviada para Alinhamento de Itens

A propriedade `place-items` é uma propriedade abreviada que permite definir `align-items` e `justify-items` em uma única declaração. O código 14.8 mostra um exemplo de como utilizar `place-items`.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      place-items: center;  
5  }
```

Exemplo de Código 14.8: Exemplo de uso de `place-items`.

Neste exemplo 14.8, a propriedade `place-items` é definida com o valor `center`, alinhando os itens do *Grid* tanto horizontalmente quanto verticalmente no centro de suas células.

14.2.8 Alinhamento Horizontal do *Grid*

A propriedade `justify-content` é utilizada para alinhar o *Grid* como um todo horizontalmente dentro do contêiner. O código 14.9 mostra um exemplo de como utilizar `justify-content`.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 100px);  
4      width: 100%;  
5      justify-content: center;  
6  }
```

Exemplo de Código 14.9: Exemplo de uso de `justify-content`.

Neste exemplo 14.9, a propriedade `justify-content` é definida com o valor `center`, alinhando o *Grid* horizontalmente no centro do contêiner.

14.2.9 Alinhamento Vertical do *Grid*

A propriedade `align-content` é usada para alinhar o espaço disponível ao longo do eixo do bloco quando há espaço extra no contêiner *Grid*. O código 14.10 mostra um exemplo de como utilizar a propriedade `align-content`.

Neste exemplo 14.10, definimos um contêiner *Grid* com três colunas e três linhas, cada uma com 100 pixels de altura. Como a altura total do contêiner é de 400 pixels, há um espaço extra de 100 pixels no eixo do bloco. A propriedade `align-content` com valor `start` alinha as linhas no início do contêiner, deixando o espaço extra no final.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      grid-template-rows: repeat(3, 100px);  
5      height: 400px;  
6      align-content: start;  
7  }
```

Exemplo de Código 14.10: Exemplo de uso de `align-content`.

14.2.10 Propriedade Abreviada para Alinhamento do Contêiner

A propriedade `place-content` é uma propriedade abreviada que permite definir `justify-content` e `align-content` simultaneamente. O código 14.11 mostra um exemplo de como utilizar a propriedade `place-content`.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(3, 1fr);  
4      grid-template-rows: repeat(3, 100px);  
5      width: 400px;  
6      height: 400px;  
7      place-content: center;  
8  }
```

Exemplo de Código 14.11: Exemplo de uso de `place-content`.

Neste exemplo 14.11, definimos um contêiner *Grid* com três colunas e três linhas, cada uma com 100 pixels de altura e largura. Como a largura e a altura total do contêiner são de 400 pixels, há um espaço extra de 100 pixels no eixo do bloco e no eixo inline. A propriedade `place-content` com valor `center` centraliza as linhas e colunas no contêiner, distribuindo o espaço extra igualmente antes e depois das linhas e colunas.

14.2.11 Geração Automática de Linhas e Colunas

As propriedades `grid-auto-columns` e `grid-auto-rows` são usadas para definir o tamanho das colunas e linhas geradas automaticamente. Isso ocorre quando os itens *Grid* são posicionados fora das áreas explicitamente definidas pelas propriedades `grid-template-columns` e `grid-template-rows`. O código 14.12 mostra um exemplo de como utilizar essas propriedades.

Neste exemplo 14.12, as colunas e linhas geradas automaticamente terão uma largura de 100px e uma altura de 50px, respectivamente.

```
1  .container {  
2      display: grid;  
3      grid-auto-columns: 100px;  
4      grid-auto-rows: 50px;  
5  }
```

Exemplo de Código 14.12: Definindo o tamanho das colunas e linhas automáticas com `grid-auto-columns` e `grid-auto-rows`.

14.2.12 Direção de Posicionamento dos Itens

A propriedade `grid-auto-flow` define a direção em que os itens *Grid* são posicionados automaticamente dentro do contêiner *Grid*. Os valores possíveis para essa propriedade são `row`, `column` ou `dense`. O valor padrão é `row`, que posiciona os itens *Grid* em linhas, preenchendo-as da esquerda para a direita e de cima para baixo. O valor `column` posiciona os itens em colunas, preenchendo-as de cima para baixo e da esquerda para a direita. Já o valor `dense` tenta preencher os espaços vazios no leiaute. O código 14.13 apresenta um exemplo de como utilizar a propriedade `grid-auto-flow`.

```
1  .container {  
2      display: grid;  
3      grid-auto-flow: column;  
4  }
```

Exemplo de Código 14.13: Definindo a direção de posicionamento automático dos itens *Grid* com `grid-auto-flow`.

Neste exemplo 14.13, os itens *Grid* serão posicionados automaticamente em colunas, preenchendo-as de cima para baixo e da esquerda para a direita.

14.3 Propriedades dos Itens Grid

Nesta seção, abordaremos as propriedades específicas dos itens *Grid*, que permitem controlar o posicionamento e o tamanho desses itens dentro do contêiner *Grid*. Essas propriedades são: `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` e `grid-area`.

14.3.1 Posicionamento no Eixo das Colunas

As propriedades `grid-column-start` e `grid-column-end` são usadas para posicionar um item *Grid* no eixo das colunas. Essas propriedades definem a linha de início e a linha de fim do item, respectivamente. O código 14.14 mostra um exemplo de como utilizar essas propriedades.

Neste exemplo 14.14, o item *Grid* começa na segunda linha vertical e termina na quarta linha vertical, ocupando duas colunas.

```
1  .item {  
2      grid-column-start: 2;  
3      grid-column-end: 4;  
4  }
```

Exemplo de Código 14.14: Posicionando um item *Grid* usando `grid-column-start` e `grid-column-end`.

14.3.2 Posicionamento no Eixo das Linhas

As propriedades `grid-row-start` e `grid-row-end` funcionam de forma semelhante às propriedades `grid-column-start` e `grid-column-end`, mas atuam no eixo das linhas. Essas propriedades definem a linha de início e a linha de fim do item no eixo das linhas. O código 14.15 mostra um exemplo de como utilizar essas propriedades.

```
1  .item {  
2      grid-row-start: 1;  
3      grid-row-end: 3;  
4  }
```

Exemplo de Código 14.15: Posicionando um item *Grid* usando `grid-row-start` e `grid-row-end`.

Neste exemplo 14.15, o item *Grid* começa na primeira linha horizontal e termina na terceira linha horizontal, ocupando duas linhas.

14.3.3 Propriedade Abreviada

A propriedade `grid-area` é uma forma abreviada de definir as propriedades `grid-row-start`, `grid-column-start`, `grid-row-end` e `grid-column-end` em uma única declaração. A sintaxe para essa propriedade é `grid-row-start / grid-column-start / grid-row-end / grid-column-end`. O código 14.16 mostra um exemplo de como utilizar a propriedade `grid-area`.

```
1  .item {  
2      grid-area: 1 / 2 / 3 / 4;  
3  }
```

Exemplo de Código 14.16: Posicionando um item *Grid* usando a propriedade `grid-area`.

Neste exemplo 14.16, o item *Grid* começa na primeira linha e na segunda coluna, termina na terceira linha e na quarta coluna, ocupando duas linhas e duas colunas.

14.4 Exemplos Práticos

Nesta seção, apresentaremos exemplos práticos do uso de *Grid* para criar diferentes tipos de leiautes. Vamos explorar como combinar as propriedades do contêiner e dos itens *Grid* que aprendemos nas seções anteriores para criar leiautes responsivos e adaptáveis.

14.4.1 Leiaute de Duas Colunas

Primeiro, vamos criar um leiaute simples de duas colunas com uma área de conteúdo e uma barra lateral. O código 14.17 mostra um exemplo de leiaute de duas colunas.

```
1  .container {
2      display: grid;
3      grid-template-columns: 2fr 1fr;
4      gap: 1rem;
5  }
6
7  .conteudo {
8      grid-column: 1 / 2;
9  }
10
11 .barra-lateral {
12     grid-column: 2 / 3;
13 }
```

Exemplo de Código 14.17: Leiaute de duas colunas.

Neste exemplo 14.17, definimos o contêiner *Grid* com duas colunas, onde a primeira coluna tem o dobro do tamanho da segunda coluna. Utilizamos a propriedade `gap` para adicionar um espaço entre as colunas. Em seguida, posicionamos a área de conteúdo na primeira coluna e a barra lateral na segunda coluna.

14.4.2 Leiaute de Cabeçalho, Conteúdo e Rodapé

Agora, vamos criar um leiaute com cabeçalho, conteúdo e rodapé. O código 14.18 mostra um exemplo de leiaute com cabeçalho, conteúdo e rodapé.

Neste exemplo 14.18, definimos o contêiner *Grid* com uma coluna e três linhas, onde a primeira e a última linhas têm altura automática e a linha do meio ocupa todo o espaço disponível. Utilizamos a propriedade `gap` para adicionar um espaço entre as linhas. Em seguida, posicionamos o cabeçalho na primeira linha, a área de conteúdo na segunda linha e o rodapé na terceira linha.


```
1  .container {
2      display: grid;
3      grid-template-columns: 1fr;
4      grid-template-rows: auto 1fr auto;
5      gap: 1rem;
6  }
7
8  .cabecalho {
9      grid-row: 1 / 2;
10 }
11
12 .conteudo {
13     grid-row: 2 / 3;
14 }
15
16 .rodape {
17     grid-row: 3 / 4;
18 }
```

Exemplo de Código 14.18: Leiaute de cabeçalho, conteúdo e rodapé.

14.5 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim você absorverá muito mais o conteúdo estudado.

Exercício 14.1 Aplique a propriedade `display` com valor `grid` em um elemento `<div>` para criar um contêiner de grid. ■

Exercício 14.2 Utilize a propriedade `grid-template-columns` para definir três colunas com larguras iguais no contêiner de grid. ■

Exercício 14.3 Aplique a propriedade `grid-template-rows` para criar duas linhas com alturas diferentes no contêiner de grid. ■

Exercício 14.4 Utilize a propriedade `grid-column` para posicionar um elemento `<div>` na segunda coluna do grid. ■

Exercício 14.5 Aplique a propriedade `grid-row` para posicionar um elemento `<p>` na primeira linha do grid. ■

Exercício 14.6 Utilize a propriedade `gap` para definir espaçamento de 10px entre as linhas e colunas do grid. ■

Exercício 14.7 Aplique a propriedade `grid-template-areas` para criar áreas nomeadas no container de grid e posicionar os elementos `<header>`, `<nav>`, `<main>` e `<footer>` de forma adequada (cabeçalho, barra de navegação lateral, conteúdo principal e rodapé, respectivamente). ■

Exercício 14.8 Utilize a propriedade `grid-auto-rows` para definir uma altura mínima para as linhas adicionais criadas automaticamente. ■

Exercício 14.9 Aplique a propriedade `grid-auto-columns` para definir uma largura mínima para as colunas adicionais criadas automaticamente. ■

Exercício 14.10 Utilize a propriedade `grid-auto-flow` com valor `row` para preencher automaticamente as lacunas no grid, criando novas linhas, se necessário. ■

14.6 Considerações Sobre o Capítulo

Neste capítulo, vimos como utilizar o *Grid* para criar leiautes responsivos e adaptáveis. O *Grid* é uma ferramenta poderosa que pode ser utilizada para criar leiautes complexos com poucas linhas de código. No entanto, o *Grid* não é a solução para todos os problemas de layout. É importante entender quando utilizar o *Grid* e quando utilizar outras ferramentas, como *Flexbox*, vista no capítulo anterior. No capítulo seguinte, veremos como criar leiautes responsivos usando *media queries*.



15. Responsividade

A responsividade é uma abordagem no design e desenvolvimento web que visa criar sites e aplicações capazes de se adaptar automaticamente a diferentes dispositivos e tamanhos de tela. Isso é alcançado através do uso de técnicas específicas, como o *media queries* em CSS e o *viewport* em HTML. O código 15.1 mostra um exemplo de uma *media query* simples.

```
1  @media screen and (max-width: 768px) {  
2      body {  
3          background-color: blue;  
4      }  
5  }
```

Exemplo de Código 15.1: Exemplo de *media query*.

Neste exemplo 15.1, utilizamos uma *media query* para verificar se a largura da tela do dispositivo é menor ou igual a 768 pixels. Se essa condição for verdadeira, a cor de fundo do elemento `body` será alterada para azul. Essa técnica permite que os estilos sejam aplicados de acordo com as características do dispositivo, como tamanho da tela, resolução e orientação.

A responsividade é crucial no desenvolvimento web atual, pois os usuários acessam a internet por meio de uma ampla variedade de dispositivos, como *smartphones*, *tablets* e *desktops*. Um site ou aplicação que não seja responsivo pode oferecer uma experiência ruim ao usuário, o que pode levar à perda de clientes e uma classificação mais baixa nos mecanismos de busca. O código 15.2 mostra um exemplo de como configurar a *viewport* em HTML.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta name="viewport" content="width=device-width, initial-scale=1.0">
5      <title>Exemplo de Responsividade</title>
6  </head>
7  <body>
8      <h1>Olá, mundo responsivo!</h1>
9  </body>
10 </html>
```

Exemplo de Código 15.2: Exemplo de viewport em HTML.

Neste exemplo 15.2, incluímos uma meta tag no cabeçalho do documento HTML, com o atributo name definido como "viewport" e o atributo content configurado para "width=device-width, initial-scale=1.0". Isso informa ao navegador para ajustar a largura do viewport à largura do dispositivo e definir o nível inicial de zoom como 1.0. Essa abordagem é fundamental para garantir que o conteúdo se adapte corretamente a diferentes dispositivos e tamanhos de tela.

15.1 Princípios Básicos da Responsividade

Nesta seção, exploraremos os princípios básicos da responsividade, que são fundamentais para criar sites e aplicações que se ajustam automaticamente a diferentes dispositivos e tamanhos de tela. Os principais conceitos abordados nesta seção incluem design fluido, imagens flexíveis, *media queries* e unidades relativas. Esses conceitos são apresentados nas subseções a seguir.

15.1.1 Design Fluido

O design fluido é uma técnica que permite que os elementos de uma página se ajustem proporcionalmente às dimensões da viewport. Isso é alcançado através do uso de unidades relativas, como porcentagens, ao invés de unidades fixas, como pixels. O código 15.3 mostra um exemplo de design fluido com CSS.

```
1  .container {
2      width: 100%;
3      max-width: 1200px;
4      margin: 0 auto;
5  }
```

Exemplo de Código 15.3: Exemplo de design fluido.

Neste exemplo 15.3, criamos uma classe `.container` com uma largura de 100% e uma largura máxima de 1200 pixels. Isso permite que o elemento se ajuste automaticamente à largura da `viewport`, mantendo uma largura máxima para evitar que o conteúdo se estenda demais em telas muito largas.

15.1.2 Imagens Flexíveis

Imagens flexíveis são imagens que se adaptam automaticamente ao tamanho da `viewport`, garantindo que não sejam maiores do que o contêiner que as contém. O código 15.4 mostra um exemplo de como tornar as imagens flexíveis com CSS.

```
1  img {  
2      max-width: 100%;  
3      height: auto;  
4  }
```

Exemplo de Código 15.4: Exemplo de imagens flexíveis.

Neste exemplo 15.4, configuramos a largura máxima das imagens para 100% e a altura para `auto`. Isso garante que a imagem sempre se ajuste à largura do contêiner, mantendo suas proporções originais.

15.1.3 Media Queries

As *media queries* permitem aplicar estilos específicos com base nas características do dispositivo, como tamanho de tela, resolução e orientação. Isso possibilita criar designs responsivos que se adaptam a diferentes dispositivos e contextos de uso. O código 15.5 mostra um exemplo de uma *media query* que aplica estilos diferentes para telas pequenas e grandes.

```
1  @media screen and (max-width: 600px) {  
2      body {  
3          font-size: 14px;  
4      }  
5  }  
6  
7  @media screen and (min-width: 601px) {  
8      body {  
9          font-size: 18px;  
10     }  
11 }
```

Exemplo de Código 15.5: Exemplo de media query para telas pequenas e grandes.

Neste exemplo 15.5, utilizamos duas *media queries* para aplicar tamanhos de fonte diferentes com base na largura da tela. Para telas com largura máxima de 600 pixels, a fonte do elemento `body` será de 14 pixels, enquanto para telas com largura mínima de 601 pixels, a fonte será de 18 pixels. Essa abordagem permite adaptar a aparência do conteúdo às características específicas do dispositivo em uso.

15.1.4 Unidades Relativas

Unidades relativas são fundamentais para a criação de designs responsivos, pois permitem que as dimensões dos elementos se ajustem proporcionalmente às dimensões da `viewport` ou de outros elementos de referência. Algumas unidades relativas comuns incluem porcentagens (%), `ems` (`em`), `rems` (`rem`) e unidades de visualização (`vw`, `vh`, `vmin`, `vmax`). O código 15.6 mostra um exemplo de como usar unidades relativas em CSS.

```
1  body {  
2      font-size: 1rem;  
3      line-height: 1.5;  
4  }  
5  
6  h1 {  
7      font-size: 2.5em;  
8      margin-bottom: 0.5em;  
9  }  
10  
11  .container {  
12      width: 90%;  
13      max-width: 80rem;  
14      margin: 0 auto;  
15  }
```

Exemplo de Código 15.6: Exemplo de unidades relativas.

Neste exemplo 15.6, utilizamos várias unidades relativas para dimensionar os elementos da página. No elemento `body`, definimos o tamanho da fonte como `1rem`, o que equivale ao tamanho da fonte base do navegador (geralmente 16 pixels). O espaçamento entre as linhas é configurado como 1.5, multiplicando o tamanho da fonte. No elemento `h1`, o tamanho da fonte é definido como 2.5 vezes o tamanho da fonte do elemento pai, e a margem inferior como 0.5 vezes o tamanho da fonte. Por fim, no elemento `.container`, a largura é definida como 90% da largura da `viewport`, com uma largura máxima de `80rem` (80 vezes o tamanho da fonte base).

15.2 Responsividade no HTML

Nesta seção, abordaremos os elementos e técnicas fundamentais para criar uma estrutura de HTML responsivo, garantindo que seu site ou aplicação seja otimizado para diferentes dispositivos e tamanhos de tela. Os principais tópicos abordados incluem a metatag `viewport`, elementos e atributos HTML para responsividade e o uso de frameworks CSS para facilitar a responsividade. Esses tópicos são explorados nas subseções a seguir.

15.2.1 Metatag de Responsividade

A metatag `viewport` é um elemento crucial para garantir que seu site seja exibido corretamente em diferentes dispositivos. Ela informa ao navegador como dimensionar e renderizar o conteúdo, adaptando-o ao tamanho da tela do dispositivo. O código 15.7 mostra um exemplo de como usar a metatag `viewport` em seu arquivo HTML.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Meu site responsivo</title>
7  </head>
8  <body>
9      <!-- Seu conteúdo aqui -->
10 </body>
11 </html>
```

Exemplo de Código 15.7: Exemplo de metatag `viewport`.

Neste exemplo 15.7, a metatag `viewport` é colocada dentro da seção `<head>` do documento HTML. Ela define o atributo `content` com os valores `"width=device-width, initial-scale=1.0"`, que informam ao navegador para dimensionar o conteúdo com base na largura do dispositivo e para definir o zoom inicial como 1.0 (sem zoom).

15.2.2 Elementos e Atributos HTML para Responsividade

Além da metatag `viewport`, existem outros elementos e atributos HTML que podem ser usados para melhorar a responsividade do seu site. O código 15.8 mostra um exemplo de como usar elementos e atributos HTML responsivos.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <!-- ... -->
5  </head>
6  <body>
7      <header>
8          <!-- Cabeçalho do site -->
9      </header>
10     <nav>
11         <!-- Navegação do site -->
12     </nav>
13     <main>
14         <article>
15             <!-- Conteúdo principal do site -->
16         </article>
17     </main>
18     <aside>
19         <!-- Conteúdo secundário do site -->
20     </aside>
21     <footer>
22         <!-- Rodapé do site -->
23     </footer>
24 </body>
25 </html>
```

Exemplo de Código 15.8: Exemplo de elementos e atributos HTML para responsividade.

Neste exemplo 15.8, utilizamos elementos HTML semânticos, como `<header>` (cabeçalho), `<nav>` (barra de navegação), `<main>` (região de conteúdo), `<footer>` (rodapé), `<aside>` (barra lateral) e `<article>` (conteúdo), para estruturar o conteúdo do site. Esses elementos não apenas melhoraram a acessibilidade e a organização do código, mas também facilitam a aplicação de estilos CSS responsivos.

15.2.3 Uso de Frameworks CSS para Facilitar a Responsividade

Frameworks CSS são bibliotecas de código prontas que fornecem uma base sólida para a criação de sites responsivos. Eles oferecem uma grade flexível, componentes prontos para uso e estilos padronizados que podem ser facilmente personalizados para atender às suas necessidades. O uso de frameworks CSS pode economizar tempo e esforço, além de garantir uma experiência de usuário consistente em diferentes dispositivos. O código 15.9 mostra um exemplo de como usar o Bootstrap 5, um popular framework CSS, em seu arquivo HTML.


```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <title>Bootstrap demo</title>
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
8     rel="stylesheet" crossorigin="anonymous">
9 </head>
10 <body>
11   <div class="container">
12     <div class="row">
13       <div class="col-md-8 bg-primary p-5">
14         <!-- Conteúdo principal do site -->
15       </div>
16       <div class="col-md-4 bg-secondary p-5">
17         <!-- Conteúdo secundário do site -->
18       </div>
19     </div>
20   </div>
21   <script crossorigin="anonymous"
22     src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js">
23   </script>
24 </body>
25 </html>
```

Exemplo de Código 15.9: Exemplo de uso do Bootstrap.

Neste exemplo 15.9, incluímos o arquivo CSS do Bootstrap 5 no <head> do documento HTML usando a tag <link>. No <body>, utilizamos as classes `.container`, `.row` e `.col-md-*` do próprio Bootstrap para criar uma grade responsiva. Foram adicionadas as classes `bg-*` para definir uma cor de fundo e `p-5` para adicionar margem interna. O arquivo JavaScript necessário para o funcionamento de alguns componentes do Bootstrap são incluídos antes do fechamento da tag <body>. Entretanto, para recursos de responsividade, esse arquivo Javascript é dispensável.

15.3 Implementando CSS Responsivo

Nesta seção, discutiremos como implementar CSS responsivo para criar sites e aplicações que se adaptam perfeitamente a diferentes tamanhos de tela e dispositivos. Abordaremos a criação de grids flexíveis, o tratamento de imagens e vídeos responsivos, a tipografia responsiva e o uso de *media queries* para adaptar estilos. Além disso, compararemos as abordagens *mobile-first* e *desktop-first* no desenvolvimento responsivo.

15.3.1 Grids Flexíveis

Grids flexíveis são fundamentais para criar leiautes responsivos. Eles permitem que você organize o conteúdo em colunas que se ajustam automaticamente de acordo com o tamanho da tela. O código 15.10 mostra um exemplo de grid flexível usando CSS.

```
1  .container {
2      display: flex;
3      flex-wrap: wrap;
4  }
5
6  .column {
7      flex-basis: 100%;
8      box-sizing: border-box;
9      padding: 10px;
10 }
11
12 @media (min-width: 768px) {
13     .column {
14         flex-basis: 50%;
15     }
16 }
17
18 @media (min-width: 1024px) {
19     .column {
20         flex-basis: 33.33%;
21     }
22 }
```

Exemplo de Código 15.10: Exemplo de grid flexível.

Neste exemplo 15.10, definimos uma classe `.container` com as propriedades `display: flex` e `flex-wrap: wrap` para criar um contêiner flexível que permita que as colunas se ajustem e se envolvam conforme necessário. A classe `.column` é usada para definir o tamanho e o espaçamento das colunas. Utilizamos *media queries* para ajustar o tamanho das colunas com base na largura da tela.

15.3.2 Imagens e Vídeos Responsivos

Imagens e vídeos responsivos são cruciais para garantir que o conteúdo multimídia seja exibido corretamente em diferentes tamanhos de tela. O código 15.11 mostra como criar imagens responsivas usando CSS.

```
1  img {  
2      max-width: 100%;  
3      height: auto;  
4  }
```

Exemplo de Código 15.11: Exemplo de imagem responsiva.

Neste exemplo 15.11, aplicamos a propriedade `max-width: 100%` às imagens para garantir que elas não excedam a largura do contêiner. A propriedade `height: auto` garante que a altura seja ajustada proporcionalmente, mantendo a proporção original da imagem.

Para vídeos, você pode usar a técnica de "embrulhar" o vídeo em um contêiner e aplicar estilos responsivos ao contêiner. O código 15.12 mostra um exemplo de vídeo responsivo.

```
1  .video-container {  
2      position: relative;  
3      padding-bottom: 56.25%;  
4      height: 0;  
5      overflow: hidden;  
6  }  
7  
8  video {  
9      position: absolute;  
10     top: 0;  
11     left: 0;  
12     width: 100%;  
13     height: 100%;  
14 }
```

Exemplo de Código 15.12: Exemplo de vídeo responsivo.

Neste exemplo 15.12, criamos um contêiner com proporções fixas usando as propriedades `height: 0` e `padding-bottom: 56.25%`. A classe `.video-container` tem a propriedade `overflow: hidden` para garantir que o conteúdo não seja exibido fora dos limites do contêiner. Em seguida, aplicamos as propriedades `position: absolute`, `top: 0`, `left: 0`, `width: 100%` e `height: 100%` ao elemento `video` para preencher completamente o contêiner, mantendo a proporção correta.

15.3.3 Tipografia Responsiva

A tipografia responsiva é uma parte essencial do design responsivo, pois permite que o texto seja facilmente legível em diferentes dispositivos e tamanhos de tela. O código 15.13 mostra como aplicar tipografia responsiva usando unidades relativas e *media queries*.

```
1  html {
2      font-size: 16px;
3  }
4
5  h1 {
6      font-size: 2rem;
7  }
8
9  p {
10     font-size: 1rem;
11     line-height: 1.5;
12 }
13
14 @media (min-width: 768px) {
15     html {
16         font-size: 18px;
17     }
18 }
19
20 @media (min-width: 1024px) {
21     html {
22         font-size: 20px;
23     }
24 }
```

Exemplo de Código 15.13: Exemplo de tipografia responsiva.

Neste exemplo 15.13, definimos o tamanho da fonte básica no elemento `html` e usamos unidades relativas, como `rem`, para definir o tamanho da fonte dos elementos `h1` e `p`. Usamos *media queries* para ajustar o tamanho da fonte básica com base na largura da tela, o que afeta o tamanho de todos os elementos que usam unidades relativas.

15.3.4 Uso de *Media Queries* para Adaptar Estilos

Media queries são fundamentais para adaptar estilos com base nas características do dispositivo, como largura da tela, altura da tela, resolução e orientação. O código 15.14 mostra como usar *media queries* para aplicar estilos diferentes para dispositivos com larguras de tela variadas.

Vale mencionar que o código 15.14 foca especificamente em tablets e telas de computadores de mesa, mas poderíamos incluir quantas *media queries* fossem necessárias para abrangermos as mais diversas configurações de tela disponíveis para visualização do conteúdo.

```
1  /* Estilos básicos para dispositivos móveis */
2  body {
3      background-color: lightgray;
4      color: black;
5  }
6
7  /* Estilos para tablets e dispositivos com largura mínima de 768px */
8  @media (min-width: 768px) {
9      body {
10         background-color: white;
11         color: black;
12     }
13 }
14
15 /* Estilos para desktops e dispositivos com largura mínima de 1024px */
16 @media (min-width: 1024px) {
17     body {
18         background-color: darkgray;
19         color: white;
20     }
21 }
```

Exemplo de Código 15.14: Exemplo de uso de media queries.

Neste exemplo 15.14, aplicamos diferentes cores de fundo e cores de texto para dispositivos móveis, tablets e desktops usando *media queries*. É muito comum configurar também tamanhos de fonte, margens de página, alturas e larguras das seções da página, entre outras coisas.

15.3.5 Mobile-first vs. Desktop-first

Ao desenvolver sites e aplicações responsivas, duas abordagens comuns são a *mobile-first* e a *desktop-first*. A abordagem *mobile-first* envolve o design e o desenvolvimento com foco nos dispositivos móveis, adaptando-se progressivamente a telas maiores. A abordagem *desktop-first* começa com o design e desenvolvimento para telas grandes e desktops e, em seguida, adapta-se progressivamente a telas menores e dispositivos móveis.

A abordagem *mobile-first* tem várias vantagens, incluindo a priorização da experiência do usuário em dispositivos móveis, que geralmente têm restrições de largura de banda e recursos, e a simplificação do design para que ele seja mais fácil de adaptar a telas maiores. No entanto, a abordagem *desktop-first* pode ser mais adequada para projetos que exigem uma experiência de usuário mais avançada em telas maiores, com recursos e funcionalidades adicionais.

Ao escolher entre *mobile-first* e *desktop-first*, é importante considerar o público-alvo, os objetivos do projeto e os recursos disponíveis. Independentemente da abordagem escolhida, o uso de técnicas de CSS responsivo, como grades flexíveis, imagens e vídeos responsivos, tipografia responsiva e *media queries*, é fundamental para criar uma experiência de usuário consistente e adaptável em todos os dispositivos e tamanhos de tela.

15.4 Técnicas Avançadas de Responsividade

Nesta seção, exploraremos técnicas avançadas de responsividade que podem ser usadas para criar designs mais sofisticados e flexíveis. Essas técnicas incluem a utilização de *Flexbox* e *CSS Grid* para criar layouts mais avançados, aprimoramento de imagens responsivas com *srcset* e *picture*, e a implementação de técnicas de carregamento condicional para otimizar o desempenho em diferentes dispositivos e condições de rede. A seguir, apresentaremos as subseções que abordam cada uma dessas técnicas avançadas.

15.4.1 Layouts com *Flexbox*

O *Flexbox* é uma poderosa ferramenta de layout do CSS que permite criar layouts responsivos e flexíveis com facilidade. Ele simplifica a organização de elementos dentro de um contêiner, permitindo que eles se ajustem automaticamente ao tamanho disponível. O código 15.15 mostra um exemplo de um layout simples usando *Flexbox*.

```
1  .container {  
2      display: flex;  
3  }  
4  
5  .item {  
6      flex: 1;  
7  }
```

Exemplo de Código 15.15: Exemplo de layout com *Flexbox*.

Neste exemplo 15.15, o contêiner possui a propriedade `display` definida como `flex`, o que o transforma em um contêiner *Flexbox*. Os itens dentro do contêiner têm a propriedade `flex` definida como 1, o que permite que eles cresçam e encolham proporcionalmente, ocupando todo o espaço disponível no contêiner.

15.4.2 Layouts com CSS Grid

O CSS Grid é outra técnica avançada de layout que oferece um maior controle sobre a organização dos elementos na página. Com o CSS Grid, é possível criar layouts complexos e responsivos usando linhas e colunas. O código 15.16 mostra um exemplo de um layout simples usando CSS Grid.

```
1  .container {  
2      display: grid;  
3      grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));  
4  }  
5  
6  .item {  
7      grid-column: span 1;  
8  }
```

Exemplo de Código 15.16: Exemplo de layout com CSS Grid.

Neste exemplo 15.16, o contêiner possui a propriedade `display` definida como `grid`, transformando-o em um contêiner de grade. A propriedade `grid-template-columns` define a quantidade de colunas e seu tamanho mínimo e máximo, criando um layout responsivo. Os itens dentro do contêiner ocupam uma coluna cada.

15.4.3 Imagens Responsivas

Para garantir que as imagens sejam carregadas de forma responsiva e otimizada, é possível usar o elemento HTML `picture` e o atributo `srcset`. O atributo `srcset` permite especificar diferentes versões de uma imagem para diferentes tamanhos de tela e densidades de pixel, enquanto a tag `picture` permite definir várias fontes de imagem e tipos de mídia. O código 15.17 mostra um exemplo de uso do `srcset` e `picture`.

```
1  <picture>  
2      <source media="(min-width: 768px)" srcset="large.jpg">  
3      <source media="(min-width: 480px)" srcset="medium.jpg">  
4        
5  </picture>
```

Exemplo de Código 15.17: Exemplo de uso de `srcset` e `picture`.

Neste exemplo 15.17, o elemento `picture` contém vários elementos `source` com atributos `media` que definem as condições de largura mínima da tela. A imagem `large.jpg` será exibida em telas com largura mínima de 768 pixels, e a imagem `medium.jpg` será exibida em telas com largura mínima de 480 pixels. O elemento `img` é usado como padrão e possui o atributo `srcset` com as versões padrão e de alta densidade para a imagem `small.jpg`.

15.4.4 Técnicas de Carregamento Condicional

O carregamento condicional é uma técnica que permite carregar recursos, como imagens, scripts e folhas de estilo, apenas quando necessário. Isso pode melhorar o desempenho e a experiência do usuário em diferentes dispositivos e condições de rede.

Um exemplo de carregamento condicional é o uso do atributo `media` nas tags `link` para folhas de estilo. O código 15.18 mostra como carregar uma folha de estilo específica para dispositivos com largura mínima de 768 pixels.

```
1 <link rel="stylesheet" href="default.css">  
2 <link rel="stylesheet" href="large-screens.css" media="(min-width: 768px)">
```

Exemplo de Código 15.18: Exemplo de carregamento condicional.

Neste exemplo 15.18, a folha de estilo `default.css` é carregada para todos os dispositivos, enquanto a folha de estilo `large-screens.css` é carregada apenas para dispositivos com largura mínima de 768 pixels, graças ao atributo `media`.

15.5 Testando e Otimizando a Responsividade

Ao desenvolver um site responsivo, é fundamental garantir que ele funcione corretamente em diversos dispositivos e navegadores. Esta seção abordará técnicas e ferramentas para testar e otimizar a responsividade do seu projeto, incluindo testes em dispositivos reais e melhorias na performance e acessibilidade.

15.5.1 Ferramentas de Teste e Simulação

Ferramentas de teste e simulação são essenciais para garantir que seu design responsivo funcione conforme o esperado. A listagem a seguir mostra um exemplo de como usar o Google Chrome DevTools para simular diferentes dispositivos e tamanhos de tela.

- a) Abra o Google Chrome.
- b) Navegue até o site que deseja testar.
- c) Pressione F12 para abrir o DevTools.
- d) Clique no ícone de "Toggle device toolbar" (alternar barra de ferramentas do dispositivo) ou pressione Ctrl + Shift + M.
- e) Selecione um dispositivo predefinido ou insira dimensões personalizadas para a tela.

Além do DevTools, outras ferramentas e navegadores também oferecem recursos semelhantes para testar a responsividade.

15.5.2 Testes em Dispositivos Reais

Embora as ferramentas de simulação sejam úteis, nada substitui o teste em dispositivos reais. Isso permite verificar o desempenho, a aparência e a usabilidade do site em diferentes dispositivos e sistemas operacionais. Para testar em dispositivos reais, considere criar um ambiente de teste local ou usar serviços de teste em nuvem que permitam acessar seu site em diversos dispositivos e navegadores.

15.5.3 Otimização de Performance

A performance é um aspecto crítico da responsividade. Algumas práticas recomendadas incluem minificar arquivos CSS e JavaScript, otimizar imagens e usar técnicas de carregamento condicional, conforme mencionado na seção 5.4.

15.5.4 Acessibilidade e Inclusão Digital

Acessibilidade e inclusão digital são aspectos fundamentais para garantir que seu site seja acessível a todos os usuários, independentemente de suas habilidades e necessidades. Algumas práticas recomendadas incluem:

- a) Usar uma estrutura semântica e clara no HTML;
- b) Garantir que o site possa ser navegado apenas com o teclado;
- c) Usar contraste adequado de cores entre texto e fundo;
- d) Fornecer descrições de texto alternativo para imagens e outros elementos não textuais.

15.6 Exercícios Propostos

Essa série de exercícios envolve os conceitos abordados neste capítulo e também **pode demandar alguma pesquisa**. Reserve um tempo e um local adequados para fazer os exercícios sem distrações. Assim, você absorverá muito mais o conteúdo estudado.

Para os exercícios deste capítulo, considere os códigos das 3 páginas HTML inseridas entre os exercícios. Alguns exercícios se aplicam a todas as páginas e outros se aplicam somente a uma página específica.

Exercício 15.1 Aplique a propriedade `max-width` no elemento `<body>` das páginas HTML para limitar sua largura máxima em 1200px. ■

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Exemplo 1</title>
7  </head>
8  <body>
9      <header>
10         <h1>Exemplo de Página 1</h1>
11     </header>
12     <nav>
13         <a href="#">Link 1</a>
14         <a href="#">Link 2</a>
15         <a href="#">Link 3</a>
16     </nav>
17     <main>
18         <p>Conteúdo da página.</p>
19     </main>
20     <footer>
21         <p>&copy; Todos os direitos reservados.</p>
22     </footer>
23 </body>
24 </html>
```

Exemplo de Código 15.19: Código da página 1.

Exercício 15.2 Utilize a propriedade `box-sizing` com valor `border-box` em todos os elementos das páginas HTML para incluir a borda e o padding no tamanho total do elemento. ■

Exercício 15.3 Aplique a propriedade `flex-wrap` com valor `wrap` no elemento `<nav>` das páginas HTML para permitir que os links se ajustem em várias linhas, se necessário. ■

Exercício 15.4 Aplique a propriedade `display` com valor `flex` no elemento `<nav>` das páginas HTML para criar um layout flexível. ■

Exercício 15.5 Utilize uma media query para aplicar a propriedade `font-size` com valor `18px` no elemento `<body>` das páginas HTML quando a largura da tela for maior que `768px`. ■

Exercício 15.6 Aplique uma media query para alterar a propriedade `flex-direction` do elemento `<nav>` das páginas HTML para `column` quando a largura da tela for menor que `480px`. ■

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Exemplo 2</title>
7  </head>
8  <body>
9      <header>
10         <h1>Exemplo de Página 2</h1>
11     </header>
12     <nav>
13         <a href="#">Link 1</a>
14         <a href="#">Link 2</a>
15         <a href="#">Link 3</a>
16     </nav>
17     <section>
18         <article>
19             <h2>Título do artigo</h2>
20             <p>Conteúdo do artigo.</p>
21         </article>
22     </section>
23     <footer>
24         <p>&copy; Todos os direitos reservados.</p>
25     </footer>
26 </body>
27 </html>
```

Exemplo de Código 15.20: Código da página 2.

Exercício 15.7 Utilize uma media query para ocultar o elemento `<nav>` das páginas HTML quando a largura da tela for menor que 320px, usando a propriedade `display` com valor `none`. ■

Exercício 15.8 Aplique uma media query para alterar a propriedade `grid-template-columns` do container de grid da página HTML 3 para ter duas colunas com larguras iguais quando a largura da tela for menor que 600px. ■

Exercício 15.9 Utilize uma media query para aplicar a propriedade `width` com valor 100 ■

Exercício 15.10 Aplique uma media query para alterar a propriedade `flex-direction` do container flexível da página HTML 2 para `column` quando a largura da tela for menor que 720px. ■

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Exemplo 3</title>
7  </head>
8  <body>
9      <header>
10         <h1>Exemplo de Página 3</h1>
11     </header>
12     <nav>
13         <a href="#">Link 1</a>
14         <a href="#">Link 2</a>
15         <a href="#">Link 3</a>
16     </nav>
17     <main>
18         <p>Conteúdo da página.</p>
19         <form>
20             <label for="nome">Nome:</label>
21             <input type="text" id="nome" name="nome">
22             <button type="submit">Enviar</button>
23         </form>
24     </main>
25     <footer>
26         <p>&copy; Todos os direitos reservados.</p>
27     </footer>
28 </body>
29 </html>
```

Exemplo de Código 15.21: Código da página 3.

15.7 Considerações Sobre o Capítulo

Ao longo deste capítulo, exploramos diversos aspectos do desenvolvimento web responsivo, incluindo fundamentos, técnicas avançadas, otimizações e testes. Em resumo, o desenvolvimento web responsivo é uma habilidade essencial para os profissionais da área. Ao aplicar as técnicas e estratégias discutidas neste capítulo, você estará preparado para enfrentar os desafios e oportunidades que a responsividade traz, garantindo que seu site seja bem-sucedido no cenário digital atual e no futuro.