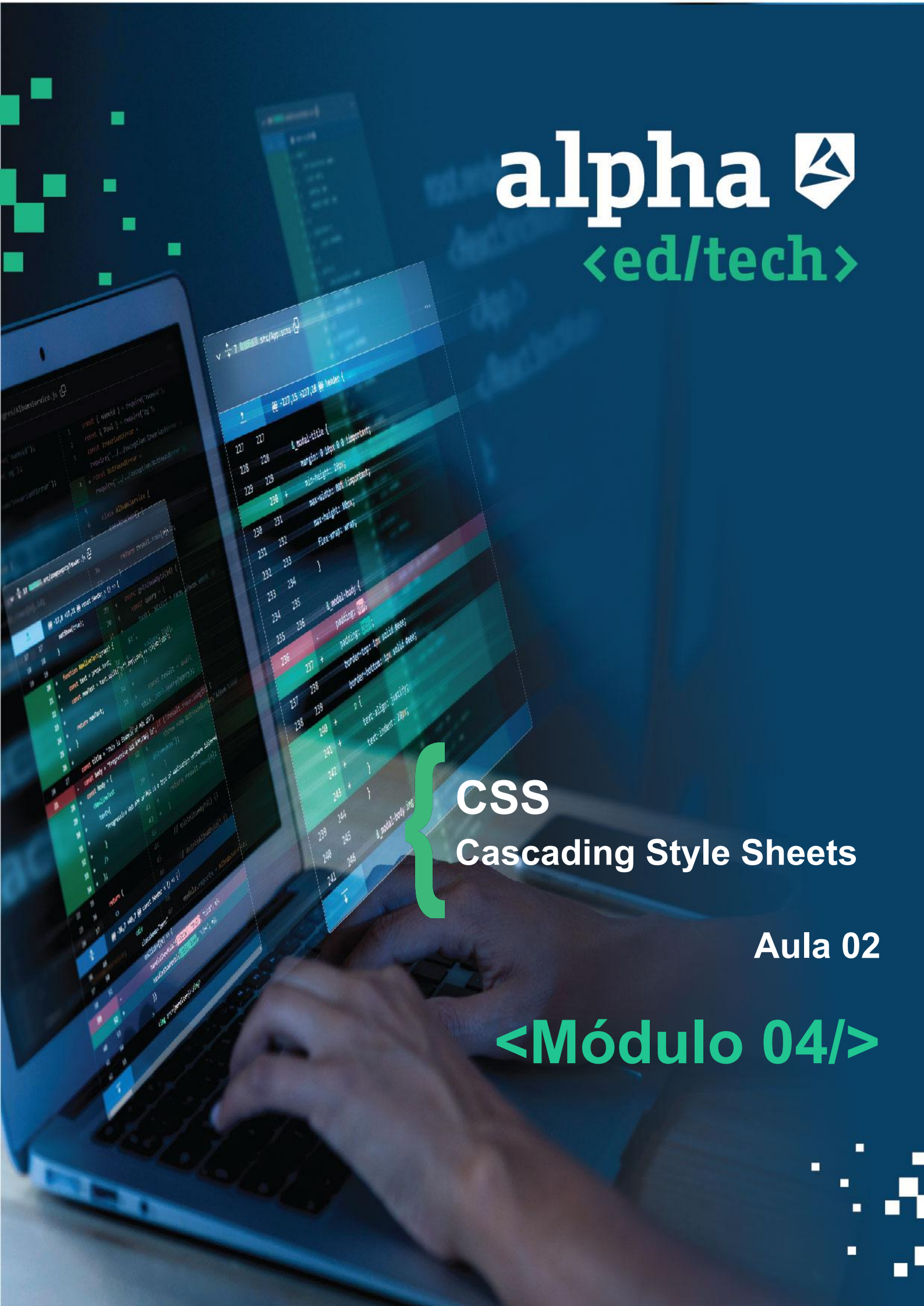

alpha
<ed/tech>
CSS
Cascading Style Sheets
Aula 02
<Módulo 04/>

Flexbox

Flexbox provê ferramentas para criação rápida de layouts complexos e flexíveis.

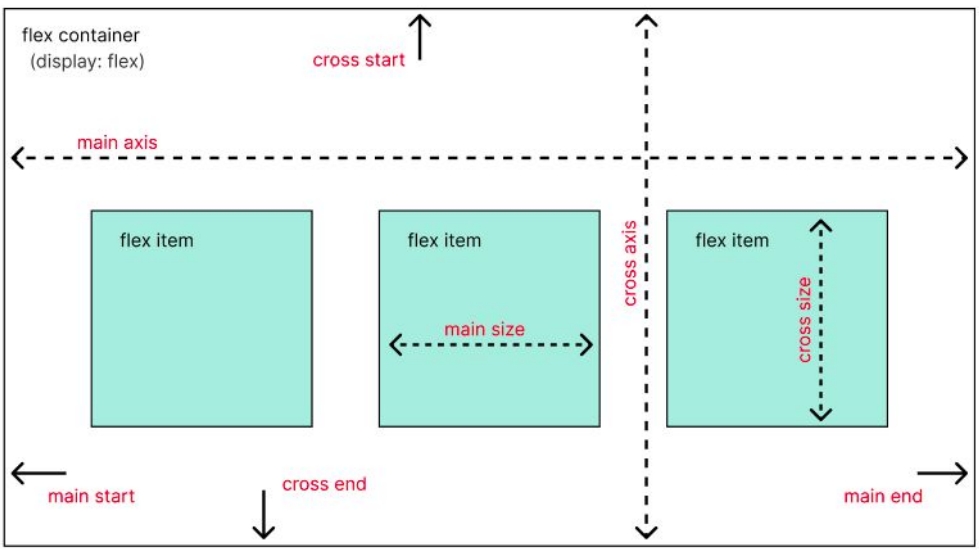
Alguns tipos de layout são muito difíceis ou impossíveis de se conseguir utilizando propriedades tais como float ou posicionamento.

Entre elas, estão:

- centralizar um bloco de conteúdo verticalmente dentro de seu pai.
- fazer com que os filhos de um container ocupe uma quantidade igual de largura/altura disponível, independente da quantidade de largura/altura disponível
- fazer todas as colunas de um layout com múltiplas colunas adotem a mesma altura, mesmo que contenham uma quantidade diferente de conteúdo.

Flexbox faz muitas tarefas de layouts de maneira mais fácil.

O modelo flex



Quando um elemento é definido como **flexible box**, terá seu conteúdo disposto em dois eixos:

- **main axis**: direção em que os itens flexíveis são dispostos, tendo início e final
- **cross axis**: eixo perpendicular ao eixo principal, tendo início e final, também chamado de eixo secundário

Para ser definido como **flexible box**, um elemento deve receber a propriedade:

Os elementos dentro do container flexível são chamados de **flexible items**.

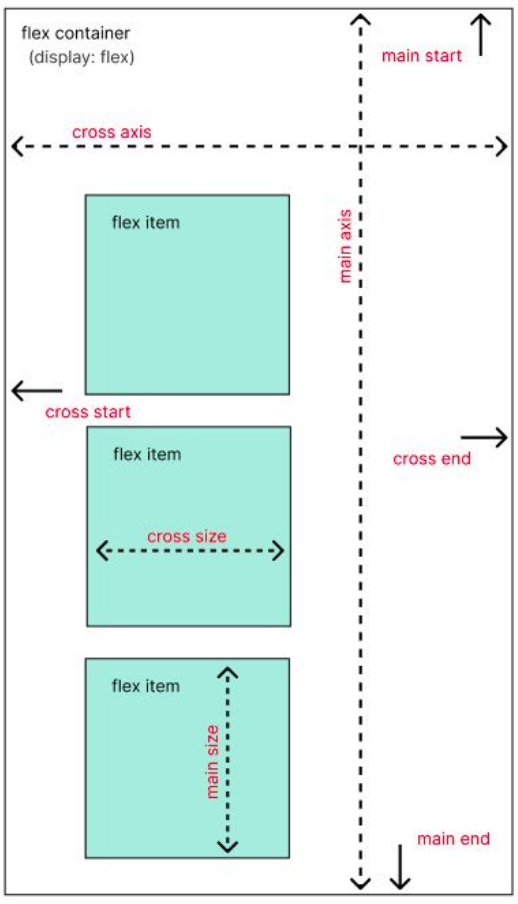
```
div {
   display: flex;
}
```

Direção principal

A direção do eixo principal em um **flex box** é definida pela propriedade flex-direction, que pode receber os seguintes valores:

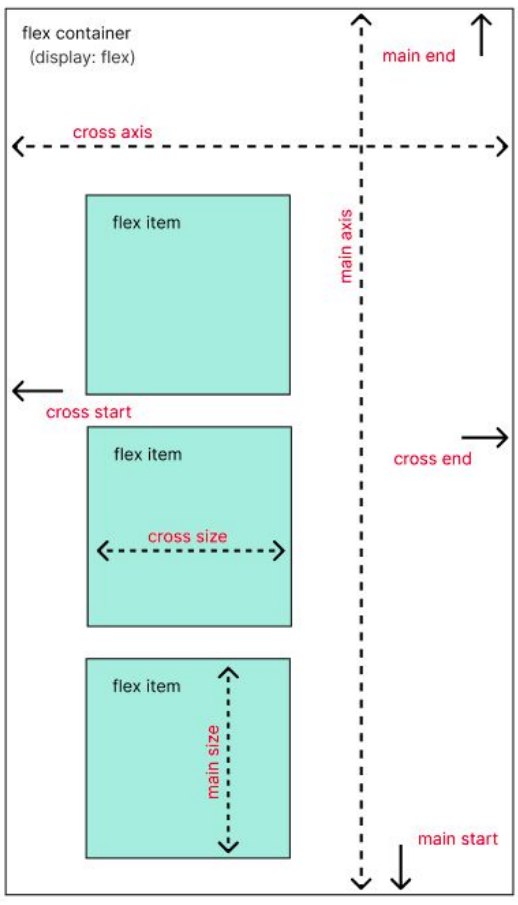
- **column**: o eixo principal é igual ao eixo do bloco, com pontos de início e final do eixo acima e abaixo do container, respectivamente

```
div {
   display: flex;
   flex-direction : column;
}
```



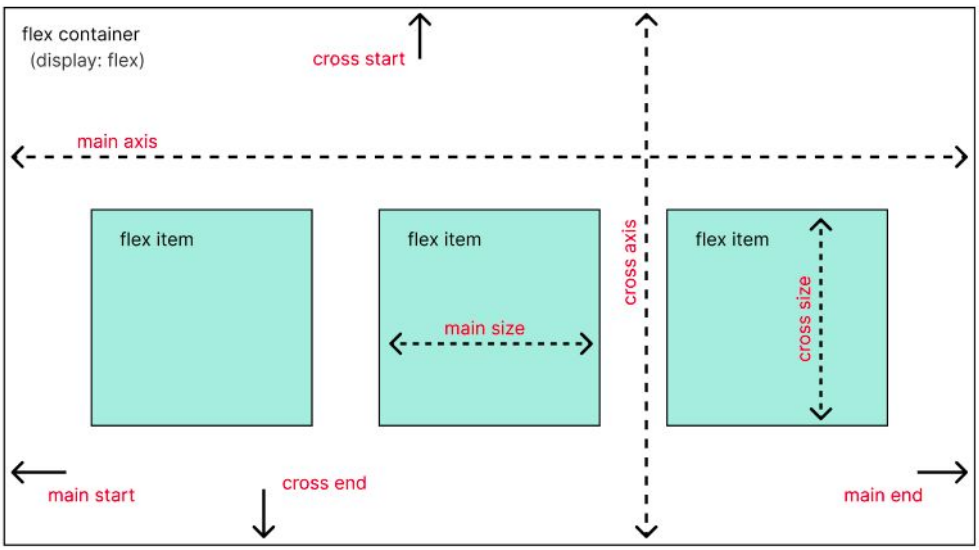
- **column-reverse**: se comporta da mesma forma que **column**, porém, com pontos de início e final invertidos

```
div {
   display: flex;
   flex-direction : column-reverse ;
}
```



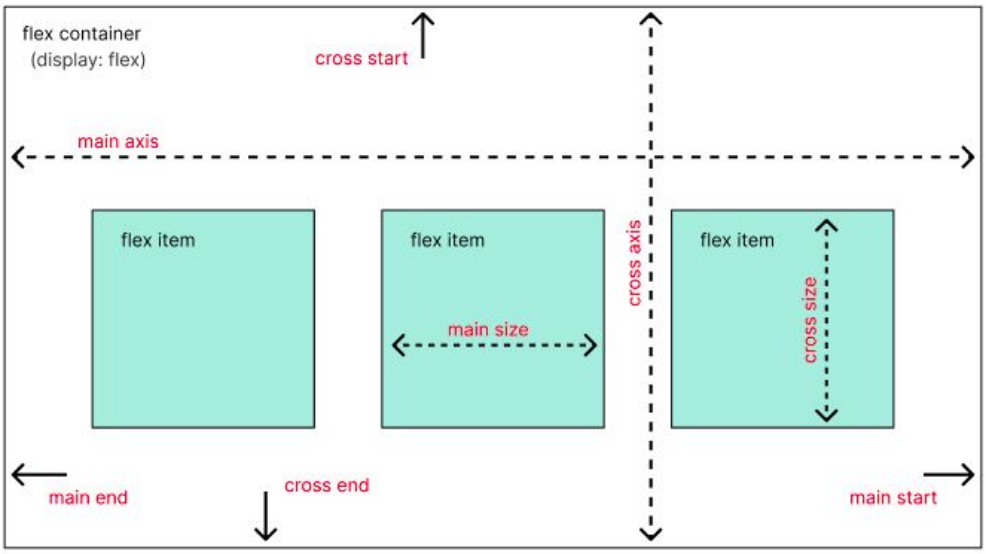
- **row**: o eixo principal é igual a direção do texto, com pontos de início e final do eixo à esquerda e à direita do container, respectivamente (valor padrão da propriedade)

```
div {
   display: flex;
   flex-direction: row;
}
```



- **row-reverse**: se comporta da mesma forma que **row**, porém, com pontos de início e final invertidos

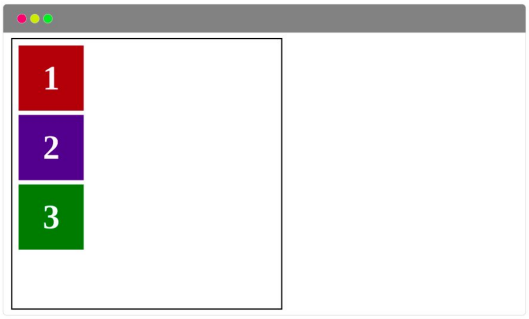
```
div {
   display: flex;
   flex-direction: row-reverse;
}
```



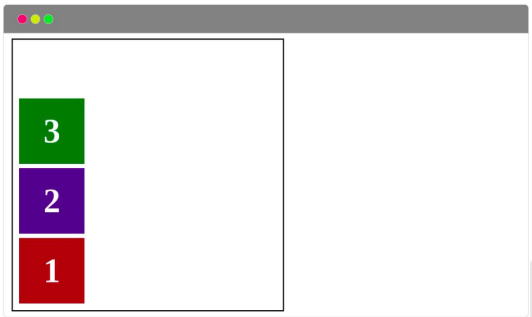
A título de exemplo, considere a seguinte página HTML:

```
<!DOCTYPE html>

<html>
    <head>
        <style>
            #container {
                display: flex;
                width: 600px;
                height: 600px;
                padding: 10px;
                border: solid 3px black;
            }

            .box {
                display: flex;
                justify-content: center;
                align-items: center;
                width: 150px;
                height: 150px;
                margin: 5px;
                color: white;
                font-size: 78px;
                font-weight: 600;
            }

            #box1 {
                background-color: darkred;
            }

            #box2 {
                background-color: darkblue;
            }

            #box3 {
                background-color: darkgreen;
            }
        </style>
    </head>

    <body>
        <div id="container">
            <div class="box"
id="box1">1</div>
            <div class="box"
id="box2">2</div>
            <div class="box"
id="box3">3</div>
        </div>
    </body>
</html>
```

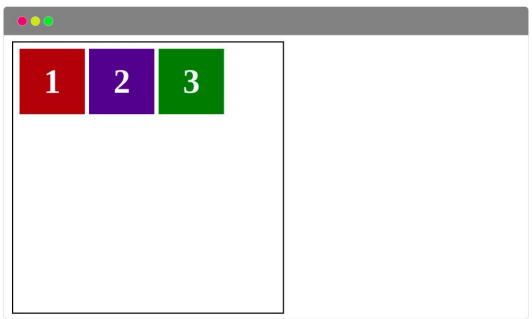
Configurando o elemento **#container** com cada valor para a propriedade **flex-direction**, teremos:



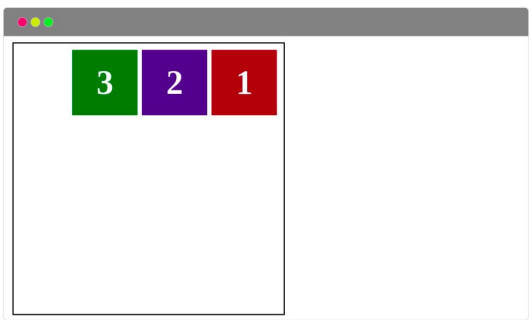
column



column-reverse



row

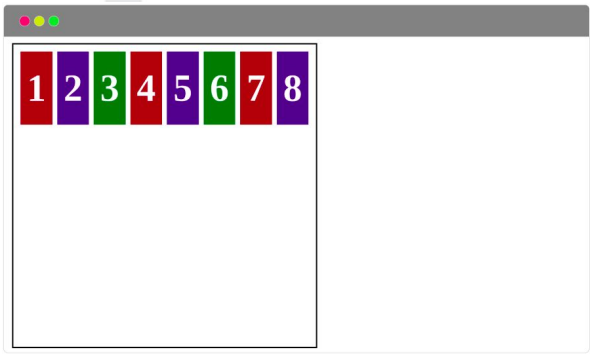


row-reverse

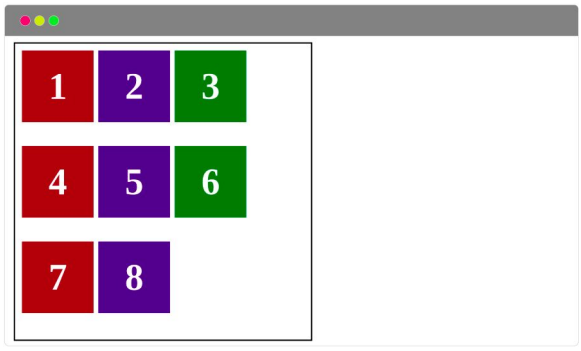
Embrulhamento

A propriedade **flex-wrap** define se os itens flexíveis são forçados a ficarem na mesma linha ou se podem ser dispostos em várias linhas.

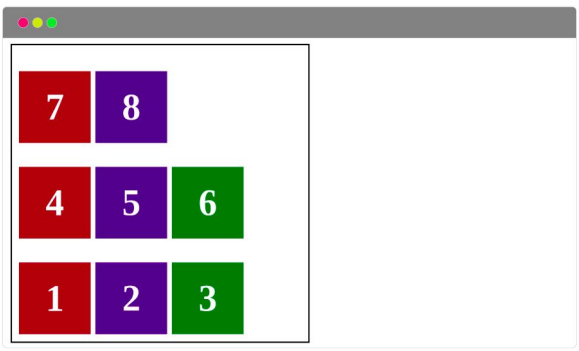
```
#container {
   display: flex;
   flex-wrap: nowrap;
}
```




```
#container {
   display: flex;
   flex-wrap: wrap;
}
```



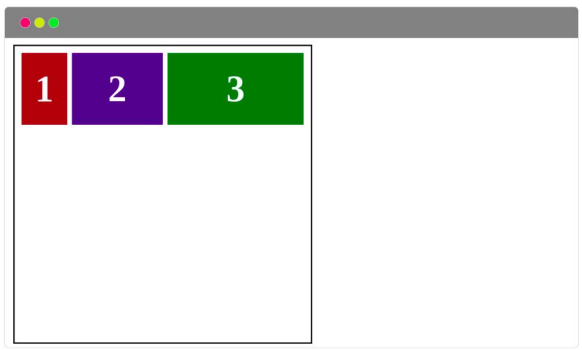

```
#container {
   display: flex;
   flex-wrap: wrap-reverse;
}
```



Dimensionamento flexível de elemento

Com a propriedade **flex** é possível definir quanto de espaço disponível pelo eixo principal cada elemento flex pode ter.

```
#box1 {
   background-color : darkred;
   flex: 1;
}

#box2 {
   background-color : darkblue;
   flex: 2;
}

#box3 {
   background-color : darkgreen;
   flex: 3;
}
```



A soma das unidades dos itens flexíveis será igual a quantidade unidades de proporção disponíveis. Cada item terá sua proporção definida pela unidade correspondente:

- **item 1**: 1/6 do espaço disponível no container
- **item 2**: 2/6 do espaço disponível no container
- **item 3**: 3/6 do espaço disponível no container

Alinhamento horizontal e vertical

As propriedades **justify-content** e **align-items** definem os alinhamentos principal e secundário dos itens flexíveis no container.

Para o eixo principal do container, temos os seguintes valores:

- **flex-start** ou **flex-end**: faz com que todos os elementos estejam no início ou no final do eixo principal, respectivamente
- **center**: fará com que os elementos flex fiquem no centro do eixo principal
- **space-around**: distribui todos os elementos igualmente pelo eixo principal, com espaço no final e no início do eixo
- **space-between**: semelhante ao space-around, mas sem o espaço no início e no final do eixo

Para o eixo secundário do container, temos os seguintes valores:

- **stretch**: estica todos os elementos flex para preencher o elemento pai na direção do eixo transversal
- **center**: faz com que os elementos mantenham suas dimensões intrínsecas, mas que seja centralizados ao longo do eixo transversal
- **flex-start** ou **flex-end**: alinham todos os elementos no início ou fim do eixo transversal, respectivamente

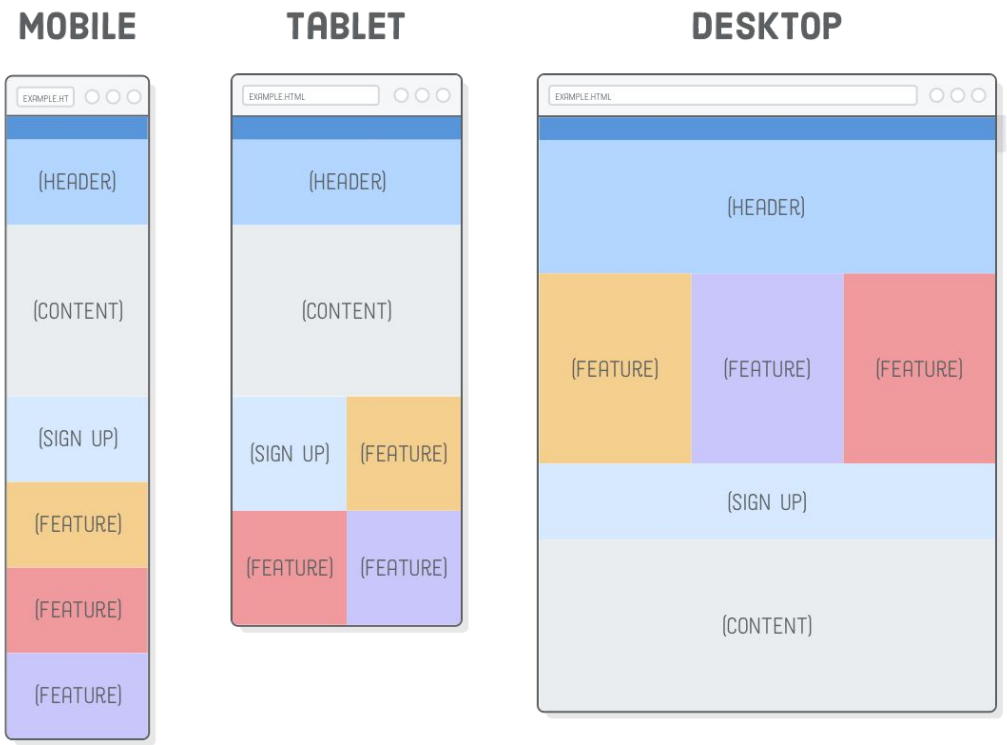
Design responsivo

O design responsivo, uma abordagem inovadora no campo do desenvolvimento web, revolucionou a maneira como os sites são visualizados e interagidos em dispositivos variados. Em um cenário digital onde a diversidade de dispositivos, tamanhos de tela e navegadores é vasta, o design responsivo emerge como uma solução inteligente e essencial.

Em sua essência, o design responsivo adapta a apresentação visual e a experiência do usuário de um site de maneira dinâmica, garantindo que o conteúdo seja exibido de forma otimizada em qualquer dispositivo, seja ele um computador desktop, tablet ou smartphone. Ao invés de criar versões diferentes do mesmo site para cada dispositivo, o design responsivo utiliza uma abordagem fluida, ajustando de forma automática elementos como layout, imagens e fontes para se adequar ao espaço disponível.

A importância do design responsivo transcende a mera estética. Com o aumento do uso de dispositivos móveis para acessar a internet, a adaptabilidade se torna crucial para oferecer uma experiência consistente e agradável aos usuários, independentemente do dispositivo que estão utilizando. Além disso, motores de busca como o Google valorizam sites responsivos, favorecendo-os em suas classificações, o que se traduz em maior visibilidade online.

O design responsivo não é apenas uma tendência, mas uma prática indispensável no desenvolvimento web contemporâneo.



Exemplo de design responsivo para diferentes dispositivos
Fonte: pinterest.com

Vantagens de um design responsivo

Alguns pontos sobre a importância de designers responsivos:

- **crescimento do mobile**: com o aumento exponencial do uso de smartphones e tablets, os usuários acessam a web a partir de uma variedade de dispositivos com diferentes tamanhos de tela. O design responsivo permite que um site se adapte harmoniosamente a essas dimensões diversas, proporcionando uma experiência consistente.

- **SEO e Visibilidade Online**: motores de busca, como o Google, valorizam sites responsivos. A adaptabilidade a diferentes dispositivos é considerada um critério importante na classificação de páginas nos resultados de pesquisa.

- **adaptação a novas tecnologias**: com o surgimento constante de novos dispositivos e tecnologias, a capacidade de adaptação é crucial. O design responsivo oferece uma base flexível que facilita a incorporação de inovações emergentes, mantendo o site relevante e funcional.

- **experiência do usuário**: o design responsivo não se trata apenas de ajustar o layout, mas de criar uma experiência fluida e coesa. Isso significa que elementos como imagens, textos e navegação são otimizados para cada dispositivo, garantindo que os usuários desfrutem de uma navegação intuitiva, independentemente do meio utilizado.

- **manutenção simplificada**: o invés de criar versões separadas para dispositivos específicos, o design responsivo simplifica a manutenção do site. As atualizações e melhorias são aplicadas globalmente, evitando a necessidade de gerenciar múltiplas versões do mesmo conteúdo.

- **economia de recursos**: a abordagem única para diferentes dispositivos simplifica o processo de desenvolvimento, resultando em economia de recursos.

Media queries

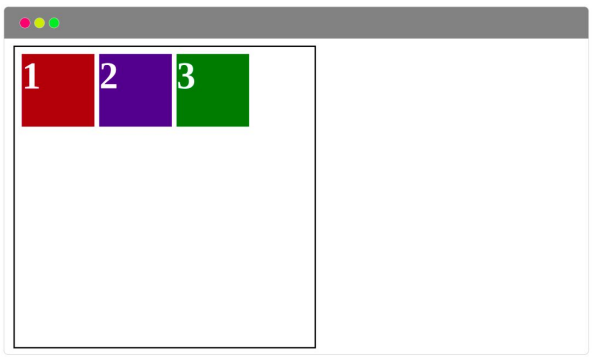
Media queries nos permitem executar uma série de testes e aplicar um CSS seletivamente para estilizar a página de acordo com as necessidades do usuário.

Media queries consistem de um **media type** e podem, a partir de uma especificação CSS, contendo uma ou mais expressões, expressas em **media features** que determinam ou verdadeiro ou falso, aplicar uma seção de estilos a uma página.

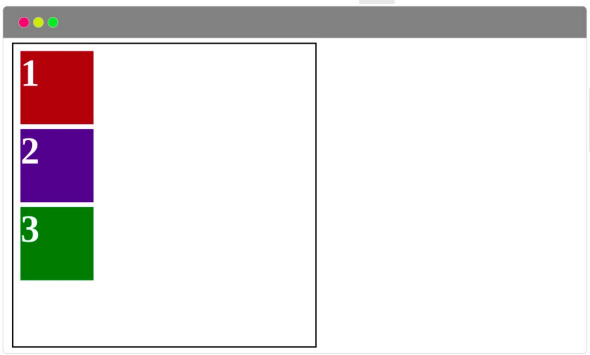
```
@media (max-width: 900px) {
   #container {
       flex-direction: column;
   }
}
```

No exemplo ao lado, para telas com no máximo **900px** de largura, o elemento **#container** terá a propriedade **flex-direction** definida com o valor **column**.

Mais de uma condição pode ser utilizada para a determinação da aplicação dos estilos. Para isso, os operadores lógicos **and** e **or** podem ser utilizados.



Tela com largura maior do que 900px



Tela com largura menor ou igual do que 900px

Múltiplos media queries podem ser adicionados a uma folha de estilo, ajustando inteiramente ao layout ou partes para que melhor se adequem a vários tamanhos de tela.

Os pontos em quem uma media query é introduzida são conhecidos como **breakpoints**.

Layouts e grids flexíveis

Sites responsivos não apenas mudam seu layout entre **breakpoints**, eles são construídos em layouts ou grids flexíveis.

Um layout ou grid flexível significa que não há necessidade de marcar todos os tamanhos possíveis existentes para as telas que podem ser utilizadas para acessar o site, e sim, construir um layout perfeito baseado em pixels que se adequa automaticamente à tela.

Flexbox

No Flexbox, os itens flexíveis irão encolher e distribuir o espaço entre os itens de acordo com o espaço em seu contêiner, conforme seu comportamento inicial.

Alterando os valores de **flex-grow** e **flex-shrink** é possível indicar como se deseja que os itens se comportem quando encontrarem mais ou menos espaço ao seu redor.

Além disso, a propriedade **flex-wrap** permite que o conteúdo seja ajustado de acordo com a capacidade do container.

Grid

No Grid Layout a unidade **fr** permite a distribuição do espaço disponível pelas trilhas da grade.

Imagens responsivas

A adoção de imagens responsivas surge como uma resposta inteligente à necessidade de apresentar conteúdo visual de forma otimizada em diferentes telas e contextos.

As imagens responsivas são projetadas para se adaptar dinamicamente aos diversos tamanhos de tela, garantindo que a qualidade visual seja mantida independentemente do dispositivo utilizado. Seja em um amplo monitor de desktop ou em uma tela pequena de smartphone, as imagens responsivas ajustam-se para proporcionar uma experiência visual consistente.

Imagens responsivas são otimizadas para reduzir o consumo de dados, carregando versões adequadas ao tamanho da tela. Isso não apenas melhora a velocidade de carregamento, mas também economiza dados, proporcionando uma experiência mais eficiente para usuários em redes móveis.

O elemento **picture** pode ser utilizado para a definição de uma série de imagens, otimizadas para cada tamanho de tela, que serão selecionadas pelo navegador, de acordo com a media feature adequada:

```
<picture>
   <source media="(max-width: 799px)" srcset="elva-480w-close-portrait.jpg" />
   <source media="(min-width: 800px)" srcset="elva-800w.jpg" />
   <img src="elva-800w.jpg" alt="Chris standing up holding his daughter Elva" />
</picture>
```

No exemplo acima, o navegador irá selecionar a imagem mais adequada, de acordo com a largura da tela.

Outra opção consiste em utilizar os novos formatos de imagens (como **WebP** e **JPEG-2000**) que podem manter baixo tamanho de arquivo e alta qualidade ao mesmo tempo. Entretanto, com menor suporte aos navegadores.

O suporte a navegadores antigos pode ser resolvido com o uso do elemento **picture**.

Tipografia responsiva

A **tipografia responsiva** descreve a alteração do tamanho das fontes nas consultas de mídia para refletir quantidades menores ou maiores de espaço na tela.

A alteração no tamanho das fontes da página pode ser configurada com a utilização da unidade de medida **rem**.

Essa unidade relaciona o tamanho da fonte com o tamanho de fonte base da página. Telas menores são configuradas com tamanho de fonte base menores, permitindo a diminuição de todas as fontes da página ao mesmo tempo.

Propriedades personalizadas

Propriedades personalizadas (às vezes chamadas de **variáveis CSS** ou **variáveis em cascata**) são entidades definidas para conter valores específicos a serem reutilizados em um documento.

O exemplo ao lado mostra como definir e como utilizar uma propriedade personalizada. Observe que a utilização se dá a partir da função **var()**.

```
#container {
   --size: 600px; /* variável CSS */

   display: flex;
   flex-direction: row;
   width: var(--size);
   height: var(--size);
   padding: 10px;
   border: solid 3px black;
}
```

Além de possibilitar reutilização, o uso de variáveis tem o benefício de criar identificadores semânticos. Por exemplo, é muito mais fácil entender que um valor representado por **--primary-color: red** representa a cor primária da página do que ler o mesmo valor de cor **red** em vários lugares diferentes e entender que essa é a cor primária da página.

É possível notar que o **escopo** de uso da variável **--size** acima consiste ao container **#container**, não sendo possível reutilizá-la em outros seletores da folha de estilo. As variáveis CSS devem ser criadas dentro de um **escopo definido**.

```
:root {
   --size: 600px; /* variável CSS */
}

#container {
   display: flex;
   flex-direction: row;
   width: var(--size);
   height: var(--size);
   padding: 10px;
   border: solid 3px black;
}
```

É uma prática recomendada comum a utilização da pseudoclasse **:root**. Dessa forma, as propriedades personalizadas poderão ser utilizadas globalmente no documento HTML.

As propriedades personalizadas são herdadas. Isso significa que, se nenhum valor for definido para uma propriedade personalizada em um determinado elemento, o valor de seu pai será usado.

Uma das principais utilizadas de propriedades personalizadas está na definição de temas. Por exemplo, imagine que você queira aplicar modo escuro na sua página. Você pode definir as seguintes propriedades personalizadas:

Qualquer elemento que possuir atributo class igual a dark-mode será estilizado de acordo com os valores das variáveis definidas ao lado.

Dessa forma, outros temas podem ser criados com diferentes valores e selecionados dinamicamente no documento HTML.

```
.dark-mode:root {
   --background: #111;
   --title: #fff;
   --paragraph: #d3d3d3;
   --element: #7357ff;
}
```

Pseudo-classes

Uma **pseudo-classe** é uma palavra-chave adicionada a selectores que especifica um estado especial do elemento selecionado.

Seu uso permite que se aplique um estilo a um elemento não apenas em relação ao conteúdo da árvore do documento, mas também em relação a fatores externos como o histórico de navegação, o status do seu conteúdo, ou a posição do mouse.

A seguir, algumas das pseudo-classes mais comuns:

- **hover**: corresponde quando o usuário designa um elemento com um dispositivo apontador, tal como um mouse, mas não necessariamente o ativa (pressiona o botão seletor do mouse, por exemplo), apenas mantendo o apontador em cima do elemento
- **checked**: representa um elemento **radio**, **checkbox** ou **option** que esteja marcado ou alterado para um estado ligado
- **first-child**: representa qualquer elemento que seja o primeiro filho de seus pais
- **last-child**: representa qualquer elemento que seja o último filho de seus pais
- **active**: corresponde quando o usuário seleciona o elemento, por exemplo, ao clicar em um botão
- **visited**: indica se um **link** já foi visitado pelo usuário
- **root**: se equipara à raiz de uma árvore, que por sua vez representa o documento
- **focus**: aplicada quando um elemento recebe foco, o que pode ocorrer quando o usuário seleciona o elemento utilizando o teclado ou ativando o mesmo com o mouse, como ao selecionar um **input** de formulário

Para adicionar estilo a partir de uma pseudo-classe, basta adicioná-la após o seletor do elemento.

No exemplo ao lado, qualquer botão que esteja sendo pressionado por um dispositivo apontador, terá o seu brilho reduzido para 90%.

```
button:active {
   filter: brightness(0.9);
}
```

Pseudo-elementos

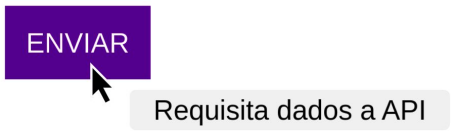
Uma **pseudo-classe** é uma palavra-chave adicionada a selectores que permite estilizar uma parte específica do elemento selecionado.

A seguir, alguns dos pseudo-elementos mais comuns:

- **after**: cria um pseudo elemento que é o último filho do elemento selecionado, com padão **inline**
- **first-letter**: seleciona a primeira letra da primeira linha de um bloco, desde que não seja precedida por outro conteúdo na mesma linha
- **before**: cria um pseudo elemento que é o primeiro filho do elemento selecionado, com padão **inline**
- **placeholder**: representa o texto do espaço reservado em um elemento **input** ou **textarea**

Para adicionar adicionar estilo a um pseudo-elemento, basta adicioná-lo após o seletor do elemento.

No exemplo ao lado, o texto "Requisita dados a API" é adicionado no pseudo-elemento **after** do botão para comunicar ao usuário uma informação importante sobre a funcionalidade do botão.




```css
button {
    width: 70px;
    height: 35px;
    position: relative;
    color: white;
    background-color : darkblue;
    border: none;
}

button:hover::after {
    content: "Requisita dados a API" ;
    width: 150px;
    padding: 2px;
    position: absolute;
    left: 60px;
    top: 40px;
    color: black;
    background-color : #f0f0f0;
    border: none;
    border-radius : 3px;
}
```

Transformações

Transformações CSS permitem que a posição do conteúdo afetado seja alterada sem afetar o fluxo de informação da página, permitindo alterações lineares em elementos HTML.

Duas propriedades são utilizadas para configurar transformações:

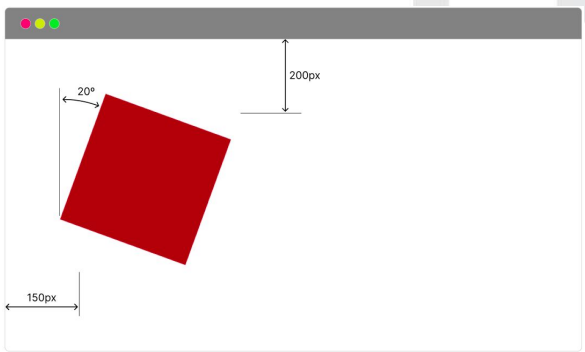
- **transform-origin**: define a posição de origem do elemento, que por padrão é a posição superior esquerda do elemento
- **transform**: define qual alteração será feita no elemento

Os seguintes tipos de transformações são possíveis:

- **translate**: movimenta o elemento na horizontal e/ou na vertical
- **rotate**: rotaciona o elemento
- **scale**: redimensiona o elemento no plano 2D
- **skewx**: inclina o elemento
- **perspective**: define perspectiva 3D no elemento

O exemplo ao abaixo movimenta o elemento 150px na horizontal e rotaciona o mesmo com um ângulo de 20 graus.

```
div {
    width: 200px;
    height: 200px;
    background-color : darkred;
    transform :
            translatex (150px)
            translatey (200px)
            rotate (20deg);
}
```



Transições

Transições possibilitam uma forma de controlar a velocidade de uma animação quando há mudanças de propriedades CSS.

Ao invés de uma propriedade entrar em vigor imediatamente, você pode fazer com que as mudanças em uma propriedade ocorram durante um periodo de tempo.

Duas propriedade precisam ser configuradas para conferir o movimento de transição a um elemento:

- **transition-property**: especifica o nome ou nomes das propriedades CSS em que as transições serão aplicadas.
- **transition-duration**: especifica a duração em que as transições devem ocorrer.

Por exemplo, imagine que você deva aplicar a pseudo-classe **hover** a um elemento que deva conferir as seguintes propriedades.

O código abaixo faz com que a transformação ocorra instantaneamente:

```
div {
    width: 200px;
    height: 200px;
    background-color : darkred;
}

div:hover {
    transform:
            translatex(150px)
            translatey(200px)
            rotate(20deg);
}
```

Ao configurar as propriedades **transition**, o movimento realizado pelo elemento pode ser executado em um período de tempo perceptível.

Como resultado, uma transição controlada é conferida ao elemento.

```
div {
    width: 200px;
    height: 200px;
    background-color : darkred;
    transition-property : transform;
    transition-duration : 2s;
}

div:hover {
    transform:
            translatex(150px)
            translatey(200px)
            rotate(20deg);
}
```

Animações

Animações CSS tornam possível animar transições de um estilo CSS para outro.

Animações consistem de dois componentes:

- um estilo descrevendo a animação
- um set de **keyframes** que indicam o estado final e inicial do estilo CSS da animação, bem como possíveis **waypoints** intermediários ao longo do caminho

Para criar uma sequência de animação, é necessário estilizar o elemento que deseja animar com a propriedade **animation** ou suas sub-propriedades.

Isso permite que seja configurada a sincronização da animação, bem como outros detalhes de como a sequência de animação deveria progredir.

As sub-propriedades da propriedade **animation** são:

- **animation-delay**: configura o delay entre o tempo em que o elemento é carregado e o início da sequência de animação
- **animation-direction**: configura se a animação deve ou não alternar a direção em cada execução durante a sequência ou voltar ao ponto inicial e se repetir
- **animation-duration**: configura o tempo que uma animação deveria levar para completar um ciclo
- **animation-iteration-count**: configura o número de vezes que uma animação deve se repetir, podendo ser configurado para repetir a animação indefinidamente.
- **animation-name**: descrevendo os keyframes da animação
- **animation-play-state**: pausa e resume a sequência da animação
- **animation-timing-function**: configura a sincronização da animação, que é como a animação transita por keyframes
- **animation-fill-mode**: configura que valores são aplicados pela animação antes e depois de se executar

Considere o seguinte exemplo:

```css
div {
    width: 200px;
    height: 200px;
    background-color : darkred;
    animation-name : movement;
    animation-duration : 3s;
    animation-direction : alternate;
    animation-iteration-count : infinite;
}
```

```css
@keyframes movement {
    from {}

    to {
        transform:
                translatex(150px)
                translatey(200px)
                rotate(20deg);
    }
}
```

O elemento **div** está configurado com a seguinte animação:

- um ciclo da animação possui um período de 3s
- o movimento é alternado e se repete infinitamente
- a animação tem como referência o keyframe **movement**
- o keyframe configura a animação como um movimento de transição que consiste em duas transações, 150px na horizontal e 200px na vertical, e uma rotação de 20º

É uma prática recomendada comum é a utilização da pseu

Filter

A propriedade **filter** aplica efeitos gráficos a um elemento, sendo comumente usados para ajustar a renderização de imagens, planos de fundo e bordas.

A propriedade recebe como valor uma série de funções:

- **blur**: aplica desfoque gaussiano ao elemento
- **brightness**: aplica um multiplicador linear, fazendo com que o elemento pareça mais ou menos brilhante
- **contrast**: ajusta o contraste do elemento
- **drop-shadow**: aplica sombreamento ao elemento
- **grayscale**: converte o elemento em tons de cinza
- **hue-rotate**: aplica rotação de matiz ao elemento
- **invert**: inverte o elemento
- **opacity**: aplica transparência ao elemento
- **saturate**: satura o elemento
- **sepia**: converte o elemento em sépia
- **url**: faz referência a um elemento de filtro SVG

É uma prática recomendada comum é a utilização da pseudo

Gradiente

Gradiente no CSS é um tipo especial que consiste em uma transição progressiva entre duas ou mais cores.

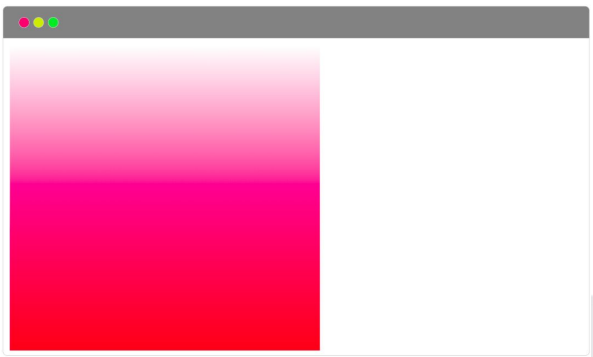
Um gradiente CSS não tem dimensões intrínsecas, isto é, não tem tamanho natural ou preferido, nem proporção preferida.

Seu tamanho concreto corresponderá ao tamanho do elemento ao qual se aplica.

Os seguintes tipos de gradientes podem ser aplicados:

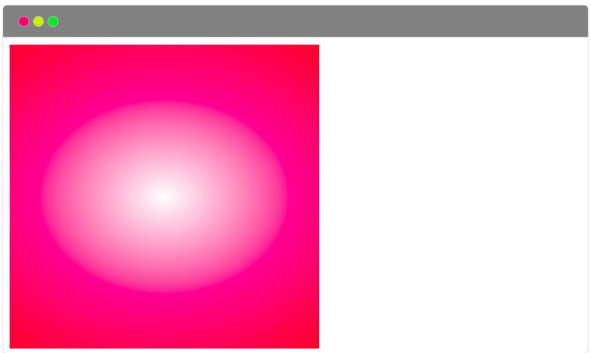
- **linear-gradient**: fazem transição de cores progressivamente ao longo de uma linha imaginária

```
div {
  width: 900px;
  height: 700px;
  background: linear-gradient (white,
red);
}
```



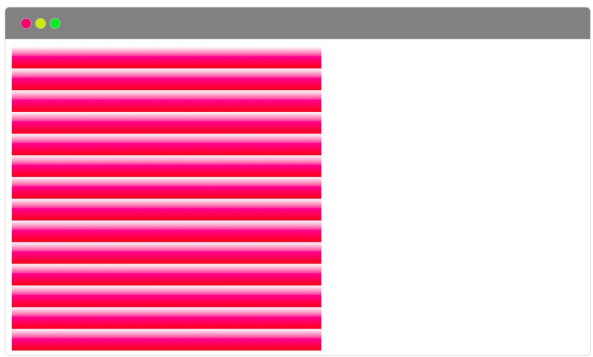
- **radial-gradient**: fazem a transição das cores progressivamente a partir de um ponto central

```
div {
  width: 900px;
  height: 700px;
  background: radial-gradient (white,
red);
}
```



- **repeating-linear-gradient**: duplica um gradiente tanto quanto necessário para preencher uma determinada área ao longo de uma linha imaginária

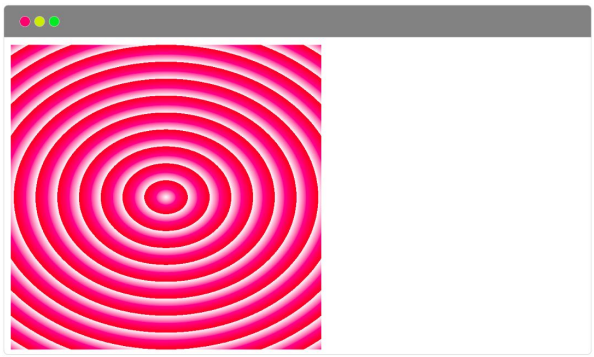
```
div {
   width: 900px;
   height: 700px;
   background:
          repeating-linear-gradient (
                 white,
                 red 50px
          );
}
```



É uma prática recomendada comum é a utilização da pseudo

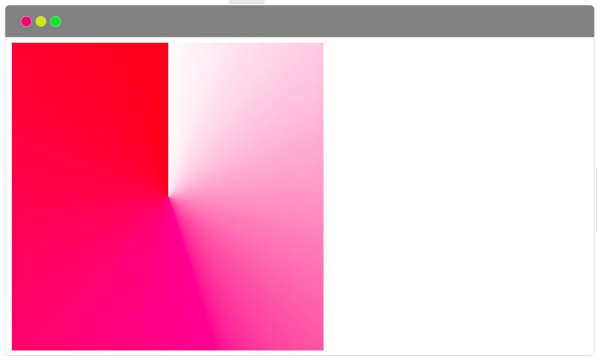
- **repeating-radial-gradient**: duplica um gradiente tanto quanto necessário para preencher uma determinada área progressivamente a partir de um ponto central

```
div {
   width: 900px;
   height: 700px;
   background:
           repeating-linear-gradient (
                   white,
                   red 50px
           );
}
```



- **conic-gradient**: fazem transição de cores progressivamente em torno de um círculo

```
div {
   width: 900px;
   height: 700px;
   background: conic-gradient (white,
red);
}
```



É uma prática recomendada comum é a utilização da pseudo

Boas práticas em CSS

Desenvolver estilos eficientes e fáceis de manter requer atenção a algumas boas práticas:

- **organização e estrutura**:
  - separe o código em arquivos modulares para facilitar a manutenção.
  - utilize uma metodologia de organização, como BEM (Block Element Modifier) ou OOCSS (Object-Oriented CSS), para criar estruturas claras e reutilizáveis

- **seletor específico**:
  - evite o uso de seletores globais, como **!important**, sempre que possível
  - prefira seletores específicos para evitar conflitos e garantir que suas regras se apliquem conforme esperado

- **responsividade**
  - adote abordagens de design responsivo para garantir uma experiência consistente em diferentes dispositivos
  - utilize unidades relativas, como **porcentagens** e **em**, para garantir que os elementos se ajustem adequadamente ao tamanho da tela

- **performance:**
  - minimize o uso de propriedades redundantes e evite estilos desnecessários
  - considere a **minificação** do seu CSS em ambientes de produção para reduzir o tempo de carregamento

- **comentários claros:**
  - documente seu código com comentários claros e informativos para facilitar a compreensão e manutenção futura
  - explique a lógica por trás de escolhas de design e soluções específicas

- **nomenclatura significativa:**
  - escolha nomes de classes e ids que reflitam o propósito do elemento, facilitando a identificação e modificação posterior
  - evite abreviações excessivas, a menos que sejam amplamente reconhecidas e compreendidas

- **uso eficiente de cores e fontes:**
  - centralize a definição de cores e fontes em variáveis para facilitar a consistência e possíveis alterações globais
  - considere o contraste para garantir a legibilidade, especialmente em textos importantes

- **teste em navegadores diferentes:**
  - verifique a compatibilidade em navegadores populares para garantir uma experiência consistente
  - utilize ferramentas como Autoprefixer para lidar com prefixos de fornecedores automaticamente

- **versionamento e controle de versão:**
  - utilize sistemas de controle de versão, como Git, para rastrear alterações e colaborar eficientemente
  - considere o uso de convenções de nomenclatura semântica para facilitar o entendimento das versões