LAYOUT COM CSS

Prof. Fabricio Freire, MSc in Cybersecurity

METAS DE APRENDIZAGEM

- Dominar estilos CSS avançados
- Compreender o Box Model profundamente
- Criar layouts com Flexbox
- Implementar layouts com CSS Grid
- Trabalhar com posicionamento CSS
- Criar designs responsivos

REVISÃO: HTML COMO BASE

Estrutura semântica que vimos:

```
<header>
  <nav>...</nav>
</header>
<main>
  <article>...</article>
  <aside>...</aside>
</main>
<footer>...</footer>
```

Agora vamos estilizar e criar layouts profissionais!

CSS - RECAPITULANDO

Três formas de aplicar:

1. **Inline** ❌ `<p style="color: red">`
2. **Internal** ⚠ `<style>p { color: red; }</style>`
3. **External** ☑ `<link rel="stylesheet" href="style.css">`

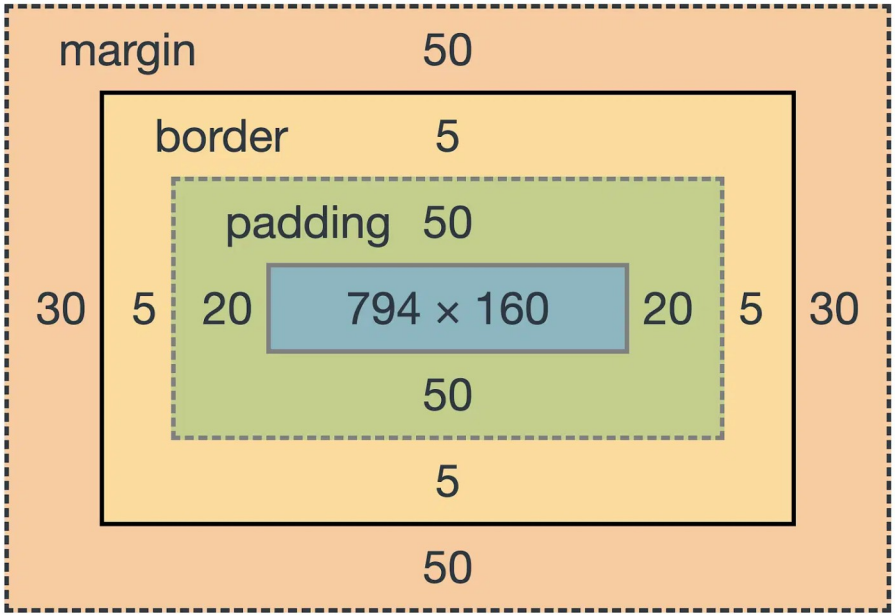
Seletores básicos:

`p` - por tag

`.classe` - por classe

`#id` - por ID

BOX MODEL - O CORAÇÃO DO LAYOUT



Cada elemento é uma caixa!

BOX MODEL NA PRÁTICA

```css
.box {
  width: 200px;              /* Conteúdo */
  height: 100px;
  padding: 20px;             /* Espaço interno */
  border: 5px solid #333;    /* Borda */
  margin: 10px;              /* Espaço externo */
}
```

Largura total: 200 + 40 + 10 + 20 = 270px
Altura total: 100 + 40 + 10 + 20 = 170px

BOX-SIZING: CONTROLANDO O CÁLCULO

box-sizing: content-box (padrão)

```
width = conteúdo apenas
```

box-sizing: border-box (recomendado)

```
width = conteúdo + padding + border
```

```
* {
  box-sizing: border-box; /* Aplicar
globalmente */
}
```

DISPLAY: CONTROLANDO O COMPORTAMENTO

Display	Quebra linha?	Width/Height?	Exemplo
`block`	☑ Sim	☑ Sim	`<div>, <p>`
`inline`	✗ Não	✗ Não	`<span>, <a>`
`inline-block`	✗ Não	☑ Sim	Híbrido

Block vs Inline vs Inline-Block

EXERCÍCIO 1

Crie três boxes com comportamentos diferentes:

```css
.block-box {
  display: block;
  width: 200px;
  height: 100px;
  background: #ff6b6b;
  margin: 10px;
}

.inline-box {
  display: inline;
  background: #4ecdc4;
  padding: 10px;
}

.inline-block-box {
  display: inline-block;
  width: 150px;
  height: 80px;
  background: #45b7d1;
}
```

⏱️ 5 minutos

POSICIONAMENTO CSS

Static (padrão)

```
position: static; /* Fluxo normal */
```

Relative

```
position: relative;
top: 10px; /* Move 10px para baixo */
left: 20px; /* Move 20px para direita */
```

Absolute

```
position: absolute;
top: 50px; /* Relativo ao pai posicionado */
right: 0;
```

POSICIONAMENTO CSS

Fixed

```
position: fixed;
top: 0;
right: 0; /* Fixo na janela */
```

Sticky

```
position: sticky;
top: 10px; /* Gruda quando rola */
```

Z-index controla sobreposição!

FLOAT: O LAYOUT CLÁSSICO

Como funciona:

```css
.left-column {
  float: left;
  width: 70%;
}

.sidebar {
  float: right;
  width: 25%;
}

.clearfix::after {
  content: "";
  display: table;
  clear: both;
}
```

⚠ Float está sendo substituído por Flexbox e Grid

FLEXBOX - LAYOUT MODERNO

Container Flex:

```
.container {
  display: flex;
  flex-direction: row; /* ou column */
  justify-content: center; /* eixo principal */
  align-items: center; /* eixo transversal */
  gap: 20px;
}
```

Itens Flex:

```
.item {
  flex: 1; /* cresce igualmente */
  flex-shrink: 0; /* não encolhe */
}
```

FLEXBOX - PROPRIEDADES PRINCIPAIS

No Container:

`flex-direction`: row | column | row-reverse | column-reverse

`justify-content`: flex-start | center | flex-end | space-between | space-around

`align-items`: stretch | center | flex-start | flex-end

`flex-wrap`: nowrap | wrap | wrap-reverse

Nos Itens:

`flex-grow`: quanto pode crescer

`flex-shrink`: quanto pode encolher

`align-self`: alinhamento individual

EXERCÍCIO 2

Crie um layout de 3 colunas com Flexbox:

```html
<div class="container">
  <div class="sidebar">Sidebar</div>
  <div class="main">Conteúdo Principal</div>
  <div class="sidebar">Sidebar 2</div>
</div>
```

```css
.container {
  display: flex;
  gap: 20px;
  min-height: 100vh;
}

.sidebar { flex: 1; background: #f0f0f0; }
.main { flex: 2; background: #fff; }
```

⏱ 5 minutos

CSS GRID - LAYOUT EM 2D

Grid Container:

```
.grid-container {
  display: grid;
  grid-template-columns: 200px 1fr 200px;
  grid-template-rows: auto 1fr auto;
  grid-gap: 20px;
  min-height: 100vh;
}
```

Grid Areas:

```
.header { grid-area: header; }
.main { grid-area: main; }
.footer { grid-area: footer; }
```

CSS GRID - TEMPLATE AREAS

```css
.grid-layout {
  display: grid;
  grid-template-areas:
    "header header header"
    "sidebar main aside"
    "footer footer footer";
  grid-template-columns: 200px 1fr 200px;
  grid-template-rows: auto 1fr auto;
}

.header { grid-area: header; }
.sidebar { grid-area: sidebar; }
.main { grid-area: main; }
.aside { grid-area: aside; }
.footer { grid-area: footer; }
```

FLEXBOX VS GRID

Flexbox (1D)

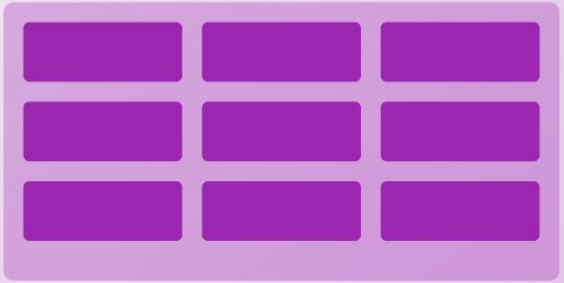
Direção principal (main axis)

Melhor para:

- Layouts unidimensionais
- Alinhamento de itens
- Distribuição de espaço
- Componentes pequenos
- Navegação horizontal

```
display: flex; justify-content: center;
```

CSS Grid (2D)



Melhor para:

- Layouts bidimensionais
- Layouts complexos
- Template areas
- Estruturas de grade
- Sobreposição
- Layouts responsivos

FLEXBOX VS GRID

Use Flexbox para:

- ☑ Layouts unidimensionais (linha OU coluna)
- ☑ Alinhamento de itens
- ☑ Distribuição de espaço
- ☑ Componentes pequenos

Use Grid para:

- ☑ Layouts bidimensionais (linhas E colunas)
- ☑ Layouts complexos de página
- ☑ Sobreposição de elementos
- ☑ Estruturas de grade definidas

EXERCÍCIO 3

Crie um layout completo com Grid:

```
.page-layout {
  display: grid;
  grid-template-areas:
    "header header"
    "sidebar main"
    "footer footer";
  grid-template-columns: 250px 1fr;
  grid-template-rows: auto 1fr auto;
  min-height: 100vh;
  gap: 20px;
}
```

⏱️10 minutos

RESPONSIVIDADE - MEDIA QUERIES

Breakpoints comuns:

```css
/* Mobile First */
.container { width: 100%; }

/* Tablet */
@media (min-width: 768px) {
  .container { width: 750px; }
}

/* Desktop */
@media (min-width: 1024px) {
  .container { width: 1000px; }
}

/* Large Desktop */
@media (min-width: 1200px) {
  .container { width: 1170px; }
}
```

UNIDADES RESPONSIVAS

Viewport Units:

`vw` - 1% da largura da viewport
`vh` - 1% da altura da viewport
`vmin` - 1% da menor dimensão
`vmax` - 1% da maior dimensão

Relative Units:

`em` - relativo ao elemento pai
`rem` - relativo ao elemento root
`%` - relativo ao elemento pai

FLEXBOX RESPONSIVO

```css
.responsive-flex {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
}

.flex-item {
  flex: 1 1 300px; /* grow shrink basis */
  min-width: 250px;
}

/* Em telas pequenas */
@media (max-width: 768px) {
  .responsive-flex {
    flex-direction: column;
  }
}
```

GRID RESPONSIVO

```
.responsive-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px,
1fr));
  gap: 20px;
}

/* Grid areas responsivas */
@media (max-width: 768px) {
  .grid-layout {
    grid-template-areas:
      "header"
      "main"
      "sidebar"
      "footer";
    grid-template-columns: 1fr;
  }
}
```

EXERCÍCIO 4

Crie um card grid responsivo:

```css
.card-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
  gap: 24px;
  padding: 20px;
}

.card {
  background: white;
  border-radius: 8px;
  padding: 20px;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  transition: transform 0.2s;
}

.card:hover {
  transform: translateY(-5px);
}
```

⏱️ 5 minutos

TÉCNICAS AVANÇADAS DE LAYOUT

Centralizando com Flexbox:

```
.center-flex {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
}
```

Centralizando com Grid:

```
.center-grid {
  display: grid;
  place-items: center;
  min-height: 100vh;
}
```

LAYOUTS COMUNS

Holy Grail Layout:

```
.holy-grail {
  display: grid;
  grid-template:
    "header header header" auto
    "nav main aside" 1fr
    "footer footer footer" auto
    / 200px 1fr 200px;
  min-height: 100vh;
}
```

Sidebar Fixa:

```
.fixed-sidebar {
  display: grid;
  grid-template-columns: 250px 1fr;
}

.sidebar {
  position: sticky;
  top: 0;
  height: 100vh;
  overflow-y: auto;
}
```

FERRAMENTAS E DEBUGGING

DevTools CSS:

Inspect Element
Box Model viewer
Grid/Flexbox overlays
Computed styles

CSS Custom Properties:

```
:root {
  --primary-color: #3498db;
  --sidebar-width: 250px;
  --gap: 20px;
}

.sidebar {
  width: var(--sidebar-width);
  background: var(--primary-
color);
}
```

PERFORMANCE E OTIMIZAÇÃO

CSS que afeta performance:

- ❌ Seletores muito específicos
- ❌ Muitos reflows/repaints
- ❌ Animações de layout

Boas práticas:

- ☑ Use `transform` e `opacity` para animações
- ☑ `will-change` para elementos animados
- ☑ Minimize seletores complexos
- ☑ Use containment quando possível

PROJETO PRÁTICO

Crie um layout de blog responsivo com:

- [x] Header fixo com navegação
- [x] Grid de posts principal
- [x] Sidebar com widgets
- [x] Footer informativo
- [x] Responsivo (mobile-first)
- [x] Hover effects nos cards
- [x] Transições suaves

Tempo: Até a próxima aula

RECURSOS AVANÇADOS

Para continuar aprendendo:

🎮 **Flexbox Froggy** - flexboxfroggy.com
🌿 **Grid Garden** - cssgridgarden.com
📖 **CSS Tricks** - css-tricks.com
🛠️ **CodePen** - codepen.io
📋 **Can I Use** - caniuse.com

Frameworks CSS:
Bootstrap, Tailwind CSS, Bulma

TENDÊNCIAS EM CSS LAYOUT

Container Queries:

```
@container (min-width: 400px) {
  .card { flex-direction: row; }
}
```

Subgrid:

```
.subgrid {
  display: grid;
  grid-template-columns: subgrid;
}
```

CSS Layers:

```
@layer base, components, utilities;
```

REVISÃO DA AULA

Aprendemos hoje:

- **Box Model** - base de todo layout
- **Flexbox** - layouts unidimensionais
- **Grid** - layouts bidimensionais

- **Responsividade** - adaptação a dispositivos
- **Posicionamento** - controle preciso
- **Técnicas avançadas** - layouts profissionais

BONS ESTUDOS