FMX Academy

INFORMAÇÕES		
MÓDULO	AULA	INSTRUTOR
03 HTML e CSS para Desenvolvedores Web	05 – Cores, Tipografia e Estilos Avançados	Pedro Henrique Amadio

CONTEÚDO

1. Nas aulas anteriores...

- **Pseudo-classes**
  :hover, :focus, :active, :nth-child().

- **Pseudo-elementos**
  ::before, ::after, ::selection.

Na última aula, vimos vários conceitos fundamentais que estruturam a forma como páginas web são organizadas e exibidas.

- **Box Model**
  Cada elemento HTML é tratado como uma "caixa", composta por:
  - Content -> o conteúdo em si (texto, imagem, etc.).
  - Padding -> espaço interno entre o conteúdo e a borda.
  - Border -> a linha que envolve o elemento.
  - Margin -> espaço externo, separando do próximo elemento.

    - **Exemplo:**
      ```
      .card {
        width: 200px;
        padding: 10px;
        border: 2px solid black;
        margin: 20px;
      }
      ```

      O espaço ocupado na tela será 200px + padding (20px) + border (4px) = 224px de largura, mais a margem externa.

- **Display**
  Controla como os elementos aparecem na tela:
  - block -> ocupa toda a linha (ex.: , <p>).
  - inline -> ocupa apenas o necessário (ex.: , <a>).
  - inline-block -> combina inline (na mesma linha) com a possibilidade de ter largura e altura definidas.
  - flex -> ativa o Flexbox.
  - grid -> ativa o Grid Layout.

    - **Exemplo:**
      ```
      <div style="display: block; background: lightblue;">Bloco 1</div>
      <div style="display: block; background: lightgreen;">Bloco 2</div>
      ```

```
<span style="display: inline; background: lightpink;">Inline 1</span>
<span style="display: inline; background: lightyellow;">Inline 2</span>
```

Os ficam um abaixo do outro e os lado a lado.

- **Position**
  Define como um elemento será posicionado na página:
  - static -> padrão.
  - relative -> em relação à posição normal.
  - absolute -> em relação ao elemento pai posicionado.
  - fixed -> fixo na tela, mesmo rolando.
  - sticky -> alterna entre relative e fixed.

  - **Exemplo:**

```
.menu-fixo {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background: #0ea5e9;
}
```

Cria uma barra fixa no topo da tela.

- **Flexbox**
  Permite organizar elementos em linha ou coluna, controlando alinhamento e espaçamento.

  - **Exemplo:**

```
.container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

Distribui os elementos lado a lado, alinhados no centro.

- **Grid**
  Divide a tela em linhas e colunas.

  - **Exemplo**

```
.grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 10px;
}
```

Cria 3 colunas iguais.

- **Media Queries**
  Permitem aplicar estilos diferentes de acordo com o tamanho da tela.

- **Exemplo:**

```
.container { flex-direction: column; }
@media (min-width: 768px) {
  .container { flex-direction: row; }
}
```

No celular, elementos em coluna; no desktop, em linha.

- **Aplicação no Portfólio**
  - Transformamos seções como Sobre, Hobbies e Contato em cards estilizados.
  - Usamos Flexbox/Grid para organizar cards em colunas no desktop e em uma coluna no mobile.
  - Criamos um layout responsivo ajustando margens, espaçamentos e tipografia.

2. Unidades de medida

- **px:** valor fixo.
- **%:** relativo ao elemento pai.
- **em:** relativo ao tamanho da fonte do pai.
- **rem:** relativo ao tamanho da fonte da raiz (html).
- **vh/vw:** relativo ao tamanho da viewport.
- **dvw, dvh**: unidades dinâmicas que consideram o espaço real da viewport, inclusive em dispositivos móveis com barras de navegação/teclado virtual.

- **Exemplo:**

```
h1 { font-size: 3rem; }
p { font-size: 1.2em; }
.box { width: 80vw; }

/* dvh e dvw (dinâmicos, para mobile) */
.fullscreen {
  width: 100dvw;
  height: 100dvh;
}
```

- **Diferença prática:**

```
- vh/vw -> consideram a viewport teórica (podem ignorar a barra do navegador).
- dvh/dvw → consideram a área realmente disponível, ajustando quando aparece o
teclado virtual no mobile.
```

3. Cores

- **Hexadecimal:** #ff0000.
- **RGB:** rgb(255, 0, 0).
- **HSL:** hsl(0, 100%, 50%).
- **Variáveis CSS com :root e var().**

4. Tipografia

A tipografia no CSS define como o texto será exibido na página.
Ela é um dos elementos mais importantes para a legibilidade, acessibilidade e estética de um site.

- **Propriedades mais usadas**
  - font-family -> define a família da fonte.
  - font-size -> tamanho da fonte.
  - font-weight -> espessura (100 a 900, ou valores como normal, bold).
  - line-height -> altura da linha (espaçamento vertical entre linhas de texto).
  - letter-spacing -> espaçamento entre letras.
  - word-spacing -> espaçamento entre palavras.
  - text-align-> alinhamento (left, right, center, justify).
  - text-transform -> maiúsculas/minúsculas (uppercase, lowercase, capitalize).

  - **Exemplo:**

```
p {
  font-family: Arial, Helvetica, sans-serif;
  font-size: 1rem;
  font-weight: 400;
  line-height: 1.6;
  letter-spacing: 0.5px;
  text-align: justify;
}
```

- **Fontes externas (Google Fonts)**
  Permitem importar fontes modernas e estilizadas:

```
<link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">
```

  Ou via @import no CSS:

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

body { font-family: 'Roboto', Arial, sans-serif; }
```

- https://fonts.google.com/

- **Sempre adicione fallback fonts (ex.: Arial, sans-serif).**

- **Unidades em tipografia**
  - **px** -> valor fixo.
  - **em** -> relativo ao tamanho do elemento pai.
  - **rem** -> relativo ao tamanho da raiz (html).
  - **clamp()** -> define mínimo, preferido e máximo → ideal para responsividade.
    - **Exemplo com clamp():**

```
h1 {
  font-size: clamp(1.8rem, 5vw, 3rem);
}
```

- **Boas práticas de tipografia**
  1. Escolha no máximo 2–3 famílias de fonte.
  2. Use hierarquia visual (títulos maiores, corpo legível).
  3. Evite fontes muito decorativas em textos longos.
  4. Priorize legibilidade (contraste adequado).
  5. Sempre defina um line-height adequado (1.4–1.8).

6. Teste em diferentes dispositivos.

- **Exemplo aplicado no portfólio:**

```
:root {
  --fonte-titulo: 'Poppins', Arial, sans-serif;
  --fonte-corpo: 'Roboto', Helvetica, sans-serif;
}

h1, h2, h3 {
  font-family: var(--fonte-titulo);
  font-weight: 700;
  text-transform: uppercase;
}

p, li {
  font-family: var(--fonte-corpo);
  font-size: 1rem;
  line-height: 1.6;
  color: #334155;
}
```

- **Sobre o exemplo acima:**
  - Usamos variáveis CSS para centralizar fontes.
  - Títulos com destaque (Poppins, bold, uppercase).
  - Corpo do texto legível com Roboto.

- **Entendendo o clamp()**

O clamp() é uma função do CSS que define um valor mínimo, ideal e máximo.

Sintaxe: clamp(min, ideal, max)

  - min → valor mínimo permitido.
  - ideal → valor preferido (geralmente relativo, como %, vw, em).
  - max → valor máximo permitido.

  - **Exemplo com tipografia:**

```
h1 {
  font-size: clamp(1.5rem, 5vw, 3rem);
}
```

  - **Interpretação:**
    - 1.5rem → nunca ficará menor que isso (mesmo em telas muito pequenas).
    - 5vw → valor fluido, baseado em 5% da largura da tela.
    - 3rem → nunca ficará maior que isso (mesmo em telas muito grandes).

  - **Vantagens:**

1. Cria responsividade automática sem várias media queries.
2. Mantém legibilidade em telas pequenas e equilíbrio em telas grandes.
3. Funciona não só em fontes, mas também em larguras, margens, paddings etc.

- **Exemplo prático com tipografia + clamp:**

```
h1 {
```

```
  font-family: var(--fonte-titulo);
  font-size: clamp(1.8rem, 4vw, 2.5rem);
  font-weight: 700;
}
p {
  font-size: clamp(1rem, 2vw, 1.2rem);
  line-height: 1.6;
}
```

“O que esse trecho faz?”
Basicamente, os títulos crescem de forma fluida até um limite e os parágrafos acompanham sem exagerar.

5. CSS Reset

Esse nome não é estranho certo?
Já comentamos sobre CSS Reset em aulas anteriores.

Mas o que consiste no CSS Reset?
Cada navegador aplica estilos padrões que podem gerar inconsistências.
Resets servem para zerar ou padronizar esses estilos.

- **Tipos principais:**
  1. Reset tradicional (Eric Meyer)
  ```
  * { margin: 0; padding: 0; box-sizing: border-box; }
  ```

  2. Normalize.css
  O **Normalize.css** é um pequeno arquivo CSS criado para **padronizar o comportamento dos navegadores**, mas **sem apagar totalmente os estilos padrões** (diferente de um reset radical).
  ```
  <link rel="stylesheet"
  href="https://necolas.github.io/normalize.css/latest/normalize.css">
  ```

  3. Modern Reset (Andy Bell)
  ```
  *, *::before, *::after { box-sizing: border-box; }
  * { margin: 0; }
  body { line-height: 1.5; -webkit-font-smoothing: antialiased; }
  img, picture, video, canvas, svg { display: block; max-width: 100%; }
  ```

- **O mais comum hoje é combinar um reset simples (*) com ajustes globais de tipografia e line-height.**

6. Pseudo-classes (interações)

São usadas para estados/interações: :hover, :focus, :active, :nth-child().

- **Exemplo:**
  ```
  button:hover { background: #2563eb; }
  input:focus { outline: 2px solid #0ea5e9; }
  ```

7. Transições

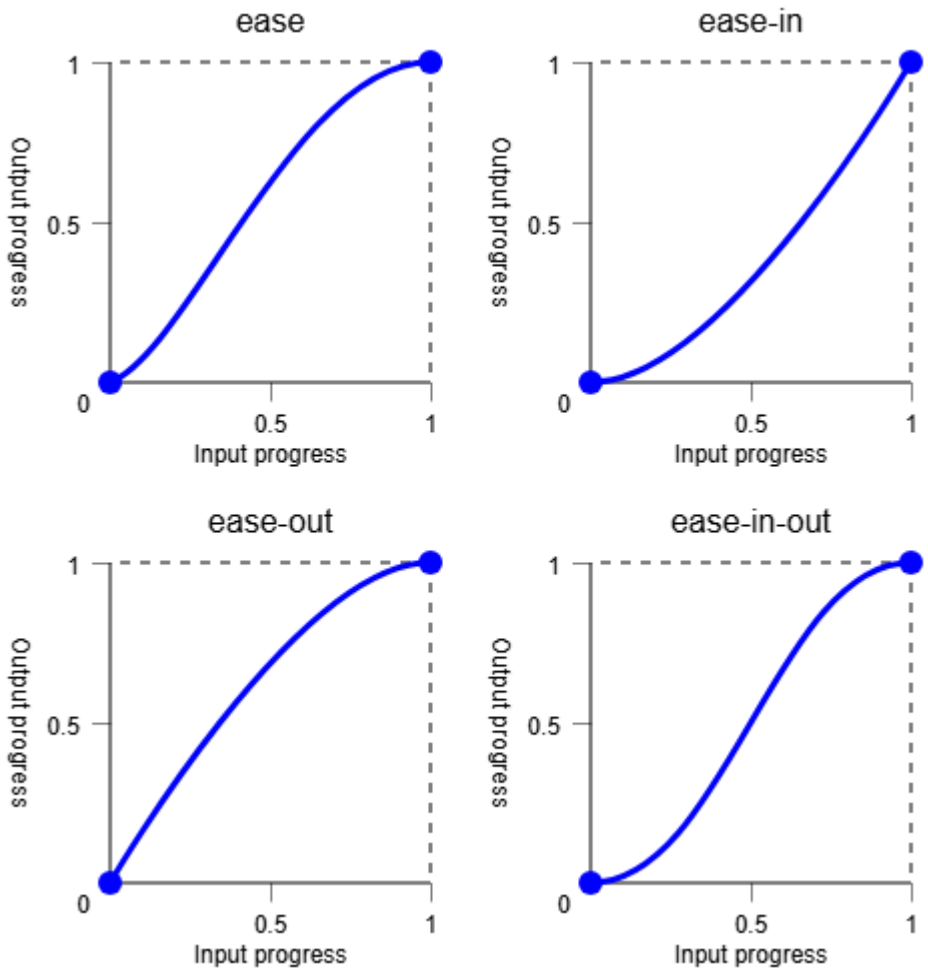
As transições criam efeitos suaves quando uma propriedade muda de valor.

Sintaxe: transition: propriedade duração timing-function delay;

- **Exemplo:**

```
button {
  transition: background 0.3s ease, transform 0.2s ease-in-out;
}
button:hover {
  background: #0369a1; transform: scale(1.05);
}
```

- **Timing-functions:**
  - ease (padrão)
  - linear
  - ease-in / ease-out / ease-in-out



https://developer.mozilla.org/en-US/docs/Web/CSS/easing-function

- **Boas práticas:**
  - Use duração curta (0.2–0.5s).
  - Prefira transform/opacity para performance.
  - Evite animar width/height em excesso.

8. Animações

As animações permitem criar mudanças contínuas de estilo controladas no tempo usando @keyframes.

- **Sintaxe básica:**

```
@keyframes nome {
  from { propriedade: valor; }
  to   { propriedade: valor; }
}

.elemento {
  animation-name: nome;
  animation-duration: 2s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
}
```

- **Propriedades principais:**
  - animation-name
    - Nome da animação
  - animation-duration
    - Duração da animação
  - animation-delay
    - Especifíca quando uma animação deve começar
  - animation-iteration-count
    - Especifíca quantas vezes uma animação se repete
  - animation-direction (normal, reverse, alternate, alternate-reverse)
    - Especifica a direção da animação

- **Exemplos:**
  - **Fade In**

    ```
    @keyframes fadeIn { from { opacity:0; } to { opacity:1; } }
    .caixa { animation: fadeIn 2s ease forwards; }
    ```
  - **Pulsar**

    ```
    @keyframes pulsar { 0%,100% { transform:scale(1); } 50% {
    transform:scale(1.1); } }
    .botao { animation: pulsar 1s ease-in-out infinite; }
    ```

- **Boas práticas:**
  - Evite animar propriedades pesadas.
  - Combine animation-delay para criar sequências.
  - Use de forma sutil para não distrair o usuário.

9. Prática guiada

1. Criar botões interativos com hover + transições.
2. Criar destaque animado (texto piscando, botão pulsando).
3. Aplicar Google Fonts ao portfólio.

10. Exercícios

1) Verdadeiro ou Falso

( ) em é relativo ao pai.
( ) linear mantém velocidade constante.
( ) animation-direction: reverse roda ao contrário.
( ) clamp() define min, ideal e max.

2) Qual valor de animation-direction alterna ida e volta?

a) normal
b) reverse
c) alternate
d) ease-in-out

11. Tarefa de Casa

- Adicionar Google Font.
- Criar 2 botões (ou estilizar os existentes) com hover + transições.
- Criar elemento animado com alternate ou reverse.