CSS3

Seletores Complexos

A sintaxe do CSS é simples:

```
seletor {
	propriedade: valor;
}
```

A propriedade é a característica que você deseja modificar no elemento. O valor é o valor referente a esta característica. Se você quer modificar a cor do texto, o valor é um Hexadecimal, RGBA ou até mesmo o nome da cor por extenso. Muitas vezes você não precisa aprender do que se trata a propriedade, basta saber que existe e se quiser decorar, decore. Propriedades são criadas todos os dias e não é um ato de heroísmo você saber todas as propriedades do CSS e seus respectivos valores.

Os seletores são a alma do CSS. É com os seletores que você irá escolher um determinado elemento dentro todos os outros elementos do site para formatá-lo. Boa parte da inteligência do CSS está em saber utilizar os seletores de uma maneira eficaz, escalável e inteligente.

O que é um seletor?

O seletor representa uma estrutura. Essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados.

Seletores encadeados e seletores agrupados são a base do CSS. Você os aprende por osmose durante o dia a dia. Para você lembrar o que são seletores encadeados e agrupados, segue um exemplo abaixo:

Exemplo de seletor encadeado:

```
div p strong a {
	color: red;
}
```

Este seletor formata o link (a), que está dentro de um strong, que está dentro de P e que por sua vez está dentro de um DIV.

Exemplo de seletor agrupado:

```
strong, em, span {
	color: red;
}
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

Estes seletores são para cobrir suas necessidades básicas de formatação de elementos. Eles fazem o simples. O que vai fazer você trabalhar menos, escrever menos código CSS e também menos código JavaScript, são os seletores complexos.

Os seletores complexos selecionam elementos que talvez você precisaria fazer algum script em JavaScript para poder marcá-lo com uma CLASS ou um ID para então você formatá-lo. Com os seletores complexos você consegue formatar elementos que antes eram inalcançáveis.

Exemplo de funcionamento

Suponha que você tenha um título (h1) seguido de um parágrafo (p). Você precisa selecionar todos os parágrafos que vem depois de um título h1. Com os seletores complexos você fará assim:

```
h1 + p {
	color:red;
}
```

Esse seletor é um dos mais simples e mais úteis. Não precisa adicionar uma classe com JQuery, não precisa de manipular o CMS para marcar esse parágrafo. Apenas aplique o seletor. Abaixo, veja uma lista de seletores complexos e quais as suas funções.

Padrão	Significado	CSS
elemento[atr]	Elemento com um atributo específico.	2
elemento[atr="x"]	Elemento que tenha um atributo com um valor específico igual a "x".	2
elemento[atr~="x"]	Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é "x".	2
elemento[atr^="x"]	Elemento com um atributo cujo valor comece exatamente com a string "x".	3
elemento[atr$="x"]	Elemento com um atributo cujo valor termina exatamente com string "x".	3
elemento[atr*="x"]	Elemento com um atributo cujo valor contenha a string "x".	3
elemento[atr|="en"]	Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EM (da esquerda para a direita).	2
elemento:root	Elemento root do documento. Normalmente o HTML.	3
elemento:nth-child(n)	Seleciona um objeto N de um determinado elemento.	3
elemento:nth-last-child(n)	Seleciona um objeto N começando pelo último objeto do elemento.	3
elemento:empty	Seleciona um elemento vazio, sem filhos, incluindo elementos de texto.	3
elemento:enabled elemento:disabled	Seleciona um elemento de interface que esteja habilitado ou desabilitado, como selects, checkboxes, radio buttons, etc.	3

elemento:checked	Seleciona elementos que estão checados, como radio buttons e checkboxes.	3
E>F	Seleciona os elementos E que são filhos diretos de F	2
E+F	Seleciona um elemento F que precede imediatamente o elemento E.	2

Acesso à lista atualizada pelo W3C clicando aqui.

Gradiente

Uma das features mais interessantes é a criação de gradientes apenas utilizando CSS. Todos os browsers mais novos como Safari, Opera, Firefox e Chrome já aceitam essa feature e você pode utilizá-la hoje. Os Internet Explorer atuais (8 e 9) não reconhecem ainda, contudo você poderá utilizar imagens para estes browsers que não aceitam essa feature. Você pode perguntar: "Mas já que terei o trabalho de produzir a imagem do gradiente, porque não utilizar imagens para todos os browsers?" Lembre-se que se utilizar uma imagem, o browser fará uma requisição no servidor buscando essa imagem, sem imagem, teremos uma requisição a menos, logo o site fica um pouquinho mais rápido. Multiplique isso para todas as imagens de gradiente que você fizer e tudo realmente fará mais sentido. Deixe para que os browsers não adeptos baixem imagens e façam mais requisições.

```
div {
	width:200px; height:200px;
	background-color: #FFF;
	/* imagem caso o browser não aceite a feature */
	background-image: url(images/gradiente.png);
	/* Firefox 3.6+ */
	background-image: -moz-linear-gradient(green, red);
	/* Safari 5.1+, Chrome 10+ */
	background-image: -webkit-linear-gradient(green, red);
	/* Opera 11.10+ */
	background-image: -o-linear-gradient(green, red);
}
```

Atenção: Até que os browsers implementem de vez essa feature, iremos utilizar seus prefixos.

radial-gradient

Veja abaixo como fazemos um gradiente radial (infelizmente não funcionará no IE):

```
background: radial-gradient(#666 40%, #FFF 40%);
```

linear-gradient nível básico

O gradiente é renderizado como uma imagem de fundo, então podemos chamá-lo assim. Dentro do () temos que definir o ângulo do degradê e depois, em vírgulas, os canais de cores. Aqui está um exemplo:

```
background-image: linear-gradient(to bottom, white, purple);
```

Mudando o ângulo do degradê

```
background-image: linear-gradient(to right, white, purple);
```

```
background-image: linear-gradient(45deg, white, purple);
```

Também podemos colocar mais vírgulas e inserir mais canais de cores para o degradê:

```
background-image: linear-gradient(to right, white, purple, yellow);
```

O problema das explicações sobre gradient é que são todas feias e não usuais, o que dificulta algumas pessoas de se sentirem confiantes em criá-las, por isso, vamos criar um degradê bem trabalhado visualmente para um botão.

```
background-image: linear-gradient(to bottom, #cf2b4f, #980021);
```

Especificaremos onde a cor vai parar (stop). Então o vermelho mais escuro não finalizará no padrão (não informado) de 100%, faremos ele parar um pouco depois, suavizando um pouco mais o gradiente.

```
background-image: linear-gradient(to bottom, #980021 130%);
```

Para ficar ainda mais visual, adicionamos alguns estilos simples para perceberem que, somados, fazem nosso elemento ficar mais atrativo.

```
height:auto;
padding: 40px 0;
color:#fff;
font-size:20px;
text-align:center;
border-radius:4px;
border:1px solid #980021;
box-shadow: inset 0 2px 3px 0 rgba(255,255,255,.3), inset 0 -3px 6px 0
rgba(0,0,0,.2), 0 3px 2px 0 rgba(0,0,0,.2);
background-image: linear-gradient(to bottom, #cf2b4f, #980021 130%);
```

Em alguns casos, você não precisa definir as cores precisamente no seu degradê, passando a usar transparência para isso. Por exemplo, você pode utilizar o background-color para definir a cor de fundo, e depois utilizar um degradê que vai do transparente para o preto com 30% de transparência, vejamos:

```
background-color:#4fd8e8;
background-image: linear-gradient(to bottom, transparent, rgba(0,0,0,.3));
```

O novo rgba é interessante para fazer diversas coisas no CSS, como, por exemplo, fazer um elemento estilo GLASS. Há alguns anos, havia uma técnica para criar botões de vidro com facilidade. Bastava começar com uma cor clara que vai para uma cor muito clara, colado a essa cor muito clara você tem a mesma cor normal, e finaliza com a cor um pouco clara, apenas essas 4 variações da mesma cor, ou da luz. Observe o exemplo de como é feito em gradiente:

```
background-color:#4fd8e8;
background-image: linear-gradient(to bottom, rgba(255,255,255,.1),
rgba(255,255,255,.4), rgba(255,255,255,0), rgba(255,255,255,.4));
```

Agora, vamos ajustar o “color stop” do degradê, em outras palavras, vamos ajustar onde cada cor vai parar.

```
background-color:#4fd8e8;
background-image: linear-gradient(to bottom, rgba(255,255,255,.1),
rgba(255,255,255,.4) 49%, rgba(255,255,255,0) 50%, rgba(255,255,255,.4));
```

O mais bacana é que, como o degradê está sendo feito com camadas de transparência, você pode fazer qualquer efeito ou adaptação do efeito através da simples mudança de cor de fundo.

Outro ponto muito importante é que o CSS transition não funciona para o linear, pois o mesmo é renderizado como IMAGEM. Sendo assim, você consegue mudar apenas a cor de fundo e utilizar o CSS transition para tornar o movimento mais divertido. Observe:

```
{height:auto;
padding: 40px 0;
color:#fff;
font-size:20px;
text-align:center;
background-color:#4fd8e8;
background-image: linear-gradient(to bottom, rgba(255,255,255,.1),
rgba(255,255,255,.4) 49%, rgba(255,255,255,0) 50%, rgba(255,255,255,.4));
transition: all .5s;}
:hover{background-color:#0b8b9a;}
```

"Stops", ou definindo o tamanho do seu gradiente

O padrão é que o gradiente ocupe 100% do elemento, mas muitas vezes queremos fazer apenas um detalhe. Nesse caso nós temos que definir um STOP, para que o browser saiba onde uma cor deve terminar para começar a outra. Perceba o 20% ao lado da cor vermelha. O browser calcula quanto é 20% da altura (ou largura dependendo do caso) do elemento, e começa o gradiente da cor exatamente ali. O código de exemplo segue abaixo:

```
/* Firefox 3.6+ */
background-image: -moz-linear-gradient(green, red 20%);
/* Safari 5.1+, Chrome 10+ */
background-image: -webkit-linear-gradient(green, red 20%);
/* Opera 11.10+ */
background-image: -o-linear-gradient(green, red 20%);
```

Se colocarmos um valor para o verde, nós iremos conseguir efeitos que antes só conseguiríamos no Illustrator ou no Photoshop. Segue o código:

```
/* Firefox 3.6+ */
background-image: -moz-linear-gradient(green 10%, red 20%);
/* Safari 5.1+, Chrome 10+ */
background-image: -webkit-linear-gradient(green 10%, red 20%);
/* Opera 11.10+ */
background-image: -o-linear-gradient(green 10%, red 20%);
```

Perceba que o tamanho da transição vai ficando menor à medida que vamos aumentando as porcentagens. Muito útil.

Columns

Com o controle de colunas no CSS, podemos definir colunas de texto de forma automática. Até hoje não havia maneira de fazer isso de maneira inteligente com CSS e o grupo de propriedades columns pode fazer isso de maneira livre de gambiarras.

column-count

A propriedade column-count define a quantidade de colunas terá o bloco de textos.

```
/* Define a quantidade de colunas, a largura é definida uniformemente. */
-moz-column-count: 2;
-webkit-column-count: 2;
```

column-width

Com a propriedade column-width definimos a largura destas colunas.

```
/* Define qual a largura mínima para as colunas. Se as colunas forem espremidas, fazendo com
que a largura delas fique menor que este valor, elas se transformam em 1 coluna
automaticamente */
-moz-column-width: 400px;
-webkit-column-width: 400px;
```

Firefox 3.5

Suponha que você tenha uma área disponível para as colunas de 100px. Ou seja, você tem um div com 100px de largura (width). E você define que as larguras destas colunas (column-width) sejam de 45px. Logo, haverá 10px de sobra, e as colunas irão automaticamente ter 50px de largura, preenchendo este espaço disponível.

Safari 4 (Public Beta)

Se você define um column-width, as colunas obedecem a esse valor e não preenchem o espaço disponível, como acontece na explicação do W3C e como acontece também no Firefox. Dessa forma faz mais sentido para mim. Como a propriedade não está 100% aprovada ainda, há tempo para que isso seja modificado novamente. Talvez a mudança da nomenclatura da classe para column-min-width ou algo parecida venha a calhar, assim, ficamos com os dois resultados citados, que são bem úteis para nós de qualquer maneira.

column-gap

A propriedade column-gap cria um espaço entre as colunas, um gap.

```
/* Define o espaço entre as colunas. */
-moz-column-gap: 50px;
-webkit-column-gap: 50px;
```

Estas propriedades não funcionam oficialmente em nenhum browser, mas já podem ser usados no Firefox e Safari.

Sombra (shadow)

box-shadow e text-shadow

No box-shadow, é aplicada sombra ao elemento que ele é atribuído. No caso do text-shadow, a sombra é aplicada ao texto. A forma de utilização é simples (no exemplo, é usado box-shadow, sendo a mesma aplicação ao text-shadow):

```
box-shadow:5px 5px 10px #ccc;
```

O primeiro valor é a posição horizontal da sombra em relação ao elemento. Valor positivo projeta a sombra para a direita e negativo para a esquerda. O segundo valor é a posição vertical da sombra em relação ao elemento. Valor positivo projeta a sombra para baixo e negativo, para cima. O terceiro valor é o blur (desfoque) da sombra e só aceita valores positivos ou 0 (zero). O quarto e último valor é a cor da sombra. A propriedade aceita múltiplas sombras. Podemos implementar inúmeros efeitos desta forma. Para isso é só separar a declaração de cada sombra com vírgulas:

```
box-shadow: 5px 5px 10px #333, -5px -5px 10px #ccc;
```

Transform 2D

A propriedade transform manipula a forma com que o elemento aparecerá na tela. Você poderá manipular sua perspectiva, escala e ângulos. Uma transformação é especificada utilizando a propriedade transform. Os browsers que suportam essa propriedade, a suportam com o prefixo especificado. Os valores possíveis até agora estão especificados abaixo:

scale - O valor scale modificará a dimensão do elemento. Ele aumentará proporcionalmente o tamanho do elemento levando em consideração o tamanho original do elemento.
skew - Skew modificará os ângulos dos elementos. Você pode modificar os ângulos individualmente dos eixos X e Y como no código abaixo:

```
-webkit-transform: skewY(30deg);
-webkit-transform: skewX(30deg);
```

translation - O translation moverá o elemento no eixo X e Y. O interessante é que você não precisa se preocupar com floats, positions, margins e etc. Se você aplica o translation, ele moverá o objeto e pronto.
rotate - O rotate rotaciona o elemento levando em consideração seu ângulo, especialmente quando o ângulo é personalizado com o transform-origin.

CSS Transform na prática

Veja o código abaixo:

```
img{
      -webkit-transform: skew(30deg); /* para webkit */
      -moz-transform: skew(30deg); /* para gecko */
      -o-transform: skew(30deg); /* para opera */
      transform: skew(30deg); /* para browsers sem prefixo */
}
```

O código acima determina que o ângulo da imagem seja de 30deg. Colocamos um exemplo para cada prefixo de browser.

Várias transformações em um único elemento

Para utilizarmos vários valores ao mesmo tempo em um mesmo elemento, basta definir vários valores separando-os com espaços em uma mesma propriedade transform:

```
img {
-webkit-transform: scale(1.5) skew(30deg); /* para webkit */
-moz-transform: scale(1.5) skew(30deg); /* para gecko */
-o-transform: scale(1.5) skew(30deg); /* para opera */
```

```
transform: scale(1.5) skew(30deg); /* para browsers sem prefixo */
}
```

transform-origin

A propriedade transform-origin define qual o ponto do elemento a transformação terá origem. A sintaxe é idêntica ao background-position. Observe o código abaixo:

```
img {
-webkit-transform-origin: 10px 10px; /* para webkit */
-moz-transform-origin: 10px 10px; /* para webkit */
-o-transform-origin: 10px 10px; /* para webkit */
transform-origin: 10px 10px; /* para webkit */
}
```

Como padrão as transições sempre acontecem tendo como ponto de âncora o centro do objeto. Há algumas situações que pode ser que você queira modificar essa âncora, fazendo com que, por exemplo, a rotação aconteça em algum dos cantos do elemento. O código de exemplo acima fará com que a transformação aconteça a partir dos 10px do topo no canto esquerdo.

A propriedade transform fica mais interessante quando a utilizamos com a propriedade transition, onde podemos executar algumas animações básicas manipulando os valores de transformação.

Transições e Animações

Desde quando o pessoal do WaSP organizou todo o movimento dos Padrões Web fazendo com que todos os desenvolvedores, fabricantes de browsers e até mesmo o W3C acreditassem no poder dos padrões, não houve grandes atualizações no CSS. Praticamente formatávamos font, background, cor, tamanhos e medidas de distância e posição.

A propriedade transition, a regra keyframe e outras propriedades vieram vitaminar a função que o CSS tem perante o HTML acrescentando maneiras de produzirmos animações e transições. Não estou dizendo que você fará animações complicadas, com diversos detalhes técnicos e etc. Para esse tipo de resultado existem outras ferramentas. Mas é fato que as animações via CSS nos ajudarão a levar a experiência do usuário para outro patamar.

Lembrando que o nível de suporte de algumas dessas técnicas ainda é muito baixo. A propriedade transition funciona em boa parte dos browsers atuais. Mas a regra keyframe que nos permite controlar as fases de uma animação ainda é muito restrito. Para uma tabela mais atual e detalhada de suporte e compatibilidade, faça uma procura no Google.

O básico: propriedade transition

A propriedade transition é praticamente auto-explicativa. Sua sintaxe tão simples que talvez até dispense explicações mais elaboradas. Vamos começar com o código abaixo:

```
a{
	color: white;
	background: gray;
}
```

No código acima, definimos que o link terá sua cor de texto igual a preta e seu background será cinza. O resultado esperado é que ao passar o mouse no link a cor do texto seja modificada, mudando do branco para o preto e que a cor de background mude de cinza para vermelho. O código abaixo faz exatamente isso:

```
a{
	color: white;
	background: gray;}
a:hover{
	color: black;
	background: red;
}
```

O problema é que a transição é muito brusca. O browser apenas modifica as características entre os dois blocos e pronto. Não há nenhuma transição suave entre os dois estados.

Podemos fazer a mudança de um estado para outro utilizando a propriedade transition. Suponha que ao passar o mouse, as mudanças acontecessem em um intervalo de meio segundo. Bastaria colocar a propriedade transition no a:hover e pronto. Ao passar o mouse, o browser modificaria as características do link com uma pequena transição de meio segundo. O código seria como se segue abaixo:

```
a:hover{
color: black;
background: red;
-webkit-transition: 0.5s;
}
```

Dessa forma a transição apenas acontece quando o hover é ativado. O problema é que ao tirar o mouse, o browser volta bruscamente para as características iniciais. Para modificar isso basta inserir também a propriedade transition no estado inicial.

```
a{
color: white;
background: gray;
-webkit-transition: 0.5s;
}


a:hover{
color: black;
background: red;
-webkit-transition: 0.5s;
}
```

O que a propriedade transition faz é comparar os valores das propriedades em comum entre os dois estados do link ou de qualquer outro elemento, assim ela modifica suavemente os valores quando há a ativação da função. Esta é uma técnica simples e que serve para manipularmos transições básicas como cor, tamanho, posição etc.

Agora suponha que em um bloco há uma determinada propriedade que no outro bloco não há, como no código abaixo:

```
a{
border:1px solid orange;
color: white;
background: gray;
-webkit-transition: 0.5s;
}

a:hover{
color: black;
background: red;
-webkit-transition: 0.5s;
}
```

Nesse caso, o browser detecta que há uma propriedade no primeiro estado, mas não no segundo, por isso ele não faz a transição desta propriedade, apenas das propriedades em comuns. Abaixo veja o código.

```
<head>
<style type=”text/css” media=”screen”>
	a {
	color:white;
	background:gray;
	-webkit-transition: 0.5s linear;
	}
	a:hover {
	color:black;
	background:red;
	-webkit-transition: 0.5s linear;
	}
</style>
</head>
<body>
	<a href=”#”>Link! Hello World!</a>
</body>
```

Propriedade animation e regra keyframe

A propriedade transition trabalha de forma muito simples e inflexível. Você praticamente diz para o browser qual o valor inicial e o valor final para que ele aplique a transição automaticamente, controlamos praticamente apenas o tempo de execução. Para termos mais controle sobre a animação, temos a propriedade animation, que trabalha juntamente com a rule keyframe.

Basicamente você consegue controlar as características do objeto e suas diversas transformações definindo fases da animação. Observe o código:

```
@-webkit-keyframes rodar{
	from{
		-webkit-transform:rotate(0deg);
	}to{
		-webkit-transform:rotate(360deg);
	}
}
```

O código acima define um valor inicial e um valor final. Agora vamos aplicar esse código a um elemento. Faremos uma div girar.

Observe o código HTML:

```
<!DOCTYPE html>
<html lang=”pt-br”>
	<head>
		<title></title>
		<meta charset=”utf-8”>
		<style>
			@-webkit-keyframes rodaroda{
				from{
					-webkit-transform:rotate(0deg);
				}to{
					-webkit-transform:rotate(360deg);
				}
			}
		</style>
	</head>
```

```
	<body>
		<div></div>
	</body>
</html>
```

Primeiro você define a função de animação, no caso é o nosso código que define um valor inicial de 0 graus e um valor final de 360 graus. Agora vamos definir as características deste DIV.

```
div{
	width:50px;
	height:50px;
	margin:30% auto 0;
	background:black;
}
```

Nas primeiras linhas defini qual será o estilo do div. Ele terá uma largura e uma altura de 50px. A margin de 30% do topo garantirá um espaço entre o objeto e o topo do documento, e background preto. A propriedade animation tem uma série de propriedades que podem ser resumidas em um shortcode bem simples. Veja a tabela logo a seguir para entender o que cada propriedade significa:

Propriedade	Definição
animation-name	Especificamos o nome da função de animação.
animation-duration	Define a duração da animação. O valor é declarado em segundos.
animation-timing-function	Descreve qual será a progressão da animação a cada ciclo de duração. Existem uma série de valores possíveis e que pode ser que o seu navegador ainda não suporte, mas são eles: ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>) [, ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>)]. O valor padrão é ease.
animation-iteration-count	Define o número de vezes que o ciclo deve acontecer. O padrão é um, ou seja, a animação acontece uma vez e pára. Pode ser também infinito definindo o valor infinite no valor.
animation-direction	Define se a animação irá acontecer ou não no sentido inverso em ciclos alternados. Ou seja, se a animação está acontecendo no sentido horário, ao acabar a animação, o browser faz a mesma animação no elemento, mas no sentido anti-horário. Os valores são alternate ou normal.
animation-play-state	Define se a animação está acontecendo ou se está pausada. Você poderá, por exemplo, via script, pausar a animação se ela estiver acontecendo. Os valores são running ou paused.
animation-delay	Define quando a animação irá começar. Ou seja, você define um tempo para que a animação inicie. O valor 0 significa que a animação começará imediatamente.

Voltando para o nosso código, vamos aplicar algumas dessas propriedades:

```
div{
width:50px;
height:50px;
margin:30% auto 0;
background:black;
-webkit-animation-name:rodaroda;
-webkit-animation-duration:0.5s;
-webkit-animation-timing-function:linear;
-webkit-animation-iteration-count:infinite;
-webkit-animation-iteration-direction:alternate;
}
```

Veja que na propriedade animation-name chamamos o mesmo nome que demos na nossa função de keyframe logo no começo da explicação. Depois definimos uma duração de ciclo de meio segundo. Definimos que o comportamento da animação será linear, e com a propriedade animation-iteration-count definimos que ele girará infinitamente. E por último definimos pelo animation-direction que a animação deverá ser alternada, ou seja, o DIV girará para um lado, e quando alcançar o final da animação, o browser deverá alternar essa animação.

Podemos melhorar esse código utilizando a versão shortcode, que é mais recomendado. Veja a ordem que devemos escrever as propriedades animation em forma de shortcode:

```
animation: [<animation-name>||<animation-duration>||<animation-timing-function>||<animation-
delay>||<animation-iteration-count>||<animation-direction>] [, [<animation-
name>||<animation-duration>||<animation-timing-function>||<animation-delay>||<animation-
iteration-count>||<animation-direction>] ]*
```

Aplicando isso ao nosso código:

```
div {
width:50px;
height:50px;
margin:30% auto 0;
background:black;
-webkit-animation: rodaroda 0.5s linear infinite alternate;
}
```

Pronto. Agora temos um elemento que gira sem parar, hora para direita hora para esquerda.

Definindo ciclos

Nós definimos no keyframe do nosso último exemplo apenas um início e um fim. Mas e se quiséssemos que ao chegar na metade da animação o nosso elemento ficasse com o background vermelho? Ou que ele mudasse de tamanho, posição e etc.? É aí onde podemos flexibilizar melhor nosso keyframe definindo as fases da animação. Por exemplo, podemos dizer para o elemento ter uma cor de background diferente quando a animação chegar aos 10% do ciclo, e assim por diante.

Veja o exemplo:

```
@-webkit-keyframes rodaroda {
	0%{
		-webkit-transform:rotate(0deg);
	}50%{
		background:red;
		-webkit-transform:rotate(180deg);
```

```
	}100%{
		-webkit-transform:rotate(360deg);
	}
}
```

Definimos acima que no início da animação o elemento começará na posição normal, 0 graus. Quando a animação chegar aos 50% do ciclo, o elemento deverá ter girado para 180 graus e o background dele deve ser vermelho. E quando a animação chegar a 100% o elemento deve ter girado ao todo 360 graus e o background, como não está sendo definido, volta ao valor padrão, no nosso caso black, que foi definido no CSS onde formatamos este DIV.

Logo nosso elemento girará pra sempre e ficará alternando a cor de fundo de preto para vermelho. Também é possível definir um position e modificar os valores de left e top para fazer o elemento se movimentar. No exemplo ficaram definidos apenas 3 estágios (0%, 50% e 100%),entretanto, é possível, também, definir um maior número de estágios: 5%, 10%, 12%, 16% e etc... Adequando as fases da animação às necessidades.

Bordas

Esta propriedade ainda está em fase de testes pelos browsers, por isso utilizaremos os prefixos para ver os resultados.Utilizarei apenas o prefixo do Safari, mas o Firefox já entende essa propriedade muito bem. A sintaxe do border-image se divide em três partes:

1. URL da imagem que será utilizada.
2. Tamanho do slice das bordas.
3. Como o browser irá aplicar a imagem na borda.

Segue um exemplo da sintaxe abaixo:

```
a{
display:block;
width:100px;
-webkit-border-image:url(border.gif) 10 10 10 10 stretch;
}
```

Acima definimos uma imagem com o nome de border.gif, logo depois definimos o width de cada uma das bordas do elemento. A sintaxe é igual a outras propriedades com 4 valores: top, right, bottom, left. E logo depois colocamos qual o tipo de tratamento que a imagem vai receber.

Dividindo a imagem

Para posicionar a imagem devidamente em seu objeto o browser divide a imagem em 9 seções: Quando a imagem é colocada no elemento, o browser posiciona os cantos da imagem juntamente com os cantos correspondentes do elemento. Ou seja, o bloco 1 da imagem é colocado no canto superior esquerdo, o 3 no canto superior direito e assim por diante. Se você fizer o teste, a imagem aparecerá no elemento como se estivesse desproporcional. Isso é normal porque a imagem deve seguir as proporções do elemento e não as suas próprias.

Comportamento da imagem

O comportamento da imagem é a parte mais importante porque define como o centro da imagem (no caso do nosso exemplo a seção de número 5), irá ser tratada. Há vários valores, mas o que é mais simples de se entender é a stretch, que estica e escala a imagem para o tamanho do elemento aplicado. Há outros valores

como round e repeat. Mas hoje, alguns browsers não tem tanto suporte e acabam ou ignorando esses valores ou, como no caso do Safari e o Chrome, interpretam o round como o repeat. Vamos nos concentrar com o stretch e você entenderá como funciona a propriedade.

Aplicação

Vamos utilizar a imagem abaixo para aplicar a propriedade. Iremos fazer um botão ao estilo iPhone. Irei aplicar o estilo em um link. O código que irei aplicar segue abaixo:

```
a{
	display:block;
	width:100px;
	text-align:center;
	font:bold 13px verdana, arial, tahoma, sans-serif;
	text-decoration:none;
	color:black; }
```

Para inserir a imagem, colocamos as duas linhas abaixo:

```
border-width:10px;
-webkit-border-image: url(button.png) 10 stretch;
```

É definido com a primeira linha que a borda teria 10px de largura com a propriedade border-width. Na segunda linha define que a imagem utilizada seria a button.png, que as áreas da imagem teriam que se estender por 10px, e o valor stretch define que a imagem cobrirá o elemento inteiro.

Temos o border-width definindo a largura da borda, mas não temos nenhum valor dizendo quanto dessa largura a imagem deve tomar. Os efeitos são bem estranhos quando esses valores estão mal formatados.

border-radius

Esta propriedade faz com que o elemento fique com seu background (ou outro, se ficar especificado) com um raio de pixels que for estabelecido. Observe o exemplo:

```
div{
	width:300px;
	height:100px;
	border-radius:4px;
}
```

Infelizmente esta propriedade não é aceita em versões do Internet Explorer, porém, foi criada uma forma de fazer isso. É preciso utilizar (e declarar) um arquivo, que pode ser baixado aqui. Depois de baixá-lo, deve declará-lo no documento, da seguinte forma:

```
div{
	width:300px;
	height:100px;
	border-radius:4px;
	behavior:url('border-radius.htc');
}
```

Nota: é necessário estabelecer o valor "limpo" de border-radius, caso contrário, o arquivo será carregado sem nenhuma informação para formatar.

Múltiplos Backgrounds

Quem nunca teve que criar um background onde havia um gradiente em cima, embaixo e dos lados? Você sabe que para criar algo parecido você não pode utilizar uma imagem só. A solução até hoje seria criar 4 elementos divs aninhados (ou seja, um dentro do outro) e aplicar um pedaço deste background em cada um destes elementos para dar a sensação de um background único. O resultado é interessante, mas o meio que isso é feito não é nada bonito. Você era obrigado a declarar 4 elementos "inúteis" no seu HTML apenas para compensar um efeito visual. A possibilidade de atribuirmos múltiplos backgrounds em apenas um elemento é a feature que vai ajudá-lo a não sujar seu código.

A sintaxe para múltiplos backgrounds é praticamente idêntica a sintaxe para definir um background. Segue abaixo um exemplo:

```
div{
width:600px;
height:500px;
background:url(img1.png) top left repeat-X,url(img2.png) bottom left repeat-Y;
}
```

Definimos apenas uma propriedade background, o valor dessa propriedade em vez de conter apenas um valor como normalmente fazemos, poderá haver vários, com suas respectivas definições de posição, repeat e attachment (fixed).

Módulo Template Layout

A parte mais fácil de desenvolver um site com CSS talvez seja o planejamento e diagramação do layout. Coincidentemente é a parte que mais os desenvolvedores têm problemas crossbrowser ou por falta de recursos mais avançados. Mas se você parar para pensar, apenas uma propriedade cuida dessa parte, que é a propriedade float. O float é a propriedade mais importante que há no CSS. Se o IE não soubesse o que é float, até hoje nós não estaríamos fazendo sites com CSS. O float cuida de toda a diagramação do site, desde os elementos que definirão as áreas mestres do site até os pequenos detalhes de imagens e ícones.

A propriedade float é muito simples de se entender. O problema não é o funcionamento, mas os efeitos que a propriedade float causa nos elementos próximos. Se você pede para duas colunas ficarem flutuando à esquerda e outra coluna à direita, o rodapé sobe. Ou se você coloca um elemento envolvendo outros elementos com float, esse elemento perde a altura. Estes são problemas corriqueiros que já tem soluções inteligentes e que não apresentam chateações mais graves.

Infelizmente o float não é o ideal para a diagramação e organização dos elementos do layout. Ele resolve muitos problemas, mas deixa a desejar em diversos sentidos. O float está completamente ligado a ordem dos elementos no HTML. Existem técnicas que você consegue fazer quase que qualquer organização visual sem encostar no código HTML. Mas há outras necessidades que invariavelmente você precisará modificar a ordem dos elementos no meio do HTML para que a diagramação do site saia conforme o esperado. Essa organização do HTML pode alterar desde programação server-side e até resultados de SEO e acessibilidade. Por isso é interessante que o HTML fique organizado de forma que ele supre as necessidades dessas bases. Sua organização visual deve ser independente desta organização.

Tendo em vista estes e outros problemas, o W3C criou um novo módulo. Na verdade ele não é o único, e nem pode ser para que tenhamos diversas formas de trabalhar. O módulo em questão é chamado de Template Layout. Esse módulo consiste em uma forma de criarmos e organizarmos os elementos e informações do layout de forma menos espartana e mais flexível.

Podemos dividir a construção do layout em duas grandes partes:

1. Definição dos elementos mestres e grid a ser seguido.
2. Formatação de font, cores, background, bordas etc.

As propriedades nesta especificação trabalham associando uma política de layout de um elemento. Essa política é chamada de template-based positioning (não tem nada a ver com a propriedade position, pelo contrário é uma alternativa) que dá ao elemento uma grid invisível para alinhar seus elementos descendentes. Porque o layout deve se adaptar em diferentes janelas e tamanhos de papéis, as colunas e linhas do grid deve ser fixas ou flexíveis dependendo do tamanho.
O W3C mostra alguns casos comuns:

- Páginas complexas com múltiplas barras de navegação em áreas com posições fixas como publicidade, submenus e etc.
- Formulários complexos, onde o alinhamento de labels e campos podem ser facilmente definidos com as propriedades deste módulo com a ajuda das propriedades de margin, padding e tables.
- GUIs, onde botões, toolbars, labels, ícones etc, tem alinhamentos complexos e precisam estar sempre alinhados caso o tamanho ou a resolução da tela mudam.
- Medias que são paginadas, como medias de impressão onde cada página são divididos em áreas fixas para conteúdos de gêneros diferentes.

Template-based positioning são uma alternativa para a propriedade position com o valor absolute. Antigamente lembro-me que quase todos os desenvolvedores tentavam organizar e diagramar layouts utilizando position. Não que seja errado, mas definitivamente não é a melhor forma. Usamos position para posicionarmos elementos que não tem relação direta com sua posição no código HTML. Ou seja, não importa onde o elemento esteja, o “position:absolute;” irá posicionar o elemento nas coordenadas que você quiser.

Sintaxe e funcionamento

O módulo Template Layout basicamente define slots de layout para que você encaixe e posicione seus elementos. O mapeamento dos slots é feito com duas propriedades que já conhecemos que este módulo adiciona mais alguns valores e funcionalidades, são as propriedades position e display.

A propriedade display irá definir como será o Grid. Quantos slots e etc. Já a propriedade position, irá posicionar seus elementos nestes slots.
Veja o código HTML abaixo:

```
<div class=“geral”>
<nav class=“menu”>...</nav>
<aside class=“menulateral”>...</aside>
<aside class=“publicidade”>...</aside>
<article class=“post”>...</article>
<footer>...</footer>
</div>
```

Agora iremos definir a posição destes elementos. O código CSS seria assim:

```
.geral {
display: “aaa”
“bcd”
“eee”;
}
nav.menu {position:a;}
aside.menulateral {position:b;}
aside.publicidade {position:d;}
article.post {position:c;}
footer {position:e;}
```

O funcionamento da propriedade display

A propriedade display define a organização dos slots. Um slot é um local onde o seu elemento ficará. Cada elemento fica em um slot.

Aqui o elemento display trabalha como um table, onde seu conteúdo será colocando em colunas e linhas. A única diferença é que o número de linhas e colunas não dependem do conteúdo, é fixa pelo valor da propriedade. E a outra principal diferença é que a ordem dos descendentes no código não importa.

Existem alguns valores para que você trabalhe: letra, @ (arroba) e “.” (pronto).

Cada letra diferente é um slot de conteúdo diferente. O @ define um sinal para um slot padrão. E o “.” (ponto) define um espaço em branco.

Quando repetimos as letras como no exemplo anterior, tanto na horizontal quanto na vertical, é formado um slot único que se expande para o tamanho da quantidade de slots. Funciona igual ao colspan ou rowspan utilizávamos nas tabelas.

Não é possível fazer um slot que não seja retangular ou vários slots com a mesma letra. Um template sem letra nenhuma também não é possível. Um template com mais de um @ também é proibido. Se houverem esses erros a declaração é ignorada pelo browser.

Pra definir a altura da linha (row-height) podemos utilizar o valor padrão “auto”, que define a altura da linha de acordo com a quantidade de conteúdo no slot. Você pode definir um valor fixo para a altura. Não é possível definir um valor negativo. Quando definimos um * (asterísco) para a altura, isso quer dizer que todas as alturas de linha serão iguais.

A largura da coluna (col-width) é definida com valores fixos, como o row-height. Podemos definir também o valor de * que funciona exatamente igual ao da altura de linha, mas aplicados a largura da coluna. Há dois valores chamados max-content e min-content que fazem com que a largura seja determinada de acordo com o conteúdo. Outro valor é o minmax(p,q) que funciona assim: a largura das colunas são fixadas para ser maiores ou iguais a p e menores ou iguais a q. Pode haver um espaço branco (white space) em volta de p e q. Se p > q, então q é ignorado e o minmax(p,q) é tratado como minmax(p,p). O valor fit-content é o equivalente a minmax(min-content, max-content).

Definindo a largura e altura dos slots

Para definir a altura dos slots, utilizamos uma sintaxe diretamente na propriedade display. Veja abaixo um exemplo de como podemos fazer isso:

```
display:“a a a” /150px
“b c d”
“e e e” /150px
100px 400px 100px;
```

Formatando de uma maneira que você entenda, fica assim:

```
display:     “a    a     a” /150px
             “b    c     d”
             “e    e     e” /150px
             100px 400px 100px;
```

Ou seja, a primeira coluna do grid terá 100px de largura, a segunda 400px e a terceira 100px. As medidas que coloquei ao lado, iniciando com uma / (barra), definem a altura das linhas. Logo, a primeira linha terá 150px e a terceira linha também. A linha do meio, como não tem altura definida, terá a altura de acordo com a quantidade de conteúdo.

O espaço entre as colunas são definidos com “.” (pontos).

```
display: “a a a” /150px
```

```
". . ." /50px
"b c d"
". . ." /50px
"e e e" /150px
100px 400px 100px;
```

No exemplo acima fiz duas colunas no código compostos por pontos em vez de fazer com letras. Isso quer dizer que entre as colunas do grid haverá um espaço em branco de 50px de altura.

O funcionamento da propriedade position

O valor da propriedade position especifica qual linha e coluna do elemento será colocado no template. Você escreve apenas uma letra por elemento. Vários elementos podem ser colocados em um mesmo slot. Logo estes elementos terão o mesmo valor de position.

Abaixo veja uma imagem que pegamos diretamente do exemplo do W3C. O layout é muito simples:

• navigation	**Weather** There will be weather	**Football** People like football. **Chess** There was a brawl at the chess tournament	**Your Horoscope** You're going to die (eventually).
		Copyright some folks	

Este layout é representado pelo código abaixo. Primeiro o HTML:

```
<ul id="nav">
	<li>navigation</li>
</ul>
<div id="content">
	<div class="module news">
		<h3>Weather</h3>
		<p>There will be weather</p>
	</div>
	<div class="module sports">
		<h3>Football</h3>
		<p>People like football.</p>
	</div>
	<div class="module sports">
		<h3>Chess</h3>
		<p>There was a brawl at the chess tournament</p>
	</div>
	<div class="module personal">
		<h3>Your Horoscope</h3>
		<p>You're going to die (eventually).</p>
	</div>
	<p id="foot">Copyright some folks</p>
</div>
```

Agora veja o CSS:

```
body {
display: "a b"
10em *;
}
```

```
#nav {
position: a;
}
#content {
position: b;
display: “c . d . e ”
“. . . . . ” /1em
“. . f . . ”
* 1em * 1em *;
}
.module.news {position: c;}
.module.sports {position: d;}
.module.personal {position: e;}
#foot {position: f;}
```

Lembre-se que não importa a posição dos elementos no código. E essa é a mágica. Podemos organizar o código HTML de acordo com nossas necessidades e levando em consideração SEO, Acessibilidade e Processo de Manutenção. O HTML fica totalmente intacto separado de qualquer formatação. Muito, mas muito interessante.

Pseudo-elemento ::slot()

Você pode formatar um slot especifico simplesmente declarando-o no CSS. Suponha que você queira que um determinado slot tenha um fundo diferente, alinhamento e etc... Essa formatação será aplicada diretamente no slot e não no elemento que você colocou lá. Fica mais simples de se formatar porque você não atrela um estilo ao elemento e sim ao slot. Se você precisar posicionar aquele elemento em outro lugar, você consegue facilmente.

```
body{
display:
“aaa”
“bcd”
}
body::slot(b){background:#FF0;}
body::slot(c),body::slot(d){vertical-align:bottom;}
```

As propriedades aplicadas no pseudo elemento slot() seguem abaixo:

- Todos as propriedades background.
- vertical-align
- overflow
- box-shadow, block-flow e direction ainda estão sendo estudados pelo W3C se elas entrarão ou não.

Cores

RGBA

Normalmente em web trabalhamos com cores na forma de hexadecimal. É a forma mais comum e mais utilizada desde os primórdios do desenvolvimento web. Mesmo assim, há outros formatos menos comuns que funcionam sem problemas, um destes formatos é o RGB. O RGB são 3 conjuntos de números que começam no 0 e vão até 255 (0% até 100%), onde o primeiro bloco define a quantidade de vermelho (Red), o segundo bloco a quantidade de verde (Green) e o último bloco a quantidade de azul (Blue). A combinação destes números formam todas as cores que você pode imaginar.

No HTML o RGB pode ser usado em qualquer propriedade que tenha a necessidade de cor, como: color, background, border etc. Exemplo:

```
p {
background:rgb(255,255,0);
padding:10px;
font:13px verdana;
}
```

Este código RGB define que o background o elemento P será amarelo.

RGBA e a diferença da propriedade OPACITY

Até então nós só podíamos escrever cores sólidas, sem nem ao menos escolhermos a opacidade dessa cor. O CSS3 nos trouxe a possibilidade de modificar a opacidade dos elementos via propriedade opacity. Lembrando que quando modificamos a opacidade do elemento, tudo o que está contido nele também fica opaco e não apenas o background ou a cor dele. Veja o exemplo abaixo e compare as imagens.

A primeira é a imagem normal, sem a aplicação de opacidade:

UM TEXTO PARA LER!

Agora com o bloco branco, marcado com um P, com opacidade definida. Perceba que o background e também a cor do texto ficaram transparentes.

Isso é útil, mas dificulta muito quando queremos que apenas a cor de fundo de um determinado elemento tenha a opacidade modificada. O RGBA funciona da mesma forma que o RGB, ou seja, definindo uma cor para a propriedade. No caso do RGBA, além dos 3 canais RGB (Red, Green e Blue) há um quarto canal, A (Alpha), que controla a opacidade da cor. Nesse caso, podemos controlar a opacidade da cor do background sem afetar a opacidade dos outros elementos:

Veja um exemplo de código aplicado abaixo:

```
p {
background:rgba(255,255,0, 0.5);
padding:10px;
font:13px verdana;
}
```

O último valor é referente ao canal Alpha, onde 1 é totalmente visível e 0 é totalmente invisível. No exemplo acima está com uma opacidade de 50%.

currentColor

O valor currentColor é muito simples de se entender e pode ser muito útil para diminuirmos o retrabalho em alguns momentos da produção. Imagine que o currentColor é uma variável cujo seu valor é definido pelo valor da propriedade color do seletor. Veja o código abaixo:

```
p{
background:red;
padding:10px;
font:13px verdana;
color: green;
border:1px solid green;
}
```

Note que o valor da propriedade color é igual ao valor da cor da propriedade border. Há uma redundância de código, que nesse caso é irrelevante, mas quando falamos de dezenas de arquivos de CSS modulados, com centenas de linhas cada, essa redundância pode incomodar a produtividade. A função do currentColor é simples: ele copia para outras propriedades do mesmo seletor o valor da propriedade color. Veja o código abaixo para entender melhor:

```
p{
background:red;
padding:10px;
font:13px verdana;
color: green;
border:1px solid currentColor;
}
```

Veja que apliquei o valor currentColor onde deveria estar o valor de cor da propriedade border. Agora, toda vez que o valor da propriedade color for modificado, o currentColor aplicará o mesmo valor para a propriedade onde ela está sendo aplicada.

Isso funciona em qualquer propriedade que faça utilização de cor como background, border, etc. Obviamente não funciona para a propriedade color. O currentColor também não funciona em seletores separados, ou seja, você não consegue atrelar o valor da propriedade color ao currentColor de outro seletor.

Paged Media

Com certeza você já deve ter tentado ler um livro ou uma apostila em algum site na web e preferiu imprimir o texto para ler off-line, no papel por ser mais confortável ou por ser mais prático quando não se está conectado. Existem vários motivos para que um leitor queira imprimir o conteúdo de um site, principalmente sites com textos longos e pesados. Durante muito tempo, o principal motivo era que ler na tela do computador era cansativo. Hoje isso ainda é um problema, mas com o avanço das telas e do aparecimento das tablets no mercado, você consegue passar mais tempo na frente de uma tela lendo grandes quantidades de texto. O problema é que geralmente a organização de páginas e o conteúdo não é exatamente confortável para passarmos horas lendo.

Outro problema comum é que nós desenvolvedores não temos uma maneira fácil de formatar páginas. Na verdade temos, mas é um pouco de gambiarra e claro, não é maneira correta. A especificação de Paged Media nos possibilita formatar as páginas, transparências (aqueles "plásticos" que usamos com retro-projetores) ou até mesmo páginas que serão vistas pelo monitor. Controlaremos suas medidas, tamanhos, margens, quebras de páginas e etc...

Nota: Quando é dito páginas, significa páginas físicas, de papel, não páginas web.

@page

Definiremos com CSS3 um modelo de página que especifica como o documento será formatado em uma área retangular, chamada de page box, com larguras e alturas limitadas. Nem sempre o page box tem referência correspondente para uma folha de papel física, como normalmente conhecemos em diversos formatos: folhas, transparências e etc. Esta especificação formata o page box, mas é o browser ou o meio de acesso que o usuário está utilizando que tem a responsabilidade de transferir o formato do page box para a folha de papel no momento da impressão. O documento é transferido no modelo da página para um ou mais page boxes. O page box é uma caixa retangular que será sua área de impressão. Isso é como se fosse um viewport do browser. Como qualquer outro box, a page box tem margin, border, padding e outras áreas. O conteúdo e as áreas de margin do page box tem algumas funções especiais:

- A área de conteúdo do page box é chamada de area da página ou page area.
- O conteúdo do documento flui na área de página.
- Os limites da área da página na primeira página estabelece o retângulo inicial que contém o bloco do documento.

A área da margem da page box é dividido em 16 caixas de margem ou margin boxes. Você pode definir para cada caixa de margem sua própria borda, margem, padding e áreas de conteúdo. Normalmente essas caixas de margem são usadas para definir headers e footers do documento. Confesso que o nome utilizado (caixas de margem) é meio estranho.

As propriedades do page box são determinadas pelas propriedades estabelecidas pelo page context, o qual é a regra de CSS @page. Para definir a largura e altura de uma page box não se usa as propriedades width e height. O tamanho da page box é especificada utilizando a propriedade size no page context.

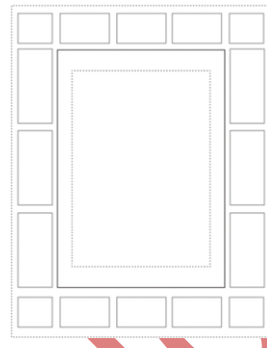
Terminologia e Page Model (modelo de página)

O page box tem algumas áreas simples de se entender, que facilitará a explicação. Veja abaixo uma imagem e uma explicação de suas respectivas áreas:

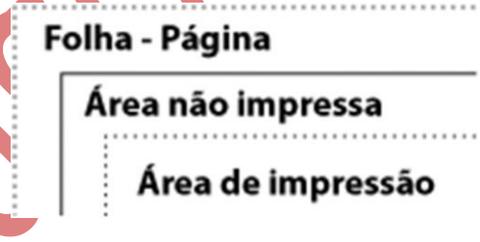
Page box: É onde tudo acontece. Tenha em mente que o page box é o viewport das medias impressas. É lá que conterá as áreas de margem, padding, border e onde o texto será consumido. A largura e altura do page box é determinada pela propriedade size. Em um caso simples, o page box tem a largura e a altura de uma folha. Entretanto, em casos complexos onde o page box difere das folhas de papel em valores e orientações, você pode personalizar de acordo com sua necessidade.

Page area: É a area de conteúdo (content area) do page box.

Margin box: Contém boxes para header e footer. São conjunto de 16 boxes onde você pode inserir conteúdo útil como número da página, título do livro, etc. Essas áreas ficam fora do Page area. Cada um tem suas margins, paddings e bordas individuais. Veja o diagrama abaixo para visualizar melhor.



Page sheet: A folha, a página e a superfície que será impresso o conteúdo. A ilustração abaixo mostra a representação de uma folha.



Non-printable area - Área não impressa: É a área onde o dispositivo de impressão não é capaz de imprimir. Esta área depende do dispositivo que você está utilizando. O page box fica dentro da área de impressão.

Área de impressão: É onde o dispositivo de impressão é capaz de imprimir. A área de impressão é o tamanho da page sheet menos a área de não impressão. Como a área de não impressão, a área útil de impressão depende muito do dispositivo. O dispositivo pode ajustar o conteúdo para que seja impresso sem problemas nessa área. Cada dispositivo tem seu meio de ajuste.

Propriedade size

A propriedade size especifica o tamanho e a orientação da área do de conteúdo, o page box. O tamanho do page box pode ser definida com valores absolutos (px) ou relativos (%). Você pode usar três valores para definir a largura e a orientação do page box:

auto: O page box irá ter o tamanho e orientação do page sheet escolhido pelo usuário.

landscape: Define que a página será impressa em modo paisagem. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o horizontal. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

portrait: Define que a página será impressa em modo retrato. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o vertical. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

Veja um exemplo abaixo:

```
@page{
size:auto; /* auto is the valor padrão */
margin:10%; /* margem */
}
```

Como nessa caso a margem é variável, ela está sendo relativa às dimensões da página. Logo, se a página é uma A4, com as dimensões: 210mm x 297mm, as margens serão 21mm e 29.7mm.

Outro exemplo:

```
@page{
size:210mm 297mm; /* definem o page-sheet para um tamanho de A4 */
}
```

Page-size

O page-size pode ser especificado utilizando um dos media names abaixo. Isso é o equivalente a utilizar os valores escritos diretamente na propriedade size. Contudo, é muito melhor utilizar o nome de um formato de papel.

Formato	Descrição
A5	A página deve ser definida para o tamanho ISO A5: 148mm x 210mm.
A4	A página deve ser definida para o tamanho ISO A4: 210mm x 297mm
A3	A página deve ser definida para o tamanho ISO A3: 297mm x 420mm.
B5	A página deve ser definida para o tamanho ISO B3 media: 176mm x 250mm
B4	A página deve ser definida para o tamanho ISO B4: 250mm x 353mm.
Letter	A página deve ser definida para o tamanho papel carta: 8.5pol x 11 pol.

Abaixo veja um exemplo de aplicação:

```
@page{
size:A4 landscape;
}
```

O W3C tem uma especificação muito extensa que pode ser encontrada aqui.

A regra @font-face é utilizada para que você utilize fontes fora do padrão do sistema em seus sites. Para que isso funcione, nós disponibilizamos as fontes necessárias em seu servidor e linkamos estas fontes no arquivo CSS. A sintaxe é bem simples e tem suporte a todos os navegadores, com algumas ressalvas.

```
@font-face {
font-family:helveticaneue;
src:url(helveticaNeueLTStd-UltLt.otf);
}
```

Na primeira linha você customiza o nome da font que você usará durante todo o projeto. Na segunda linha definimos a URL onde o browser procurará o arquivo da font para baixar e aplicar no site. Você aplica a fonte como abaixo:

```
p{font:36px helveticaneue, Arial, Tahoma, Sans-serif;}
```

Suponha que o usuário tenha a fonte instalada, logo ele não precisa baixar a font, assim podemos indicar para que o browser possa utilizar o arquivo da própria máquina do usuário. Menos requisições, mais velocidade. Veja o código:

```
@font-face {
	font-family:helveticaneue;
	src:local(HelveticaNeueLTStd-UltLt.otf), url(HelveticaNeueLTStd-UltLt.otf);
}
```

Abaixo segue uma série de formatos que podem ser usados e que os browsers podem adotar:

String	Font Format	Common Extensions
truetype	TrueType	.ttf
opentype	OpenType	.ttf,.otf
truetype-aat	TrueType with Apple Advanced Typography extensions	.ttf
embedded-opentype	Embedded OpenType	.eot
woff	WOFF (Web Open Font Format)	.woff
svg	SVG Font	.svg,.svgz

Compatibilidade

As versões 7, 8 e 9 do Internet Explorer aceitam o @font-face apenas se a font for EOT. Você pode encontrar qualquer conversor online que esse problema é resolvido. Você pode converter suas fontes para EOT diretamente no Font Squirrel. O Safari, Firefox, Chrome e Opera aceitam fontes em TTF e OTF.

Para suprir a necessidade de atenção do Internet Explorer, você precisa especificar a URL da font .eot para que o Internet Explorer possa aplicar a font corretamente. A sintaxe fica desta forma:

```
@font-face {
font-family: helveticaneue;
src: url(helveticaNeueLTStd-UltLt.eot);
src: url(helveticaNeueLTStd-UltLt.otf);
}
```

Kit de sobrevivência

O Font Squirrel é um sistema que converte suas fontes para os formatos necessários e devolve para você utilizar em seus sites. Acesse clicando aqui.

Presentation-Levels

A informação na web é reutilizada de diversas maneiras. Toda informação publicada é reutilizada por diversos meios de acesso, seja o seu browser, leitor de tela ou robôs de busca. O HTML proporciona essa liberdade para a informação. Por ser uma linguagem muito simples, podemos reutilizar a informação marcada com HTML em diversos meios de acesso. Mas o HTML não cuida da forma com que o usuário vai visualizar a informação em seu dispositivo. O HTML apenas exibe a informação. A maneira que o usuário consome essa informação é diferente em cada um dos meios de acesso e dispositivos. O CSS formata a informação para que ela possa ser acessível em diversos usar agents (meios de acesso). Se você acessa o site do seu banco pelo monitor de 22'' da sua casa ou pelo seu celular, a informação tem que aparecer bem organizada em ambos cenários. É o CSS que organiza visualmente essas informações.

Além disso, podemos apresentar a informação de diversas formas em um mesmo dispositivo. Por exemplo: você pode ver uma galeria de imagens da maneira convencional, clicando nas thumbs das fotos ou ver em forma de slideshow. Podemos levar essas experiências para websites de conteúdo textual também. A especificação de presentation-levels é uma das especificações que levam o usuário a terem conteúdo mostrados de uma outra forma da qual estamos acostumados. É muito útil para apresentações de slides, com efeitos, transições ou qualquer documento que seria mais bem apresentado no formato de apresentação, como uma proposta, documentos técnicos e etc.

Como funciona o modelo

O modelo por trás da especificação é simples. Cada elemento no documento é definido como um "elemento de apresentação" ou no formato original "element's presentation level" - EPL.

O EPL pode ser explícito em uma folha de estilo ou calculado automaticamente baseado na posição do elemento pela estrutura do documento. É assim que o browser calcula para mostrar os elementos progressivamente, como se faz normalmente em programas de apresentação. O elemento fica em um dos três seguintes níveis que também são representadas por classes: below-level, at-level e above-level. Dependendo da pontuação de EPL que o browser dá, o elemento fica em um determinado nível. Essas pseudo-classes podem e devem ser modificadas via CSS.

A propriedade presentation-level

A propriedade presentation-level define como os valores de apresentação (EPL) de um determinado objeto devem ser calculados. São três valores possíveis: números inteiros, increment e same.

Quando definimos um valor inteiro, o elemento tem aquele valor fixo. Quando colocamos increment, o valor do objeto aumenta um ponto em relação ao objeto anterior. Suponha que há duas LI em uma UL. A primeira LI tem o valor de 1, a segunda tem valor de 2 e assim por diante. Quando definimos o valor same, o browser computa o mesmo valor do objeto anterior.

Utilizando o mesmo exemplo da especificação do W3C, temos o código abaixo:

```
<!DOCTYPE html>
<html>
	<body>
		<h1>strategies</h1>
```

```
            <h2>our strategy</h2>
            <ul>
                  <li>divide</li>
                  <li>conquer
                        <p>(in that order)</p>
                  </li>
            </ul>
            <h2>their strategy</h2>
            <ul>
                  <li>obfuscate</li>
                  <li>propagate</li>
            </ul>
      </body>
</html>
```

Vamos definir o CSS de presentation-levels para esse HTML adicionado o código CSS:

```
@media projection{
h1{page-break-before:always;}
li{presentation-level:increment;}
:below-level{color:black;}
:at-level{color:red;}
:above-level{color:silver;}
}
```

Definimos que o H1 irá sempre iniciar em uma nova página. Mas o mais importante é a propriedade presentation-level que definimos para a LI. Isso quer dizer que a cada LI o browser contará mais um ponto.

As três pseudo-classes que falamos no começo do texto (below-level, at-level, above-level) que formata os elementos que foram mostrados, o elemento que está sendo mostrado e o próximo elemento.

Sendo assim, o browser calcula a pontuação de cada um dos elementos utilizados no exemplo como mostra abaixo:

HTML	Valor de EPL
strategies	0
our strategy	0
	0
divide	1
conquer	2
their strategy	0
	0
obfuscate	1
propagate	2
	0

Temos um outro exemplo, segue abaixo o HTML e logo depois a tabela com os valores de EPL:

```
<!DOCTYPE html>
<html>
      <style>
            @media projection {
            h1 { presentation-level: 0; }
            h2 { presentation-level: 1; }
            h3 { presentation-level: 2; }
            body * { presentation-level: 3; }
            :above-level { display: none; }
            }
      </style>
      <body>
            <h1>strategies</h1>
```

```
            <h2>our strategy</h2>
            <ul>
                  <li>divide</li>
                  <li>conquer</li>
            </ul>
            <h2>their strategy</h2>
            <ul>
                  <li>obfuscate</li>
                  <li>propagate</li>
            </ul>
      </body>
</html>
```

Perceba que agora definimos no CSS que tudo dentro de body tem o valor de 3. Logo o H1 que foi definido como 0 pela propriedade presentation-level tem o valor de 3.

Definimos também display:none; para os próximos elementos. Agora veja a pontuação aplicada:

HTML	Valor de EPL
strategies	3
our strategy	2
	0
divide	0
conquer	0
their strategy	2
	0
obfuscate	0
propagate	0
	0

O W3C tem uma especificação completa e em constante atualização do presentation-levels aqui.