3
CSS: Dando Estilo e Vida às Páginas Web

Se no capítulo anterior construímos o esqueleto da nossa página com HTML, agora vamos aprender a vesti-lo. Bem-vindo ao mundo do **CSS (*Cascading Style Sheets*, ou Folhas de Estilo em Cascata)**! O CSS é a linguagem que usamos para controlar toda a apresentação visual de um documento HTML: cores, fontes, espaçamentos, layouts e até animações.

Enquanto o HTML se preocupa com a **estrutura** e o **significado** do conteúdo, o CSS cuida da **estética** e da **forma**. Separar essas duas responsabilidades é uma das práticas mais importantes do desenvolvimento web moderno.

3.1 Nosso Laboratório: O HTML da "Esquina do Sabor"

Para explorarmos os conceitos de CSS, vamos usar como base a página HTML que estruturamos para a nossa lanchonete. Este arquivo contém todos os tipos de elementos que precisaremos para aplicar nossos estilos: cabeçalho, seções, artigos, um grid, um formulário e muito mais.

Listing 3.1 – Código HTML base para as práticas de CSS.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
       initial-scale=1.0">
    <title>Esquina do Sabor - A Melhor Lanchonete</title>

    <style>
        /* Estilos podem ser adicionados aqui ou definidos em
        arquivos externos e incluidos usando a tag link*/
    </style>
</head>
<body>

    <header class="cabecalho-principal">
        <img src="imagens/logo.png" alt="Logomarca da Esquina do
           Sabor" id="logo">
```

```
17          <h1>Esquina do Sabor</h1>
18          <nav>
19              <ul>
20                  <li><a href="#promocao">Promoções</a></li>
21                  <li><a href="#cardapio">Cardápio</a></li>
22                  <li><a href="#contato">Contato</a></li>
23              </ul>
24          </nav>
25      </header>
26
27      <main>
28          <section id="promocao" style="background-color: #fff8e1;">
               <h2>Promoção do Dia!</h2>
29              <p>Na compra de qualquer pastel, o <strong>caldo de
                   cana</strong> sai pela metade do preço!</p>
30              <p>Aproveite já esta oferta por tempo limitado.</p>
31          </section>
32
33          <section id="cardapio" class="secao-cardapio">
34              <h2>Nosso Cardápio</h2>
35              <div class="cardapio-grid">
36                  <article class="item-cardapio">
37                      <img src="imagens/pastel.png" alt="Um pastel de
                           queijo dourado e crocante.">
38                      <h3>Pastel de Queijo</h3>
39                      <p>Crocante por fora, cremoso por dentro. <span
                           class="preco">R$ 7,00</span></p>
40                  </article>
41                  <article class="item-cardapio">
42                      <img src="imagens/coxinha.png" alt="Uma coxinha
                           de frango bem recheada.">
43                      <h3>Coxinha de Frango</h3>
44                      <p>A tradicional receita brasileira com muito
                           recheio. <span class="preco">R$
                           6,00</span></p>
45                  </article>
46                  <article class="item-cardapio">
47                      <img src="imagens/caldo-de-cana.png" alt="Um
                           copo de caldo de cana gelado com limão.">
48                      <h3>Caldo de Cana</h3>
49                      <p>Gelado e refrescante, moído na hora. <span
                           class="preco">R$ 5,00</span></p>
50                  </article>
51              </div>
52          </section>
53
54          <section id="contato">
55              <h2>Faça seu Pedido</h2>
56              <form action="#" method="post">
57                  <fieldset>
58                      <legend>Seus Dados</legend>
59                      <label for="nome">Nome:</label>
```

```
60                      <input type="text" id="nome" name="nome"
                           required placeholder="Digite seu nome
                           completo">
61
62                      <label for="telefone">Telefone:</label>
63                      <input type="tel" id="telefone" name="telefone"
                           required pattern="[0-9]{10,11}"
                           placeholder="Apenas números">
64                  </fieldset>
65
66                  <fieldset>
67                      <legend>Seu Pedido</legend>
68                      <label for="item">Escolha o item:</label>
69                      <select id="item" name="item">
70                          <option value="pastel">Pastel de
                               Queijo</option>
71                          <option value="coxinha">Coxinha de
                               Frango</option>
72                          <option value="caldo">Caldo de Cana</option>
73                      </select>
74
75                      <label for="quantidade">Quantidade:</label>
76                      <input type="number" id="quantidade"
                           name="quantidade" min="1" max="10" value="1">
77
78                      <p>Para viagem?</p>
79                      <input type="radio" id="viagem_sim"
                           name="viagem" value="sim"> <label
                           for="viagem_sim">Sim</label>
80                      <input type="radio" id="viagem_nao"
                           name="viagem" value="nao" checked> <label
                           for="viagem_nao">Não</label>
81                  </fieldset>
82
83                  <button type="submit">Enviar Pedido</button>
84              </form>
85          </section>
86      </main>
87
88      <footer>
89          <p>&copy; 2025 Esquina do Sabor. Todos os direitos
                reservados.</p>
90          <p>Aracaju, Sergipe</p>
91      </footer>
92
93  </body>
94  </html>
```

Se você transcrever o conteúdo HTML em um arquivo index.html, você verá uma formatação como o apresentado do lado esquerdo da Figura 5. O resultado da formatação que faremos ao longo deste capítulo será como o apresentado do lado direito da Figura.

Figura 5 – Esquerda: HTML puro, Direita: após a formatação com CSS.

3.2 Como Incluir CSS em uma Página

Existem três maneiras de aplicar regras CSS a um documento HTML. A escolha depende do escopo e da organização que você deseja.

1. **CSS Externo (*External*):** Esta é a **forma recomendada e mais utilizada**. Criamos um arquivo separado com a extensão `.css` e o vinculamos ao nosso HTML através da tag `<link>` dentro do `<head>`. Vantagens: organização, reutilização (uma folha de estilo para várias páginas) e melhor desempenho (o navegador pode armazenar o arquivo em cache).

2. **CSS Interno (*Internal*):** As regras são escritas diretamente dentro de uma tag `<style>` no `<head>` do documento HTML. É útil para estilos que são específicos de uma única página.

3. **CSS Em Linha (*Inline*):** O estilo é aplicado diretamente a uma tag HTML através do atributo `style`. Deve ser evitado, pois mistura conteúdo e apresentação, tornando a manutenção difícil. Use apenas em situações muito específicas.

Volte ao código HTML apresentado. Você consegue identificar onde o CSS foi incluído usando a forma inline?

3.3 A Sintaxe do CSS: Seletores e Declarações

Importante!!! Para acompanhar melhor o que será explicado daqui para frente, recomendo fortemente que você construa a árvore do DOM da página HTML de referência. Você pode usar o indicado no capítulo anterior, ou usar um novo, como o *https://codu-code.github.io/dom-visualizer/index.html*. Na verdade, o que não falta é opção, basta pesquisar no Google por "DOM Visualizer".

Uma regra CSS é composta por duas partes principais: um **seletor** e um **bloco de declaração**.

```
seletor {
    propriedade: valor;
    outra-propriedade: outro-valor;
}
```

- **Seletor:** Aponta para o elemento HTML que você quer estilizar (ex: `h1`, `.minha-classe`, `#meu-id`).
- **Declaração:** Consiste em uma **propriedade** (o que você quer mudar, ex: `color`) e um **valor** (como você quer mudar, ex: `blue`).

3.3.1 Seletores Básicos: Tipo, Classe e ID

- **Seletor de Tipo (Tag):** Seleciona todos os elementos de um determinado tipo. Ex: `p` seleciona todos os parágrafos.
- **Seletor de Classe (`.`):** Seleciona todos os elementos que possuem um determinado atributo `class`. É a forma mais flexível e reutilizável de aplicar estilos. Ex: `.item-cardapio` seleciona todos os artigos do nosso cardápio.
- **Seletor de ID (`#`):** Seleciona um **único** elemento que possui um determinado atributo `id`. O ID deve ser exclusivo na página. Ex: `#promocao` seleciona apenas a seção de promoção.

3.3.2 Combinadores: Refinando a Seleção

Você pode filtrar a seleção dos elementos usando combinadores, que podem ser especificados conforme segue.

- **Descendente (espaço):** Seleciona elementos que estão dentro de outro. Ex: `nav a` seleciona todos os links dentro da navegação.
- **Filho Direto (>):** Seleciona apenas os filhos diretos. Ex: `ul > li` seleciona os itens de lista que são filhos diretos de uma <ul>.
- **Irmão Adjacente (+):** Seleciona o elemento que vem *imediatamente depois* de outro. Ex: `h2 + p` seleciona o primeiro parágrafo logo após um <h2>.
- **Irmãos em Geral (~):** Seleciona todos os irmãos que vêm depois de um certo elemento.

3.3.3 Cascata e Especificidade: A Regra do Jogo

O "C"em CSS vem de "Cascata". Isso significa que a ordem e a especificidade das regras importam.

- **Especificidade:** O navegador calcula um "peso"para cada seletor. Quanto mais específico, mais peso ele tem. A ordem de força é: **Seletor de Tipo < Seletor de Classe < Seletor de ID < Estilo Inline**.
- **Cascata:** A última regra declarada vence (se a especificidade for a mesma). Um estilo em um arquivo externo pode ser sobrescrito por um estilo interno, que por sua vez pode ser sobrescrito por um estilo inline. Ou seja, na ordem de prioridade, temos que: inline > interno > externo.
- **`!important`:** Uma declaração com `!important` fura a fila e é aplicada, não importa a especificidade. **Deve ser evitado ao máximo!**, pois torna a depuração do código um pesadelo.

3.4 Fundamentos do Estilo

Entendidos os conceitos mais básicos, hora de aplicar os fundamentos usando nossa página de referência como guia. Uma boa forma de acompanhar esse assunto é criar o arquivo CSS com o nome indicado em uma pasta chamada "css"no mesmo diretório onde está o arquivo index.html. Então deve-se transcrever o conteúdo dos arquivos CSS e incluir no <head> do HTML usando a tag link à medida em que surgirem no texto, conforme no trecho de código HTML que segue:

Listing 3.2 – Código HTML para inclusão dos arquivos externos CSS presentes neste capítulo.

```
<link rel="stylesheet" href="css/00-variaveis-e-globais.css">
<link rel="stylesheet" href="css/01-seletores-e-texto.css">
<link rel="stylesheet" href="css/02-layout-e-boxmodel.css">
<link rel="stylesheet" href="css/03-flexbox.css">
<link rel="stylesheet" href="css/04-grid.css">
<link rel="stylesheet" href="css/05-formularios.css">
<link rel="stylesheet" href="css/06-efeitos-visuais.css">
<link rel="stylesheet" href="css/07-responsividade.css">
```

3.4.1 Variáveis, Reset e Estilos Globais

Antes de começar, vamos definir as bases do nosso design. Usamos **variáveis CSS** (declaradas dentro de `:root`, usando o prefixo – , seguido do nome da variável, seguido de :, seguido do seu valor) para armazenar valores reutilizáveis, como cores e fontes. Também aplicamos um "reset"básico para remover margens e paddings padrão dos navegadores.

Listing 3.3 – 00-variaveis-e-globais.css: As fundações do nosso design.

```
/* --- Variáveis e Boas Práticas (Conceito: Variáveis CSS) --- */
:root {
    --cor-primaria: #c0392b; /* Vermelho escuro */
    --cor-secundaria: #27ae60; /* Verde */
    --cor-texto: #333333; /* Cinza escuro para texto */
    --cor-fundo: #fdfaf6;
    --fonte-principal: Arial, Helvetica, sans-serif;
    --sombra-padrao: 0 4px 8px rgba(0, 0, 0, 0.1);
}

/* --- Estilos Globais e Box Model --- */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box; /* Conceito: Propriedade box-sizing */
}

body {
    font-family: var(--fonte-principal);
    color: var(--cor-texto);
    background-color: var(--cor-fundo);
    line-height: 1.6;
}
```

3.4.2 Unidades de Medida e Cores

Em CSS especificamos os tamanhos dos elementos (tamanho da fonte, largura da borda, altura da imagem...) utilizando unidades, que podem ser absolutas e relativas. Os tamanhos absolutos normalmente são adotados poucas vezes e servem como referência para as dimensões relativas usadas nos demais elementos.

- **Unidades Absolutas:** São as unidades que indicam um tamanho fixo. `px` (pixels) é a mais comum, mas também é possível usar cm, mm, in (inches polegadas), além de outros menos usuais.
- **Unidades Relativas:** São unidades relativas ao tamanho de outro elemento (dependem de outro elemento). `%` (porcentagem), `em` (relativo ao tamanho da fonte do elemento pai), `rem` (relativo ao tamanho da fonte do elemento raiz, <`html`>), `vh` (viewport height) e `vw` (viewport width).

Já as cores em CSS podem ser definidas de várias maneiras, cada uma com sua utilidade. Adicionar um "a"no final de alguns formatos (como em RGBA ou HSLA) permite definir

a transparência. Mas como essas cores podem ser definidas, de fato? Embora você possa pesquisar paletas de cores prontas na internet, é importante entender a estrutura por trás de cada especificação.

Vamos usar a cor laranja como nosso exemplo para explorar os principais formatos:

- **Por Nome (*Color Name*):** É a forma mais simples e legível. O CSS possui cerca de 140 nomes de cores pré-definidos que o navegador entende diretamente. É ótimo para cores comuns e prototipagem rápida.
  - Exemplo: `color: orange;`
- **Por Hexadecimal (*HEX*):** Este é o formato mais comum na web. Ele representa a cor pela mistura de Vermelho, Verde e Azul (RGB) em base 16. A estrutura é `#RRGGBB`. Cada par de caracteres (de `00` a `FF`) indica a intensidade de uma das três cores primárias.
  - `RR`: Intensidade de Vermelho.
  - `GG`: Intensidade de Verde.
  - `BB`: Intensidade de Azul.

  Para o nosso laranja, temos uma alta intensidade de vermelho (`FF`), uma intensidade média de verde (`A5`) e nenhuma intensidade de azul (`00`). Para acompanhar melhor o que esses códigos significam, abra a calculadora do seu computador no modo programador. Selecione a base hexadecimal (base 16) e então digite A5 no display. Você verá que o equivalente em decimal para esse valor é 165. Se fizer o mesmo para FF, verá que o equivalente decimal é 255.
  - Exemplo: `background-color: #FFA500;`
- **Por RGB (*Red, Green, Blue*):** Similar ao HEX, mas usa a base decimal (valores de 0 a 255) para definir a intensidade de Vermelho, Verde e Azul. É um pouco mais fácil para humanos lerem e ajustarem do que o hexadecimal. A estrutura é `rgb(R, G, B)`.
  - Exemplo: `background-color: rgb(255, 165, 0);`

  Transparência com RGBA: Adicionando o canal "A"(Alfa), podemos controlar a opacidade. O valor de alfa vai de `0.0` (totalmente transparente) a `1.0` (totalmente opaco).
  - Exemplo: `background-color: rgba(255, 165, 0, 0.5);` *(Laranja com 50% de transparência)*
- **Por HSL (*Hue, Saturation, Lightness*):** Este é o formato mais intuitivo para o cérebro humano, pois se aproxima de como percebemos as cores. A estrutura é `hsl(H, S, L)`.
  - **H (Matiz / *Hue*):** É o grau na roda de cores (de 0 a 360). 0 é vermelho, 120 é verde, 240 é azul. O laranja fica em torno de 39.
  - **S (Saturação / *Saturation*):** É a pureza da cor, em porcentagem. `0%` é um tom de cinza, e `100%` é a cor mais vibrante possível.
  - **L (Luminosidade / *Lightness*):** É o brilho, em porcentagem. `0%` é preto, `100%` é branco, e `50%` é a cor pura.

Este formato é excelente para criar paletas de cores, pois você pode manter o mesmo matiz (H) e apenas variar a saturação (S) e a luminosidade (L) para obter tons diferentes da mesma cor.

- Exemplo: `border-color: hsl(39, 100%, 50%);`

Transparência com HSLA: Assim como no RGBA, podemos adicionar um quarto valor para a transparência.

- Exemplo: `border-color: hsla(39, 100%, 50%, 0.8);` *(Laranja vibrante com 80% de opacidade)*

Na prática mesmo, recorremos a ferramentas como o *https://colorpicker.me/* para selecionarmos a cor que queremos.

3.4.3 O Box Model: Entendendo as Caixas da Web

Um dos conceitos mais fundamentais do CSS é o **Box Model** (Modelo de Caixa). A melhor forma de entendê-lo é pensar que **todo elemento na sua página HTML é uma caixa retangular**. Seja um título, um parágrafo, uma imagem ou uma seção, o navegador o enxerga como uma caixa. O Box Model é o conjunto de regras que define como o tamanho dessa caixa e o espaço ao redor dela são calculados.

Essa "caixa"é composta por quatro camadas, como se fossem as camadas de uma cebola. De dentro para fora, elas são:

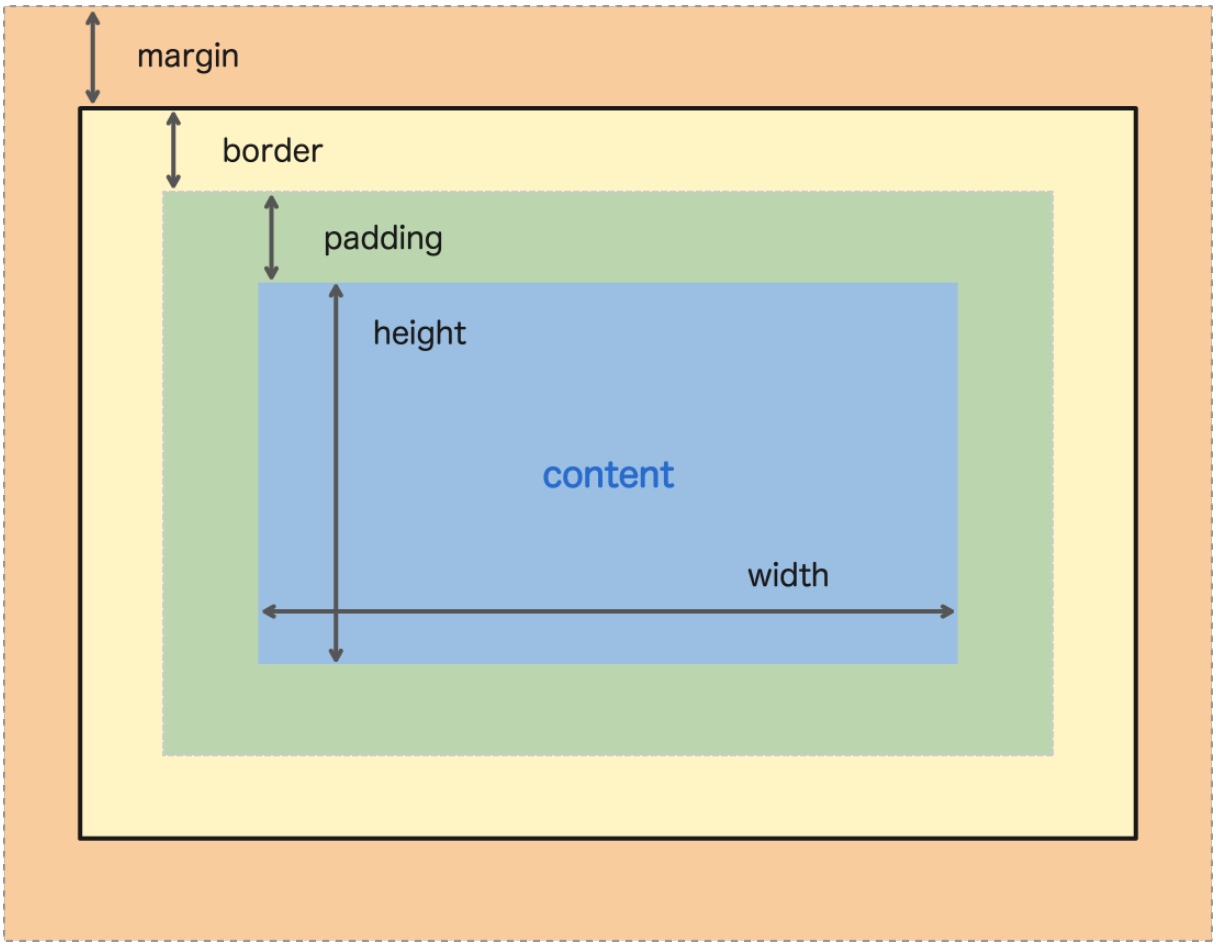
1. **Conteúdo (*Content*):** Esta é a camada mais interna, onde o seu conteúdo de fato reside. Pode ser o texto de um parágrafo, uma imagem, o título de uma seção, etc. Suas dimensões são definidas pelas propriedades `width` (largura) e `height` (altura).
2. **Preenchimento (*Padding*):** É o espaçamento **interno** da caixa, uma área transparente que fica entre o conteúdo e a borda. Pense nele como o "respiro"do seu conteúdo. Se você define um `padding` de `10px`, você está empurrando a borda para longe do conteúdo em 10 pixels de todos os lados.
3. **Borda (*Border*):** É a linha que envolve o conteúdo e o preenchimento. A borda pode ter diferentes estilos (sólida, tracejada), espessuras e cores. Ela fica exatamente entre o `padding` e a `margin`.
4. **Margem (*Margin*):** É o espaçamento **externo** da caixa, uma área transparente que fica do lado de fora da borda. A margem é responsável por empurrar os outros elementos para longe, criando o espaço *entre* as caixas na sua página.

3.4.3.0.1 Uma Propriedade Mágica: `box-sizing`

Por padrão, quando você define a largura (`width`) de um elemento, esse valor se aplica apenas à área do *conteúdo*. O `padding` e a `border` são adicionados a essa largura, o que pode tornar o cálculo do tamanho total da caixa um pouco confuso.

Para simplificar nossa vida, usamos uma regra de ouro no início do nosso CSS:

Figura 6 – Modelo de caixa do CSS




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

Com `box-sizing: border-box;`, a largura que você define para um elemento passa a ser a largura **total** da caixa, incluindo o `padding` e a `border`. O navegador então ajusta o espaço do conteúdo internamente para que tudo caiba. Isso torna a criação de layouts muito mais intuitiva e previsível.

Listing 3.4 – 02-layout-e-boxmodel.css: Definindo a estrutura de caixas.

```
1  /* --- Estrutura e Layout Principal (Box Model) --- */
2  header, main, footer, section {
3      width: 90%;
4      max-width: 960px;
5      margin: 20px auto; /* Margin auto centraliza elementos de bloco
           com largura definida */
6      padding: 20px;
7  }
8
9  .cabecalho-principal {
10     background-color: white;
11     border: 1px solid #ddd;
```

```
    border-radius: 8px;
    box-shadow: var(--sombra-padrao);
}

#promocao {
    border-radius: 8px;
    text-align: center;
    border: 2px dashed var(--cor-primaria);
    background-color: rgba(224, 83, 64, 0.1);
}

.item-cardapio {
    background-color: white;
    border: 1px solid #eee;
    padding: 15px;
    text-align: center;
    box-shadow: var(--sombra-padrao);
    border-radius: 8px;
}

.item-cardapio img {
    max-width: 100%;
    height: auto;
    border-radius: 5px;
    margin-bottom: 10px;
}

.item-cardapio .preco {
    font-weight: bold;
    color: var(--cor-secundaria);
    display: block; /* Conceito: Display */
    margin-top: 10px;
}
```

3.5 Construindo Layouts Modernos

3.5.1 Display e Posicionamento

3.5.2 Display e Posicionamento: Controlando o Fluxo e a Posição dos Elementos

Até agora, vimos os elementos como caixas que se empilham e se organizam de forma previsível. Mas e se quisermos quebrar esse padrão? E se precisarmos que um elemento fique ao lado do outro, ou flutue sobre a página? É aqui que entram duas das propriedades mais poderosas do CSS: `display` e `position`.

3.5.2.0.1 A Propriedade `display`: O Comportamento da Caixa

A propriedade `display` define como um elemento se comporta no fluxo do layout. Ela controla se o elemento começa em uma nova linha, se respeita a largura e altura definidas e como

ele interage com os elementos ao seu redor. Os valores mais fundamentais são:

- `display: block;`

  Pense nos elementos de bloco como "antissociais". Eles gostam de ter seu próprio espaço.

  - Sempre começam em uma **nova linha**.
  - Ocupam toda a **largura disponível** do seu contêiner pai, por padrão.
  - Respeitam as propriedades `width`, `height`, `margin` e `padding` em todas as direções.
  - Exemplos comuns: `<h1>` a `<h6>`, `<p>`, `<div>`, `<section>`, `<form>`.

- `display: inline;`

  Elementos em linha são "sociais". Eles gostam de ficar juntos na mesma linha, um após o outro.

  - **Não começam** em uma nova linha; eles fluem com o texto.
  - Ocupam apenas a **largura necessária** para o seu conteúdo.
  - **Não respeitam** `width` e `height`. Além disso, `margin` e `padding` verticais (top/bottom) não afetam o layout dos elementos ao redor.
  - Exemplos comuns: `<a>`, `<span>`, `<strong>`, `<img>`.

- `display: inline-block;`

  Este valor é um híbrido, o "melhor dos dois mundos".

  - Se comporta como um elemento `inline` externamente (fica na mesma linha que outros elementos).
  - Se comporta como um elemento `block` internamente (respeita `width`, `height`, `margin` e `padding`).
  - É extremamente útil para criar itens de navegação ou pequenos cards que precisam ficar lado a lado, mas com dimensões e espaçamentos definidos.

- `display: none;`

  Remove o elemento completamente da página. Ele não apenas fica invisível, mas também não ocupa nenhum espaço. É como se nunca tivesse existido no HTML.

Valores modernos como `display: flex;` e `display: grid;` transformam o elemento em um contêiner especial, nos dando superpoderes para alinhar e distribuir os elementos filhos, como veremos a seguir.

3.5.2.0.2 A Propriedade `position`: Tirando a Caixa do Fluxo

Se `display` define o comportamento padrão, `position` nos permite quebrar as regras e colocar um elemento exatamente onde queremos. Para isso, usamos as propriedades de deslocamento: `top` (topo), `right` (direita), `bottom` (em baixo) e `left` (esquerda).

- `position: static;`

  Este é o valor padrão. O elemento simplesmente segue o fluxo normal da página. As propriedades `top`, `right`, etc., não têm nenhum efeito sobre ele.

- `position: relative;`

  Este é o ponto de partida para o posicionamento avançado. O elemento permanece no fluxo normal da página, mas agora podemos "deslocá-lo"*relativamente* à sua posição original usando `top`, `left`, etc. O mais importante é que, ao se tornar "relativo", ele cria um **contexto de posicionamento** para seus elementos filhos que tiverem `position: absolute`.

- `position: absolute;`

  Aqui a mágica acontece. Um elemento com posição absoluta é **removido do fluxo normal** da página. Ele passa a "flutuar"e não afeta mais a posição dos outros elementos. Sua posição é calculada com base no seu **ancestral posicionado mais próximo** (qualquer ancestral que tenha uma posição diferente de `static`, geralmente `relative`). Se nenhum ancestral posicionado for encontrado, ele se posiciona em relação ao próprio corpo da página (<body>). É perfeito para criar ícones, selos ou janelas modais que precisam ficar sobre outros conteúdos.

- `position: fixed;`

  Similar ao `absolute`, o elemento é removido do fluxo. A diferença é que ele se posiciona em relação à **janela do navegador (*viewport*)**. Isso significa que ele ficará fixo na tela, mesmo quando o usuário rolar a página. É ideal para criar cabeçalhos fixos, menus laterais ou botões de "Voltar ao Topo".

- `position: sticky;`

  Um híbrido inteligente. O elemento se comporta como `static` até que a rolagem da página o faça atingir um determinado ponto (definido por `top`, `bottom`, etc.). A partir daí, ele "gruda"na tela e se comporta como `fixed`. É muito usado para cabeçalhos de tabela ou barras de busca que devem ficar visíveis após o usuário rolar uma certa distância.

3.5.2.0.3 A Propriedade `z-index`: Empilhando as Camadas

Quando usamos posicionamentos que tiram os elementos do fluxo (como `absolute` ou `fixed`), eles podem acabar se sobrepondo. A propriedade `z-index` controla a ordem de empilhamento dessas camadas, como se fossem folhas de papel transparentes.

- Só funciona em elementos que tenham uma propriedade `position` diferente de `static`.
- Aceita um valor numérico (pode ser negativo).
- Elementos com um `z-index` maior ficam na frente de elementos com um `z-index` menor.

Tente você mesmo!

Modifique as definições para o id #promocao, adicionando a propriedade display e vinculando a ela o valor none. Atualize a página. O que acontece com a promoção? Agora volte ao CSS e altere a propriedade display para block. O que acontece?

3.5.3 Flexbox: Layouts em Uma Dimensão

O Flexbox é um modelo de layout projetado para organizar itens em uma única linha ou coluna. É perfeito para alinhar itens em um cabeçalho, um menu de navegação ou um rodapé. Ativamos com `display: flex;` no contêiner pai e controlamos o alinhamento com propriedades como `justify-content` e `align-items`.

Listing 3.5 – 03-flexbox.css: Organizando a navegação.

```
/* --- Cabeçalho e Navegação (Conceito: Flexbox) --- */
.cabecalho-principal nav ul {
    list-style: none;
    display: flex; /* Ativando o Flexbox */
    justify-content: center; /* Alinha os itens no centro do eixo
       principal */
    gap: 20px;
}

/* Seletor de Filho (>) */
.cabecalho-principal nav > ul > li > a {
    padding: 10px 15px;
    border-radius: 5px;
}
```

3.5.4 Grid Layout: Layouts em Duas Dimensões

O Grid Layout é um sistema poderoso para criar layouts complexos em linhas e colunas, como uma grade de revista. Ativamos com `display: grid;` e definimos a estrutura das colunas e linhas com `grid-template-columns` e `grid-template-rows`.

Listing 3.6 – 04-grid.css: Criando o layout do cardápio.

```
/* --- Cardápio (Conceito: Grid Layout) --- */
.cardapio-grid {
    display: grid;
    /* Cria colunas responsivas que se adaptam ao espaço disponível
       */
    grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
    gap: 25px;
}
```

Discussão importante

É fundamental entender a diferença entre o Box Model, o Flexbox Layout e o Grid Layout, pois cada um tem um papel diferente na construção de uma página. O CSS Box Model não é um sistema de layout ativo, mas sim o princípio fundamental de que todo elemento HTML é uma caixa retangular com quatro camadas: conteúdo, padding (preenchimento), border (borda) e margin (margem), conforme já discutimos. O comportamento padrão do navegador, onde elementos de bloco se empilham verticalmente e elementos em linha fluem horizontalmente, é conhecido como "Flow Layout"e é a consequência natural do Box Model antes de aplicarmos sistemas mais avançados.
O Flexbox Layout e o Grid Layout, por outro lado, são sistemas de layout ativos e intencionais. O Flexbox é um modelo unidimensional, projetado para distribuir espaço e alinhar itens ao longo de um único eixo, seja uma linha ou uma coluna. Ele é perfeito para componentes como menus de navegação, alinhar itens dentro de um card ou centralizar um elemento. Já o Grid Layout é um modelo bidimensional, criado para gerenciar linhas e colunas simultaneamente. Ele permite construir uma grade complexa para a página inteira, posicionando os elementos com precisão dentro dessa estrutura. Em resumo: enquanto o Box Model descreve as propriedades de cada caixa, o Flexbox as organiza em uma única dimensão (linha ou coluna) e o Grid as organiza em duas dimensões (uma grade completa).

3.6 Efeitos e Interatividade

3.6.1 Estilizando Formulários

Podemos usar seletores de atributo para estilizar diferentes tipos de `<input>`, melhorando drasticamente a usabilidade dos nossos formulários.

Listing 3.7 – 05-formularios.css: Melhorando a aparência do formulário.

```
/* --- Estilos de Formulários --- */
form {
    background-color: white;
    padding: 25px;
    border-radius: 8px;
    box-shadow: var(--sombra-padrao);
}

fieldset {
    border: 1px solid #ddd;
    padding: 15px;
    margin-bottom: 20px;
    border-radius: 5px;
}

legend {
    padding: 0 10px;
    font-weight: bold;
    color: var(--cor-primaria);
}

input[type="text"],
```

```
input[type="tel"],
input[type="number"],
select,
textarea {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    margin-bottom: 15px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button[type="submit"] {
    display: block;
    width: 100%;
    padding: 12px;
    background-color: var(--cor-secundaria);
    color: white;
    border: none;
    border-radius: 5px;
    font-size: 1.1em;
    cursor: pointer;
}
```

3.6.2 Pseudo-classes e Pseudo-elementos: Estilizando o Dinâmico e o Específico

Até agora, nossos seletores focaram em elementos que *existem* de forma estática no HTML (uma tag, uma classe, um ID). Mas e se quiséssemos estilizar um link apenas quando o mouse está sobre ele? Ou colorir apenas a primeira letra de um parágrafo? Para isso, usamos pseudo-classes e pseudo-elementos, que nos permitem selecionar elementos com base em seu **estado** ou em uma **parte específica** deles.

3.6.2.0.1 Pseudo-classes: Selecionando por Estado ou Posição

Uma pseudo-classe é uma palavra-chave, iniciada por dois pontos (`:`), que adicionamos a um seletor para especificar um estado especial do elemento. Elas nos permitem criar interfaces dinâmicas e interativas.

- `:hover`

  Provavelmente a pseudo-classe mais famosa. Ela aplica um estilo quando o cursor do mouse está **sobre** o elemento. É essencial para dar feedback visual ao usuário em links, botões e qualquer elemento clicável.

  - Exemplo: `a:hover { color: red; }` (Muda a cor do link para vermelho quando o mouse passa por cima).

- `:active`

Aplica um estilo no exato momento em que o elemento está sendo **ativado** pelo usuário, ou seja, enquanto o botão do mouse está pressionado sobre ele. É útil para dar um feedback instantâneo de "clique".

  - Exemplo: `button:active { transform: scale(0.98); }` (Diminui ligeiramente o tamanho do botão enquanto ele é pressionado).

- `:focus`

  Aplica um estilo quando um elemento está **em foco**. Isso acontece quando o selecionamos com o mouse (clicando em um campo de formulário) ou através da navegação pelo teclado (usando a tecla Tab). É crucial para a acessibilidade, pois mostra ao usuário qual elemento está ativo.

  - Exemplo: `input:focus { border-color: blue; }` (Muda a cor da borda do campo de input quando ele está selecionado).

- `:nth-child(n)`

  Esta é uma pseudo-classe estrutural poderosa. Ela seleciona elementos com base em sua **posição** entre um grupo de irmãos. O `n` pode ser um número (`1`, `2`, `3`...), uma palavra-chave (`odd` para os ímpares, `even` para os pares) ou uma fórmula (`2n+1`). É fantástica para criar tabelas ou listas com "zebrado"(cores de fundo alternadas).

  - Exemplo: `tbody tr:nth-child(even) { background-color: #f2f2f2; }` (Aplica um fundo cinza a todas as linhas pares do corpo de uma tabela).

3.6.2.0.2 Pseudo-elementos: Selecionando Partes ou Criando Conteúdo

Um pseudo-elemento, iniciado por dois pontos duplos (`::`), funciona como um "elemento fantasma"que você pode estilizar. Ele permite selecionar uma *parte* de um elemento ou *adicionar conteúdo* antes ou depois do conteúdo real do elemento.

- `::before`

  Cria um pseudo-elemento que se torna o **primeiro filho** do elemento selecionado. Ele é renderizado "antes"do conteúdo real. Para que ele apareça, é obrigatório usar a propriedade `content`. O conteúdo pode ser um texto, um ícone ou até mesmo uma string vazia se for usado apenas para fins decorativos (como formas geométricas).

  - Exemplo: `h2::before { content: '* '; color: gold; }` (Adiciona um asterisco dourado antes de todo título <h2>).

- `::after`

  Funciona da mesma forma que o `::before`, mas cria um pseudo-elemento que se torna o **último filho** do elemento selecionado, aparecendo "depois"do conteúdo real. Também requer a propriedade `content`.

  - Exemplo: `a.link-externo::after { content: 'º '; }` (Adiciona uma bolinha depois de todo link com a classe "link-externo").

- `::first-letter` e `::first-line`

  Permitem estilizar, respectivamente, a **primeira letra** ou a **primeira linha** de um elemento de bloco, como um parágrafo. São ótimos para criar efeitos tipográficos, como um capitular (letra maior no início de um capítulo de livro).

  - Exemplo: `article p::first-letter { font-size: 3em; color: red; }` (Torna a primeira letra de cada parágrafo dentro de um artigo três vezes maior e vermelha).

3.6.3 Transições e Animações: Adicionando Movimento e Fluidez

Um design web moderno vai além de cores e layouts estáticos; ele se move e reage ao usuário. O CSS nos dá duas ferramentas poderosas para controlar o movimento e criar experiências mais ricas e intuitivas: **Transições** e **Animações**. Embora ambas criem movimento, elas servem a propósitos diferentes.

3.6.3.0.1 Transições: Suavizando Mudanças de Estado

Pense em uma transição como a diferença entre ligar uma lâmpada com um interruptor (uma mudança instantânea de "desligado"para "ligado") e usar um *dimmer* (uma mudança suave e gradual). As transições em CSS fazem exatamente isso: elas suavizam a mudança de uma propriedade de um estado para outro.

Uma transição é sempre acionada por uma **mudança de estado**, que geralmente ocorre através de uma pseudo-classe como `:hover`, `:focus` ou `:active`, ou quando uma classe é adicionada ou removida via JavaScript.

Para controlar uma transição, usamos principalmente quatro propriedades:

- `transition-property`: Especifica *qual* propriedade CSS deve ser animada (ex: `background-color`, `transform`, `opacity`). Você pode especificar uma, várias ou usar `all` para todas as propriedades que puderem ser transicionadas.
- `transition-duration`: Define *quanto tempo* a transição deve levar, geralmente em segundos (`s`) ou milissegundos (`ms`). Ex: `0.3s`.
- `transition-timing-function`: Descreve a "curva de aceleração"da transição. Ela controla o ritmo da mudança. Valores comuns são `ease` (padrão: começa lento, acelera e termina lento), `linear` (ritmo constante), `ease-in`, `ease-out` e `ease-in-out`.
- `transition-delay`: Especifica um *atraso* antes que a transição comece.

Sintaxe Abreviada (*Shorthand*): É muito mais comum usar a propriedade `transition` para abreviar tudo em uma única linha, na ordem: `property duration function delay`.

```
.item-cardapio {
    /* ... outros estilos ... */
    /* Define uma transição para as propriedades 'transform' e 'box-shadow'
       que durará 0.3 segundos com uma curva de aceleração 'ease'. */
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
```

```
.item-cardapio:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 15px rgba(0, 0, 0, 0.2);
}
```

No exemplo acima, em vez de o card "pular"instantaneamente para cima ao passar o mouse, ele se moverá e sua sombra se expandirá suavemente ao longo de 0.3 segundos, criando um efeito muito mais agradável.

3.6.3.0.2 Animações: Criando Sequências de Movimento Complexas

Se as transições são para mudanças simples de A para B, as animações são como criar um pequeno filme ou um flipbook. Elas permitem definir múltiplos passos e controlar o ciclo de movimento de forma muito mais detalhada, sem a necessidade de um gatilho como o `:hover` (embora também possam ser acionadas por ele).

A criação de uma animação CSS envolve duas partes:

1. **A Regra `@keyframes`:** Aqui nós definimos a "cena". Damos um nome à animação e descrevemos a aparência do elemento em diferentes estágios do seu ciclo, usando porcentagens (de `0%` a `100%`) ou as palavras-chave `from` (equivalente a 0%) e `to` (equivalente a 100%).
2. **A Propriedade `animation`:** Depois de definir os `@keyframes`, nós "aplicamos"essa animação a um seletor e configuramos como ela deve se comportar: seu nome, duração, quantas vezes deve se repetir, etc.

Exemplo de `@keyframes`:

```
/* Define uma animação chamada 'fadeIn' */
@keyframes fadeIn {
    /* No início (0%), o elemento está totalmente transparente */
    from {
        opacity: 0;
    }
    /* No final (100%), o elemento está totalmente visível */
    to {
        opacity: 1;
    }
}
```

Aplicando a Animação:

```
body {
    /* Aplica a animação 'fadeIn', com duração de 0.8s e curva 'ease-in-out' */
    animation: fadeIn 0.8s ease-in-out;
}
```

Com este código, ao carregar a página, todo o conteúdo do <body> aparecerá suavemente em vez de surgir instantaneamente. As propriedades de animação (como `animation-name`, `animation-duration`, `animation-iteration-count`, etc.) também podem ser escritas de forma abreviada na propriedade `animation`.

3.6.3.0.3 Resumo da Diferença:

- Use **Transições** para suavizar mudanças de estado simples (A → B), geralmente acionadas pela interação do usuário.
- Use **Animações** para criar sequências de movimento complexas com múltiplos passos (A → B → C...), que podem ser executadas automaticamente e repetidas em loop.

Listing 3.8 – 06-efeitos-visuais.css: Adicionando interatividade e decoração.

```
/* --- Pseudo-classes (:hover, :focus) --- */
.cabecalho-principal nav a:hover {
    background-color: var(--cor-secundaria);
    color: white;
}

.item-cardapio:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 15px rgba(0, 0, 0, 0.2);
}

input:focus, select:focus, textarea:focus {
    outline: none;
    border-color: var(--cor-secundaria);
    box-shadow: 0 0 5px rgba(39, 174, 96, 0.5);
}

button[type="submit"]:hover {
    background-color: #2ecc71;
}

/* --- Pseudo-elementos (::before, ::after) e Posicionamento --- */
.secao-cardapio h2::before {
    content: '\2605 '; /* estrela solida */
    color: var(--cor-primaria);
}

.item-cardapio:first-child {
    position: relative; /* Pai precisa ser relativo */
}

.item-cardapio:first-child::after {
    content: 'Novo!';
    position: absolute; /* Filho absoluto */
    top: 10px;
    right: 10px;
```

```
    background-color: var(--cor-primaria);
    color: white;
    padding: 5px 8px;
    border-radius: 5px;
    font-size: 0.8em;
    z-index: 10;
}

/* --- Transições e Animações --- */
.cabecalho-principal nav a,
.item-cardapio,
button[type="submit"] {
    transition: all 0.3s ease-in-out;
}

@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}

body {
    animation: fadeIn 0.8s ease-in-out;
}
```

3.7 Design Responsivo com Media Queries

Hoje, os sites são acessados em uma infinidade de tamanhos de tela. O **Design Responsivo** é a prática de criar layouts que se adaptam a diferentes dispositivos. A principal ferramenta para isso são as **Media Queries**.

Uma Media Query é uma regra `@media` que aplica um bloco de estilos CSS somente se uma determinada condição for verdadeira (geralmente, a largura da tela).

Listing 3.9 – 07-responsividade.css: Adaptando o layout para telas menores.

```
/* --- Responsividade e Media Queries --- */
@media (max-width: 768px) {
    h1 { font-size: 2rem; }
    h2 { font-size: 1.5rem; }

    /* A navegação vira uma coluna em telas menores */
    .cabecalho-principal nav ul {
        flex-direction: column;
        align-items: center;
    }

    /* O grid quebra para uma única coluna */
    .cardapio-grid {
        grid-template-columns: 1fr;
    }
}
```