APOSTILA DE REVISÃO V1

HTML 5 & CSS 3

Módulo 10: CSS 3 Moderno e Avançado

FORMULA WEB PRO-MAX

Selectores Direcionados

Em CSS3, os "selectores direcionados" não são uma categoria de seletores específica. No entanto, os seletores em CSS3 incluem uma variedade de técnicas para direcionar elementos específicos em uma página da web com base em critérios específicos. Abaixo você tem algumas das técnicas de seleção direcionada mais comuns em CSS3:

Tipo de Elemento (Element Type Selector):

O seletor de tipo direciona todos os elementos de um tipo específico. Por exemplo, p selecionaria todos os parágrafos na página.

Seletor de Classe (Class Selector):

O seletor de classe permite direcionar elementos com base na classe atribuída a eles. Por exemplo, .destaque direcionaria todos os elementos com a classe "destaque".

Seletor de ID (ID Selector):

O seletor de ID direciona um elemento com base em seu ID único. Por exemplo, #cabecalho direcionaria um elemento com o ID "cabecalho".

Seletor de Atributo (Attribute Selector):

O seletor de atributo permite direcionar elementos com base em um valor de atributo específico. Por exemplo, [type="text"] direcionaria todos os elementos de entrada com o atributo "type" definido como "text".

Seletor Universal (Universal Selector):

O seletor universal (*) direciona todos os elementos na página.

Seletor Filho Direto (Child Selector):

O seletor filho direto (>) direciona elementos que são filhos diretos de outro elemento. Por exemplo, ul > li direcionaria todas as listas de itens (<li>) que são filhas diretas de uma lista não ordenada (<ul>).

Seletor Adjacente (Adjacent Selector):

O seletor adjacente (+) direciona um elemento que é diretamente adjacente a outro elemento. Por exemplo, h2 + p direcionaria todos os parágrafos que são irmãos imediatos de um cabeçalho de segundo nível (<h2>).

Seletor de Pseudo-classes:

Pseudo-classes são usadas para direcionar elementos com base em estados específicos ou posições no documento. Alguns exemplos incluem :hover (quando o mouse está sobre o elemento) e :nth-child(n) (para selecionar elementos com base na ordem dentro de um pai).

Seletor de Pseudo-elementos ou Pseudo Selectores:

Pseudo-elementos permitem estilizar partes específicas de um elemento, como o primeiro caractere de um elemento ou a primeira linha de um parágrafo. Exemplos incluem **::first-letter** e **::before**.

Seletor de Descendência (Descendant Selector):

O seletor de descendência (espaço) permite direcionar elementos que são descendentes de outro elemento. Por exemplo, div p direcionaria todos os parágrafos que estão dentro de um elemento .

Estas são algumas das técnicas de seleção direcionada em CSS3. Elas permitem que você aplique estilos a elementos específicos em seu documento HTML, tornando a estilização de páginas da web mais flexível e personalizável. Você pode combinar esses seletores para criar regras CSS complexas que atendam às suas necessidades de design.

Pseudo Selector nth-child

O :nth-child é um seletor de pseudo-classe em CSS que permite selecionar elementos com base na sua posição relativa dentro de um pai. Ele é usado para aplicar estilos a elementos que correspondem a uma fórmula matemática especificada, definida por an + b, onde n representa o índice do elemento (começando em 1), a e b são números inteiros. Isso permite que você aplique estilos a elementos específicos com base em sua posição dentro de um elemento pai.

A sintaxe básica do :nth-child é a seguinte:

```
:nth-child(an + b) {
  /* Estilos a serem aplicados aos elementos correspondentes */
}
```

Aqui estão alguns exemplos de como o **:nth-child** pode ser usado:

Selecionar elementos pares/ímpares:

Você pode usar 2n para selecionar todos os elementos pares e 2n+1 para selecionar todos os elementos ímpares. Por exemplo:

```
/* Estilos para elementos pares */
:nth-child(2n) {
  background-color: #f2f2f2;
}
/* Estilos para elementos ímpares */
:nth-child(2n+1) {
  background-color: #ffffff;
}
```

Para selecionar elementos ímpares e pares você ainda usar even (par) e odd (impar) de forma mais fácil, exemplo:

```
/* Estilos para elementos pares */
:nth-child(even) {
  background-color: #f2f2f2;
}
/* Estilos para elementos ímpares */
:nth-child(odd) {
  background-color: #ffffff;
}
```

Selecionar o primeiro elemento:

Para selecionar o primeiro elemento dentro de um pai, use :nth-child(1):

```
/* Estilos para o primeiro elemento */
:nth-child(1) {
  font-weight: bold;
}
```

Selecionar elementos a partir de uma determinada posição:

Você pode selecionar elementos a partir de uma posição específica com an + b, onde a e b são números inteiros. Por exemplo, para selecionar elementos a partir do terceiro elemento em diante:

```
/* Estilos para elementos a partir do terceiro */
:nth-child(n+3) {
  color: #ff0000;
}
```

Selecionar um elemento em uma posição específica:

Para selecionar um elemento em uma posição específica, defina a como 0 e b como o índice do elemento desejado. Por exemplo, para selecionar o quarto elemento:

```
:nth-child(4) { /* Estilos para o quarto elemento */
  text-decoration: underline;
}
```

Selecionar vários elementos variando com base a um número:

Vamos supor que você selecionar todos os elementos na posição 3 ou melhor ainda múltiplos de 3 Isto seria assim:

```
:nth-child(3n) { /* Seleciona todos elementos cuja sua posição é múltiplo de 3 */
  text-decoration: underline;
}

:nth-child(3n+9) { /* Seleciona todos elementos cuja sua posição é múltiplo de 3 apartir do elemento número 9 */
  text-decoration: underline;
}
```

O **:nth-child** é uma ferramenta poderosa para aplicar estilos com base na posição dos elementos, permitindo uma personalização mais refinada da aparência de elementos em uma página da web. Você pode usar essa pseudo-classe em combinação com outros seletores e propriedades CSS para criar layouts e estilos complexos e responsivos.

Pseudo Selectores Before e After

::before e **::after** - Permitem inserir conteúdo antes ou depois do conteúdo de um elemento, criando elementos virtuais. Você pode estilizar esses elementos adicionais como se fossem parte do documento.

```
p::before {
  content: "Antes: ";
  font-weight: bold;
}
```

```
p::after {
  content: " Depois";
  font-style: italic;
}
```

Isso adicionará "Antes: " antes de cada parágrafo e " Depois" após cada parágrafo.

Pseudo Selectores not, first-child e last-child

:first-child - Seleciona o primeiro elemento filho de um elemento pai. É útil para aplicar estilos específicos ao primeiro elemento em um container.

```
li:first-child {
}
```

Isso aplicará a formatação em negrito ao primeiro elemento <li> que é filho direto de um elemento pai, como uma lista não ordenada (<ul>) ou uma lista ordenada (<ol>).

:last-child - Seleciona o último elemento filho de um elemento pai. Pode ser usado para estilizar o último item em uma lista, por exemplo.

```
li:last-child {
}
```

:not - Selecionar elementos que não correspondem a um seletor específico:

```
a:not(.btn) {
}
```

Neste exemplo, todos os links (<a>) que não têm a classe "botao" terão uma linha de sublinhado.

Pseudo Selectores Mais Usados Para Textos

::first-line para estilizar a primeira linha de um parágrafo:

```
p::first-line {
  font-weight: bold;
  color: blue;
}
```

Isso tornará a primeira linha de cada parágrafo em negrito e com cor azul.

::first-letter para estilizar a primeira letra de um parágrafo:

```
p::first-letter {
  font-size: 150%;
  color: red;
}
```

Isso aumentará o tamanho e alterará a cor da primeira letra em cada parágrafo.

::selection para estilizar texto selecionado:

```
::selection {
  background-color: yellow;
  color: black;
}
```

Isso fará com que o texto selecionado pelo usuário tenha um fundo amarelo e texto preto.

::placeholder para estilizar o texto de um campo de entrada HTML:

```
input::placeholder {
  color: gray;
}
```

Isso alterará a cor do texto de espaço reservado (placeholder) em campos de entrada para cinza.

::marker para estilizar os marcadores em listas:

```
li::marker {
  color: green;
}
```

Isso alterará a cor dos marcadores em listas para verde.

Lembre-se de que a compatibilidade com navegadores pode variar para alguns pseudo-elementos, especialmente os mais avançados. Certifique-se de verificar a compatibilidade do navegador ao usar pseudo-elementos em seu projeto.

Sombras em CSS 3

Em CSS 3, você pode criar sombras para elementos de várias maneiras usando as propriedades **box-shadow** e **text-shadow**. As sombras são usadas para adicionar profundidade e destacar elementos em uma página da web. Vou explicar cada uma dessas propriedades em detalhes:

box-shadow (sombra em elementos/caixas)

A propriedade box-shadow é usada para criar sombras em torno de um elemento (como caixas ou divs). Ela aceita valores que controlam a posição, o desfoque, o espalhamento e a cor da sombra. Aqui está a sintaxe básica:

box-shadow: [horizontal] [vertical] [desfoque] [espalhamento] [cor];

- **horizontal:** A posição horizontal da sombra em relação ao elemento. Valores positivos deslocam a sombra para a direita, enquanto valores negativos a deslocam para a esquerda.
- **vertical:** A posição vertical da sombra em relação ao elemento. Valores positivos movem a sombra para baixo, enquanto valores negativos movem a sombra para cima.
- **desfoque:** Controla o desfoque da sombra. Quanto maior o valor, mais difusa será a sombra.
- **espalhamento:** Controla o espalhamento da sombra. Valores positivos aumentam a sombra e valores negativos a reduzem.
- **cor:** Define a cor da sombra.

Aqui está um exemplo de como aplicar uma sombra a um elemento:

```
.box {
  box-shadow: 10px 10px 20px rgba(0, 0, 0, 0.5);
}
```

Isso criará uma caixa com uma sombra que se desloca 10 pixels para a direita e 10 pixels para baixo, com um desfoque de 20 pixels e uma cor preta semitransparente.

text-shadow (sombra de texto)

A propriedade text-shadow é usada para criar sombras em torno do texto dentro de um elemento. Sua sintaxe é semelhante à box-shadow:

text-shadow: [horizontal] [vertical] [desfoque] [cor];

Por exemplo:

```
h1 {
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
}
```

Isso aplicará uma sombra de texto ao título <h1>, deslocando-o 2 pixels para a direita, 2 pixels para baixo, com um desfoque de 4 pixels e uma cor preta semitransparente.

Ambas as propriedades **box-shadow** e **text-shadow** são altamente personalizáveis, e você pode ajustar os valores para alcançar o efeito desejado. Eles são amplamente usados para criar elementos com uma aparência tridimensional e destacar partes importantes do conteúdo em uma página da web.

Variáveis CSS 3

Variáveis em CSS, também conhecidas como variáveis personalizadas ou variáveis CSS, são recursos introduzidos no CSS3 que permitem armazenar valores para reutilização em todo o seu arquivo CSS. Elas são denotadas por nomes que começam com dois hífens (--) e podem conter qualquer valor CSS válido. As variáveis tornam o código CSS mais organizado, legível e fácil de manter, permitindo que você defina valores em um único lugar e os utilize em várias regras CSS.

Aqui está como você pode definir, usar e beneficiar-se das variáveis CSS:

Definindo uma Variável

Para definir uma variável, use a sintaxe --nome-da-variavel seguida pelo valor que você deseja atribuir a ela. Isso normalmente é feito dentro de um seletor raiz, como o **:root**

pois colocando dentro do seletor raiz as variáveis poderão ser usadas em qualquer elemento.

```
:root {
  --cor-primaria: #007bff;
  --tamanho-fonte: 16px;
}
```

Neste exemplo, criamos duas variáveis: --cor-primaria com a cor azul #007bff e

--tamanho-fonte com o valor 16px. Lembrando que você pode ainda definir uma variável dentro de um seletor específico por exemplo:

```
.caixa{
--largura: 300px;
}
```

A variável –-largura poderá ser utilizada apenas por elementos que conterem a classe caixa independentemente de tal elemento conter outras classes como por exemplo:

```
<div class=”caixa coluna cartaz”></div>
```

Usando uma Variável

Para usar uma variável, você a referencia com a função var() e fornece o nome da variável como argumento. Por exemplo:

```
.botao {
  background-color: var(--cor-primaria);
  font-size: var(--tamanho-fonte);
}
```

Neste caso, a classe .botao usará a cor definida na variável --cor-primaria para o fundo e o tamanho de fonte definido na variável --tamanho-fonte.

Benefícios das Variáveis CSS

- **Reutilização e consistência:** Variáveis permitem que você reutilize valores em várias partes do seu CSS. Se você deseja alterar uma cor ou tamanho, basta atualizar a variável, e todas as instâncias da variável serão atualizadas automaticamente.
- **Manutenção simplificada:** Manter um código CSS organizado e atualizado é mais fácil com variáveis, pois você pode fazer alterações em um único local e ver os efeitos em todo o estilo.
- **Leitura e compreensão aprimoradas:** Variáveis tornam seu código mais legível e compreensível, pois fornecem nomes significativos para os valores em vez de valores numéricos ou hexadecimais diretamente no código.
- **Escopo local:** Variáveis CSS têm escopo local, o que significa que você pode definir variáveis em um escopo específico, como dentro de um componente, sem afetar outras partes do seu código.
- **Facilitação de temas:** Variáveis são úteis para criar temas em seu site, onde você pode alternar facilmente entre esquemas de cores ou estilos ao alterar os valores das variáveis.

As variáveis CSS são uma adição poderosa ao arsenal de ferramentas de estilo, tornando seu código mais eficiente, flexível e fácil de manter. Elas são suportadas em todos os navegadores modernos, tornando-as uma técnica amplamente utilizada na criação de estilos em páginas da web.

Funções Matemáticas em CSS 3

As funções matemáticas do CSS permitem realizar operações matemáticas em valores numéricos nas suas Cascading Style Sheets (CSS) para obter efeitos de estilo e layout dinâmicos. Essas funções são particularmente úteis quando você precisa calcular valores com base na entrada do usuário, dimensões da viewport ou outras variáveis. Aqui estão algumas funções matemáticas CSS comumente usadas:

1. **calc():** A função calc() é a função matemática CSS mais versátil. Ela permite realizar adição, subtração, multiplicação e divisão em valores numéricos e pode combinar diferentes unidades (por exemplo, pixels e porcentagens). Por exemplo:

   ```
   width: calc(50% - 20px);
   ```

2. **min() e max():** Essas funções retornam o valor mínimo ou máximo de uma lista de valores. Você pode usá-las para garantir que uma propriedade nunca fique abaixo ou acima de um determinado limite. Por exemplo:

   ```
   width: min(300px, 50%);
   height: max(200px, 30%);
   ```

3. **clamp():** A função clamp() ajuda a definir um valor dentro de uma faixa especificada. Ela recebe três argumentos: um valor mínimo, um valor preferido e um valor máximo. O valor preferido é usado se estiver dentro da faixa especificada; caso contrário, o valor mínimo ou máximo é usado. Por exemplo:

   ```
   font-size: clamp(16px, 5vw, 24px);
   ```

Propriedade Overflow

A propriedade CSS overflow é usada para controlar como o conteúdo que excede as dimensões de um elemento deve ser tratado. Ela é especialmente útil quando você tem elementos com dimensões fixas e o conteúdo dentro desses elementos é maior do que o espaço disponível. A propriedade overflow pode ter os seguintes valores:

- **visible (padrão):** O conteúdo que excede as dimensões do elemento é visível fora de suas bordas. Isso significa que o conteúdo pode se sobrepor a outros elementos na página.
- **hidden:** O conteúdo que excede as dimensões do elemento é cortado e não é visível. Isso cria uma "janela" que mostra apenas a parte do conteúdo dentro do elemento e oculta o restante.
- **scroll:** Uma barra de rolagem é exibida para permitir que o usuário role para ver o conteúdo que excede as dimensões do elemento. Mesmo se o conteúdo não exceder as dimensões, uma barra de rolagem inativa pode ser mostrada.
- **auto:** Semelhante a scroll, uma barra de rolagem é exibida apenas se o conteúdo exceder as dimensões do elemento. Caso contrário, não há barra de rolagem.

Aqui está um exemplo de uso da propriedade overflow:

```
div {
  width: 200px;
  height: 100px;
  border: 1px solid #ccc;
  overflow: auto; /* Pode ser qualquer um dos valores acima */
}
```

Neste exemplo, se o conteúdo dentro da exceder as dimensões especificadas (200px de largura por 100px de altura), uma barra de rolagem será exibida automaticamente para permitir que o usuário role e veja o conteúdo oculto.

A propriedade overflow é especialmente útil em elementos como , <textarea>, ou qualquer outro elemento que possa conter conteúdo que não caiba inteiramente dentro de suas dimensões especificadas. Ela permite um melhor controle sobre como o conteúdo excedente é tratado e exibido.

Cores Gradientes

Cores gradientes permitem criar efeitos visuais suaves e atraentes em elementos HTML usando uma transição de cor de um ponto para outro. Em CSS, você pode definir gradientes de duas maneiras principais: gradientes lineares e gradientes radiais. Aqui está uma explicação sobre cada um deles:

Gradientes Lineares:

Os gradientes lineares criam uma transição de cor em uma linha reta, definindo um ponto de partida e um ponto de término para a transição. Você pode especificar as cores, direção e o ponto onde a transição ocorre. Aqui está um exemplo de um gradiente linear simples:

```
/* Gradiente linear de cima para baixo, de vermelho para azul */
background-image: linear-gradient(to bottom, red, blue);
```

Neste exemplo, o gradiente começa com a cor vermelha no topo e termina com a cor azul na parte inferior. Você pode personalizar a direção e adicionar mais cores para criar efeitos mais complexos.

Gradientes Radiais:

Os gradientes radiais criam uma transição de cor a partir de um ponto central em direção às bordas do elemento. Você pode definir a posição central, as cores e o tamanho da transição. Aqui está um exemplo de um gradiente radial:

```
/* Gradiente radial do centro para fora, de amarelo para verde */
background-image: radial-gradient(circle, yellow, green);
```

Neste caso, o gradiente começa no centro com a cor amarela e se espalha para fora, terminando com a cor verde. Você pode ajustar o tamanho e a forma do gradiente radial conforme necessário.

Gradientes com Cores Personalizadas:

Além de usar cores simples como no exemplo acima, você também pode criar gradientes com cores personalizadas ou até mesmo gradientes transparentes. Por exemplo:

```
/* Gradiente linear personalizado de vermelho para amarelo para verde */
background-image: linear-gradient(to right, red, yellow, green);
/* Gradiente radial com cores transparentes */
background-image: radial-gradient(circle, transparent, rgba(0, 0, 0, 0.5));
```

Com gradientes, você pode criar uma ampla variedade de efeitos visuais em elementos HTML, como botões, fundos de divs, cabeçalhos, e muito mais. Experimente diferentes cores, direções e tamanhos para alcançar o efeito desejado em seu design web.

Visibilidade e Opacidade

A visibilidade e a opacidade são usados para controlar a visibilidade de elementos HTML em uma página da web, mas eles têm diferentes efeitos e propósitos. Para definir a visibilidade e opacidade são usada 3 propriedades:

1. visibility (visibilidade): ela controla a visibilidade de um elemento, mas não afeta o espaço que o elemento ocupa na página. Ela aceita dois valores principais:

visible: O elemento é visível (esse é o valor padrão).

hidden: O elemento fica invisível, mas ainda ocupa espaço na página.

Exemplo:

```
.box {
  visibility: hidden;
}
```

Nesse exemplo, o elemento com a classe .box fica invisível, mas ainda deixa um espaço vazio na página onde ele deveria estar.

2. opacity (opacidade): a propriedade CSS opacity controla o quão transparente um elemento é. Ela varia de 0 (totalmente transparente) a 1 (totalmente opaco). Valores intermediários, como 0.5, resultam em uma transparência parcial. Exemplo:

```
.opacity-example {
  opacity: 0.5;
}
```

Nesse exemplo, o elemento com a classe .opacity-example fica com 50% de opacidade, tornando-o parcialmente transparente

3. display: none (exibição: nenhum): A propriedade CSS display controla como um elemento é exibido na página. Quando definida como none, o elemento é completamente removido do fluxo de layout da página e não ocupa nenhum espaço.

Exemplo:

```
.box {
  display: none;
}
```

Nesse exemplo, o elemento com a classe .box não será exibido na página e não ocupará espaço na renderização.

A escolha entre essas propriedades depende dos requisitos específicos do seu design e de como você deseja controlar a visibilidade e a interação com os elementos em sua página da web.

Propriedade Object-fit

object-fit é controla como uma imagem (ou outro elemento substituído, como um vídeo) é redimensionada e ajustada dentro de seu container. Ela é muito útil quando você deseja controlar o comportamento de dimensionamento e corte de imagens em elementos, como <img> ou <video>. Existem várias opções para esta propriedade:

1. **fill (preencher)**: A imagem é dimensionada para preencher completamente o contêiner, mesmo que isso signifique distorcer a proporção original. Isso pode resultar em uma imagem esticada ou comprimida.

```
.object-fit-example {
object-fit: fill;
}
```

2. **contain (contido)**: A imagem é dimensionada para que seu conteúdo seja completamente visível dentro do container, mantendo a proporção original. Pode haver espaços vazios nas bordas do container.

```
.object-fit-example {
  object-fit: contain;
}
```

3. **cover (cobrir):** A imagem é dimensionada para cobrir completamente o container, mantendo a proporção original. Isso pode resultar em parte da imagem sendo cortada se a proporção do container for diferente da imagem.

```
.object-fit-example {
  object-fit: cover;
}
```

4. **none (nenhum):** A imagem não é dimensionada ou ajustada de forma alguma. Ela mantém seu tamanho original e pode transbordar o container se for maior.

```
.object-fit-example {
  object-fit: none;
}
```

5. **scale-down (escala para baixo):** A imagem é dimensionada para que seu conteúdo seja completamente visível dentro do container, mantendo a proporção original, mas nunca será maior do que o tamanho original da imagem.

```
.object-fit-example {
  object-fit: scale-down;
}
```

A propriedade object-fit é especialmente útil em situações em que você deseja controlar como uma imagem se ajusta a um espaço específico sem distorcer sua proporção original. Ela oferece flexibilidade no dimensionamento de imagens dentro de elementos e pode melhorar a consistência visual em layouts responsivos.

Proporção em CSS

A propriedade aspect-ratio é uma propriedade CSS relativamente nova que permite definir a proporção de aspecto de um elemento. Ela foi introduzida como parte do CSS Contain, uma especificação que visa fornecer maior controle sobre o dimensionamento e layout de elementos.

A proporção de aspecto (aspect ratio em inglês) é a relação entre a largura e a altura de um elemento. Por exemplo, uma proporção de aspecto comum é 16:9, que é frequentemente usada em vídeos e telas widescreen. Outras proporções de aspecto incluem 4:3 (usada em monitores antigos), 1:1 (uma proporção de aspecto quadrada), entre outras.

A propriedade **aspect-ratio** permite definir essa relação diretamente em um elemento, sem precisar usar truques de CSS ou elementos extras para atingir o mesmo efeito. Ela aceita dois valores: o primeiro valor define a largura e o segundo valor define a altura da proporção de aspecto. Aqui está um exemplo de como usar a propriedade aspect-ratio:

```
div {
  width: 200px;
  aspect-ratio: 16/9; /* Define uma proporção de aspecto de 16:9 */
}
```

Neste exemplo, a terá uma largura de 200px e uma altura calculada automaticamente com base na proporção de aspecto 16:9. Isso significa que a altura será ajustada automaticamente para manter a proporção correta.

A propriedade aspect-ratio é particularmente útil para criar layouts responsivos e controlar o dimensionamento de elementos, especialmente quando se trabalha com imagens ou vídeos incorporados. Ela simplifica o processo de manter proporções de aspecto corretas em vários tamanhos de tela. É importante observar que, como uma propriedade relativamente nova, a compatibilidade com navegadores pode variar, e é sempre aconselhável verificar a documentação atualizada e usar fallbacks quando necessário para garantir uma experiência consistente em todos os navegadores.

Animações com keyframe

As animações em CSS3 permitem criar efeitos de animação suaves e interativos em elementos HTML sem a necessidade de JavaScript ou bibliotecas externas. As animações em CSS3 são baseadas em transições de propriedades ao longo do tempo e podem ser usadas para adicionar elementos de design atraentes e melhorar a experiência do usuário em uma página da web.

Aqui estão os principais conceitos e propriedades relacionados a animações em CSS3:

@keyframes

@keyframes é uma regra CSS especial que define os estágios de uma animação. Ela especifica como uma propriedade deve mudar em diferentes momentos durante a animação.

Exemplo de uso de @keyframes:

```
@keyframes moverElemento {
  from {
    transform: translateX(0);
  }
  to {
    transform: translateX(200px);
  }
}
.elemento {
  width: 50px;
```

```
  height: 50px;
  background-color: blue;
  animation: moverElemento 3s ease-in-out 1s infinite alternate;
}
```

Neste exemplo:

@keyframes moverElemento - define a animação chamada "moverElemento".

from - representa o estado inicial da animação, onde o elemento começa com transform: translateX(0);.

to - representa o estado final da animação, onde o elemento se move para a direita com transform: translateX(200px);.

.elemento é o seletor do elemento HTML que deseja animar.

animation: moverElemento 3s ease-in-out 1s infinite alternate; aplica a animação "moverElemento" ao elemento. A animação dura 3 segundos, tem uma função de temporização "ease-in-out", um atraso de 1 segundo, é repetida infinitamente e alterna entre ida e volta.

Este exemplo fará com que o elemento se mova suavemente da esquerda para a direita e de volta infinitamente, com uma pausa de 1 segundo antes de começar novamente.

Para além de definir estágio inicial e estágio final da animação como demonstrado no exemplo acima nesse from (de) e to (para), os keyframes podem definir estágios de uma animação usando percentagem que representa nesse caso a duração da animação.

Exemplo de uso de **@keyframes:**

```
@keyframes mover {
  0% {
```

```
    transform: translateX(0);
  }
  50% {
    transform: translateX(100px);
  }
  100% {
    transform: translateX(200px);
  }
}
```

animation – a propriedade animation é usada para aplicar animações a elementos HTML. Ela é uma combinação de várias propriedades, como **animation-name**, **animation-duration**, **animation-timing-function**, **animation-delay**, **animation-iteration-count**, **animation-direction**, **animation-fill-mode**, e **animation-play-state**. Lembrando que keyframes e animation andam sempre juntos, keyframes para definir os estágios da animação e animation para poder animar o determinado elemento e definir outras propriedades da animação do geral.

Exemplo do uso de animation e keyframes:

```
@keyframes mover {
  0% {
    transform: translateX(0);
  }
  50% {
    transform: translateX(100px);
  }
```

```
  100% {
    transform: translateX(200px);
  }
}
.element {
  animation: mover 3s ease-in-out 1s infinite alternate;
}
```

Neste exemplo quando a animação é aplicada ao elemento .element, ele começa na posição inicial (0% nos keyframes) e, ao longo de 3 segundos, move-se para a direita até 100 pixéis (50% nos keyframes) e, em seguida, retrocede para 200 pixéis (100% nos keyframes).

A animação continua indefinidamente devido à propriedade infinite, e o efeito de ida e volta é criado devido ao valor alternate. Portanto, o elemento se move da esquerda para a direita e vice-versa em um loop infinito.

Se você prestou atenção no exemplo anterior reparou que na propriedade animation nos temos diversos valores, e nos vamos agora ver o que eles valores significam e quais as suas funções nas animações em CSS 3.

Propriedades de animação individuais

- **animation-name:** Define o nome do conjunto de keyframes que será usado para a animação.
- **animation-duration:** Define o tempo que a animação deve levar para ser concluída.
- **animation-timing-function:** Controla a aceleração ou desaceleração da animação ao longo do tempo. Pode ser "linear", "ease-in", "ease-out", ou "ease-in-out", entre outros.
- **animation-delay:** Define o tempo que a animação deve esperar antes de começar.

- **animation-iteration-count:** Especifica o número de vezes que a animação deve ser repetida (ou "infinite" para repetição infinita).
- **animation-direction:** Define a direção da animação, que pode ser "normal", "reverse", "alternate", ou "alternate-reverse".
- **animation-fill-mode:** Controla como a animação deve tratar os estados inicial e final. Pode ser "none", "forwards", ou "backwards".
- **animation-play-state:** Define o estado de execução da animação como "running" ou "paused".

Explicando as propriedades de animação de forma descomplicada

Neste caso, vamos animar um elemento HTML quadrado para que ele se mova horizontalmente. Vamos usar as propriedades de animação separadas para controlar a animação:

```
.elemento {
  width: 50px;
  height: 50px;
  background-color: blue;
  position: relative;
  animation-name: mover;
  animation-duration: 3s;
  animation-timing-function: ease-in-out;
  animation-delay: 1s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
  animation-fill-mode: both;
  animation-play-state: running;
}
```

```
@keyframes mover {
  0% {
    left: 0;
  }
  50% {
    left: 100px;
  }
  100% {
    left: 200px;
  }
}
```

Aqui estão as explicações das propriedades usadas no exemplo acima:

- **position:** relative; é usado para posicionar o elemento de forma relativa ao seu local original.
- **animation-name:** mover; define o nome da animação, que é "mover".
- **animation-duration:** 3s; define a duração da animação em 3 segundos.
- **animation-timing-function:** ease-in-out; especifica uma função de temporização que causa uma aceleração gradual no início e no fim da animação.
- **animation-delay:** 1s; define um atraso de 1 segundo antes de a animação começar.
- **animation-iteration-count:** infinite; faz com que a animação seja repetida infinitamente.
- **animation-direction:** alternate; faz com que a animação alterne entre a reprodução normal e reversa após cada iteração.
- **animation-fill-mode:** both; faz com que o estado inicial e final da animação sejam aplicados ao elemento.
- **animation-play-state:** running; define o estado de execução da animação como "running".

Agora, vamos criar o mesmo exemplo usando a propriedade animation shorthand que é a propriedade que usa todas a propriedades acima de forma abreviada em uma única linha:

```
.elemento {
  width: 50px;
  height: 50px;
  background-color: blue;
  position: relative;
  animation: mover 3s ease-in-out 1s infinite alternate both running;
}
@keyframes mover {
  0% {
    left: 0;
  }
  50% {
    left: 100px;
  }
  100% {
    left: 200px;
  }
}
```

Neste exemplo, todas as propriedades de animação são combinadas em uma única linha usando a propriedade **animation**. O resultado é o mesmo que o exemplo anterior, mas a declaração é mais concisa.

Ambos os exemplos fazem com que o elemento se mova horizontalmente de ida e volta. Você pode escolher qual abordagem usar com base na legibilidade e na facilidade de manutenção do seu código. O **shorthand/abreviação** é útil quando você deseja manter seu código CSS mais enxuto.

Transições em CSS

As transições em CSS são uma forma de animar propriedades de elementos HTML quando ocorrem mudanças de estado, como hover, focus, click, entre outras. Elas permitem que você suavize as mudanças de estilo, criando efeitos de animação mais agradáveis para os usuários. Aqui estão os conceitos básicos das transições em CSS:

Propriedades de Transição:

Para usar transições em CSS, você precisa definir as propriedades que deseja animar. As propriedades mais comuns incluem background-color, color, width, height, opacity, transform, etc.

Duração da Transição:

Você define a duração da transição usando a propriedade transition-duration. Ela especifica quanto tempo a transição deve levar para ser concluída, geralmente em segundos ou milissegundos.

Tipo de Transição:

O tipo de transição especifica como a animação ocorrerá. Os valores comuns para a propriedade **transition-timing-function** incluem **ease** (padrão), **linear**, **ease-in**, **ease-out**, **ease-in-out**, **cubic-bezier**, entre outros. Cada um deles produz diferentes curvas de animação.

Propriedades de Transição Abreviadas:

Você também pode usar a propriedade abreviada transition para definir todas as propriedades de transição de uma só vez. Por exemplo:

```
transition: property duration timing-function delay;
```

Eventos de Ativação:

As transições são acionadas por eventos, como :hover, :focus, :active, ou quando você adiciona ou remove classes de elementos com JavaScript.

Aqui está um exemplo simples de como usar transições em CSS para criar um efeito de hover suave em um botão:

```
.button { /* Define as propriedades de transição para o botão */
  background-color: #3498db;
  color: #fff;
  padding: 10px 20px;
  border: none;
  transition-property: background-color, color; /* Propriedades a serem animadas */
  transition-duration: 0.3s; /* Duração da animação */
  transition-timing-function: ease-in-out; /* Tipo de animação */
}
/* Define o estilo quando o mouse passa sobre o botão */
.button:hover {
  background-color: #2980b9;
  color: #fff;
}
```

Neste exemplo, quando o mouse passa sobre o botão, as propriedades background-color e color mudam suavemente ao longo de 0,3 segundos, criando um efeito de transição agradável.

As transições em CSS podem ser aplicadas a uma variedade de elementos e propriedades, permitindo que você crie uma experiência de usuário mais envolvente e interativa em seu site ou aplicação web.

Propriedades Transacionáveis

Abaixo você tem uma lista de propriedades CSS que é possível identificar um ponto intermediário para transições:

- background-color
- background-position
- border-color
- border-width
- border-spacing
- bottom
- color
- font-size
- font-weight
- height left
- letter-spacing
- line-height
- margin
- max-height
- max-width
- min-height
- min-width
- opacity
- outline-color
- outline-offset
- outline-width
- padding right
- text-indent
- text-shadow
- top
- vertical-align
- visibility
- width
- word-spacing

- z-index

Propriedade Transform

A propriedade **transform** em CSS é uma poderosa ferramenta que permite aplicar transformações geométricas a elementos HTML. Essas transformações incluem rotação, escala, translação (movimento), inclinação (skew), entre outras. A propriedade transform torna possível criar efeitos visuais dinâmicos e interativos em elementos da sua página da web. Vamos explorar em detalhes como a propriedade transform funciona:

```
selector{
  transform: transform-function;
}
```

selector: O seletor CSS que se refere ao elemento HTML que você deseja transformar.

transform-function: A função de transformação que especifica o tipo de transformação a ser aplicada.

Tipos de transformações

1. **Rotação (rotate):** A função rotate permite girar um elemento em torno de seu ponto de origem (normalmente o centro). Exemplo:

   ```
   selector {
     transform: rotate(45deg); /* Rotaciona o elemento 45 graus sentido horário */
   }
   ```

2. **Escala (scale):** A função scale controla o dimensionamento do elemento, permitindo que você o aumente ou reduza em tamanho.

   ```
   element { /* Aumenta o elemento para 150% do tamanho original */
           transform: scale(1.5);
   ```

```
}
```

3. **Translação (translate):** A função translate move um elemento horizontalmente e/ou verticalmente. Exemplo:

```
element {  /* Move o elemento 20px para a direita e 30px para baixo */
        transform: translate(20px, 30px);
}
```

4. **Inclinação (skew):** inclina um elemento em relação ao seu eixo X e/ou Y. Exemplo:

```
element { /* Inclina o elemento 30 graus no eixo X e 20 graus no eixo Y */
        transform: skew(30deg, 20deg);
}
```

5. **Transformação 2D vs. 3D:** Você pode aplicar transformações tanto em 2D quanto em 3D. As 2D afetam apenas os eixos X e Y, enquanto as 3D também afetam o eixo Z, adicionando profundidade. Exemplo de transformação 3D:

```
element { /* Rotação 3D em torno dos eixos X e Y */
  transform: rotateX(45deg) rotateY(45deg);
}
```

Prioridade de Estilos

Em CSS, prioridade refere-se à ordem de precedência que as regras de estilo têm quando se aplicam a elementos HTML. É importante entender a prioridade para resolver conflitos e determinar como as regras de estilo serão aplicadas aos elementos da página da web. A prioridade é geralmente determinada da seguinte maneira, da mais alta para a mais baixa:

1. **Estilos Inline:** São definidos diretamente no elemento usando o atributo style. Eles têm a mais alta prioridade e substituirão todos as outras regras. **Exemplo:**

```
<p style="color: blue; font-size: 16px;">Este é um parágrafo com estilo inline.</p>
```

2. **Seletores de ID:** os estilos associados a um ID no HTML. Os seletores de ID têm uma prioridade mais alta do que os seletores de classe e de tag. **Exemplo:**

```
#minha-div {
        background-color: yellow;
}
<div id="minha-div">Este é um elemento com um ID específico.</div>
```

3. **Seletores de Classe:** Regras de estilo associadas a classes específicas no HTML. Os seletores de classe têm uma prioridade mais alta do que os seletores de tag.

```
.botao-destaque {
        background-color: orange;
}
<button class="botao-destaque">Clique em mim</button>
```

4. **Seletores de Tag:** Regras de estilo aplicadas a elementos HTML específicos usando seus nomes de tag. Essas regras têm a mais baixa prioridade.

```
p {
font-weight: bold;
}
<p>Este é um parágrafo normal.</p>
```

5. **Especificidade:** A especificidade é um conceito importante na resolução de conflitos de regras de estilo. Ela se baseia na contagem de seletores em uma regra. Quanto mais seletores específicos houver, maior será a prioridade. Por exemplo, uma regra com um seletor de ID (#minha-div) é mais específica do que uma regra com três seletores de classe (.classe1 .classe2 .classe3).
6. **Ordem de Declaração:** Quando todas as outras coisas são iguais, a ordem em que as regras de estilo são declaradas no arquivo CSS pode afetar a prioridade. A regra que aparece por último terá precedência sobre as anteriores.

   Por exemplo, se tivermos o seguinte HTML:

```
<div id="minha-div" class="botao-destaque">
        Este é um elemento com um ID e uma classe.
```

```
</div>
```

As regras de estilo serão aplicadas na seguinte ordem de prioridade:

- Estilos Inline (mais alta prioridade)
- Regras de ID (#minha-div)
- Regras de Classe (.botao-destaque)
- Regras de Tag ( )
- Especificidade e ordem de declaração podem ser usadas para resolver conflitos entre regras de mesma prioridade.

É importante lembrar que entender a prioridade em CSS é fundamental para criar estilos consistentes e resolver conflitos de estilo de forma eficaz em páginas da web complexas.