**Píxeis e Porcentagens**

A primeira página que desenvolvemos foi a da biografia do João da Silva.



Podemos notar, ao olharmos para o código CSS, que utilizamos com frequência a medida "*px*", ou seja, a medida *pixel*. Por exemplo:

.titulo-principal {

font-size: 68px;

text-align: center;

background-color: #851944;

color: #FFF;

border-botton: 10px solid #000;

}

.subtitulo-texto {

font-size: 38px;

clear: both;

}

E assim por diante.

O que representa *1* (um) *pixel*? Se olharmos bem de perto um monitor, veremos que sua tela possui diversos micro pontos, estes são os *píxeis*.

No exemplo acima, passamos que o título principal da página deve ter uma fonte de *68 pixeis* de tamanho. Tal medida é muito boa, porque nos permite um controle dos tamanhos dos elementos. Porém não é apenas essa que estamos usando. Também temos a porcentagem, como em:

body {

font-family: "Crimson Text", "Times New Roman", serif;

background-color: #F2FFFC;

font-size: 120%;

line-height: 1.5;

}

Utilizamos a porcentagem para termos um controle do tamanho da fonte da página com base no tamanho padrão da fonte do navegador.

Outro caso do uso de porcentagem é para definir larguras. Por exemplo:

main {

width: 86%;

}

Se você baixar o arquivo do projeto, pode ser que essa porcentagens estejam diferentes. Isso acontece porque elas devem variar de monitor para monitor. Então dê uma modificada nelas para adaptá-las ao seu. Mais para frente veremos que existe um meio de não precisar fazer isso.

O tamanho das fontes ou das larguras dos objetos irá variar, ao utilizarmos a porcentagem, de acordo com o navegador e o monitor que estamos usando, ou seja, o contexto em que a fonte se insere.

Se inspecionarmos o elemento *aside*:



Observamos:

.navegacao-site {

width: 14%;

}

Ele possui uma largura de 14% da tela. Mas será que é sempre assim? Vamos pegar outro caso, o do *blockquote*:



Observamos:

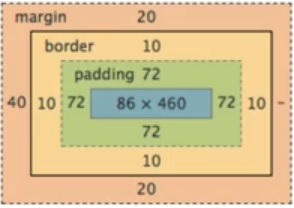
.citacao-bio {

...

padding: 20px;

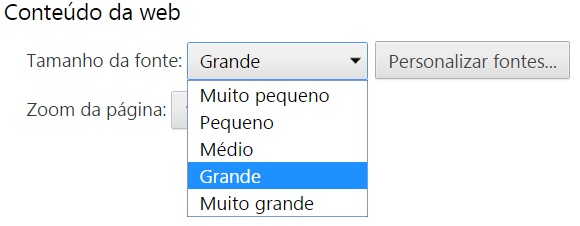
}

Se mudarmos de 20 px para 10%, será que será 10% da tela? Se inspecionarmos os tamanhos dos elementos:



Perceba que o *padding* está com 72 pixeis. Como nossa tela não tem 720 píxeis, é muito maior, esses 10% não estão relacionados a ela, mas sim ao *container*, ao pai do elemento que se quer modificar o tamanho. E lembre-se que a relação é sempre entre as larguras.

Voltando aos píxeis, nós temos um pequeno problema ao trabalhar com essa medida. O próprio usuário pode mudar o tamanho da fonte utilizando as configurações do navegador:



Se voltarmos à nossa página, verificamos que o texto principal ficou maior, porém título e subtítulos mantiveram seu tamanho original. Isso acontece porque o texto principal está definido por uma porcentagem, enquanto que o título e subtítulos estão estritamente definidos pelo número de píxeis. Não importa o quanto você modifique o tamanho da fonte nas configurações. Se o tamanho do elemento for definido por píxeis, ele manterá esse tamanho.

O que podemos fazer é nos utilizar do mesmo recurso do *body*:

.titulo-principal {

font-size: 300%;

}

**A medida *rem***

Podemos aplicar as porcentagens em todos os elementos, mas sempre lembrando que para alguns ela está relacionada à largura do elemento pai. Isto pode causar alguma confusão, uma vez que teríamos que aplicar uma porcentagem em cima de outra, no caso do *padding*.

Utilizaremos, então, uma nova medida que usa o tamanho da fonte como base, o ***rem***:

*1,25 rem* = *1,25* x *tamanho da fonte*

Por exemplo, se a fonte do navegador for de 20px, temos:

*20px* x *1,25* = *25px*

O *rem* é uma medida muito boa para quando quisermos que os elementos sigam o padrão do texto principal.

**A medida *ch***

Voltando ao *blockquote*, vemos que seus elementos filhos são baseados na sua largura. Então não faz sentido utilizar o *rem* aqui, uma vez que este é baseado na altura da fonte.

Então vamos utilizar uma medida que facilita principalmente na edição de textos dentro de caixas, a medida ***ch***:

*1 ch* = *largura do caractere "zero" da fonte utilizada*

Modificando o tamanho da fonte do navegador, esta medida também modifica-se. Essa medida é boa porque não importa muito a fonte que você está utilizando, ela se adequará muito bem.

**A medida *em***

Agora vamos tentar aumentar o tamanho da fonte do *blockquote*:

.citacao-bio {

...

font-size: 200%;

}

Ao fazermos isso, nada acompanha proporcionalmente esse aumento. Seria interessante que todos os elementos modificassem de tamanho com base na fonte do elemento pai.

Para isso, utilizamos a medida ***em*** que indica o tamanho da fonte do elemento, não do navegador como é o caso do *rem*.

Claro que se aumentarmos a fonte do navegador, como todos os elementos estão ligados a ela, o *blockquote* e seus elementos irão ficar ainda maiores.

***Font-size* no HTML**

No *body* sua *font size* era de 120%. Se aumentarmos para 220%, verifique que o tamanho da borda não aumentou. Isso acontece porque a fonte que padroniza tudo não é a do body, mas sim a do navegador, aquela que modificamos nas configurações.

Porém conseguimos modificar com base no elemento HTML, declarando:

html {

font-size: 200%

}

Se agora voltarmos ao título principal e declararmos:

.titulo-principal {

...

border-bottom: 0.5rem

}

A borda fica mais larga.

Então conseguimos controlar o *rem* tanto pela configuração do navegador quanto pelo *font-size* do HTML (não do *body*). Tanto é que é melhor excluirmos o *font-size do*body*para que toda declaração de*rem\* faça sentido.

**Resumindo**

Vimos, então, três novas medidas:

* *rem*: tem como padrão a fonte do navegador;
* *em*: tem como padrão a fonte do elemento pai;
* *ch*: tem como base a largura do caractere zero da fonte usada;

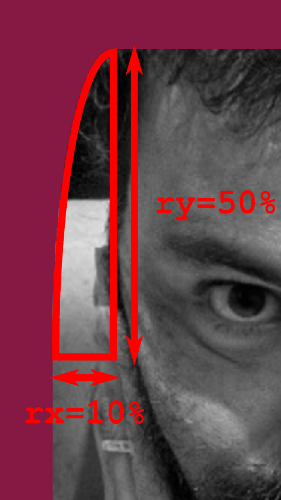
Vimos que a porcentagem tem como base ou a largura do elemento pai ou o *font-size* do elemento anterior.

Utilizando essas medidas conseguimos um CSS mais flexível, principalmente se quisermos manter um padrão de tamanho de fonte. Utilizando o *font-size* do HTML, é apenas ali que será necessário fazer alterações.

Como vimos, o border-radius nos permite fazer não apenas bordas redondas como também bordas elípticas. Conseguimos fazer o efeito abaixo na foto do João da Silva, dando a impressão de que ela está inclinada:



Na imagem abaixo é possível ver melhor como estão definidas as bordas:



Observe que as elipses estão em orientações diferentes: nos cantos superior-esquerdo e inferior-direito, a elipse tem o eixo vertical maior enquanto que, nos cantos superior-direito e inferior-esquerdo, a elipse tem o eixo horizontal maior.

A sintaxe para o border-radius criar bordas elípticas é a seguinte:

border-radius: <raios na horizontal> / <raios na vertical>;

Na verdade, quando declaramos

border-radius: 50%;

É como se tivéssemos feito:

border-radius: 50% 50% 50% 50% / 50% 50% 50% 50%;

Sabendo disso, qual das declarações abaixo cria o efeito desejado na imagem?

border-radius: 10% 50% / 50% 10%

**Prefixos em valores**

Não apenas propriedades recebem prefixos quando ainda estão em fase beta de implementação. Algumas propriedades vão ganhando **novos valores** com a evolução do CSS.

Um exemplo de novo valor que já pode ser usado em diversas propriedades e que estudaremos mais adiante é o valor calc, que nos permite fazer contas no CSS! Podemos fazer, por exemplo, com que um elemento tenha metade da largura da tela menos 10 pixels:

div {

width: calc(50% - 10px);

}

Agora, nem sempre essa propriedade foi bem suportada; dê uma olhada no [Can I Use](http://caniuse.com/#feat=calc). Com base nele, como devemos fazer a declaração acima para que ela funcione em todas as versões do Firefox e do Chrome possíveis?

**d**iv {

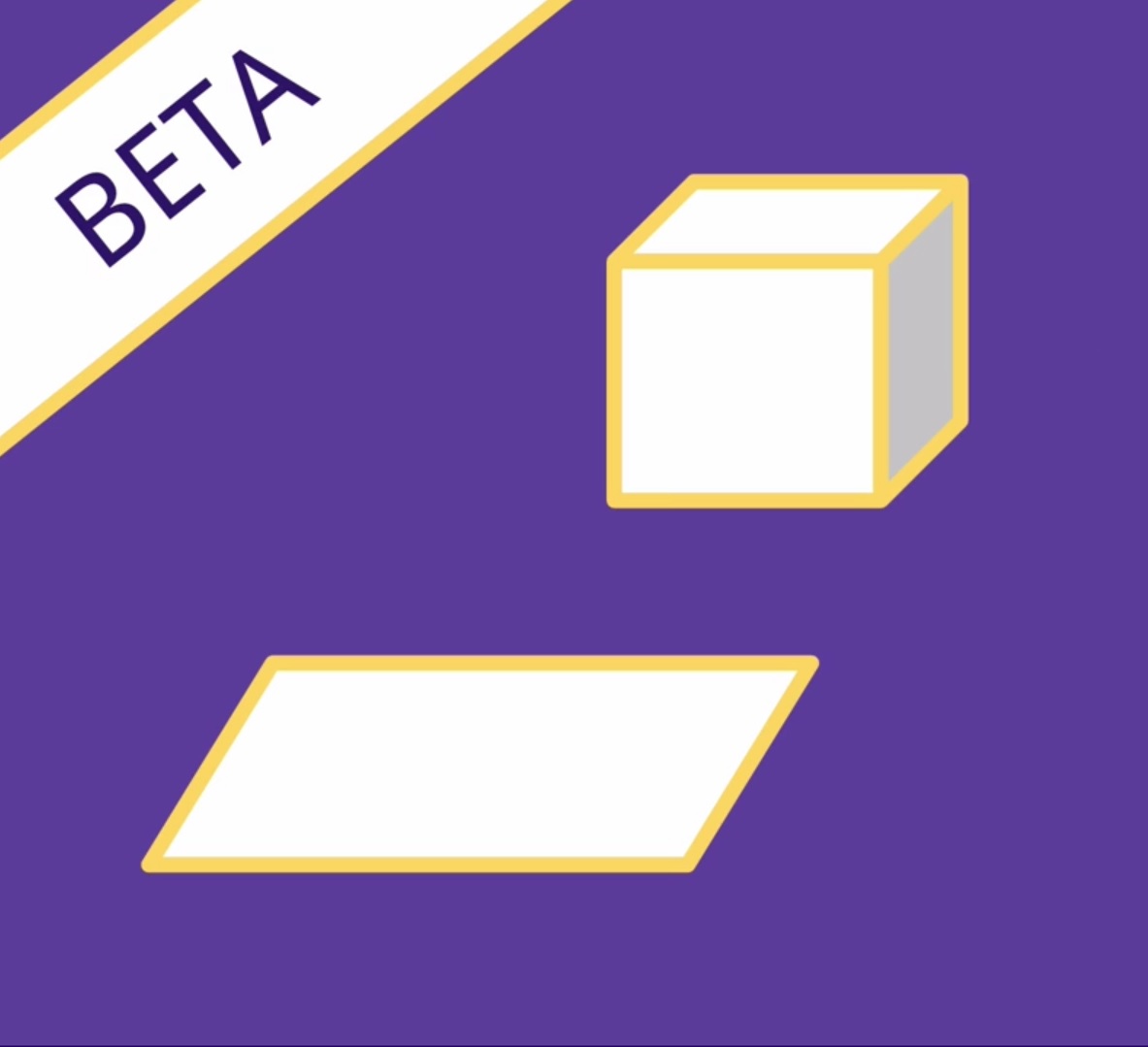
width: -webkit-calc(50% - 10px);

width: -moz-calc(50% - 10px);

width: calc(50% - 10px);

}

Vamos continuar estudando as propriedades do CSS3. Nessa aula veremos as propriedades ***transform*** e ***perspective***, o que nos permitirá, por exemplo, inserir textos rotacionados, caixas "tortas" e objetos 3D:



**A propriedade *transform***

O *transform* é uma propriedade poderosa, mas também muito simples de usar. Basta especificar o tipo de transformação que queremos fazer com o objeto:

div {

transform: ...;

}

Vamos ver alguns exemplos práticos:

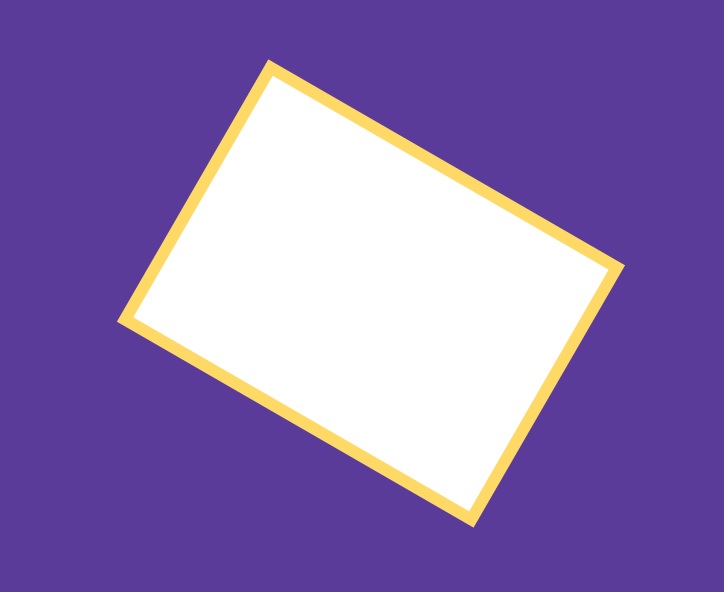
* Rotacionar o objeto

Aqui o objeto está sendo rotacionado em 30 graus:

div {

transform: rotate(30deg);

}



* Aumentar ou diminuir o tamanho do objeto

Aqui o objeto aumenta em 1,5 vezes:

div {

transform: scale(1.5);

}



* "Entortar" o objeto

Aqui os ângulos do objeto crescem ou diminuem em 20 graus:

div {

transform: skew(20deg);

}



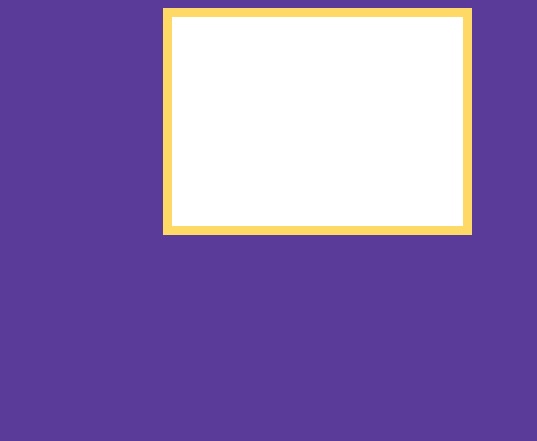
* Transladar o objeto

Aqui o objeto foi transladado 10 pixeis para a direita e 50 para baixo:

div {

transform: translate(10px, 50px);

}



* Fazer tudo ao mesmo tempo

Aqui o objeto foi: entortado em 20 graus, rotacionado em 30 graus e aumentado em 1,2 vezes:

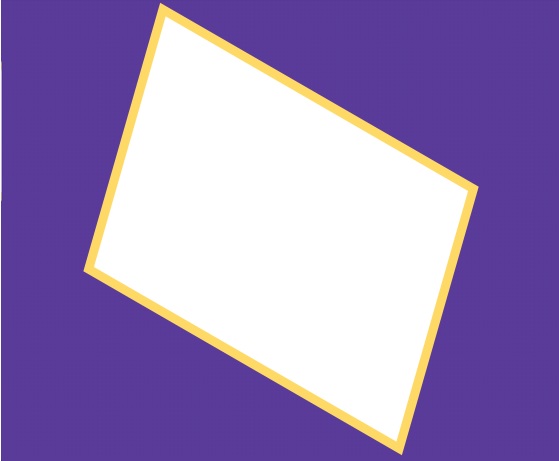
div {

transform: skew(20deg)

rotate(30deg)

scale(1.2);

}



Como essa propriedade é relativamente nova, talvez seja interessante acrescentar antes dela o prefixo *BETA*, para versões mais antigas de navegadores:

div {

-webkit-transform: skew(20deg)

rotate(30deg)

scale(1.2);

transform: skew(20deg)

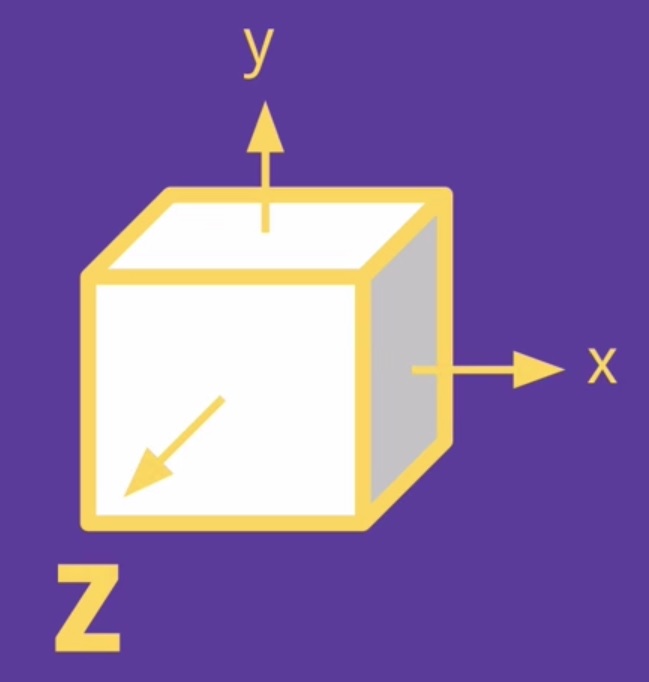
rotate(30deg)

scale(1.2);

}

**Transformações 3D / propriedade *perspective***

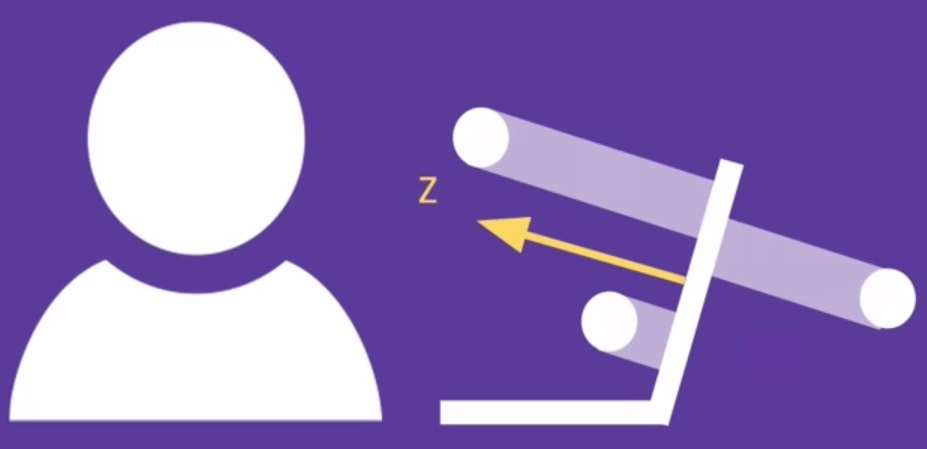
Nas transformações em 3D trabalhamos com três eixos para definir o tamanho dos nossos objetos: *x*, *y* e *z*. Este último é novidade, ele indica a profundidade do objeto. Os dois primeiros são eixos que já vínhamos trabalhando.



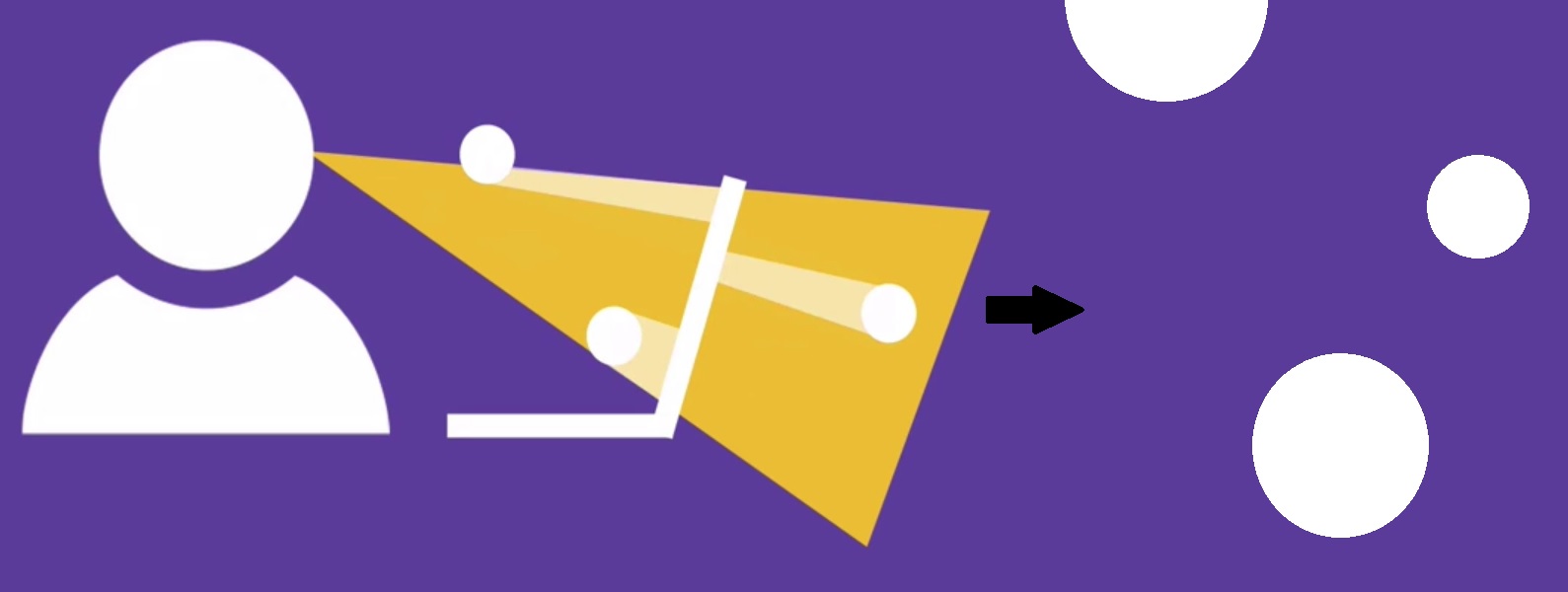
Agora especificamos para cada eixo suas transformações:

* translateX
* translateY
* translateZ

Se quisermos, por exemplo, que o objeto pareça estar saindo "para fora" da tela, mexemos no eixo *z*. Para isso usamos um efeito de projeção de luz e sombras, que dá essa impressão de tridimensionalidade.



Outro efeito importante para dar essa impressão é modificar algumas dimensões do objeto e suas sombras, pois apenas utilizando efeitos de luz, ainda teremos a sensação de que o objeto tem o mesmo tamanho. As sombras devem ter tamanhos diferentes.



Teremos esse resultado acima se projetarmos as sombras corretamente.

Porém, se nos afastarmos muito da tela do computador, essa diferença some e os círculos parecerão ter o mesmo tamanho. Tudo isso está relacionado à *perspectiva*.

Então, antes de especificarmos a cena 3D, devemos primeiro especificar a *perspectiva*que queremos usar: a que distância o usuário estará da tela para que possamos fazer o cálculo da projeção da cena 3D.

Conseguimos isso por meio da propriedade ***perspective***, que usa o sistema de cone de luz. Além, é claro, de nos utilizarmos das transformações já vistas.

Um outro detalhe importante é a angulação com que o usuário está olhando para a tela, o que causa efeitos diferentes de perspectiva. Para lidar com essas diferenças de posição do usuário, usamos a propriedade ***perspective-origin***.

**Implementando o 3D no código**

A cena que vimos com os círculos poderia ser um HTML desse tipo:

<div class="bolas">

<div class="bola1"></div>

<div class="bola2"></div>

<div class="bola3"></div>

</div>

O primeiro passo para criarmos uma cena 3D com essas bolinhas é deslocá-las conforme a distância (profundidade) que queremos que elas estejam da tela:

.bola1 {

transform: translateZ(2px);

}

.bola2 {

transform: translateZ(-1px);

}

As translações positivas trazem o objeto para "perto" da tela e as negativas para longe.

A perspectiva é implementada no elemento pai, ou seja, em "bolas":

.bolas {

perspective: 4px;

}

.bola1 {

transform: translateZ(2px);

}

.bola2 {

transform: translateZ(-1px);

}

Os "4px" não significa que o usuário estará visualizando a cena a 4 pixeis de distância, mas apenas para facilitar os cálculos, para ter uma ideia do que está perto e do que está longe de um ponto de referência.

Nós podemos ter várias cenas 3D independentes, cada uma com sua perspectiva:

<main>

<section>

...

perspective: 8px;

...

</section>

<section>

...

perspective: 80px;

...

</section>

<section>

...

perspective: 42px;

...

</section>

</main>

E, dentro de cada cena, com sua própria perspectiva, também não precisamos nos limitar a um elemento. Podemos, por exemplo, definir propriedades para cada face de um cubo. Configuramos a *perspectiva* da cena, *transformamos* o objeto fazendo uma rotação ou deslocamento e *transformamos* cada face do cubo, rotacionando junto com o cubo, por exemplo:

<main>

<section> <- [perspective: ...]

<div class="cubo"> <- [transform: ...]

<span class="face1">...</span> <- [transform: ...]

<span class="face2">...</span>

<span class="face3">...</span>

<span class="face4">...</span>

<span class="face5">...</span>

<span class="face6">...</span>

</div>

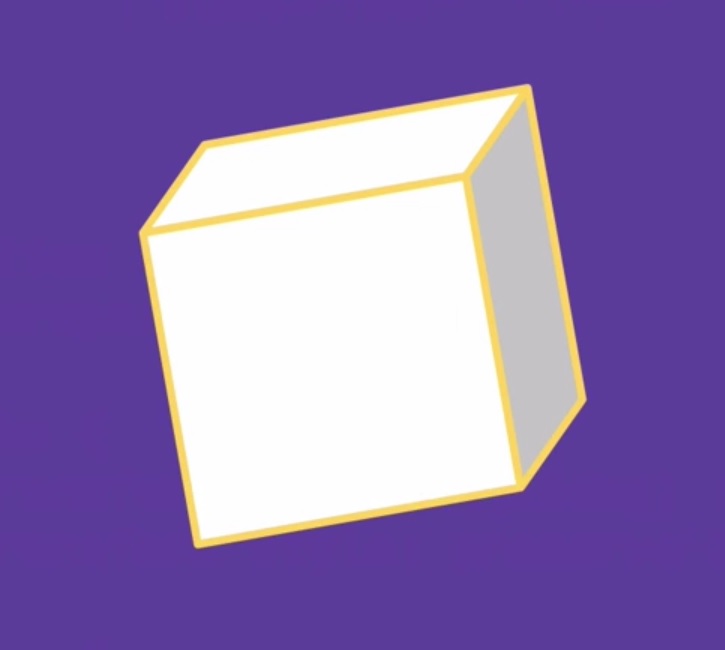
</section>

...

</main>

O elemento *section* precisa pegar a transformação de dentro do cubo também. Para isso vamos utilizar a propriedade ***transform-style: preserve-3d*** dentro da classe "cubo", para que o navegador use as transformações feitas com cubo nos elementos dentro dele. Dessa forma preserva-se o efeito 3D de todos os elementos. Sem isso as faces não aparecem com esse efeito conjunto. O cubo rotacionaria, mas as faces continuariam com uma impressão achatada.

Nos utilizando de todas essas ferramentas, chegamos a um resultado como este:



Mais para frente de nosso curso, veremos como fazer para animar esse objeto.

**Barra de rolagem para cenas em 3D / propriedade *overflow***

Como vimos, podemos ter diversas cenas 3D em nossa página. Cada uma delas pode ter sua própria barra de rolagem. Para tal, utilizamos a propriedade ***overflow*** com o valor *scroll*. O conteúdo desse elemento vai aparecer cortado e o usuário poderá rolar para visualizá-lo

<main>

<section>

...

overflow: scroll

...

</section>

<section>

...

overflow: scroll

...

</section>

<section>

...

overflow: scroll

...

</section>

</main>

Se tivéssemos uma dessas *sections* com *10px* de altura e, dentro dela, um cubo de *100px*de altura, o usuário veria uma barra de rolagem e poderia movê-la para visualizá-lo.

É importante notar que esse *scroll* influencia na *perspective-origin*, ou seja, ao rolarmos a barra em uma cena 3D é como se estivéssemos nos deslocando, mudando a perspectiva. Os objetos que estiverem "mais perto" se deslocarão mais rápido que aqueles que estiverem "mais longe", dando o efeito de uma movimentação. Chamamos esse efeito de *Paralaxe*.

Vimos que é possível combinar mais de uma transformação em um elemento. Por exemplo, para rotacionar e transladar uma div, podemos fazer

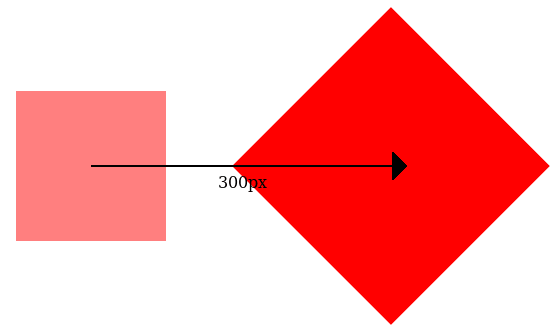
div {

transform: translateX(20px) rotate(45deg);

}

Porém, quando colocamos mais de uma transformação, a ordem entre elas importa, já que o navegador as aplica **da direita para a esquerda**. Ou seja, no exemplo acima, a divserá primeiro rotacionada e depois transladada.

Sabendo disso, qual declaração devemos fazer para conseguir o resultado abaixo?



*Selecione 2 alternativas*

transform: translateX(300px) scale(1.5) rotate(45deg);

transform: translateX(300px) rotate(45deg) scale(1.5);

Outro lugar da página inicial em que uma palavra rotacionada chama a atenção é na mensagem de saudação. O texto já é bem colorido, com tamanhos diferentes de fonte, mas o que deve chamar a atenção mesmo é a palavra "maravilhosos".



Para conseguir o efeito desejado, vamos aplicar duas transformações. A primeira é a skew, para entortar um pouco a palavra. Com isso, vamos dar a impressão de que as letras estão subindo na página, quando combinarmos a skew com a rotate.

Porém, há um **detalhe** nesse caso. Estamos aplicando transformações num elemento que é inline por padrão, e **transformações não funcionam com elementos inline**. Então precisaremos transformá-lo num inline-block.

Implemente essas transformações no texto, acrescentando o código no index.css.

Resposta do instrutor: Podemos usar valores diferentes de rotação e cisalhamento para a palavra, mas o código final ficará bem próximo do abaixo.

.saudacao em {

color: #851944;

font-size: 4em;

transform: rotate(-5deg) skew(-10deg);

display: inline-block;

}

Repare que usamos o display: inline-block para que a transformação funcione e fazemos **primeiro** o skew para **depois** fazer o rotate. Se fizermos na ordem inversa, ficaremos com um resultado estranho, apesar de a diferença ser sutil.

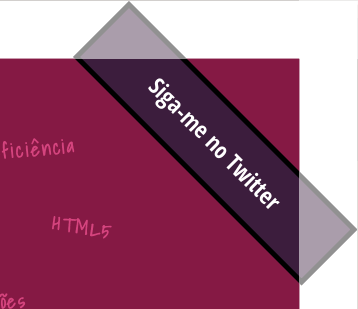
**Banner do Twitter**

Um efeito bem interessante que a propriedade transform nos permite criar e colocar um banner rotacionado no canto da tela, como se fosse uma etiqueta colada na página. No caso da página do João da Silva, podemos colocar um banner como esse para chamar a atenção do visitante para o perfil do João no Twitter.

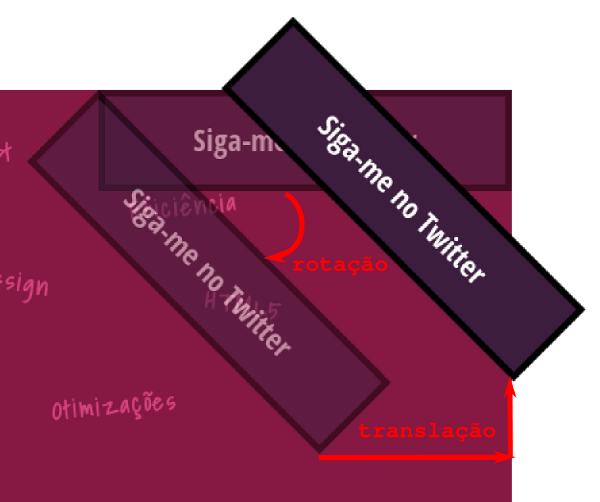


Para conseguir esse efeito, precisamos criar um link, adicionar uma cor de fundo e bordas para ele e posicioná-lo no canto superior direito da tela (o position: absoluteresolve isso para nós).

A parte complicada aqui é rotacionar e reposicionar o link de modo que não sobre nem falte espaço ao redor dele. Apenas as pontas superiores da caixa do link devem ficar para fora, como mostra a figura abaixo.



Não basta rotacionar o link, é preciso também fazer uma translação, tanto na vertical como na horizontal.



Repare que a rotação foi feita pelo canto superior esquerdo do link, e não pelo centro do elemento, que é o padrão do navegador. Para mudarmos esse padrão, usamos a propriedade transform-origin. Como queremos que o canto superior seja a base para a rotação, usamos a declaração

transform-origin: 0 0;

Agora resta a questão: como fazer a translação? Com um pouco de trigonometria, chegamos nas seguintes fórmulas para o quanto temos que transladar na horizontal e na vertical.

Na horizontal: cos(45) \* (altura - largura) + largura

Na vertical: -1 \* sen(45) \* altura

**Para simplificar**, podemos definir uma largura para o link, independente da largura do texto dentro dele. Já a altura conseguimos calcular com facilidade se usarmos a medida em e a declaração line-height: 1 para que o texto tenha a altura de 1em exatamente.

Com base nessas informações, **implemente** esse link na página inicial do João da Silva.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

O primeiro passo é adicionar o link ao cabeçalho da página:

<header class="titulo-principal">

<img class="foto-home" src="imagens/eu.jpg" alt="Foto de João da Silva">

<h1>João da Silva</h1>

<p class="subtitulo-principal">Desenvolvedor web</p>

<ul class="palavras-home">

<!-- palavras aqui -->

</ul>

<a class="banner-twitter" href="http://twitter.com/joaodasilva">Siga-me no Twitter</a>

</header>

Em seguida, configuramos cores, fonte, bordas e tamanho, usando *opcionalmente* a propriedade box-sizing para facilitar as contas depois:

.banner-twitter {

width: 14em;

line-height: 1;

padding: 1em 0;

box-sizing: border-box;

text-align: center;

text-decoration: none;

background-color: #3C1D3D;

color: white;

border: .25em solid black;

font-size: 1.2em;

font-family: "Open Sans Condensed", sans-serif;

}

Fazendo assim, sabemos que a caixa tem largura de 14em e altura de 3.5em, já que as bordas somam 0.5em, o texto tem 1em de altura e temos mais 1em de padding em cima e abaixo do texto.

Substituindo na fórmula do enunciado, vemos que precisamos transladar o link 6.5765em na horizontal e -2.4745em na vertical. Então agora podemos incluir o posicionamento e a transformação do link no CSS, não esquecendo que precisamos fazer **primeiro** a rotação, e tudo isso com base no canto superior esquerdo do elemento:

.banner-twitter {

position: absolute;

top: 0;

right: 0;

transform-origin: 0 0;

transform: translate(6.5754em, -2.4749em) rotate(45deg);

}

**Palavras em 3D**

Já conseguimos um efeito legal rotacionando as palavras aleatoriamente no cabeçalho da página inicial. Agora vamos além: vamos deslocá-las no eixo Z, isto é, vamos entrar no mundo 3D!

Para isso, como vimos, precisamos definir uma perspectiva. Então **adicione** uma perspectiva ao elemento <ul> que contém as palavras, faça com que esse elemento **ocupe o cabeçalho inteiro** (você pode usar o position: absolute para isso) e **translate** as palavras aleatoriamente no eixo Z para conseguir um efeito como o da imagem abaixo.



*Talvez* você tenha que ajustar a posição das palavras após adicionar a translação no eixo Z. Para evitar que isso aconteça novamente mais para a frente, **adicione também** a transformação translate(-50%, -50%) **após** as outras transformações. Isso vai fazer com que o top e o left que estamos usando para posicionar as palavras definam a posição central delas, e não o canto superior esquerdo. Assim, se as palavras mudarem de tamanho, não será necessário reposicioná-las.



VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

Primeiro, adicionamos a perspectiva ao elemento <ul> que contém as palavras e fazemos com que ele ocupe o cabeçalho todo. Podemos adicionar uma classe a ele palavras-home, por exemplo para, no CSS, adicionarmos o seguinte código:

.palavras-home {

perspective: 6px;

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

}

E o HTML:

<ul class="palavras-home">

<li class="palavra-home eficiencia">Eficiência</li>

<li class="palavra-home boas-praticas">Boas práticas</li>

<li class="palavra-home codigo-limpo">Código limpo</li>

<li class="palavra-home css3">CSS3</li>

<li class="palavra-home html5">HTML5</li>

<li class="palavra-home javascript">JavaScript</li>

<li class="palavra-home acessibilidade">Acessibilidade</li>

<li class="palavra-home responsivo">Responsivo</li>

<li class="palavra-home otimizacoes">Otimizações</li>

<li class="palavra-home agilidade">Agilidade</li>

<li class="palavra-home design">Design</li>

</ul>

Outra forma de fazer isso é definindo o bottom e o right como zero também. Isso faz com que o elemento estique para grudar nos quatro cantos do cabeçalho:

.palavras-home {

perspective: 6px;

position: absolute;

top: 0;

left: 0;

bottom: 0;

right: 0;

}

Com isso, agora conseguimos aplicar as transformações nas palavras. Fazemos elas na seguinte ordem: translação no eixo Z, rotação e translação nos eixos X e Y. O código deve ficar mais ou menos assim:

.eficiencia {

top: 50%;

left: 60%;

transform: translate(-50%, -50%) rotate(-5deg) translateZ(4px);

}

.boas-praticas {

top: 70%;

left: 20%;

transform: translate(-50%, -50%) rotate(10deg) translateZ(1px);

}

.codigo-limpo {

top: 45%;

left: 25%;

transform: translate(-50%, -50%) rotate(-10deg) translateZ(2px);

}

.css3 {

top: 55%;

left: 35%;

transform: translate(-50%, -50%) rotate(5deg) translateZ(2px);

}

.html5 {

top: 30%;

left: 65%;

transform: translate(-50%, -50%) rotate(10deg) translateZ(2px);

}

.javascript {

top: 15%;

left: 60%;

transform: translate(-50%, -50%) rotate(-5deg) translateZ(1px);

}

.acessibilidade {

top: 30%;

left: 35%;

transform: translate(-50%, -50%) rotate(-5deg) translateZ(3px);

}

.responsivo {

top: 40%;

left: 40%;

transform: translate(-50%, -50%) rotate(10deg) translateZ(3px);

}

.otimizacoes {

top: 70%;

left: 65%;

transform: translate(-50%, -50%) rotate(-5deg) translateZ(2px);

}

.agilidade {

top: 75%;

left: 35%;

transform: translate(-50%, -50%) rotate(5deg) translateZ(1px);

}

.design {

top: 40%;

left: 60%;

transform: translate(-50%, -50%) rotate(10deg) translateZ(2px);

}

**Melhorando a qualidade das palavras**

Após o exercício anterior, você deve ter notado que as palavras ficaram meio borradas. Isso acontece porque o navegador tenta aumentar as palavras que estão mais próximas na perspectiva. Só que o navegador não faz isso aumentando a fonte; ele faz isso aumentando os *pixels* da fonte, e por isso a qualidade não é tão boa.

Podemos contornar isso **aumentando** o tamanho da fonte fora da transformação. Assim, na transformação, podemos diminuir a palavra e depois aumentá-la o quanto necessário. A fonte original, por ter uma resolução maior, vai garantir que, nesse segundo aumento, a qualidade será mantida.

Ou seja, no nosso código, queremos aumentar quatro vezes (ou mais) o tamanho da fonte para depois usar a transformação scale e voltar ao tamanho original **antes** de fazer as outras transformações:

.palavras-home li {

font-size: 400%;

}

.eficiencia {

top: 50%;

left: 60%;

transform: translate(-50%, -50%) rotate(-5deg) translateZ(4px) scale(0.25);

}

/\* mesma coisa para outras palavras \*/

Faça essas alterações no seu código.

Na aula, vimos que é possível montar um cubo com HTML e CSS. Vamos fazer isso agora. Para facilitar, vamos fazer isso num serviço online para testar código HTML e CSS. Dois bem legais são o [Dabblet](http://dabblet.com/) e o [CSS Deck](http://cssdeck.com/).

No HTML, precisamos de um elemento para cada face do cubo e de um elemento para representar o cubo inteiro. Podemos ter, então:

<div class="cubo">

<div class="face"></div>

<div class="face"></div>

<div class="face"></div>

<div class="face"></div>

<div class="face"></div>

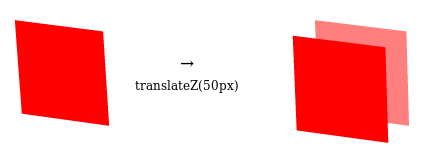
<div class="face"></div>

</div>

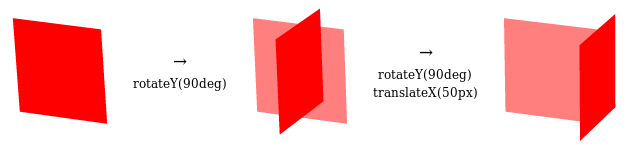
No CSS, precisamos definir o tamanho das faces e posicioná-las com relação ao cubo. Então podemos deixar o elemento com a classe cubo com position: relative e, dentro dele, as faces com position: absolute e um tamanho. Podemos também colocar uma cor em cada face, para um efeito visual mais legal.

Mas o mais importante aqui é posicionar essas faces no espaço 3D! Para isso precisamos rotacionar e transladar cada uma das faces para o seu devido lugar.

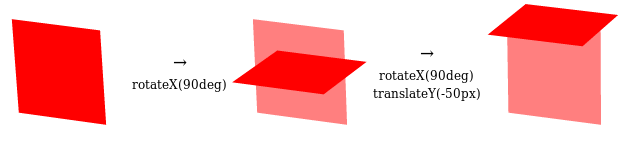
A primeira face temos que trazer para mais perto do usuário. Afinal, agora estão todas no meio do cubo! Então fazemos a translação dela no eixo Z.



A segunda face será a lateral direita do cubo. Então precisamos virá-la para o lado direito primeiro. Para isso, fazemos uma rotação no eixo Y de 90 graus. Depois disso, transladamos ela no eixo X para colocá-la no lado do cubo.



Fazemos o mesmo para a face da lateral esquerda, porém rotacionando -90 graus. E fazemos algo parecido com as faces superior e inferior: rotacionamos agora no eixo X 90 (ou -90) graus e transladamos no eixo Y.



Por fim, precisamos rotacionar a face traseira 180 graus (no eixo X ou Y) para que ela fique virada para trás do cubo e, depois, transladá-la no eixo Z para deixá-la mais distante do usuário.

Agora, **nada disso adianta** se não definirmos a perspectiva. E aqui vale a pena definir a perspectiva no body! Assim, podemos mover e rotacionar nosso cubo inteiro à vontade depois, e **vamos querer isso**. Só que isso vai nos obrigar a usar o transform-style: preserve-3d no elemento com a classe cubo para que a perspectiva seja aplicada às faces do cubo.

Com isso definido, agora podemos fazer uma rotação do cubo (10 graus nos eixos X e Y) para enxergar melhor nosso cubo.

**Implemente** esse código.(Páginas diversos.html)

No final da aula, vimos que é possível recriar o efeito de paralaxe (objetos mais próximos têm velocidade aparente maior) usando apenas CSS. Vamos fazer isso agora com as palavras da página inicial. Elas já têm um translateZ configurado, então basta fazer com que o navegador considere que a rolagem da página seja interpretada como uma movimentação na cena 3D.

Para isso, precisamos definir a perspectiva não mais no cabeçalho, mas no body. Além disso, precisamos definir a altura dos elementos html e body como 100%. Por fim, após o exercício do link para o Twitter, você deve ter notado que apareceu uma barra de rolagem horizontal. Isso acontece porque estamos transladando parte do link para fora da página, e isso vai impedir o efeito de paralaxe de funcionar. Vamos dizer, então, para o navegador ignorar o que fica para fora da página colocando a declaração overflow-x: hidden nos dois elementos (html e body).

Depois, para testar, basta fazer a rolagem da página *vagarosamente*. Deveremos ver as palavras se movimentando em velocidades diferentes.

**Adicione** essas declarações ao index.css e **não esqueça** que vamos precisar usar a declaração transform-style: preserve-3d depois de **mover a declaração da perspectiva para o body**. E o perspective-origin: 50% 160px no body também.

Primeiramente, adicionamos a perspectiva ao body (removendo-a do cabeçalho) e adicionamos as regras height: 100% e overflow-x: hidden nele e no html:

html, body {

height: 100%;

overflow-x: hidden;

}

body {

perspective: 6px;

perspective-origin: 50% 160px;

}

Agora, para manter as transformações 3D funcionando nas palavras, precisamos adicionar a declaração transform-style: preserve-3d; nos elementos acima delas na hierarquia, que são a ul com a classe palavras-home e o header com a classe titulo-principal:

.palavras-home, .titulo-principal {

transform-style: preserve-3d;

}

Nessa aula estudaremos mais alguns efeitos para enriquecer nosso site na web. Dois deles são as sombras e a opacidade. Eles destacam os elementos ou dão efeito de iluminação e tridimensionalidade.

### Sombras

Vamos utilizar o título da página do João da Silva e aplicar uma sombra nele.



Para isso usamos a propriedade **text-shadow**, que recebe algumas configurações para dar impressão de tridimensionalidade, perspectiva, difusão de luz, além de escolher a cor dessa sombra:

h1 {

text-transform: uppercase;

font-size: 3em;

text-shadow: 5px 5px #000;

}

Deslocamos a sombra para a direita e para baixo e sua cor é preta. Veja o resultado:



Lembre-se que podemos aplicar a sombra em qualquer texto do site. E vamos além: também conseguimos usar o efeito em outros elementos que não são texto utilizando o **box-shadow**. Façamos isso com a foto do João da Silva:

.foto-home {

height: 200px;

border-radius: 50%;

box-shadow: 0 0 1em #000;

}

Não deslocamos a sombra, a deixamos mais difusa e sua cor é preta. O resultado:



Não precisamos nos limitar a apenas uma sombra por elemento. Pode-se haver mais de uma. Vamos testar com o "botao-index":

.botao-index {

font-size: 1.25em;

background-color: #851944;

color: #FFF;

padding: .5em;

border: .2em solid black;

width: 40ch;

margin: 2em auto;

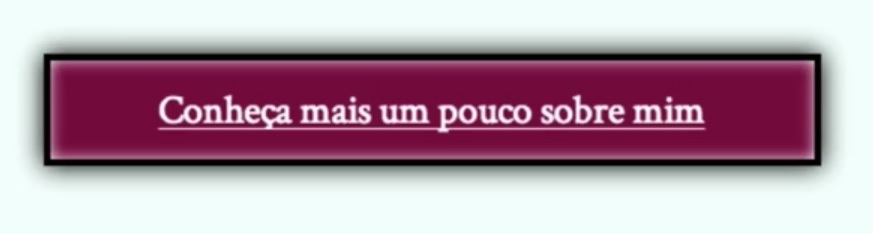
display: block;

text-align: center;

box-shadow: 0 0 1em #000, inset 0 0 .5em #FFF;

}

Inserimos uma sombra preta externa no botão e uma branca interna. O resultado:



### Opacidade e cores semitransparentes

Outro efeito que podemos inserir nos elementos é a opacidade, com a propriedade **opacity**. Vamos testá-la com algumas palavras de nossa página:

.javascript {

top: 15%;

left: 60%;

opacity: .3;

...

}

.acessibilidade {

top: 30%;

left: 35%;

opacity: .8;

...

}

Sua escala vai de 0 a 1, sendo 0 totalmente invisível e 1 totalmente visível.

Vamos aplicar essa propriedade para o rodapé da página, que nesse momento está totalmente preto, não permitindo a leitura do texto que está atrás dele:

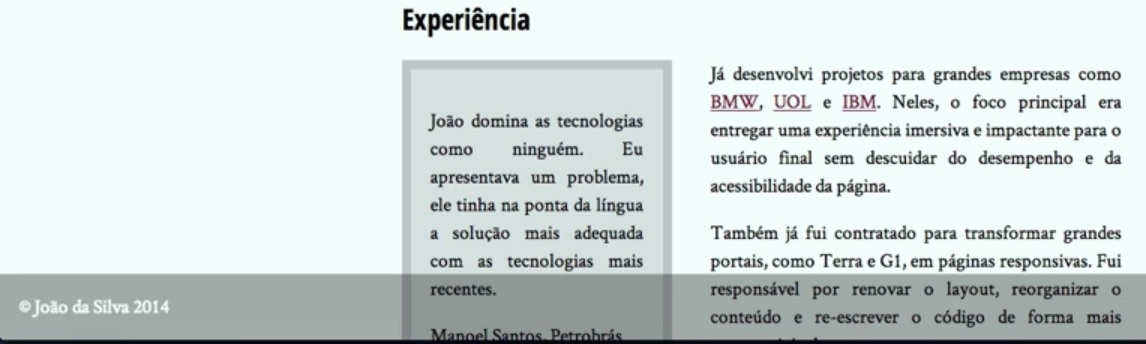
.rodape-pagina {

...

opacity: .3;

}

Perceba que agora conseguimos ver através dele:



Ao utilizar essa propriedade em objetos que não são textos, mas que possuam textos, podemos cair no erro de escondê-los, pois o texto acompanha a opacidade definida para o elemento pai. Neste caso, se o texto também estivesse preto, ele sumiria, pois acompanhou a opacidade do rodapé inteiro.

A opacity não permite que modifiquemos apenas o fundo, para isso utilizaremos uma outra ferramenta do CSS, um truque com as cores. Vamos escolher uma que seja semitransparente, na escala RGB:

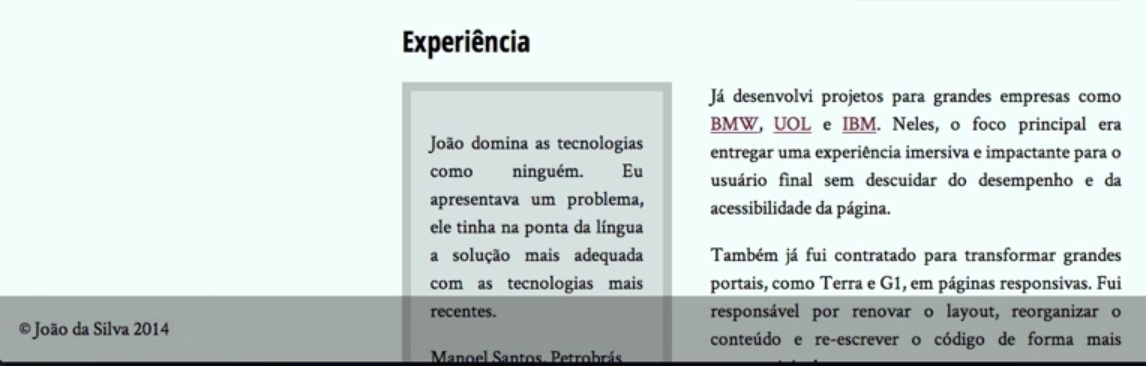
.rodape-pagina {

background-color: rgba(0, 0, 0, .3);

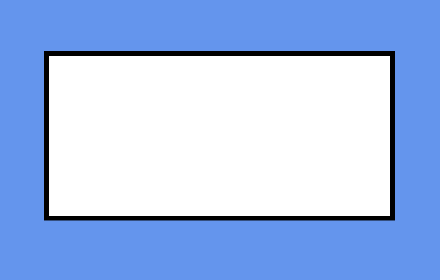
...

}

O "a" logo após a sigla RGB significa a opacidade da cor e ela é definida após definirmos o vermelho (R), o verde (G) e o azul (B). O resultado:



Com a propriedade box-shadow conseguimos alguns efeitos bem interessantes! Um efeito que muitas vezes vem a calhar é usar o box-shadow como uma borda no nosso elemento. Quando a sombra não tem borramento, seu visual é bastante parecido com o de uma borda, como mostra a imagem abaixo.

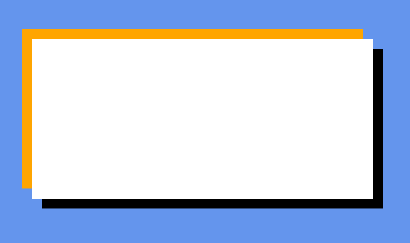


Esse efeito acima pode ser obtido com a declaração box-shadow: 0 0 0 5px black: fazemos a sombra *crescer* ao redor do elemento.

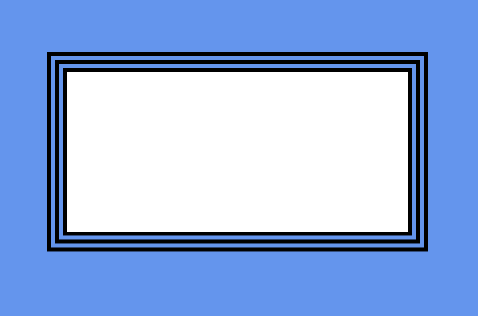
Além disso, podemos ter mais de uma sombra num mesmo elemento. Basta separar as sombras por vírgula na declaração da propriedade:

box-shadow: 10px 10px black, -10px -10px orange;

E temos o seguinte resultado:



Com base nessas informações, tente construir o efeito abaixo, que simula múltiplas bordas usando a propriedade box-shadow.



Para adicionar uma "borda", já vimos que basta aumentar a sombra ao redor do elemento:

box-shadow: 0 0 0 4px black;

Agora, para adicionar uma segunda "borda", queremos adicionar mais uma sombra. Essa segunda sombra tem que ser transparente e ser **maior** do que a primeira. Então fica:

box-shadow: 0 0 0 4px black,

0 0 0 8px transparent;

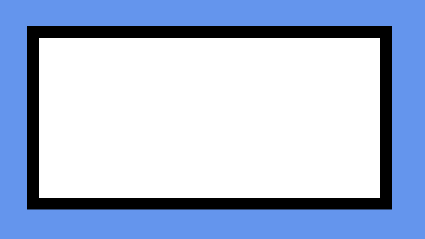
A mesma coisa para a terceira borda, novamente preta:

box-shadow: 0 0 0 4px black,

0 0 0 8px transparent,

0 0 0 12px black;

Opa! Nossa sombra transparente não aparece no meio:



Isso acontece porque a sombra transparente **é transparente**. Precisamos, na verdade, que ela seja da mesma cor do fundo por trás da caixa:

box-shadow: 0 0 0 4px black,

0 0 0 8px blue,

0 0 0 12px black;

Repetindo mais uma vez a combinação "borda azul" e "borda preta", chegamos no resultado desejado.

box-shadow: 0 0 0 4px black,

0 0 0 8px blue,

0 0 0 12px black,

0 0 0 16px blue,

0 0 0 20px black;

Para diminuir a visibilidade das palavras, podemos usar a propriedade opacity ou a propriedade color com uma cor semitransparente. Essa última solução, porém, vai deixar nosso código mais complicado de manter pois teremos a cor repetida várias vezes; se quisermos alterá-la, teremos muito trabalho.

A ideia é deixar as palavras mais distantes menos visíveis. Assim, as palavras com translateZ menor (que são as mais distantes do usuário) devem ter um valor de opacity menor.

O CSS final para as palavras deve ficar algo como:

.eficiencia {

top: 50%;

left: 60%;

transform: translate(-50%, -50%) translateZ(4px) rotate(-5deg) scale(.25);

opacity: .3;

}

.boas-praticas {

top: 70%;

left: 20%;

transform: translate(-50%, -50%) translateZ(1px) rotate(10deg) scale(.25);

opacity: .3;

}

.codigo-limpo {

top: 45%;

left: 25%;

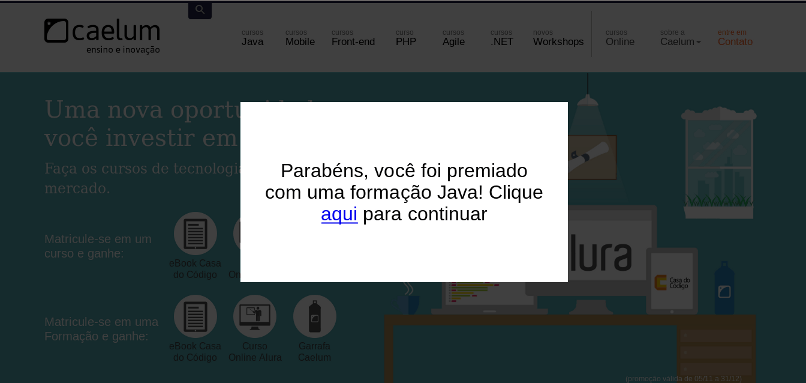
transform: translate(-50%, -50%) translateZ(2px) rotate(-10deg) scale(.25);

opacity: .6;

}

...

Além do efeito de múltiplas bordas que fizemos anteriormente com a propriedade box-shadow, um outro efeito bastante interessante que conseguimos com essa propriedade é esconder uma página quando um diálogo modal é exibido, como na imagem abaixo.



**Usando o console do navegador**, vamos criar esse efeito no site do Alura. Para isso, o primeiro passo é adicionar um novo elemento ao body da página (pode ser o primeiro elemento para facilitar). Depois estilizamos esse elemento dando tamanho, cor de fundo e posicionamento. Por fim, para criar o efeito de esconder a página, usamos o box-shadow com uma cor semitransparente. Para que ela ocupe toda a página, fazemos com que ela **cresça** bastante (um número mágico como 9999px deve resolver). E, para garantir que ele vai aparecer acima de todos os outros elementos da página, usamos a propriedade z-index com um valor bem grande (outro número mágico como 9999resolve). Não vamos ver essa propriedade a fundo, mas são poucos os casos em que precisamos dela.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

Primeiro, definimos tamanho e cor de fundo para a nossa caixa de diálogo:

width: 40%;

background-color: #FFF;

Em seguida, podemos partir para o posicionamento. Queremos que a caixa fique sempre fixa no meio da tela, **independente da rolagem da página**. Para isso, precisamos usar o position: fixed. E como queremos que ela apareça no meio da tela, podemos pensar em usar top: 50% e left: 50%.

position: fixed;

top: 50%;

left: 50%;

Porém, isso vai fazer com que ela apareça no canto inferior direito da tela, pois com o position e as outras duas propriedades estamos dizendo que *o canto superior esquerdo*da caixa deve ficar na metade da tela. Para corrigir isso, podemos usar o truque de **transladar** a caixa para cima e para a direita:

transform: translate(-50%, -50%);

E, para garantir que ela vai aparecer acima de todos os outros elementos da página, usamos a propriedade z-index:

z-index: 9999;

Por fim, adicionamos a sombra. Como queremos que ela cresça bastante, **temos que mexer no quarto número** do box-shadow; os outros podem todos ser zero:

box-shadow: 0px 0px 0px 9999px rgba(0, 0, 0, 0.75);

Para melhorar um pouco o visual, podemos adicionar um espaçamento interno na caixa, aumentar um pouco a fonte e centralizar o texto:

padding: 10% 3%;

text-align: center;

font-size: 2em;

Veremos mais alguns efeitos que enriquecem nossa página, para os quais antigamente precisávamos usar imagens.

Antes, para criar um efeito gradiente num elemento, nós criávamos uma imagem em um editor e a exportávamos para o CSS, o que causava uma série de complicações em relação ao seu tamanho e posição.

### Gradientes

Hoje em dia o próprio navegador cria esses efeitos visuais utilizando a propriedade background-image, só que, ao invés de colocar o caminho para uma imagem, nós vamos colocar uma instrução para criar um gradiente.

No CSS, chamamos o título principal e escolhemos uma imagem de fundo com o valor **linear-gradient()**, que receberá alguns parâmetros:

.titulo-principal {

background-image: linear-gradient(to bottom, #F00, #000);

}

O "to bottom" indica que o gradiente acontece de cima para baixo. Logo depois escolhemos as cores dos gradientes, neste caso do vermelho (#F00) ao preto (#000). Como resultado temos:



Perceba que o gradiente não precisa ser apenas de duas cores, podemos acrescentar mais uma cor central, o cinza, por exemplo:

.titulo-principal {

background-image: linear-gradient(to bottom, #F00, #888, #000);

}



Podemos modificar o lugar em que ocorre a transição de cores. Queremos aqui que a mudança de vermelho para cinza ocorra a uns 80% da altura da imagem, de cima para baixo:

.titulo-principal {

background-image: linear-gradient(to bottom, #F00, #888 80%, #000);

}



Por padrão, o gradiente ocupa 100% do elemento. Mas, com o CSS3, podemos agora definir quanto do elemento o gradiente deve pegar. Usamos a propriedade background-size para redimensionar a imagem ou o gradiente:

.titulo-principal {

background-image: linear-gradient(to bottom, #F00, #888 80%, #000);

background-size : 100% 200px;

}

Aqui ele terá 100% da largura do background e 200 pixeis de altura. O resultado:



O gradiente se repete automaticamente. Podemos desligar esse comportamento utilizando a propriedade background-repeat:

.titulo-principal {

background-image: linear-gradient(to bottom, #F00, #888 80%, #000);

background-size : 100%, 200px;

background-repeat: no-repeat;

}



Também podemos controlar o gradiente para repetir em apenas um sentido (eixo x ou eixo y).

Com todas essas ferramentas podemos criar fundos muito criativos, e é o que veremos nos exercícios.

Por último vejamos a propriedade background-attachment, com o valor "fixed". Ela dá aquele efeito de paralaxe na imagem ou na textura do background. É como se elas estivessem fixas à tela do computador.

.titulo-principal {

...

background-attachment: fixed;

}

Qual das declarações abaixo cria um gradiente como o da imagem, com a cor branca no canto superior esquerdo, a cor vermelha no meio e a cor preta no canto inferior direito?



background-image: linear-gradient(to bottom right, white, red, black);

Podemos usar um código parecido com o do último exercício para substituir a borda sólida que existe atualmente entre os posts na página blog.html por uma borda mais suave, como mostram as imagens abaixo.

**Antes:**



**Depois**



Conseguimos isso **trocando** a declaração border-bottom por um gradiente. **Implemente**essa borda suave.

VER OPINIÃO DO INSTRUTOR

### Opinião do Instrutor

Para fazer o efeito de borda suave, começamos com um gradiente da esquerda para a direita, indo de transparente para preto e de volta para transparente:

background-image: linear-gradient(to right, transparent, black, transparent);

Isso vai fazer com que o gradiente ocupe o elemento inteiro. Para deixá-lo apenas na parte **inferior** do post, precisamos usar as propriedades background-size, background-repeat e background-position. Reduzimos o tamanho do gradiente para 1 pixel de altura com o background-size:

background-size: 100% 1px;

Para que o gradiente não fique repetindo no elemento inteiro, usamos o background-repeat:

background-repeat: no-repeat;

Por fim, para fazer com que ele apareça na parte inferior do post, usamos o background-position:

background-position: bottom center;

Podemos deixar a borda mais suave usando um cinza claro no lugar do preto:

background-image: linear-gradient(to right,

rgba(200, 200, 200, 0),

rgba(200, 200, 200, 1),

rgba(200, 200, 200, 0));

O navegador também é capaz de gerar *gradientes radiais*, nos quais a cor muda a partir de um ponto em todas as direções.



Para conseguir o efeito acima, basta fazer a declaração

background-image: radial-gradient(yellow, red);

Assim como no gradiente linear, é possível especificar várias paradas de cor. É possível ainda mudar a posição central do gradiente com relação ao elemento e restringir o gradiente à forma redonda ou permitir um gradiente oval. Para isso, passamos um primeiro parâmetro com essas especificações:

background-image: radial-gradient(circle at bottom left, yellow, red);



Sabendo disso, qual das declarações abaixo gera o seguinte resultado?



Parte superior do formulário

* Alternativa correta



background-image: radial-gradient(circle at bottom right,

yellow,

yellow 20%,

black 20%,

black 30%,

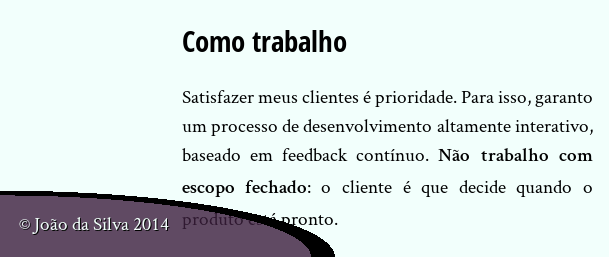
yellow 30%,

yellow);

Parte inferior do formulário

Vamos agora aplicar um gradiente radial no site do João da Silva. O rodapé das páginas já ficou mais discreto com a cor semitransparente mas, do jeito que está agora, ainda ocupa muito espaço na tela!

Para melhorar isso, vamos deixar um fundo semitransparente apenas na região da mensagem de copyright, como na imagem abaixo, além de adicionar uma sombra discreta ao texto.



O gradiente ocupa 25% da largura do rodapé e toda a altura dele, e tem a cor rgba(60, 29, 61, 0.8) na parte interna e borda preta.

Implemente essa alteração no arquivo site.css.

**Dica:** Não esqueça de remover o background-color que atualmente está no rodapé da página.

VER OPINIÃO DO INSTRUTOR

### Opinião do Instrutor

O gradiente é radial, oval (elíptico) e centrado no canto inferior esquerdo do rodapé. Então começamos com

background-image: radial-gradient(ellipse at left bottom,

/\* cores aqui \*/);

A cor rgba(60, 29, 61, 0.8) vai do centro até aproximadamente 65% do tamanho total do gradiente, trocando em seguida imediatamente para preto, então continuamos com

background-image: radial-gradient(ellipse at left bottom,

rgba(60, 29, 61, 0.8),

rgba(60, 29, 61, 0.8) 65%,

black 65%,

/\* mais cores aqui \*/);

Por fim, essa cor preta fica por aproximadamente 5% do gradiente e depois desaparece, ou seja, é imediatamente substituída por uma cor transparente. Então temos

background-image: radial-gradient(ellipse at left bottom,

rgba(60, 29, 61, 0.8),

rgba(60, 29, 61, 0.8) 65%,

black 65%,

black 70%,

transparent 70%,

transparent);

Porém, se fizermos simplesmente assim, a elipse ocupará a largura inteira da página. Para restringir o gradiente para apenas 25% de 100% da largura da página, definimos o tamanho e a repetição do fundo:

background-image: radial-gradient(ellipse at left bottom,

rgba(60, 29, 61, 0.8),

rgba(60, 29, 61, 0.8) 65%,

black 65%,

black 70%,

transparent 70%,

transparent);

width:100%;

background-size: 25% 100%;

background-repeat: no-repeat;

Crie o efeito de fundo listrado no fundo das seções **Trabalhos** e **Blog** da página inicial. As linhas da seção **Trabalhos** podem ser da cor rgba(0, 0, 0, 0.1) e as linhas da seção **Blog**, da cor rgba(255, 255, 255, 0.1).

Além disso, as linhas de ambas as seções devem ser na diagonal, porém a seção **Blog**deve ter linhas na diagonal contrária à seção **Trabalhos**. E, por fim, **as linhas devem ficar fixas** quando a página é rolada.

VER OPINIÃO DO INSTRUTOR

### Opinião do Instrutor

Para a seção **Trabalhos**, podemos fazer o gradiente direcionado para o canto inferior direito. Para isso, começamos nosso gradiente com a direção to bottom right ou, se quisermos usar a notação de graus, 135deg:

.trabalhos {

background-image: linear-gradient(135deg,

/\* cores aqui \*/);

}

Queremos criar uma linha fina com a cor rgba(0, 0, 0, 0.1) nessa seção. Para isso, até 14 pixels vamos usar a cor transparente, trocando subitamente para a cor semitransparente, usando-a por 1 pixel e voltando subitamente para transparente:

.trabalhos {

background-image: linear-gradient(135deg,

transparent,

transparent 14px,

rgba(0, 0, 0, 0.1) 14px,

rgba(0, 0, 0, 0.1) 15px,

transparent 15px,

transparent);

}

Agora, para que essa linha se repita pelo elemento todo, definimos o tamanho da imagem de fundo. Vamos usar 20 pixels. E, para que as linhas não se movimentem com a rolagem, adicionamos também o background-attachment: fixed. Então ficamos com:

.trabalhos {

background-image: linear-gradient(135deg,

transparent,

transparent 14px,

rgba(0, 0, 0, 0.1) 14px,

rgba(0, 0, 0, 0.1) 15px,

transparent 15px,

transparent);

background-size: 20px 20px;

background-attachment: fixed;

}

Para a seção **Blog**, o código é bem parecido. Mudam apenas o ângulo do gradiente e as cores:

.blog {

background-image: linear-gradient(45deg,

transparent,

transparent 14px,

rgba(255, 255, 255, 0.1) 14px,

rgba(255, 255, 255, 0.1) 15px,

transparent 15px,

transparent);

background-size: 20px 20px;

background-attachment: fixed;

}

Uma das coisas importantes de se trabalhar com CSS é selecionar elementos. Precisamos selecionar os elementos para aplicar os estilos.

No curso passado vimos quatro formas de selecionar elementos:

* tag
* id
* classe
* hierarquia pai-filho

Será que esses seletores conseguem resolver todos os problemas? Observe o seguinte caso:

Temos uma lista de elementos e queremos espaçá-los:

* Item 1
* Item 2
* Item 3

Para isso poderíamos usar o margin-top:

* Item 1

margin-top

* Item 2

margin-top

* Item 3

Implementando fica:

li {

margin-top: 1 em;

}

O problema é que não podemos esquecer que o margin-top se aplica para todos os itens, então o primeiro também terá essa distância a mais do topo da caixa de texto, o que não deixa muito bonito. Para solucionar isso, precisamos de um jeito para que a propriedade seja aplicada a partir do segundo item em diante.

A melhor maneira de se fazer isso é se utilizando do caractere "~" (til), cuja função é selecionar elementos depois de outros elementos:

li ~ li {

margin-top: 1em;

}

Um outro caso é quando queremos que o primeiro parágrafo de um texto logo após uma foto, por exemplo, não fique indentado, não possua aquele recuo na primeira linha, mas os outros sim.

<p>...</p>

<p>...</p>

<p>...</p>

p {

text-indent: 4ch;

}

Do modo acima todos os parágrafos estão indentados. Se usarmos o "~" (img ~ p) ele se aplicará a todos os parágrafos depois da imagem. E não é bem isso que queremos.

Para que só o primeiro parágrafo se modifique, utilizaremos o símbolo "+" (mais). Ele seleciona apenas o elemento imediatamente depois da imagem:

<img>

<p>...</p>

<p>...</p>

<p>...</p>

p {

text-indent: 4ch;

}

img + p {

text-ident: 0;

}

Perceba que existirá um problema se os parágrafos estiverem dentro de uma div:

<img>

<div>

<p>...</p>

<p>...</p>

<p>...</p>

</div>

p {

text-indent: 4ch;

}

img + p {

text-ident: 0;

}

Nesse caso o p está dentro de um elemento que está depois da imagem e o seletor será aplicado na div inteira. Ele funciona hierarquicamente entre elementos "irmãos".

### O ">"

Vamos ver outro caso, onde queremos um block-quote sem indentação, no meio de parágrafos indentados

<div>

<p>...</p>

<blockquote>

<p>...</p>

</blockquotes>

<p>...</p>

</div>

Se fizermos div p o seletor vai ser aplicado em todos os parágrafos, inclusive aquele dentro do block-quote. Para resolvermos isso utilizamos outro caractere, o ">" (maior).

Esse seletor indica que queremos aplicar a propriedade apenas no primeiro grau da hierarquia. Dessa maneira indentamos apenas os parágrafos fora do block-quote.

Então agora temos mais seletores que facilitarão nossa vida, além daqueles que vimos no curso anterior:

* tag
* id
* classe
* ancestral descendente
* irmão ~ irmão
* irmão + irmão seguinte
* pai > filho

Veja que podemos ainda combinar os seletores:

.noticia > h1 + p

Aqui estamos pegando todos os títulos "h1" dentro da classe ".noticia" e os parágrafos imediatamente depois desses "h1".

Esses novos seletores juntos com os já vistos nos dá muito poder para implementar os códigos. Porém pode ser complicado trabalhar com eles, a manutenção fica difícil. Além de que se quisermos acrescentar novos elementos, pode haver quebra da estrutura ou diminuição do desempenho.

A melhor alternativa é trabalharmos com as classes.

Além dos seletores que vimos em aula, existe mais um tipo de seletor: o de atributo. Assim como já conseguimos selecionar por id e por classes, conseguimos selecionar por qualquer outro atributo usando essa sintaxe. Por exemplo, dado o elemento

<a href="http://www.alura.com.br">

Podemos selecioná-lo com o seletor:

[href="http://www.alura.com.br"]

O seletor de atributo é bem poderoso! Podemos, além de selecionar pelo valor inteiro do atributo, selecionar apenas por uma parte dele ou simplesmente pela presença dele.

Por exemplo, se quisermos selecionar todas as imagens com a extensão .jpg de uma página, podemos usar o seletor:

[src$=".jpg"]

O cifrão representa que queremos que o atributo **termine com** um valor.

Se quisermos selecionar todos os links que começam com http://, podemos usar o seletor:

[href^="http://"]

Ou, se quisermos selecionar todos os elementos da página que tenham qualquer classe, basta usar o seletor [class]. Repare que é só o nome do atributo; se ele estiver presente, não importa seu valor, selecionamos ele.

Na aula passada vimos alguns seletores novos, o "~", o "+" e o ">". Apesar de muitas vezes nos ajudar, eles possuem alguns problemas como:

* Legibilidade: é difícil agrupar mais de um símbolo e deixar claro o que queremos com eles;
* Desempenho: o navegador terá muito trabalho para analisar toda a estrutura do HTML;
* Fragilidade: a chance de errarmos no código na hora de implementá-lo é grande.

Por essas e outras razões, vamos utilizar uma outra ferramenta que facilitará nossa vida e terá resultados parecidos com os dos seletores.

Imaginemos que vamos montar esta tabela zebrada, com as linhas de cores diferentes:



Seu HTML seria neste formato:

<table>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
</table>

Queremos dar um estilo para as linhas pares (*even*) e outro para as ímpares (*odds*). Podemos fazer isso criando classes:

<table>  
    <tr class="odd">...</tr>  
    <tr class="even">...</tr>  
    <tr class="odd">...</tr>  
    <tr class="even">...</tr>  
    <tr class="odd">...</tr>  
</table>

Essa formatação é muito comum, então não vamos inserir as classes dentro do HTML. Vamos, por sua vez, criar as ***pseudoclasses***.

As *pseudoclasses* são chamadas com os ":" (dois pontos):

:odd      
...  
:even

Na linguagem que estamos usando, generalizamos essas *pseudoclasses* com "*:nth-child()*". Dessa maneira, podemos estilizar não só as linhas pares ou ímpares:

* **:nth-child(odd)** - linhas ímpares;
* **:nth-child(even)** - linhas pares;
* **:nth-child(3)** - terceira linha;
* **:first-child** - primeira linha;
* **:last-child** - última linha.

Também podemos inserir fórmulas dentro do *:nth-child()* como *nth-child(2n)* que nada mais é do que as linhas pares. Ou *nth-child(2n+1)*, que são as ímpares.

Generalizando: **:nth-child(xn + c)**

Em que:

* **x** é a periodicidade, de quantos em quantos elementos será aplicado o estilo;
* **c** é o ponto de partida;
* **n** é a variável, que começa em 0 (zero).

É importante lembrar que não temos apenas a tabela na nossa página. Podemos ter parágrafos, por exemplo. Se aplicarmos **:nth-child(odd)** em um HTML desse tipo:

<p>...</p>  
<p>...</p>  
<table>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
    <tr>...</tr>  
</table>  
<p>...</p>  
<p>...</p>

O estilo será aplicado em todos os elementos ímpares, ou seja, no primeiro parágrafo, na tabela e no quinto parágrafo, além dos elementos filhos ímpares dentro da tabela. Então é comum especificarmos esse seletor agrupando-o com outro, a *tag*:

tr:nth-child(odd)

ou

table:nth-child(odd)

Neste caso usamos as *tags* tr e table para especificar o seletor na tabela, só aplicando nas linhas ímpares.

Não confunda o agrupamento de seletores, que é específico para o mesmo elemento e são escritos juntos:

table:first-child (a tabela que é o primeiro filho)

Com os seletores funcionando em nível hierárquico (elementos diferentes), escritos separadamente:

tr :first-child (a tabela é pai e "first-child" é seu primeiro filho)

Vamos pegar outro caso para resolvermos. Na página de biografia do João da Silva havia os *block-quotes* organizados dessa maneira:

1 <p>  
 2 <p>  
 3 <h2>  
 4 <blockquote>  
 5 <p>  
 6 <p>  
 7 <h2>  
 8 <blockquote>  
 9 <p>  
10 <p>  
11 <h2>  
12 <blockquote>

Tínhamos que inserir classes específicas para cada um. Vamos limpar o código utilizando o *nth-child* dessa maneira:

blockquote:nth-child(even)

Pode parecer que agora todos os *block-quotes* de número par terão o mesmo estilo. Porém a contagem da pseudoclasse vale para todo mundo, ou seja, ela pegará todos os *block-quotes* que estão em posições pares, nesse caso englobaria os três. Todos são filhos pares do *body*. Para resolver por fim esse problema, podemos pedir que conte apenas os *block-quotes* utilizando

blockquote:nth-of-type(even)

Dessa maneira a contagem da pseudoclasse fica

<p>  
  <p>  
  <h2>  
1 <blockquote>  
  <p>  
  <p>  
  <h2>  
2 <blockquote>  
  <p>  
  <p>  
  <h2>  
3 <blockquote>

Revendo, vimos nesta aula quatro pseudoclasses:

* **:first-child**
* **:last-child**
* **:nth-child**
* **:nth-of-type**

Com o uso das pseudoclasses, mantemos o poder que os seletores avançados possuíam, porém com elas temos uma melhor legibilidade e, por sua vez, um melhor desempenho.

Um outro recurso das pseudoclasses é que algumas delas são dinâmicas, o que nos permite fazer efeitos de interatividade. Imagine um exemplo onde temos um botão que muda de cor quando o ponteiro do *mouse* fica sobre ele:



Uma forma de fazer isso é via *javascript*:

<a href="...">Compre já</a>  
  
<a href="..." class="destaque">

Este é outro efeito muito comum nas páginas da web, então o próprio navegador pode inserir essa classe para nós:

<a href="..." class="hover">

A pseudoclasse *hover* (passar por cima) que nos permite esse efeito. O código para fazer funcionar recebe a classe ".botao" com a formatação que quisermos:

.botao {  
    border: 8xp solid purple;  
}



Quando o *hover* estiver no mesmo elemento, ou seja, o seletor será aplicado na classe *.botão* e na pseudoclasse *:hover*, o estilo muda:

.botao {  
    border: 8px solid purple;  
}  
  
.botao:hover {  
    background-color: purple;  
}



Com isso temos o efeito que queríamos sem precisar do *javascript*.

Então, quando estamos falando de *pseudoclasses* não nos limitamos às *estruturais*, temos uma gama de *dinâmicas* também:

<table>  
    <thead>  
        <tr>  
            <th style="text-align:center">Estruturais</th>  
            <th style="text-align:center">Dinâmicas</th>  
        </tr>  
    </thead>  
    <tbody>  
        <tr>  
            <td style="text-align:center">:first-child</td>  
            <td style="text-align:center">:hover</td>  
        </tr>  
        <tr>  
            <td style="text-align:center">:last-child</td>  
            <td style="text-align:center">:focus</td>  
        </tr>  
        <tr>  
            <td style="text-align:center">:nth-child</td>  
            <td style="text-align:center">:active</td>  
        </tr>  
        <tr>  
            <td style="text-align:center">:nth-of-type</td>  
            <td style="text-align:center">:checked</td>  
        </tr>  
    </tbody>  
</table>

## **Além dos seletores estruturais que vimos, temos ainda o :nth-last-child e o :nth-last-of-type, que fazem o mesmo que o :nth-child e o :nth-of-type, respectivamente, porém contando na ordem inversa. Ou seja, no HTML abaixo, o elemento com o id escolhido é o :nth-child(3)** e **o :nth-last-child(1).**

## **<article>    <p>Texto</p>    <p>Texto</p>    <p id="escolhido">Texto mais legal</p> </article>**

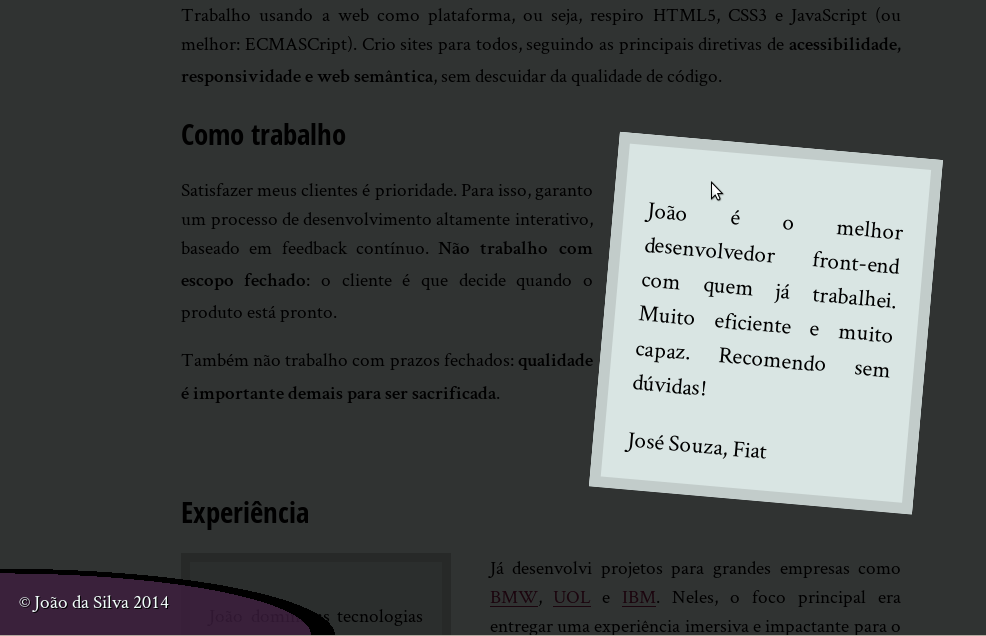
## **Vale observar que podemos selecionar por mais de uma classe ou pseudoclasse ao mesmo tempo! O seletor p:nth-child(3):nth-last-child(1) é perfeitamente válido, e seleciona os parágrafos que são o** terceiro e último **filhos de seus pais.**

## **Sabendo disso, qual seletor abaixo seleciona elementos que são** filhos únicos **de seus pais?**

nth-child(1):nth-last-child(1)

Numa das aulas anteriores, vimos em um exercício como fazer o efeito de uma caixa de diálogo modal com a propriedade box-shadow. E vimos também como aumentar e rotacionar elementos usando a propriedade transform.

Vamos combinar todos esses efeitos e a pseudoclasse :hover para criar um efeito interessante: quando o visitante passar o mouse sobre uma citação da página da biografia, a citação vai parecer saltar da página.



Mais do que isso, as citações do lado esquerdo da tela devem girar para um lado e as citações do lado direito, para outro.

Ou seja, quando o visitante passar o mouse sobre uma citação, devem acontecer as seguintes mudanças:

* Uma sombra deve cobrir o resto da página
* A citação deve ser aumentada 20% e levemente rotacionada (para a direita se estiver na direita e para a esquerda se estiver na esquerda)

Implemente esses efeitos no bio.css.

**VER OPINIÃO DO INSTRUTOR**

### Opinião do Instrutor

Para conseguir o efeito da sombra, adicionamos o box-shadow às citações quando elas tiverem a pseudoclasse :hover:

.citacao-bio:hover {  
    box-shadow: 0 0 0 99999px rgba(0,0,0,.8);  
}

Para adicionar o efeito de escala e rotação, mudando o lado da rotação de acordo com o lado da caixa, precisamos combinar a pseudoclasse :hover com a pseudoclasse :nth-of-type. Para selecionar os elementos do lado direito, usamos novamente :nth-of-type(odd) e, para rotacioná-los para a direita, passamos um valor positivo.

.citacao-bio:nth-of-type(odd):hover {  
    transform: scale(1.2) rotate(5deg);  
}

Para o lado esquerdo, usamos :nth-of-type(even) e um valor negativo para a rotação:

.citacao-bio:nth-of-type(even):hover {  
    transform: scale(1.2) rotate(-5deg);  
}

O efeito que acabamos de fazer é bem interessante, mas apenas visível para quem usa o mouse para navegar. Quem usa apenas teclado ou acessa de um celular ou tablet não vai conseguir ver esse efeito.

No entanto, podemos corrigir esse problema tornando as citações selecionáveis com o atributo tabindex do HTML. Esse atributo pode ser adicionado a qualquer elemento, tornando-o acessível por teclado e toque.

<blockquote tabindex="1">  
    <p>João é o melhor</p>  
    <cite>José Souza, Fiat</cite>  
</blockquote>

Repare que o atributo tabindex recebe um número. Esse número permite mudar a ordem de navegação pelo teclado, fazendo o foco do teclado ir para o final da página antes de ir para começo, por exemplo. Se colocarmos sempre o mesmo valor para todos os itens na página, a navegação segue a ordem do código HTML.

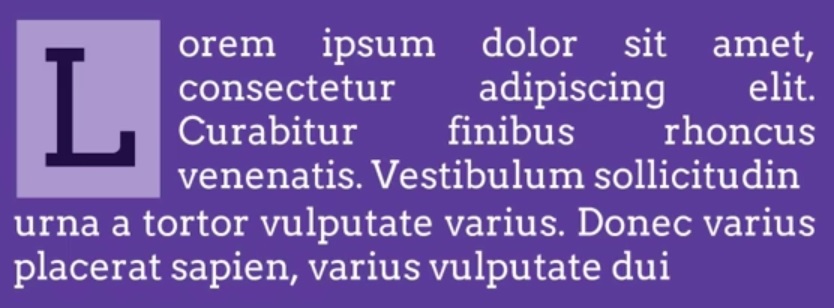
**Adicione** esse atributo aos <blockquote> da página bio.html e altere o CSS para que o efeito apareça quando o elemento ganha o foco do teclado (vá apertando a tecla TAB até chegar nele).

Qual alteração você teve que fazer no CSS?

.citacao-bio:hover,  
.citacao-bio:focus {  
    box-shadow: 0 0 0 99999px rgba(0,0,0,.8);  
}  
  
.citacao-bio:nth-of-type(odd):hover,  
.citacao-bio:nth-of-type(odd):focus {  
    transform: scale(1.2) rotate(5deg);  
}  
  
.citacao-bio:nth-of-type(even):hover,  
.citacao-bio:nth-of-type(even):focus {  
    transform: scale(1.2) rotate(-5deg);  
}

Vimos na aula passada que as *pseudoclasses* ajudam bastante no desenvolvimento de nosso site. Veremos que existem outras ferramentas para o melhorar ainda mais.

Veja este cenário comum, em que queremos deixar a primeira letra do primeiro parágrafo maior que as outras, a *Letra capitular*:



Precisamos achar um modo de selecionar apenas a primeira letra do parágrafo. Com o que aprendemos até agora, uma possível maneira de implementar esse efeito é criar um elemento ao redor da letra, por exemplo um *span*:

<p><span>L</span>orem ipsum...  
  
p span {  
    font-size: 3em;  
}

Porém, se esse é um efeito comum, será que o navegador não resolveria de outro jeito, como foi visto com as *pseudoclasses*? Sim, com os ***pseudoelementos***.

No nosso exemplo, o *pseudoelemento* que vamos usar é o "*:first-letter*":

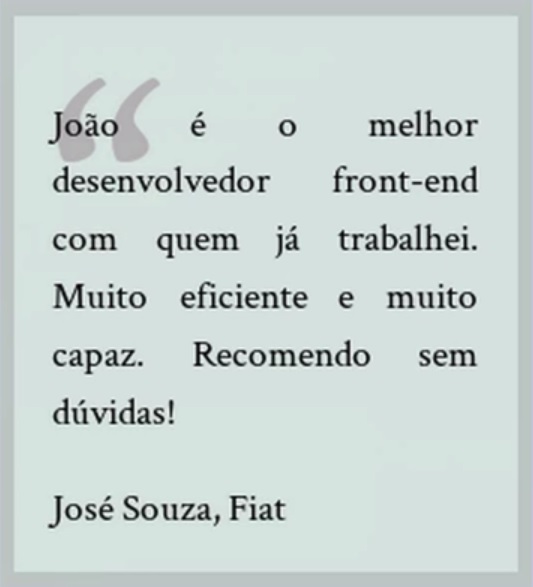
<p>Lorem ipsum...  
  
p:first-letter {  
    font-size: 3em;  
}

Ou

<p>Lorem ipsum...  
  
p::first-letter {  
    font-size: 3em;  
}

Essa segunda maneira, com "::", surgiu com o CSS3 e é o que vale daqui para frente.

Um outro caso em que os pseudoelementos nos ajuda é quando queremos dar um toque estético nas *block-quotes*:



Perceba o desenho de aspas apareceu ao fundo da caixa. Uma solução é colocarmos uma imagem de fundo e, como sabemos, é uma requisição a mais para o navegador. A alternativa para isso é estilizar usando o próprio texto com um pseudoelemento.

Sabemos que é possível com a propriedade *span* dentro do HTML, mas sempre devemos nos perguntar se esse elemento é um conteúdo ou se é apenas decorativo. As aspas são apenas decorativas, então não tem por que implementá-las no HTML, mas sim no CSS.

Então fazemos

blockquote:before

O problema é que o navegador não traduz diretamente isso como a inserção de aspas. Para ele poderia ser qualquer elemento, incluindo imagens. Então devemos inseri-las via CSS com a propriedade "*content*":

blockquote:before {  
    content: '"';  
}

Com isso, estilizamos o elemento como quisermos:

blockquote:before {  
    content: '"';  
    font-size: 10em;  
    position: absolute;  
}

O "*content*" é muito poderoso. Com ele podemos trabalhar com imagens, textos, marcadores, etc, além de combiná-lo com pseudoclasses como o "*:hover*". Ele é obrigatório mesmo se não for usar nenhum texto na estilização

Assim como as *pseudoclasses*, os *pseudoelementos*" permitem que nosso HTML fique muito mais limpo e semântico.

Quando usamos os pseudoelementos ::before e ::after e sua propriedade content, essa propriedade pode assumir valores baseados em atributos do elemento da página. Por exemplo, se tivermos uma <div> na página com a seguinte declaração:

<div title="Teste">

Conseguimos usar o texto desse atributo title nos pseudoelementos. Basta, no CSS, usar:

div::before {

content: attr(title);

}

Desta forma, conseguimos transformar o atributo title, que originalmente serve apenas para mostrar uma legenda quando o mouse é colocado sobre o elemento, em algo visual.

Vamos usar essa ideia para criar legendas interessantes para os links das redes sociais, que aparecerão quando passarmos o mouse ou o foco do teclado sobre eles.

Para evitar conflito com a funcionalidade do atributo title, vamos usar um recurso do HTML 5 que são os **atributos customizados**. A partir dessa versão do HTML, podemos criar nossos próprios atributos em qualquer elemento do HTML sem deixá-lo inválido; basta que esse atributo comece com data-. Por exemplo:

<!-- a declaração abaixo é válida a partir do HTML 5 -->

<a href="#" data-legenda="Clique em mim">Link</a>

A legenda, no final, deve aparecer como na imagem abaixo, com texto branco, fundo preto e posicionada abaixo do ícone.



**Observe que**, como usamos a propriedade text-indent para fazer o *image replacement*nos ícones das redes sociais, precisamos **desativar** esse text-indent no pseudoelemento e **tirar** o pseudoelemento do fluxo do documento.

Além disso, observe-que, como queremos que a legenda apareça apenas quando o usuário selecionar o elemento com o teclado ou passar o mouse sobre ele, precisaremos combinar pseudoclasses e pseudoelementos no seletor. Por exemplo: a:hover::before.

Implemente esse efeito usando o pseudoelemento ::after e o que foi discutido, adicionando-o ao arquivo site.css.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

O primeiro passo que precisamos fazer é adicionar o atributo customizado aos links das redes sociais. O nome do atributo pode ser qualquer um, desde que se mantenha consistente no HTML e no CSS posteriormente:

<ul class="icones-redes-sociais">

<li>

<a href="https://github.com/joaodasilva" class="github" data-tooltip="Veja meus projetos opensource">

Github

</a>

</li>

<li>

<a href="https://twitter.com/joaodasilva" class="twitter" data-tooltip="Siga-me no Twitter">

Twitter

</a>

</li>

<li>

<a href="https://br.linkedin.com/pub/joão-da-silva/32/4/508" class="linkedin" data-tooltip="Veja meu currículo">

LinkedIn

</a>

</li>

</ul>

Tendo esses atributos no HTML, podemos usá-los no pseudoelemento ::after de cada link. Porém, queremos que esse pseudoelemento apareça apenas quando passarmos o mouse ou o foco do teclado sobre o elemento. Então começamos o código com:

.icones-redes-sociais a:hover::after,

.icones-redes-sociais a:focus::after {

content: attr(data-tooltip);

}

O texto ainda não vai aparecer pois precisamos tirá-lo do fluxo do link que o text-indent impõe. Então definimos o posicionamento do link como relativo, para que ele sirva de referência para o pseudoelemento, e o posicionamento do pseudoelemento como absoluto. E, para que a legenda apareça embaixo do link, definimos que esse pseudoelemento deve estar a 100% do topo e a 0 pixels da direita do link. Já vamos desativar, também, o text-indent negativo do link, colocando 0 para essa propriedade também.

.icones-redes-sociais a {

position: relative;

}

.icones-redes-sociais a:hover::after,

.icones-redes-sociais a:focus::after {

content: attr(data-tooltip);

position: absolute;

top: 100%;

right: 0;

text-indent: 0;

}

Isso vai fazer o texto aparecer abaixo do link, porém esse texto vai aparecer num espaço bem pequeno, já que o link tem uma largura pequena definida. Podemos aumentar a largura da legenda usado a largura do link como base:

.icones-redes-sociais a {

position: relative;

}

.icones-redes-sociais a:hover::after,

.icones-redes-sociais a:focus::after {

content: attr(data-tooltip);

position: absolute;

top: 100%;

right: 0;

text-indent: 0;

width: 300%;

}

Por fim, para melhorar o visual da legenda, diminuimos um pouco o tamanho da fonte e colocamos um fundo preto:

.icones-redes-sociais a {

position: relative;

}

.icones-redes-sociais a:hover::after,

.icones-redes-sociais a:focus::after {

content: attr(data-tooltip);

position: absolute;

top: 100%;

right: 0;

text-indent: 0;

width: 300%;

background: black;

font-size: .75em;

}

Um efeito muito interessante que conseguimos criar apenas com CSS com a ajuda de pseudoelementos é transformar uma caixa num balão de diálogo, como em quadrinhos:



O pseudoelemento ajuda a fazer a ponta que indica quem está falando. Na figura abaixo, é possível ver o pseudoelemento mais claramente.



Adicione uma ponta como essa nas legendas que fizemos agora há pouco usando o pseudoelemento ::before.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

Para criar o elemento que vamos transformar na ponta, precisamos adicionar a declaração do ::before no :hover dos links. Esse pseudoelemento não vai ter texto algum mas, como já vimos, é necessário declarar a propriedade content para que o pseudoelemento seja adicionado à página. Então começamos com

.icones-redes-sociais a:hover::before,

.icones-redes-sociais a:focus::before {

content: '';

}

Queremos que ele seja uma caixinha preta, então adicionamos tamanho e definimos a cor de fundo:

.icones-redes-sociais a:hover::before,

.icones-redes-sociais a:focus::before {

content: '';

width: 1em;

height: 1em;

background-color: black;

}

Precisamos posicioná-la logo acima da legenda, um pouco afastada do canto direito dela. Para isso, usamos o position: absolute e, usando o link como referência, dizemos que queremos ela grudada no canto inferior do link e no canto esquerdo também; isso vai deixá-la suficientemente afastada do canto direito da legenda.

.icones-redes-sociais a:hover::before,

.icones-redes-sociais a:focus::before {

content: '';

width: 1em;

height: 1em;

background-color: black;

position: absolute;

bottom: 0;

left: 0;

}

Por fim, precisamos rotacionar e transladar a caixa que criamos para que ela encaixe como uma ponta na legenda. Para isso, usamos a propriedade transform:

.icones-redes-sociais a:hover::before,

.icones-redes-sociais a:focus::before {

content: '';

width: 1em;

height: 1em;

background-color: black;

position: absolute;

bottom: 0;

left: 0;

transform: translateY(.5em) rotate(45deg);

}

Nesta aula vamos aprender como criar uma página de contato no site do João da Silva, com um ***formulário***.

Na página inicial encontramos um link para a página de contato:



Ao entrarmos na página, vemos que ainda está vazia de conteúdo. Possui apenas a estrutura básica.

O primeiro passo para criarmos um formulário em HTML é usar a *tag* "*form*":

<div class="container">

<form>

...

</form>

</div>

Vamos aos poucos implementando-a. Começando pelos espaços onde o usuário colocará seu nome, seu e-mail:

<form>

<label>Nome:</label>

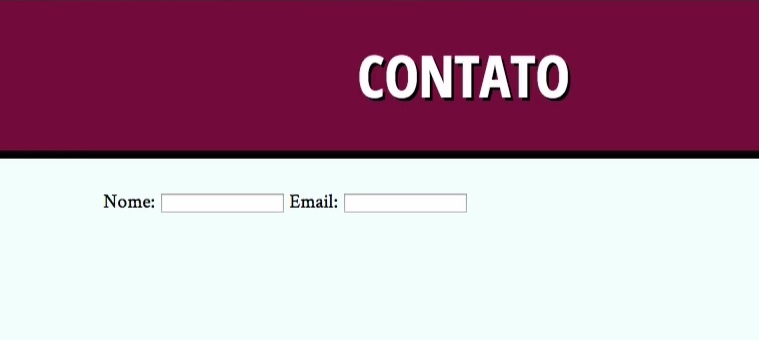
<input>

<label>Email:</label>

<input>

</form>

Até agora nosso formulário está assim:



O usuário também poderá escolher o assunto do qual se trata a mensagem. Isso será feito por meio de botões:

<form>

<label>Nome:</label>

<input>

<label>Email:</label>

<input>

<input type="radio">

<label>Consultoria</label>

</form>

Antes de continuarmos, note que o navegador não saberá a que campo se referem os *inputs*. Para isso a *tag* "*label*" deve possuir o atributo"*for*" e o *input* o atributo "*id*", exatamente para identificar a qual campo ele se refere.

<form>

<label for="nome">Nome:</label>

<input id="nome">

<label for="email">Email:</label>

<input id="email">

<input type="radio" id="consultoria">

<label for="consultoria">Consultoria</label>

</form>

Visualmente nada muda, porém muda-se a semântica e a acessibilidade, o que ajuda o usuário a selecionar os campos.

Se quisermos, também podemos deixar o *input* dentro do *label*:

<label>

<input type"radio">

Consultoria

</label>

Vamos acrescentar os outros assuntos:

<form>

<label for="nome">Nome:</label>

<input id="nome">

<label for="email">Email:</label>

<input id="email">

<label>

<input type="radio">

Consultoria

</label>

<label>

<input type="radio">

Blog

</label>

<label>

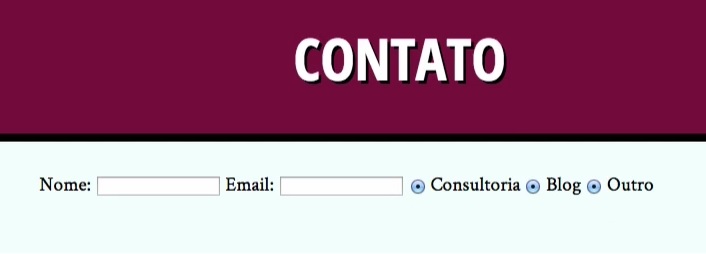
<input type="radio">

Outro

</label>

</form>

Nesse momento nosso formulário está assim:



Perceba que conseguimos selecionar mais de um assunto. Isto acontece porque o navegador não sabe que existe uma relação entre eles. Para arrumar, precisamos adicionar mais um atributo a esses assuntos, que é o "*name*". Ele manda a informação para o servidor. Ou seja, todos os *inputs* precisam dele.

<form>

<label for="nome">Nome:</label>

<input id="nome" name="nome">

<label for="email">Email:</label>

<input id="email" name="e-mail">

<label>

<input type="radio" name="assunto">

Consultoria

</label>

<label>

<input type="radio" name="assunto">

Blog

</label>

<label>

<input type="radio" name="assunto">

Outro

</label>

</form>

Agora apenas um assunto poderá ser selecionado de cada vez.

Os nomes que demos para cada assunto são apenas rótulos que aparecem na página. O navegador ainda não os associa com o valor que eles devem ter e, por sua vez, não os envia para o servidor. Então precisamos do atributo "*value*":

<form>

<label for="nome">Nome:</label>

<input id="nome" name="nome">

<label for="email">Email:</label>

<input id="email" name="e-mail">

<label>

<input type="radio" name="assunto" value="consult">

Consultoria

</label>

<label>

<input type="radio" name="assunto" value="blog">

Blog

</label>

<label>

<input type="radio" name="assunto" value="outro">

Outro

</label>

</form>

Também poderíamos passar os valores para os campos "Nome" e "Email", porém existe um atributo específico para eles, o "*placeholder*":

<form>

<label for="nome">Nome:</label>

<input id="nome" name="nome" placeholder="Seu nome aqui">

<label for="email">Email:</label>

<input id="email" name="e-mail" placeholder="seu@email.com">

<label>

<input type="radio" name="assunto" value="consult">

Consultoria

</label>

<label>

<input type="radio" name="assunto" value="blog">

Blog

</label>

<label>

<input type="radio" name="assunto" value="outro">

Outro

</label>

</form>

O *placeholder* permite que as frases escritas no HTML ("Seu nome aqui", por exemplo) apareçam nos campos do formulário. Quando clicamos elas não impedem a escrita e apagam-se quando começamos a escrever.

http://i.imgur.com/wmlQZIW.jpg

Falta ainda a parte mais importante do formulário: o lugar onde o usuário irá digitar sua mensagem. Usamos a *tag* "*textarea*". Ela pode ser configurada para ter o tamanho que queiramos usando "*cols*" (colunas) e "*rows*" (linhas):

<form>

<label for="nome">Nome:</label>

<input id="nome" name="nome" placeholder="Seu nome aqui">

<label for="email">Email:</label>

<input id="email" name="e-mail" placeholder="seu@email.com">

<label>

<input type="radio" name="assunto" value="consult">

Consultoria

</label>

<label>

<input type="radio" name="assunto" value="blog">

Blog

</label>

<label>

<input type="radio" name="assunto" value="outro">

Outro

</label>

<label for="msg">Mensagem:</label>

<textarea id="msg" name="mensagem" cols="30" rows="6"></textarea>

</form>

Nesse momento o formulário está com essa configuração:



Falta agora uma coisa muito importante: o usuário enviar o formulário. Para isso vamos criar um botão. O jeito tradicional de fazer isso é

<input type="submit" value="Enviar">



A *tag* *input* é limitada para fazermos botões, pois não conseguimos colocar uma imagem ou sublinhar o texto, etc. Então usamos a *tag* "*button*". Com ela podemos formatar o texto do botão.

<button type="submit">Enviar <strong>para mim</strong></button>



Vamos trabalhar um pouco mais em cima da semântica desses elementos que criamos para o fomulário. Para os botões de assuntos, a área do texto da mensagem e o botão de Enviar nós declaramos seus tipos. Façamos o mesmo com o Nome e o E-mail. O nome poderá ser qualquer texto, porém o e-mail tem uma característica diferente da do nome. Para diferenciá-lo, então, vamos acrescentar um *type*:

<label for="email">Email:</label>

<input id="email" name="e-mail" placeholder="seu@email.com" type="email"required>

Visualmente, a tela de formulário não muda. Mas se tentarmos escrever um e-mail falso, sem "@" por exemplo, aparecerá uma mensagem de erro e o formulário não será enviado. Essa validação de formulários é algo novo trazido pelo HTML5. O "*required*" também faz com que o campo seja obrigatório o preenchimento.

Outro atributo interessante para inserirmos no formulário é o "*autofocus*". Ele faz com que toda vez que a página é recarregada o ponteiro de escrita apareça no campo indicado

<label for="nome">Nome:</label>

<input id="nome" name="nome" placeholder="Seu nome aqui" autofocus>

Para melhorar ainda mais a semântica do formulário, vamos agrupar os campos em dois conjuntos: dados pessoais e assunto. Utilizamos para tal as *tags* "*fieldset*" e "*legend*"

<fieldset>

<legend>Dados pessoais</legend>

<label for="nome">Nome:</label>

<input id="nome" name="nome" placeholder="Seu nome aqui" autofocus>

<label for="email">Email:</label>

<input id="email" name="e-mail" placeholder="seu@email.com" type="email"required>

</fieldset>

<fieldset>

<legend>Assunto:</legend>

<label>

<input type="radio" name="assunto" value="consult">

Consultoria

</label>

<label>

<input type="radio" name="assunto" value="blog">

Blog

</label>

<label>

<input type="radio" name="assunto" value="outro">

Outro

</label>

</fieldset>

O formulário fica assim:



Apesar de termos criado ferramentas para validar as informações do formulário, não podemos esquecer de validá-las também no servidor.

O que vimos foi o básico da construção de formulários: validação, usabilidade, semântica. Existem muitas outras ferramentas para deixá-lo mais rico.

Para tornar o campo "Nome" obrigatório e fazer o cursor do teclado já estar nele assim que a página carregar, adicionamos os atributos required e autofocus nele:

<input id="nome" type="text" name="nome-contato" required autofocus>

Para o campo "E-mail", queremos que ele seja obrigatório e tenha uma mensagem explicativa. Para a mensagem explicativa, usamos o atributo placeholder:

<input id="email" type="email" name="email-contato" required placeholder="Digite seu e-mail">

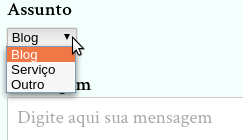
Para adicionar uma mensagem explicativa no campo "Mensagem", usamos novamente o atributo placeholder:

<textarea id="mensagem" name="msg" placeholder="Digite aqui sua mensagem"></ textarea>

Dos tipos com casos de uso mais interessantes, temos:

* **date**, usado para que o usuário possa fornecer uma data como seu aniversário ou data de nascimento.
* **datetime**, para que o usuário possa fornecer, além da data, um horário, útil para que ele forneça um momento para ser lembrado de algo.
* **datetime-local**, para quando a data e o horário que o usuário fornecer sejam independentes de local ou quando o sistema é feito para trabalhar sempre no mesmo fuso horário, como por exemplo quando o usuário precisa fornecer a data e o horário de nascimento dele no horário de Brasília.
* **hidden**, usado para enviar informações extra para o servidor além daquelas fornecidas pelo usuário, como um identificador do usuário gerado pelo sistema.
* **image**, usado para enviar os dados do formulário com um botão gráfico, pode ser usado para não apenas ter um botão mais bonito, como também para fazer uma interação mais dinâmica com o usuário, em que o resultado da ação depende do local onde ele clica na imagem, uma vez que esse tipo de botão envia as coordenadas do clique para o servidor.
* **month**, usado para que o usuário forneça um mês e um ano, útil para que ele forneça a validade de um cartão de crédito, por exemplo.
* **number**, para que o usuário forneça um número como sua idade ou a quantidade de um produto num carrinho de compras.
* **range**, usado para que se possa escolher um valor numérico numa faixa de valores possíveis, normalmente sem muita preocupação com a precisão, como por exemplo quando se deseja que o usuário dê uma nota para um serviço.

Na nossa página, usamos botões do tipo radio para mostrar as opções de assunto da mensagem, mas também poderíamos ter usado um menu de opções como o da imagem abaixo.



Para criar um componente como esse para os assuntos da mensagem, qual código HTML você teria que escrever? [Consulte a documentação da Mozilla sobre o elemento select](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/select)) para ver como ele funciona.

VER OPINIÃO DO INSTRUTOR

### Opinião do Instrutor

O HTML desse componente seria, na sua versão mais simples, algo como:

<select>

<option>Blog</option>

<option>Serviço</option>

<option>Outro</option>

</select>

É possível incrementar esse código adicionando os dados necessários para enviar a opção escolhida para o servidor:

<select name="assunto">

<option value="blog">Blog</option>

<option value="servico">Serviço</option>

<option value="outro">Outro</option>

</select>

Quando usamos um tipo mais específico de campo, estamos agregando **semântica** ao nosso formulário, e isso traz diversas vantagens na prática.

Uma primeira vantagem é a **validação** dos campos. O campo do tipo email garante que o usuário deve digitar um e-mail válido na hora de enviar o formulário. O campo do tipo number garante que o valor digitado será numérico.

Outra grande vantagem é uma **usabilidade** melhor do formulário, principalmente em dispositivos móveis. Quando usamos um tipo date, por exemplo, o navegador pode mostrar um controle para selecionarmos uma data. Ou, quando usamos um tipo number, o dispositivo móvel pode mostrar um teclado numérico, facilitando a digitação.

Além dessas vantagens, temos todas as vantagens de ter uma página mais semântica: mais acessibilidade, melhor indexação pelas ferramentas de busca etc.

Vimos que o HTML5 trouxe validação para formulários sem a necessidade de JavaScript, com novos tipos e o atributo required. Além disso, temos também o atributo pattern.

No atributo pattern, podemos especificar qual padrão deve seguir um determinado campo, para que o usuário preencha-o corretamente. Especificamos esse padrão usando uma linguagem chamada expressão regular. Para garantir que um campo só tenha letras e números, por exemplo, podemos fazer:

<input pattern="[A-Za-z0-9]\*">

Repare que colocamos os conjuntos de caracteres permitidos entre colchetes (letras maiúsculas, minúsculas e dígitos, nessa ordem) e um asterisco depois. Esse asterisco significa que podemos repetir quantas vezes quisermos um caractere desse conjunto. Se quisermos garantir uma quantidade mínima de letras e números, podemos fazer

<input pattern="[A-Za-z0-9]{8,}">

Com esse padrão, agora nosso usuário precisa digitar pelo menos 8 letras ou números para a entrada ser válida.

Com base nisso, altere o campo "Nome" para que o usuário só possa entrar um nome contendo letras (maiúsculas e minúsculas), espaços e apóstrofe, com no mínimo 4 letras. Como resposta, forneça o código do <input> alterado.

Para mais informações sobre expressões regulares veja o [curso desse assunto aqui da Alura](https://cursos.alura.com.br/course/expressoes-regulares).

Fizemos um padrão personalizado para o campo "Nome", garantindo dados mais confiáveis dos nossos visitantes. Porém, quando um dado inválido é inserido nesse campo, a mensagem de erro é muito genérica! Vamos melhorar isso.

Num campo como o "Nome", com o atributo pattern, podemos usar o atributo titlepara descrever esse padrão. Por exemplo:

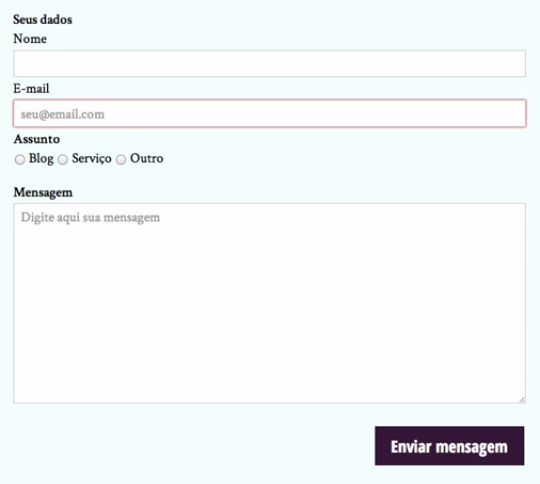
<input pattern="[0-9]\*" title="Preencha somente com números">

Com base nisso, altere o código do campo "Nome" colocando uma mensagem descritiva do formato.

Vamos deixar nosso formulário de contato mais bonito para nossos visitantes. Queremos mudar os seguintes aspectos:

* As legendas dos grupos de campo e do campo "Mensagem" devem estar em negrito e espaçadas do grupo de campos anterior.
* Deixar os rótulos dos campos sempre em cima deles, *exceto* para os assuntos.
* Os campos de texto devem ocupar a largura inteira da tela (*exceto* o campo para o assunto "Outro"), ter uma borda cinza clara, espaçamento interno e usar a mesma fonte do resto da página.
* O campo de texto que tiver o foco do teclado deve ficar com o fundo levemente amarelo.
* Os campos que estiverem inválidos devem ter uma sombra vermelha (*dica*: use a pseudoclasse :invalid).
* O botão para enviar o formulário deve ficar na parte inferior direita da página, após o formulário, e ter o mesmo esquema de cores da barra lateral com a fonte usada nos títulos e levemente maior do que o texto normal. Além disso, o botão não deve ter borda.
* Quando o botão for selecionado pelo teclado ou quando o mouse estiver sobre ele, o fundo deve mudar para a cor #8C1D3D.

O resultado final deve ser parecido com o da imagem abaixo:



Seguindo passo a passo os requisitos do enunciado:

* As legendas dos grupos de campo e do campo "Mensagem" devem estar em negrito e espaçadas do grupo de campos anterior.

/\* Podemos usar um seletor de atributo para selecionar

apenas a legenda desse campo e não dos outros \*/

legend,

label[for="mensagem"] {

font-weight: bold;

margin-top: 1em;

}

* Deixar os rótulos dos campos sempre em cima deles, *exceto* para os assuntos.

/\* As tags <label> que têm o atributo "for" são

os rótulos dos campos sem ser os assuntos \*/

label[for] {

display: block;

}

* Os campos de texto devem ocupar a largura inteira da tela (*exceto* o campo para o assunto "Outro"), ter uma borda cinza clara, espaçamento interno e usar a mesma fonte do resto da página.

/\* Selecionamos apenas os campos de texto \*/

input[type="text"],

input[type="email"],

textarea {

/\* Força o navegador a usar a mesma fonte da página \*/

font-family: inherit;

/\* Inclusive o tamanho dela \*/

font-size: 100%;

padding: .25em .5em;

width: 100%;

box-sizing: border-box;

border: 1px solid #ccc;

}

/\* alguns ajustes para o campo "Outro": deixar ele na mesma linha dos outros assuntos \*/

fieldset > fieldset {

display: inline;

}

/\* deixar o rótulo dele inline de novo \*/

label[for="outro"] {

display: inline;

}

/\* deixar só esse campo sem ocupar a tela toda \*/

input[name="outro-assunto"] {

width: auto;

}

* O campo de texto que tiver o foco do teclado deve ficar com o fundo levemente amarelo.

input:focus,

textarea:focus {

background-color: #FFD;

}

* Os campos que estiverem inválidos devem ter uma sombra vermelha.

input:invalid {

box-shadow: 0 0 3px red;

}

* O botão para enviar o formulário deve ficar na parte inferior direita da página, após o formulário, e ter o mesmo esquema de cores da barra lateral com a fonte usada nos títulos e levemente maior do que o texto normal. Além disso, o botão não deve ter borda.

button {

/\* Precisamos tirar a borda que o navegador coloca por padrão \*/

border: 0;

padding: .5em 1em;

font-family: "Open Sans Condensed", sans-serif;

background-color: #3C1D3D;

color: white;

font-size: 1.2em;

/\* Faz com que o botão apareça na direita \*/

margin-left: auto;

margin-top: 1em;

display: block;

}

* Quando o botão for selecionado pelo teclado ou quando o mouse estiver sobre ele, o fundo deve mudar para a cor #8C1D3D.

button:hover,

button:focus {

background-color: #8C1D3D;

}

Adicionamos um campo de texto ao lado da opção "Outros" para que o usuário especifique o assunto quando selecionar esta opção. Sendo assim, o campo não é útil antes que esta opção seja selecionada.

Tendo isso em mente, podemos esconder o campo por padrão, mostrando-o somente quando a opção "Outros" for selecionada, e podemos fazer isso com o que vimos até agora e a pseudoclasse :checked, que o navegador adiciona a elementos de formulário selecionados.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

O *radio* usado para o assunto "Outro" tem o atributo value com esse texto. Então, usando os seletores de atributo e de irmão, conseguimos selecionar o campo e escondê-lo com a propriedade display:

input[value="Outro"] ~ input {

display: none;

}

Agora, quando esse *radio* for selecionado, a pseudoclasse será adicionada a ele. Podemos usar essa pseudoclasse para selecionar novamente o campo de texto ao lado e mudar o valor da propriedade display:

input[value="Outro"]:checked ~ input {

display: inline;

}

Um problema que apareceu desde o início do curso foi o tamanho da barra lateral, a qual não se ajusta conforme a resolução do monitor.



No CSS3 temos uma ferramenta nova que nos permite fazer algumas contas que resolverão esse problema.

Vamos abrir o CSS e tentar manter a proporção entre as larguras da foto e do texto da barra.

.navegacao-site {

width: 270px;

...

}

Se fizermos isso, a caixa de texto da barra lateral irá se deslocar. Não sabemos o quanto que *270 px* é do total da tela, e para deixar mais genérico, podemos dizer que 270 é 90% de 300px (o tamanho da foto).

No *main* fazemos:

main {

width: calc(100% - 0.9 \* 300px);

...

}

Agora a caixa de texto está de volta no lugar. E lá no ".navegação-site" também podemos generalizar:

.navegacao-site {

width: calc(0.9 \* 300px);

...

}

Podemos refazer, inclusive, o código do posicionamento da *banner* do Twitter com a conta explícita. Se quisermos mudar o tamanho desse *banner*, só será preciso mexer em uma variável da conta.

Essa propriedade de cálculo no CSS também poderia nos ajudar se quisermos colocar, por exemplo, um ícone antes dos *inputs* do formulário. Como é apenas decorativo, vamos inserir a imagem do ícone no CSS por meio de um *pseudoelemento* com a *taglabel*. Na seção de contatos fazemos:

label[for="nome"] {

position: relative;

}

label[for="nome"]:after {

content: "";

background-image: [o destino da imagem aqui]

background-size: 50% 50%;

width: 2em;

height: 2em;

background-position: center

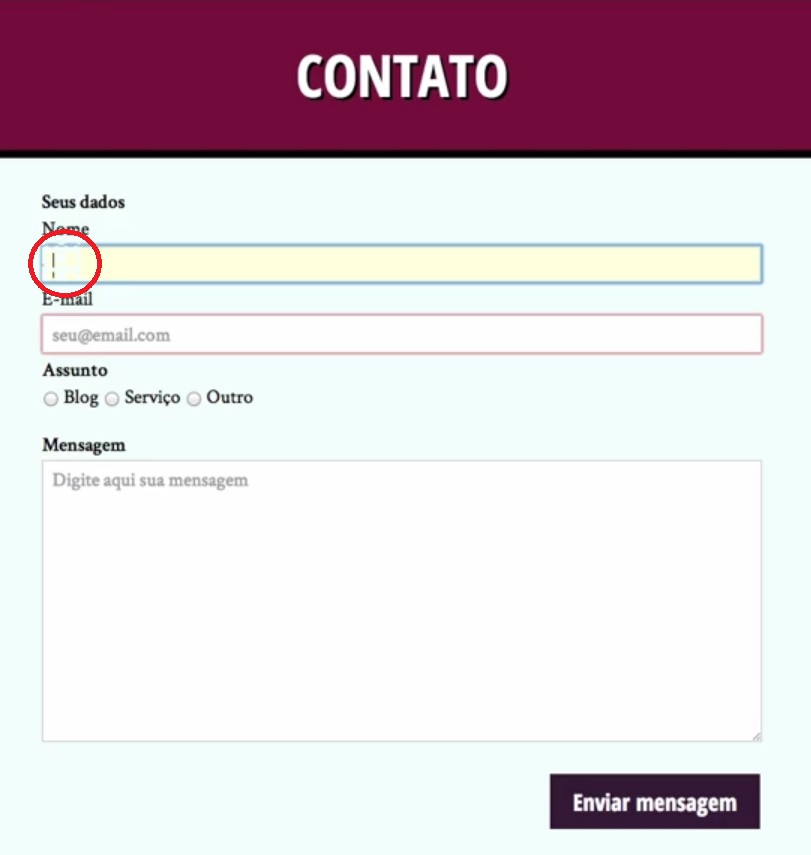
position: absolute;

top: 100%;

left:0;

}

Nós pegamos o pseudoelemento, posicionamos ele relativo à *label* 100% para baixo dela, deixando-o em cima do *input* (também indicamos seu tamanho). Nesse momento o ícone aparece dessa forma:



Perceba que ele realmente está bem posicionado, porém o campo de *input* do nome está logo atrás, vamos tirá-lo de lá e vamos mudar a cor da imagem, que é igual à do *background*. Também devemos dizer que não queremos que ele repita:

label[for="nome"] {

position: relative;

}

label[for="nome"]:after {

content: "";

background-color: #666;

background-repeat: no-repeat;

background-image: [o destino da imagem aqui]

background-size: 50% 50%;

width: 2em;

height: 2em;

background-position: center;

position: absolute;

top: 100%;

left:0;

}

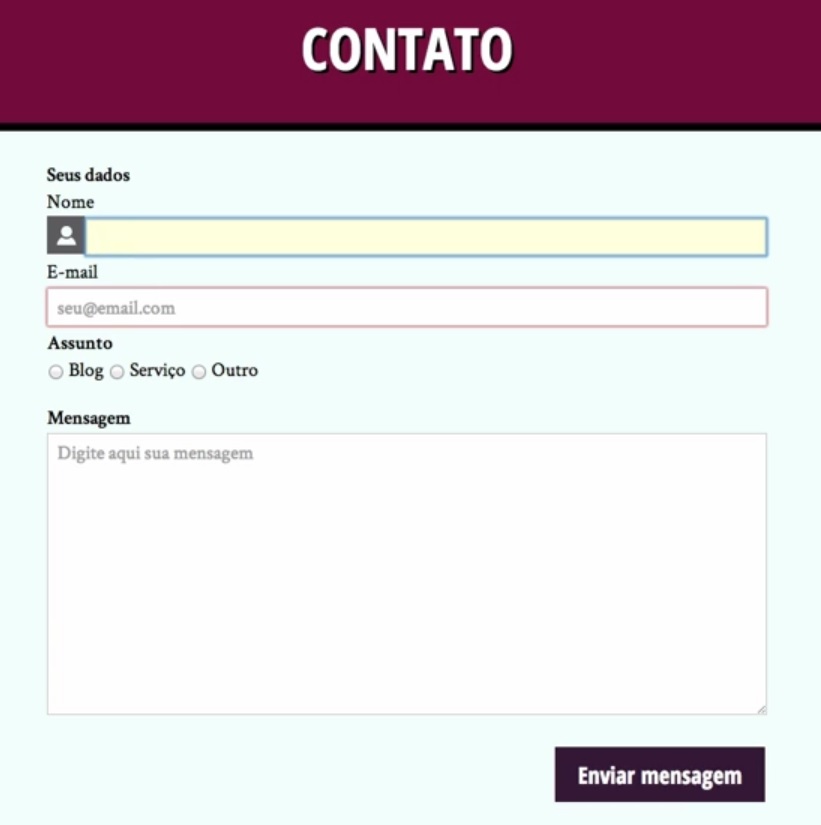
#nome {

width: calc(100% - 2em);

position: relative;

left: 2em;

}



Perceba que se clicarmos no ícone, o foco vai para a caixa de *input*. Isso acontece porque ele é um elemento da *label*.

O cálculo no CSS é muito importante para podermos colocar ícones nos *inputs*, melhorar os estilos dos formulários e, principalmente, para deixarmos as contas mais claras, facilitar refatorações (como veremos mais para frente) e resolver alguns casos complicados como o da barra lateral, a qual precisava ter um tamanho fixo.

Vamos usar a função calc() para deixar o código que fizemos para o banner do Twitter, na página inicial, mais legível.

Colocamos, no CSS, o resultado das contas necessárias para transladar corretamente o banner após a rotação. As contas que fizemos são:

* Eixo X: 0.707 \* (altura - largura) + largura
* Eixo Y: -0.707 \* altura

O número 0.707 ainda vai ter que continuar lá, já que o CSS ainda não suporta contas com senos e cossenos; a altura do elemento nós sabemos que é 3.5em e, a largura, podemos usar 100%.

Troque os números que você colocou direto no seu CSS pela conta usando a função calc().

Traduzindo a conta para o CSS, teremos o seguinte código:

transform: translate(calc(.707 \* (3.5em - 100%) + 100%), calc(-.707 \* 3.5em)) rotate(45deg);

Cuidado com os espaços! Eles são importantes.

A propriedade box-sizing nos permite controlar o modo como o navegador mede a largura de elementos e, com ela, conseguimos dizer para o navegador medir a largura que especificamos com a propriedade width considerando ou não o espaçamento interno e as bordas do elemento.

Um cenário que ela nos permite resolver é criar um elemento que ocupa 50% da tela e tem espaçamento interno de 1em.

div {

box-sizing: border-box;

width: 50%;

padding: 1em;

}

Porém dessa maneira acabamos com o elemento 2em além do esperado. Como poderíamos corrigir a largura?

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

Para conseguir o mesmo resultado usando a função calc(), precisamos observar que o comportamento padrão do navegador é **não contar** o espaçamento interno na hora de definir a largura do elemento. Ou seja, quando fazemos

div {

width: 50%;

padding: 1em;

}

O elemento acaba 2em mais largo do que deveria. Para resolver isso, podemos usar a função calc() para *subtrair* esses 2em da largura definida. Dessa forma, a largura final será exatamente 50%. Ou seja:

div {

width: calc(50% - 2em);

padding: 1em;

}

Um problema atualmente no nosso layout é que estamos usando porcentagens para definir a largura do conteúdo principal e da barra de navegação lateral do site. Com isso, conseguimos flexibilidade, mas temos uma imagem que não é flexível no cabeçalho! Acabamos com o seguinte resultado em telas mais largas, por exemplo:



Queremos, então, que a barra lateral tenha uma largura fixa, baseada na largura da foto (90% da largura dela), e que o conteúdo principal ocupe o resto da página.

Com base nessas observações, altere o código do arquivo site.css para corrigir esse problema.

VER OPINIÃO DO INSTRUTOR

**Opinião do Instrutor**

Para que a barra lateral tenha sempre 90% da largura da foto, que tem 300 pixels de altura, podemos definir a largura dela com calc(), deixando explícita essa relação:

.navegacao-site {

width: calc(0.9 \* 300px);

}

Sendo assim, para que a parte principal ocupe *o resto* da tela, dizemos que queremos a largura total da tela *menos* essa largura que acabamos de definir para a barra lateral. Ou seja:

main {

width: calc(100% - 0.9 \* 300px);

}

A função calc() traz três grandes vantagens para nosso código.

A primeira é a **legibilidade**: podemos deixar algumas contas explícitas, facilitando o entendimento e a manutenção do código.

A segunda é o **poder**: conseguimos criar layouts que não eram possíveis sem JavaScript antes, como elementos que ocupam um espaço que outros deixaram sobrando.

A terceira é a **flexibilidade**: como podemos misturar unidades nas contas, conseguimos deixar nosso layout ainda mais flexível, com um comportamento adequado em tamanhos de tela diferentes e com fontes diferentes.

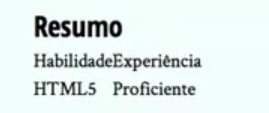
No site do João da Silva, a página de biografia está quase completa. Vamos ver como fazemos para adicionar um resumo das habilidades do dono do site. Uma maneira de apresentar essas informações semanticamente é fazendo uma tabela:

<h2 class="subtitulo-texto">Resumo</h2>  
<table>  
    <tr>  
        <td>Habilidade</td>  
        <td>Experiência</td>  
    </tr>  
</table>

Até agora a tabela possui uma linha (<tr>) com duas colunas (<td>). colocando mais uma linha:

<h2 class="subtitulo-texto">Resumo</h2>  
<table>  
    <tr>  
        <td>Habilidade</td>  
        <td>Experiência</td>  
    </tr>  
    <tr>  
        <td>HTML5</td>  
        <td>Proficiente</td>  
    </tr>  
</table>

Recarregando a página teremos:



Apesar de esteticamente nossa tabela ainda estar ruim, a informação já aparece de uma forma tabular. Perceba que a linha onde estão escritas "Habilidade" e "Experiência" são os cabeçalhos, então vamos destacá-la com thead e th, enquanto que o resto da tabela é o corpo dela, vamos destacá-lo com "*tbody*":

<h2 class="subtitulo-texto">Resumo</h2>  
<table>  
    <thead>  
        <tr>  
            <th>Habilidade</th>  
            <th>Experiência</th>  
        </tr>  
    </thead>  
    <tbody>  
        <tr>  
            <th>HTML5</th>  
            <td>Proficiente</td>  
        </tr>  
    </tbody>  
</table>

Vamos começar a estilizar a tabela, como colocar cores de fundo. Lembrando que agora vamos inserir elementos decorativos, logo as mudanças ocorrerão no CSS.

th {  
    background-color: red;  
    border: 5px solid black;  
}  
  
td {  
    background-color: blue;  
    border: 5px solid black;  
}

Nossa tabela está assim:



Perceba que as bordas pretas elas sobrepõem-se onde as células se encontram. Podemos separá-las com

table {  
    border-collapse: separate;  
}

Um outro efeito é espaçá-las:

table {  
    border-collapse: separate;  
    border-spacing: 5px;  
}



Conseguimos trabalhar com muitas propriedades da tabela, além de outras do CSS, como por exemplo as pseudoclasses, criando tabelas zebradas.

Antigamente se usava muito as tabelas no HTML para posicionar elementos, às vezes até a página inteira era construída dentro de uma tabela. Isso acontecia pois o suporte de CSS não era bom. Tal uso das tabelas para *layouts* prejudica muito a semântica e a manutenção. Hoje, com um bom suporte, é necessário fugirmos desses truques.

Conseguimos criar tabelas com *layouts* mais complicados usando as propriedades rowspan e colspan. Elas nos permitem dizer que uma célula deve ocupar mais de uma linha ou mais de uma coluna, respectivamente. Por exemplo, com o código

<table>  
    <tr>  
        <td colspan="2">Grande</td>  
    </tr>  
    <tr>  
        <td>Pequena</td>  
        <td>Pequena</td>  
    </tr>  
</table>

temos o seguinte resultado visual:

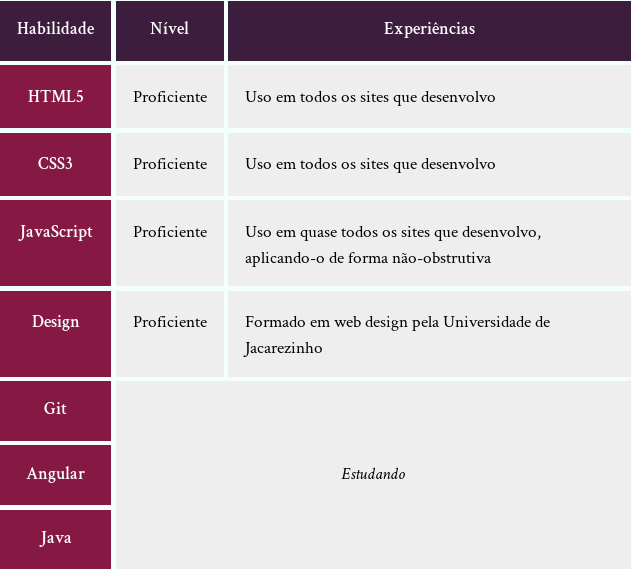
Primeira linha da tabela apenas com o texto 'Grande' ocupando duas colunas e segunda linha com duas colunas com o mesmo texto: 'Pequena'

Quando usamos a propriedade rowspan, não precisamos colocar um <td> para aquela coluna na linha seguinte. Por exemplo:

<table>  
    <tr>  
        <td rowspan="2">Coluna da esquerda</td>  
        <td>Coluna da direita</td>  
    </tr>  
    <tr>  
        <td>Coluna da direita</td>  
    </tr>  
</table>

Coluna da esquerda ocupando duas linhas e duas células na coluna da direita

Sabendo disso, adicione mais três linhas à tabela de habilidades do João da Silva com as habilidades Git, Angular e Java. Nas outras duas colunas dessas três linhas, deve aparecer o texto "Estudando" centralizado e itálico, como na imagem abaixo.



Para centralizar *verticalmente* o texto dentro da célula, podemos usar a propriedade vertical-align. Dentro de células de tabelas, a propriedade vertical-align ganha um significado ligeiramente diferente do que o padrão, alinhando os textos de uma linha inteira da tabela.

Na resposta, forneça o código atualizado da tabela e o código CSS que você acrescentou ao arquivo bio.css.

Para adicionar essas três novas linhas, precisaremos de mais três elementos <tr>. Porém, apenas no primeiro vai aparecer o <td> contendo o texto "Estudando", já que esse elemento deve ocupar as três linhas e, usando a propriedade rowspan, não precisamos repeti-lo nas demais linhas.

Além disso, como ele deve ocupar as duas colunas além do nome da habilidade, vamos ter só um <td> nessa primeira linha nova.

Nosso HTML fica assim, então:

<table class="habilidades">  
    <thead>  
        <tr>  
            <th>Habilidade</th>  
            <th>Nível</th>  
            <th>Experiências</th>  
        </tr>  
    </thead>  
    <tbody>  
        <tr>  
            <th>HTML5</th>  
            <td>Proficiente</td>  
            <td>Uso em todos os sites que desenvolvo</td>  
        </tr>  
        <tr>  
            <th>CSS3</th>  
            <td>Proficiente</td>  
            <td>Uso em todos os sites que desenvolvo</td>  
        </tr>  
        <tr>  
            <th>JavaScript</th>  
            <td>Proficiente</td>  
            <td>Uso em quase todos os sites que desenvolvo, aplicando-o de forma não-obstrutiva</td>  
        </tr>  
        <tr>  
            <th>Design</th>  
            <td>Proficiente</td>  
            <td>Formado em web design pela Universidade de Jacarezinho</td>  
        </tr>  
        <tr>  
            <th>Git</th>  
            <td colspan="2" rowspan="3">Estudando</td>  
        </tr>  
        <tr>  
            <th>Angular</th>  
        </tr>  
        <tr>  
            <th>Java</th>  
        </tr>  
    </tbody>  
</table>

Para conseguir centralizar o texto dentro dessa célula grande, podemos usar as propriedades text-align (centralização horizontal) e vertical-align (centralização vertical).

Agora, para selecionar **apenas essa célula**, podemos adicionar uma classe a ela ou recorrer novamente a seletores de atributo. Neste caso, eles nos darão um resultado bem interessante, permitindo a estilização baseada no tipo de célula que temos.

Quando uma célula ocupa várias colunas (tem a propriedade colspan declarada), podemos centralizar o texto horizontalmente (e deixá-lo em itálico):

.habilidades [colspan] {  
    text-align: center;  
    font-style: italic;  
}

E, quando temos uma célula ocupando várias linhas, podemos centralizar o texto verticalmente:

.habilidades [rowspan] {  
    vertical-align: middle;  
}

No nosso caso, essas regras serão aplicadas em conjunto, mas elas podem ser reaproveitadas separadamente depois!

Se quisermos incluir numa página diversas informações sobre um usuário (nome, idade, email, foto, etc), qual das tags abaixo é mais adequada para representar essas informações semanticamente?

Neste caso, o elemento <dl> é o mais adequado. Estamos descrevendo **um único item**, detalhando características dele.

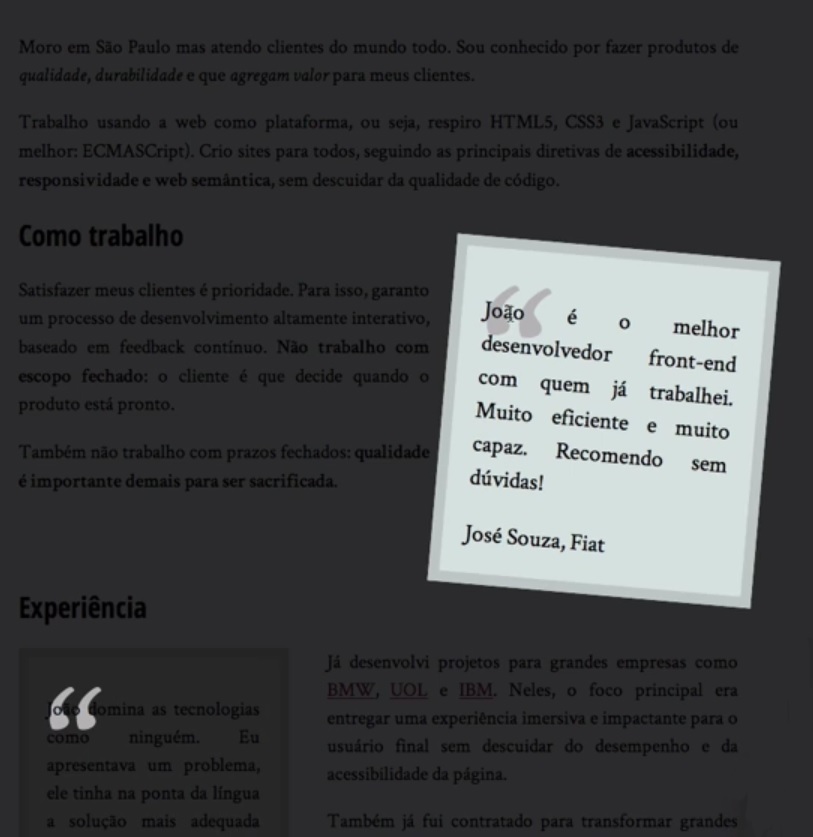
As tags <ul> e <ol> se encaixariam melhor se estivéssemos listando **uma única característica de vários itens** (uma lista de nomes, por exemplo).

Uma tabela seria adequada caso estivéssemos representando dados de vários produtos. Quando essa tabela é dinâmica, podendo ter um ou mais itens de acordo com o sistema, é justo que ela tenha apenas um item sem perder a semântica.

Novamente, as tags <div> e <section> são genéricas demais para esse caso, mas nada impede que o elemento <dl> seja colocado dentro de um desses dois.

Vimos diversas propriedade do CSS3 que permitem deixar a nossa página com uma cara mais moderna. Hoje veremos mais algumas que deixarão nosso site mais interativo.

Quando passamos o *mouse* por cima das citações na página de biografia do João da Silva, fizemos um efeito de destaque:



Mas tal efeito é muito brusco, não existe uma transição da página normal para o efeito de destaque. Não é bom para a experiência do usuário, de um modo geral.

Antigamente, para conseguirmos um efeito de transição, tínhamos que nos utilizar do *javascript*, hoje não mais. Um recurso do CSS nos permite ter esse controle. A propriedade que usamos é ***transition***. Vamos aplicá-la na citação:

.citacao-bio {  
    ...  
    transition: 1s;  
}

Isso significa que a transição de antes para depois de passar o *mouse* durará 1 segundo. O inverso também ocorre nesse tempo.

Queremos um controle maior sobre essa transição: que a sombra de fundo apareça em 2 segundos, mas a rotação ocorra no tempo estipulado anteriormente, ou seja, a transição deverá ser aplicada nas propriedades *transform* e *box-shadow* separadamente:

.citacao-bio {  
    ...  
    transition: transform 1s, box-shadow 2s;  
}

Também podemos controlar o início das transições. Elas podem começar com um atraso. Para isso usamos a propriedade transition-delay:

.citacao-bio {  
    ...  
    transition: transform 1s, box-shadow 2s;  
    transition-delay: 0s, 1s;  
}

O *transform* começará instantaneamente e o *box-shadow* 1 segundos após passarmos o *mouse* por cima do elemento.

Perceba que essas transições possuem uma aceleração. Sua velocidade varia conforme elas vão ocorrendo. Existe uma suavização no final de cada uma. Podemos mudar isso:

.citacao-bio {  
    ...  
    transition: transform 1s linear, box-shadow 2s ease-in;  
    transition-delay: 0s, 1s;  
}

Aqui a velocidade é constante (*linear*) para a transformação e a sombra tem uma aceleração no começo (*ease-in*). Podemos, então mudar a aceleração com o *ease-in, o* ease-out *(fim) e o* ease-in-out\* (começo e fim).

Como vimos, é necessária uma interação do usuário com o elemento para que aconteça a animação. Mas e se quisermos fazer alguma que seja contínua ou que não dependesse do usuário, mas que, por exemplo, aconteça quando carregamos a página? A propriedade *transition* não suporta isso, teremos que usar a ***animation***.

Vamos colocar uma animação de 2 segundos no título principal da página:

h1 {  
    animation: aparece 2s;  
}

Agora vamos especificar essa animação utilizando o símbolo "@", que nos permite defini-la:

h1 {  
    animation: aparece 2s;  
}  
  
@keyframes aparece {      
    0% {  
        opacity: 0;  
    }  
    100% {  
        opacity: 1;  
    }  
}

O "*@keyframes*" nos permite especificar o que acontece a cada *frame* da animação. Nesse caso apenas queremos dizer o que acontecerá no primeiro *frame* (0%) e no último *frame* (100%) dela.

Se estivermos usando o *Google Chrome*, tal propriedade ainda não será suportada, ela está em *BETA*. Precisamos daquele prefixo "*-webkit-*" visto em aulas anteriores:

h1 {  
    -webkit-animation: aparece 2s;  
    animation: aparece 2s;  
}  
  
@-webkit-keyframes aparece {      
    0% {  
        opacity: 0;  
    }  
    100% {  
        opacity: 1;  
    }  
}  
  
@keyframes aparece {      
    0% {  
        opacity: 0;  
    }  
    100% {  
        opacity: 1;  
    }  
}

A propriedade de animação também permite modificar a aceleração, utilizando *ease-in*, *ease-out* e *ease-in-out*.

A animação que criamos ocorre uma vez, apenas quando carregamos a página. E para deixar uma animação contínua, com repetição? Vejamos como implementar isso no *block-quote*:

.citacao-bio {  
    ...  
    position: relative;  
    -webkit-animation: quica 1s infinite;  
    animation: quica 1s infinite;  
}

Depois de definirmos sua posição, a animação dará "pulinhos" de 1 segundo infinitamente. Vamos especificá-la (sempre lembrando do prefixo, pois a propriedade está em *BETA*):

.citacao-bio {  
    ...  
    position: relative;  
    -webkit-animation: quica 1s infinite;  
    animation: quica 1s infinite;  
}  
  
@-webkit-keyframes quica {  
    from {  
        top: 0;  
    }  
    to {  
        top: 1 em;  
    }  
}  
  
@keyframes quica {  
    from {  
        top: 0;  
    }  
    to {  
        top: 1 em;  
    }  
}

Se carregarmos a página veremos que a animação funciona, porém ela sempre volta para o início bruscamente. Conseguimos fazer com que ela se alterne da maneira início-fim e fim-início com a propriedade "*animation-direction*":

.citacao-bio {  
    ...  
    position: relative;  
    -webkit-animation: quica 1s infinite;  
    animation: quica 1s infinite;  
    -webkit-animation-direction: alternate;  
    animation-direction: alternate;  
}

Precisamos de outra propriedade para modificar a aceleração dessa alternância, a "*animation-timing-function*":

.citacao-bio {  
    ...  
    position: relative;  
    -webkit-animation: quica 1s infinite;  
    animation: quica 1s infinite;  
    -webkit-animation-direction: alternate;  
    animation-direction: alternate;  
    -webkit-animation-timing-function: ease-in;  
    animation-timing-function: ease-in;  
}

A animação não impede que a transição criada anteriormente ocorra. Vamos ver como fazê-la pausar quando passamos o *mouse* em cima do *block-quote*:

.citacao-bio:hover,  
.citacao-bio:focus {  
    box-shadow: 0 0 0 99999px rgba(0,0,0,.8);  
    -webkit-animation-play-state: paused;  
    animation-play-state: paused;  
}

Vamos deixar a animação ainda mais rebuscada. Além de dar os "pulos", o objeto irá mexer um pouco:

@-webkit-keyframes quica {  
    from {  
        top: 0;  
    }  
    90% {  
        transform: rotate(-2deg);  
    }  
    95% {  
        transform: rotate(2deg);  
    }  
    to {  
        top: 1 em;  
    }  
}

Com isso conseguimos fazer páginas bem dinâmicas e divertidas. Explore um pouco mais as transições e animações!

O banner do Twitter fica aparecendo antes da animação começar. Isso indica que o estilo inicial da animação não é aplicado ao elemento antes que ela comece por padrão. Porém, podemos mudar esse comportamento.

A especificação de animações define uma propriedade chamada animation-fill-mode. Essa propriedade nos permite fazer com que o navegador aplique o estilo inicial de uma animação antes mesmo de ela começar (valor backwards), mantenha o estilo final de uma animação mesmo após ela terminar de executar (valor forwards) ou ainda faça as duas coisas (valor both).

Adicione, então, a declaração adequada ao estilo do banner do Twitter.

Como queremos que o estilo inicial da animação (elemento transparente e pequeno) seja aplicado antes mesmo de a animação começar, precisamos adicionar a declaração:

animation-fill-mode: backwards;

Ao elemento. Podemos também usar o valor both sem problemas.

Para incrementar um pouco mais o efeito que acabamos de fazer, vamos dar a possibilidade de o usuário parar a flutuação quando colocar o mouse em cima **do parágrafo**.

**VER OPINIÃO DO INSTRUTOR**

**Opinião do Instrutor**

Para parar uma animação, podemos usar a propriedade animation-play-state, passando para ela o valor paused. Queremos fazer isso quando o mouse é posicionado sobre o **parágrafo**. Então fazemos:

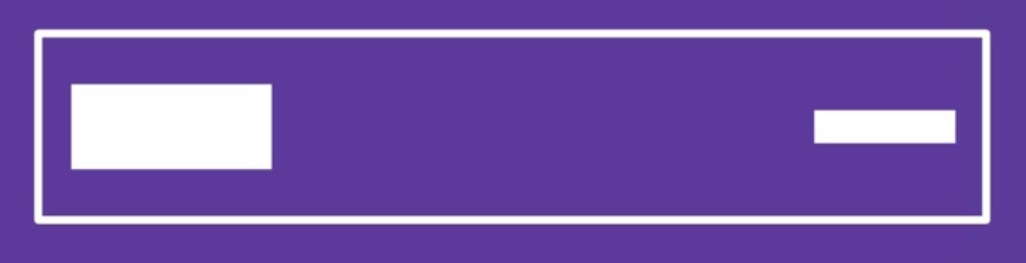
.saudacao:hover em {  
    animation-play-state: paused;  
}

Ao longo desse curso aprofundamos nossos conhecimentos em CSS e HTML. Vimos como o CSS é poderoso para trabalharmos com *layouts* dinâmicos e interativos e como o HTML pode ser muito mais semântico. Mesmo assim, ainda existe alguns problemas difíceis de serem solucionados.

O maior desses problemas ainda é o de *layout*. Nós utilizamos de três propriedades para criá-lo:

* *position*
* *display*
* *float*

Mesmo com elas ainda sofremos bastante. Queremos, por exemplo, criar algo desse tipo:



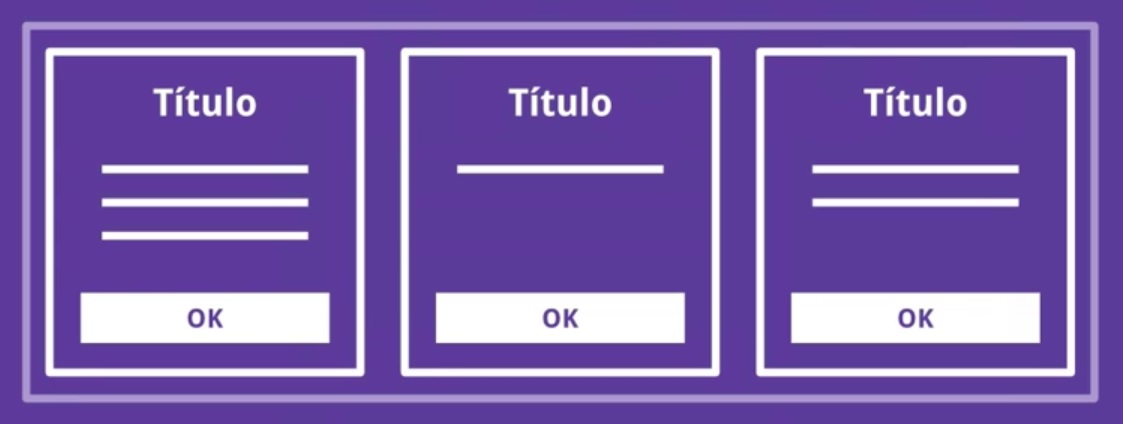
Uma caixa com um elemento ocupando toda sua altura à esquerda e à direita um outro ocupando metade da altura. Ambos devem estar centralizados verticalmente.

Outro exemplo:



Uma caixa com um botão sempre na parte de baixo.

Ou ainda:



Várias caixas de mesma altura lado a lado dentro de um mesmo *container*, independente do conteúdo que elas possuem.

Esses elementos são muito complicados de serem feitos no CSS, pois as regras que vimos até agora servem para controlar os elementos filhos em relação ao pai. Nenhuma delas nos permite controlá-los a partir do pai.

Porém, recentemente, surgiu uma nova especificação para CSS chamada ***flexbox***, a qual permite termos esse controle a partir de um elemento pai.

Pegando nosso primeiro exemplo, podemos resolvê-lo usando três linhas no pai:

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

* "display: flex": faz com que o *container* se comporte como um *flexbox*, ou seja, ele controlará o *layout* de seus filhos. Por padrão, os filhos terão a mesma altura e alinhados à esquerda;
* "align-items": center": centraliza verticalmente os elementos filhos dentro do pai;
* "justify-content: space-between": distribui entre os elementos o espaço que está sobrando.

Perceba que em nenhum momento passamos a informação no código que se tratavam de dois elementos, ou que o espaço entre eles valia alguma porcentagem do total. Isso acontece porque o *flexbox* automaticamente sabe quantos filhos ele terá. É tão dinâmico o *layout* que se colocássemos um terceiro elemento, este já fica alinhado e justificado de acordo:



No outro caso, conseguimos resolver o problema das três caixas fazendo

.container {  
    display: flex;  
}

Isto serve para que as caixas tenham a mesma altura, já que é padrão do *flexbox* fazê-lo. Agora, para manter o conteúdo das caixas justificado verticalmente basta repetir a ideia para elas só que utilizando a propriedade "*flex-direction*":

.box {  
    display: flex;  
    flex-direction:column;  
}

Porém, as caixas com mais texto ficarão com largura maior, então devemos defini-la usando a noção de proporção. Como as queremos do mesmo tamanho, essa proporção é igual a 1:

.box {  
    display: flex;  
    flex-direction:column;  
    flex: 1;  
}

Se quisermos que uma das caixas tenha o dobro de tamanho das duas, por exemplo, a segunda, basta fazermos:

.box:nth-child(2) {  
    flex: 2;  
}



Perceba que se adicionarmos uma quarta caixa, o *flexbox* automaticamente redimensiona todas. O nosso *layout* é muito flexível.

Um outro recurso dessa propriedade é pegar esse elemento que deixamos maior e deixá-lo na frente, sem mexer no HTML:

.box:nth-child(2) {  
    flex: 2;  
    order: 1;  
}

Isso é muito importante do ponto de vista de acessibilidade e de semântica.

Falta agora deixarmos os botões dentro das caixas sempre em baixo:

.box p {  
    flex: 1;  
}

Porque deixar os botões em baixo nada mais é do que deixar os parágrafos do mesmo tamanho.

Vamos rever alguns recursos do *flexbox*:

* *display: flex*: ativa o *flexbox* no elemento;
* *align-items*: distribui verticalmente os elementos dentro de um *container flex*;
* *justify-content*: distribuir os elementos espaçadamente dentro do *container*;
* *flex-direction*: permite inverter o *align-items*;
* *order*: muda a ordem dos elementos;
* *flex*: indica a proporção do tamanho dos elementos. Ele é um atalho para mais três propriedades:
  + *flex-grow*: determina quanto o elemento deve crescer;
  + *flex-shrink*: determina quanto o elemento deve diminuir;
  + *flex-basis*: determina o tamanho mínimo do elemento.

O *flexbox* é uma dentre várias outras especificações que estão surgindo no CSS para facilitar nosso trabalho com *layout* e deixar o código mais simples.

Para que serve a propriedade flex?

Controlar o tamanho de elementos internos a um container flex.

A propriedade flex aceita até 3 valores, que correspondem a 3 propriedades da especificação Flexbox. Quais propriedades são essas?

flex-grow, flex-shrink e flex-basis

A propriedade flex é um atalho para as propriedades flex-grow, flex-shrink e flex-basis, nessa ordem. Ou seja, a declaração:

flex: 3 1 200px;

É equivalente a:

flex-grow: 3;  
flex-shrink: 1;  
flex-basis: 200px;

**Apague** o código para posicionar os itens com display: inline-block e, no lugar, use a especificação Flexbox para ajustá-los.

**Não esqueça** que, por padrão, os itens de um container flex ficam todos na mesma linha; use a propriedade flex-wrap para mudar esse comportamento. Use, também, a propriedade flex para regular o tamanho dos elementos.

A primeira coisa que queremos fazer é ativar o modelo flex na lista, com suporte a quebra de linha. Para isso, precisamos das seguintes declarações:

.portfolio {  
    display: flex;  
    flex-wrap: wrap;  
}

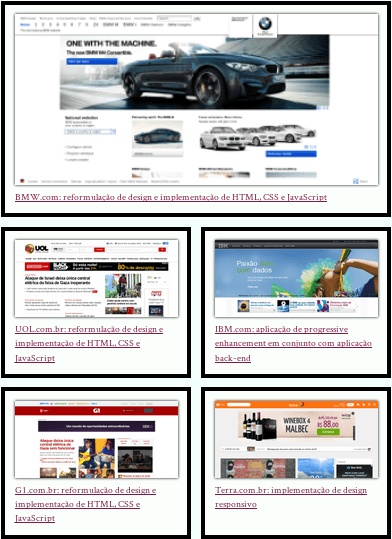
Agora, para regular o tamanho dos elementos, podemos usar a propriedade flex especificando que o elemento deve crescer e diminuir na mesma proporção que os outros, mas com um mínimo de 45% de largura. Então temos:

.portfolio li {  
    /\* não precisamos mais do display: inline-block aqui \*/  
    flex: 1 1 45%;  
}

Por fim, para fazer o destaque, basta mudar o flex-basis (último valor da propriedade flex) do primeiro item da lista:

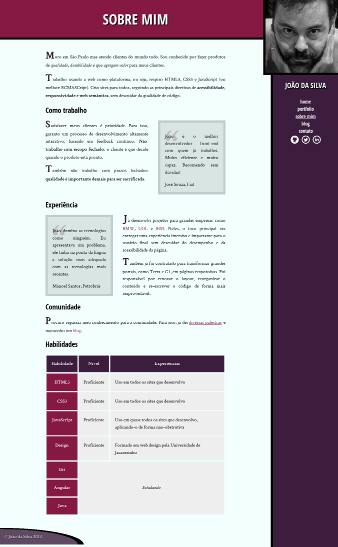
.portfolio li:first-child {  
    flex-basis: 100%;  
}

Devemos chegar ao mesmo resultado de antes, porém agora com os itens alinhados.



Podemos usar a especificação Flexbox para facilitar o código que posiciona os links de navegação na lateral das páginas internas do site. Conseguimos usá-la para retirar os *floats* e diminuir a quantidade de cálculos que precisamos fazer no CSS.

De quebra, conseguimos fazer com que a barra de navegação ocupe a página inteira, como mostra a figura abaixo.



Implemente essa refatoração para conseguir esse resultado.

**Dica**: para evitar que sobre espaço no final da página, troque o position: relative e o top: 300pxda barra lateral por um padding-top: 300px (adicione 1em a esse valor para conseguir exatamente o mesmo resultado). E, para evitar que a página inicial fique quebrada, desative o Flexbox apenas nela usando display: block no body apenas no index.css.

**VER OPINIÃO DO INSTRUTOR**

**Opinião do Instrutor**

O primeiro passo é transformar o <body> num container flex com suporte a quebras de linha, para que o cabeçalho fique numa linha diferente dos elementos <main> e <aside>. Para isso, adicionamos o seguinte código ao arquivo site.css:

body {  
    display: flex;  
    flex-wrap: wrap;  
}

Isso vai quebrar a página inicial, mas a dica resolve nosso problema: basta adicionar a seguinte declaração ao index.css:

body {  
    /\* sobrescreve o site.css \*/  
    display: block;  
}

Com isso, agora podemos remover as declarações de float no <main> e no <aside>. Também podemos remover a declaração de largura no <main>. No lugar, queremos apenas garantir que esse elemento ocupe todo o espaço que a barra lateral não ocupa. Para isso, usamos a propriedade flex-grow:

main {  
    flex-grow: 1;  
}

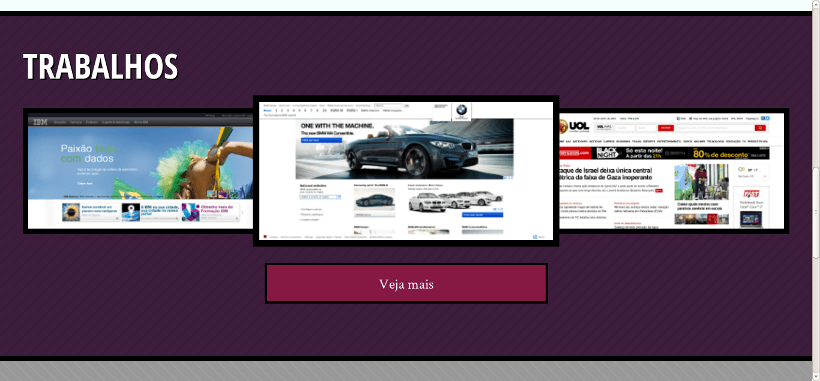
A barra lateral já vai automaticamente crescer para ocupar toda a altura da página. Podemos apenas fazer alguns ajustes para melhorar o resultado. Removemos a borda inferior dela, **deixando apenas na foto**, e usamos a dica para remover espaço sobrando no final da página:

.navegacao-site {  
    /\* removemos position e top, além do float, já removido acima \*/  
    padding-top: calc(300px + 1em);  
}  
  
  
.minha-foto,  
.navegacao-site {  
    /\* removemos a borda aqui \*/  
}  
  
.minha-foto {  
    border-bottom: .5em solid black;  
}

Vimos que a propriedade order permite mudar a ordem em que elementos dentro de um container flex são exibidos. Podemos usá-la, então, para dar mais destaque ao primeiro item da lista de trabalhos na página inicial.

Queremos que ele continue em primeiro na lista pois, pela semântica, ele é o mais importante. Porém, visualmente, colocá-lo no meio da página (e aumentá-lo ligeiramente) dará mais destaque a ele.

Use a propriedade order e a transformação de escala para criar esse efeito, chegando num resultado como o da figura abaixo.



**VER OPINIÃO DO INSTRUTOR**

**Opinião do Instrutor**

Para conseguir mudar a ordem dos elementos, precisamos definir a ordem nos três. O primeiro item deve aparecer no meio da lista, portanto o valor da propriedade order para ele será 2. Além disso, ele deve aparecer ligeiramente maior que os outros, então usamos a propriedade transform com scale:

.trabalhos li:first-child {  
    order: 2;  
    transform: scale(1.2);  
}

Nos outros dois elementos, precisamos apenas definir a propriedade order. O segundo item da lista deve aparecer em primeiro. Portanto, temos que fazer:

.trabalhos li:nth-child(2) {  
    order: 1;  
}

E, por fim, o último elemento deve aparecer depois de todos. Então precisamos definir seu ordercomo 3:

.trabalhos li:last-child {  
    order: 3;  
}

Por fim, para conseguir as bordas ao redor da imagem central, podemos tirar a borda e o *overflow*da lista e colocar apenas a borda em cada um de seus itens:

.trabalhos ul {  
    /\* removemos a borda e o overflow daqui \*/  
}  
  
.trabalhos li {  
    border: .5em solid #000;  
}