



DIO
CEZAR
GO

PRÉ-COMPILADORES CSS

Escreve menos e faça mais com
SASS



APRESENTAÇÃO

quem sou eu?

APRESENTAÇÃO

- olá, sou o **Diogo Cezar!**
 - mestre em IA pela *UFPR* (2012);
 - tecnologia em Informática *UTFPR* (2007);
 - Full Stack Web Developer;
 - professor;

FUI PARAR NO BBB!

- mas... antes dos autógrafos!
- Hackaton Globo 2016
 - O que é?
 - Como funciona?
 - Como foi a experiência?



COISAS IMPORTANTES

- <http://www.diogocezar.com> - este é meu site;
- <http://www.creativetroops.com.br> - hub de profissionais de tecnologia;
- <http://github.com/diogocezar> - este é meu github;
- diogo@diogocezar.com - este é o meu e-mail;
- diogoc@utfpr.edu.br - este também (usem esse);
- sala: **K001**;

UM POUCO DO QUE JÁ FIZ





I'M BACK!

- estive na UTFPR em **2008** e **2009**;
- dei aulas na pós-graduação em **2013**;
- **2017** de volta a UTFPR;

MATERIAIS UTFWARE

<https://github.com/diogocezar/dctb-utfware>

DICAS GERAIS

o que eu já aprendi com a vida!

DICA 01

Se você é programador, deve passar boa parte do seu tempo atrás de uma tela de terminal!

Sério! aprendam e usem o terminal.

DICA 02

Só programar não é suficiente :(você precisa aprender a criar e configurar os seus ambientes de desenvolvimento e produção.

DICA 03

Aprendam Linux/Unix!

A maioria dos ambientes (reais) de desenvolvimento e produção estão abaixo de um ecossistema Linux/Unix.

Você irá precisar disso, mais cedo ou mais tarde.

Windows? Máquina Virtual ou Bash.

DICA 04

**A melhor forma de aprender (com pessoas) é fazendo
amizades**

Arquivo confidencial: aprendi pela dor!

Você confia, ajuda e compartilha mais com seus amigos.

DICA 05

Hierarquia? Deixe para quando você for chefe.

Estou aqui para aprender (junto) com vocês!

Todos estamos aqui para aprender.

DICA 06

Se eu não souber, vou procurar.

Vocês, façam o mesmo!

DICA 07

A melhor skill que um programador pode ter não é saber de tudo, é saber se virar e fazer acontecer!

Atualmente, as coisas evoluem MUITO rápido! Acreditem, por experiência própria, vocês não vão conseguir dominar tudo!

Mas alivia saber que, se você precisar, vai saber como encontrar e como resolver.

Treine essa habilidade! Vai ser a sua principal durante sua vida de desenvolvedor;

DICA 08

As pessoas... bem elas são complicadas!

Tenha calma, trabalhe o seu psicológico para compreender que as pessoas são diferentes, nem sempre estão em um bom dia.

DICA 09

Aprenda um pouco sobre design!

Isso fez muita diferença durante todo o tempo que trabalhei.

Programadores nerds são ótimos, mas programadores que manjam de design conseguem entregar algo bonito, que é (as vezes) mais importante.

DICA 10

Seja responsável!

Nossa área está cheia de *sobrinhos*!

Clientes e empresas acabam sempre se decepcionando com entregas fora de prazo ou com escopo diferente do combinado, seja **responsável** e cumpra com suas obrigações.

Assim será reconhecido rapidamente.

DICA 11

Arrisque!

Nunca mais pense: será que consigo fazer isso?

Yes, you can!

É importante aprender a aprender, hoje, tem tudo no Google, ou melhor, no stackoverflow!

DICA 12

Faça contatos.

Hey nerdão! Saia de casa!

Conheça pessoas, mostre seus trabalhos, faça amizades indique parceiros, aumente a sua rede de contatos, e sempre terá oportunidades de trabalho.

DICA 13

Crie o seu GitHub hoje!

Empresas estão sempre de olho no que vocês andam “codando”.

Mas ninguém vai abrir o seu Moodle para ver se vc entrou a atividade ou não!
Exponha todos seus estudos! Crie repositório com links, tutoriais e coisas do tipo.

Sabe aquele problema que você demorou um monte para resolver? Faça um repositório com esse tutorial. Além de ajudar outras pessoas, pode ajudar você mesmo no futuro!

DICA 14

Contribua com a comunidade!

Trabalhar em projetos *OpenSource* é uma ótima maneira de mostrar o que você sabe.

Contribua sempre que possível com qualquer tipo de projeto que achar que pode melhorar.

Seja traduzindo, seja refatorando, sei lá... melhore as coisas!

DICA 15

A melhor linguagem é o Inglês!

Nunca mais escreva uma variável em HUE, digo.. PT-BR.

Sério, isso é importante. Quando tiver que mostrar seu código, pode ser para uma empresa de fora... Sempre faça tudo em Inglês!

Aprenda a ler e escrever em inglês, falar também é bom!

A maioria do conteúdo relevante na internet está em Inglês.

DICA 16

Acompanhe as tendências!

Você não vai conseguir aprender a programar em todas as novas linguagens que saírem.

Entenda como elas funcionam, se prepare para estar pronto caso alguém pergunte você conhece “Cobol” (zuera) NodeJs, GoLang, Angular, Vue, React, Kotlin... essas coisas...

DICA 17

Estude, sempre que possível

Está na faculdade, beleza, é hora de se concentrar no estudo. Terminou e acabou? Se engana meu jovem gafanhoto...

Você terá que estudar todos os dias se quiser acompanhar a próxima geração!
Eles estão cada vez mais fortes e rápidos, mal posso ver seus movimentos!

DICA 18

Ajude!

Pode parecer bobagem, mas... seja uma pessoa boa! Ajude sempre que possível.

Você nunca perderá nada com isso.

Não existem perguntas idiotas, apenas respostas idiotas

DICA 19

Seja um membro ativo em fóruns

Fóruns e Grupos de discussões contribuem muito para ficar ligado nas últimas tendências, e lá é um bom lugar para ajudar e ser ajudado também.

DICA 20

Academicamente

Leia artigos científicos, eles é que mudam o mundo!

Participe de eventos;

Participe de algum projeto de pesquisa;

Publique um artigo;

Aprenda a escrever! É sério, é difícil e você vai precisar!

Professores? Respeite-os.



CSS3

**revisando e apresentando
novidades**

O QUE É?

- o CSS **formata a informação** entregue pelo HTML;
 - essa informação pode ser qualquer coisa:
 - imagem, texto, vídeo, áudio...
- o **HTML5** trouxe poucas alterações;
 - novas tags;
 - alteração do significado de algumas tags;
 - outras foram descontinuadas;
- já o **CSS3** trouxe mudanças drásticas para a manipulação visual dos elementos do HTML;

O QUE É?

- com as versões anteriores nós podíamos formatar algumas características básicas:
 - cores;
 - backgrounds;
 - fonts;
 - margins;
 - paddings;
 - e uma maneira bem artesanal de controlar os posicionamentos;

O QUE É?

- a idéia do **CSS3**: trazer mais funcionalidades e menos dependências de tecnologias “auxiliares”:
 - *Photoshop* para criar detalhes de *layout*;
 - *JavaScript* para tratarmos comportamentos ou manipularmos elementos específicos no *HTML*;
 - acessibilidade e *SEO*;

QUAIS AS NOVIDADES?

- com as atualizações do **CSS3**, conseguimos:
 - selecionar primeiro e último elemento;
 - selecionar elementos pares ou ímpares;
 - selecionarmos elementos específicos de um determinado grupo de elementos;
 - gradiente em textos e elementos;
 - bordas arredondadas;
 - sombras em texto e elementos;
 - manipulação de opacidade;
 - controle de rotação;
 - controle de perspectiva;
 - animação;
 - estruturação independente da posição no código HTML;

UMA RÁPIDA REVISÃO

a sintaxe dos seletores é bem simples:

SINTAXE BÁSICA DE UM SELETOR EM CSS

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

- *seletor* → é um forma de indicar qual elemento será modificado;
- *propriedade* → é a característica que deseja modificar no elemento;
- *valor* → é o valor referente a esta característica;

UMA RÁPIDA REVISÃO

- o que é um seletor?
 - um seletor representa uma estrutura;
 - essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados;
 - seletores agrupados e encadeados são a base do CSS;
 - e acredite... você só aprende por *osmose* durante o dia-a-dia de desenvolvimento;

DICA → sempre pesquise uma forma de selecionar elementos específicos para fixar cada uma de suas utilidades.

UMA RÁPIDA REVISÃO

um exemplo de seletor encadeado:

SELETOR ENCADEADO

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

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

UMA RÁPIDA REVISÃO

um exemplo de seletor agrupado:

SELETOR AGRUPADO

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

*este seletor formata todos elementos **strong**, **em** ou **span***

DICA → agrupe elementos separados por vírgula para que herdem a mesma formatação.

ID E CLASS

- é a forma básica de se nomear os elementos em HTML5;
- um **ID** deve ser atribuído à um elemento único;
 - nunca deve ser repetido;
 - exemplo de utilização: `<div id="my-id"></div>`;
- uma **class** deve ser atribuída a elementos que têm propriedades em comum:
 - podem ser utilizadas por mais de um elemento;
 - pode-se ter mais de uma classe por elemento;
 - exemplo de utilização: `<div class="color-blue big-size"></div>`

SELETORES COMPLEXOS

- os seletores básicos são para cobrir suas necessidades básicas de formatação de elementos;
- mas o que fará você escrever menos código, e principalmente menos *JavaScript* são os seletores *complexos*;
 - eles selecionam elementos que talvez você precisaria fazer algum *script* em JavaScript para poder marcá-lo com uma CLASS ou um ID para então você formatá-lo.

SELETORES COMPLEXOS

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

SELETOR AGRUPADO

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

tornam-se muito úteis ao trabalhar com animações e pseudo-seletores
lista atualizada pelo **W3C** → <http://www.w3.org/TR/css3-selectors/#selectors>

SELETORES COMPLEXOS

```
<div data-MyEl="teste"></div>
```

PADRÃO	SIGNIFICADO	CSS
elemento[atr]	Elemento com um atributo específico.	2
elemento[atr="x"]	Elemento que tenha um atributo com um valor específico igual a "x".	2
elemento[atr~="x"]	Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é "x".	2
elemento[atr^="x"]	Elemento com um atributo cujo valor começa exatamente com string "x".	3
elemento[atr\$="x"]	Elemento com um atributo cujo valor termina exatamente com string "x".	3

SELETORES COMPLEXOS

PADRÃO	SIGNIFICADO	CSS
elemento[atr*="x"]	Elemento com um atributo cujo valor contenha a string "x".	3
elemento[atr]="en"]	Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EN (da esquerda para direita).	2
elemento:root	Elemento root do documento. Normalmente o HTML.	3
elemento:nth-child(n)	Selecione um objeto N de um determinado elemento.	3
elemento:nth-last-child(n)	Seleciona um objeto N começando pelo último objeto do elemento.	3

SELETORES COMPLEXOS

PADRÃO	SIGNIFICADO	CSS
elemento:empty	Seleciona um elemento vazio, sem filhos. Incluindo elementos de texto.	3
elemento:enabled elemento:disabled	Seleciona um elemento de interface que esteja habilitado ou desabilitado, como selects, checkbox, radio button etc.	3
elemento:checked	Seleciona elementos que estão checados, como radio buttons e checkboxes.	3
E > F	Seleciona os elementos E que são filhos diretos de F.	2
E + F	Seleciona um elemento F que precede imediatamente o elemento E.	2

O QUE SÃO PREFIXOS E PORQUE USAR?

- é uma forma de manter a compatibilidade da propriedade em vários navegadores;
- propriedades que ainda não estão totalmente especificadas;
- note que a última chamada é a “oficial” e “sobrescreve” todas as anteriores;

PSEUDO SELETORES

```
div {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  -o-border-radius: 10px;  
  border-radius: 10px;  
}
```

PSEUDO SELETORES

especificam um estado especial do elemento selecionado;

PSEUDO SELETORES

```
a:hover {  
    color:red;  
}
```


PSEUDO SELETORES

dois seletores especiais: after e before;
“criam” um novo elemento antes ou depois do elemento selecionado:

PSEUDO SELETORES

```
div:after {  
    content : 'Uma string'  
}  
  
a:before{  
    content : url('img.jpg');  
}  
  
section:after{  
    content : content: "[line " attr(data-line) "]";  
}
```

PRINCIPAIS PSEUDO SELETORES

PADRÃO	SIGNIFICADO
elemento:active	aplica as regras quando um elemento é ativado. Por exemplo, quando clicamos em um link e não soltamos o botão do mouse.
elemento:hover	quando passamos o mouse em cima do elemento.
elemento:visited	quando o link é visitado.
elemento:focus	quando um elemento recebe foco. Muito utilizado em campos de texto.

EXEMPLOS RÁPIDOS - GRADIENTE

o que é? → possibilita a criação de um background degradê;

COMO FAZER?

```
div {  
    background: red;  
    background: -webkit-linear-gradient(red, yellow);  
    background: -o-linear-gradient(red, yellow);  
    background: -moz-linear-gradient(red, yellow);  
    background: linear-gradient(red, yellow);  
}
```

EXEMPLOS RÁPIDOS - TRANSFORM

o que é? → manipula a forma com que o elemento aparecerá na tela;

scale → modifica as dimensões de um elemento;

skew → modifica os ângulos de um elemento;

translation → move o elemento no eixo X e Y;

rotate → rotaciona o elemento;

COMO FAZER?

```
div {  
  transform: scale(1.5) skew(30deg) rotate(20deg);  
  -webkit-transform: scale(1.5) skew(30deg) rotate(20deg);  
  -moz-transform: scale(1.5) skew(30deg) rotate(20deg);  
  -o-transform: scale(1.5) skew(30deg) rotate(20deg);  
}
```

EXEMPLOS RÁPIDOS - TRANSIÇÕES

o que é? → possibilita a criação de transições;

COMO FAZER?

```
div {  
    color: white;  
    background: gray;  
    transition: all 0.5s linear;  
    -webkit-transition: all 0.5s linear;  
}  
div:hover {  
    color: white;  
    background: gray;  
    transition: all 0.5s linear;  
    -webkit-transition: all 0.5s linear;  
}
```

EXEMPLOS RÁPIDOS - ANIMAÇÕES

o que é? → possibilita criar animações em css;

DEFININDO A ANIMAÇÃO

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

APLICANDO A ANIMAÇÃO

```
div:hover{  
  -webkit-animation-name: rotate;  
  -webkit-animation-duration: 0.5s;  
  -webkit-animation-timing-function: linear;  
  -webkit-animation-iteration-count: infinite;  
  -webkit-animation-direction: alternate;  
}
```

EXEMPLOS RÁPIDOS - FONT-FACE

o que é? → possibilita utilizar uma fonte específica em seu site;

COMO FAZER?

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

EXEMPLOS RÁPIDOS - MEDIA-QUERIES

o que é? → aplica a regra para determinado tipo e tamanho de tela;

COMO FAZER?

```
@media screen and (min-width: 768px) {  
  a {color: yellow;}  
}  
  
@media screen and (min-width: 992px) {  
  a {color: green;}  
}
```


PROPRIEDADE FLOAT

- é a propriedade que define onde um elemento deverá “**flutuar**”
- é um tanto quanto imprevisível;
- por definição:
 - `float:left` → alinha o elemento a esquerda;
 - `float:right` → alinha o elemento a direita;
 - `float:none` → restaura o valor original;

PROPRIEDADE POSITION

- **por que ainda utilizar?**
 - retrocompatibilidade;
- **problemas conhecidos** → centralização de elementos;
- essencialmente **não** é só ela quem diagrama os *layouts*;
 - isso se faz em conjunto com a propriedade: *float*;

PROPRIEDADE POSITION

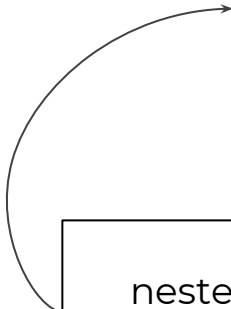
- **coordenadas!**

- são a base para o entendimento dos elementos em css através de: `relative`, `absolute` e `fixed`;
- utilizam as propriedades: `top`, `left`, `right` ou `bottom`;
- todos os valores de `position` trabalham com essas coordenadas;
- obviamente se você define `left`, não precisa definir `right`;
- e o mesmo serve para `top` e `bottom`;

PROPRIEDADE POSITION

UTILIZANDO AS COORDENADAS PARA POSICIONAR UMA DIV

```
div {  
  position: absolute;  
  top: 150px;  
  left: 150px;  
}
```



neste caso, a div está posicionada a *150px* do topo e *150px* da esquerda

agora podemos falar dos tipos de position: **relative**, **absolute** e **fixed**

PROPRIEDADE POSITION : FIXED

- esta propriedade irá **fixar** a posição do elemento na coordenada que você definir;
- a medida que se faz o *scroll* o elemento continuará fixo na posição que foi definida;
- o conteúdo continua rolando normalmente;
- geralmente é utilizado para fixar cabeçalhos ou sidebars;

PROPRIEDADE POSITION : FIXED

EXEMPLO DE UTILIZAÇÃO DO FIXED

```
div#sidebar {  
    width: 200px;  
    height: 300px;  
    background: #75AD45;  
    border: 2px solid black;  
    position: fixed;  
    top: 30px;  
    right: 30px;  
    color: white;  
    padding: 10px;  
}
```

PROPRIEDADE POSITION : RELATIVE

- todos os *positions* precisam de um **ponto** para iniciar o cálculo da coordenada;
- utilizam este ponto para posicionar o elemento na tela;
- este ponto é o canto **superior esquerdo** do elemento;
- a partir deste ponto, o navegador irá calcular a coordenada que você definiu e irá posicionar o elemento no *viewport* (elemento externo ao em questão).
- o **position relative** posiciona o elemento em relação a **si mesmo**;
- utilizado para posicionar elementos lado a lado, e deixá-los quebrar a linha automaticamente (quando não houver mais espaço);
- utilizado quando não se necessita de um posicionamento preciso, ele se orienta por seu próprio ponto de referência;

PROPRIEDADE POSITION : ABSOLUTE

- o **position absolute** posiciona o elemento em relação ao seu pai (que contenha um `position absolute` ou `relative`);
- se nenhum div pai for encontrado, utiliza como referência o body;
- posiciona exatamente o elemento com base nos atributos de posicionamento: `top`, `left`, `right` ou `bottom`;

PROPRIEDADE Z-INDEX

- ocasionalmente você precisará colocar um elemento acima do outro;
- funciona apenas para elementos com position: fixed ou absolute;
- isso se consegue com a propriedade z-index;
- quem tiver maior número fica acima;

EXEMPLO Z-INDEX

```
div {  
    position: absolute;  
    top: 150px;  
    left: 150px;  
    z-index: 150;  
}
```

UNIDADES DE MEDIDA INTERESSANTES

algumas medidas interessantes: **em**, **rem**, **vh** e **vw**

COMO FAZER?

```
html{ font-size: 14px }  
body { font-size: 14px }  
body div{ font-size: 2em; }  
body div div { font-size: 2em; }  
a { font-size: 5rem; }  
.container { width: 100vh; height: 100vw; }
```

UTILIZANDO O CALC

com o **calc**, podemos realizar cálculos no próprio css

COMO FAZER?

```
div {  
    width: calc(100vh/2);  
}  
  
a{  
    margin-left: calc(200px*3);  
}
```

HACK PARA CENTRALIZAR AS COISAS

podemos utilizar alguns hacks para centralizar as coisas com *absolute*:

HACK 1

```
div {  
    position: absolute;  
    height: 150px;  
    width: 150px;  
    left: 50%;  
    margin-left: calc(150px/2);  
}
```

HACK 2

```
div {  
    position: absolute;  
    height: 150px;  
    width: 150px;  
    left: 50%;  
    transition: transformX(-50%);  
}
```

MATERIAIS COMPLEMENTARES CSS3

- <https://goo.gl/6VzDj4>
- <https://goo.gl/bHhALv>
- <https://goo.gl/Y2KXxK>
- <https://goo.gl/zDxSoz>
- <https://goo.gl/CDfSMV>
- <https://goo.gl/NUApxh>
- <https://goo.gl/kssk6Z>
- <https://goo.gl/9TgCe6>
- <https://goo.gl/up9GTz>
- <https://goo.gl/6AZ7ii>
- <https://goo.gl/1vW22L>

CSS FLEX e CSS GRID

**as novas formas de se *layoutar*
com CSS**

O QUE SÃO ESSES CONCEITOS?

- são basicamente implementações na linguagem CSS3;
- por que utilizar? → cansado de ficar ajustando tudo e não saber o que aconteceu como funciona?
 - problema → float;
- aplicam novos conceitos de formatação do seu html e css;
- já estão preparados para responsividade;
- aqui estão apresentados conceitos superficiais das tecnologias, e materiais para aprofundamento;

CSS FLEXBOX

O QUE É CSS FLEXBOX?

- faz parte da especificação do CSS3 que promete organizar elementos na página previsivelmente quando o *layout* precisa ser visualizado em diversos tamanhos de tela e em diversos dispositivos;
- nos ajuda a organizar elementos sem a ajuda do float;
- também nos ajudam a resolver problemas de *box model* → quando acrescentamos, *padding*, *margin* e *border* além da largura do elemento.

COMO FUNCIONA CSS FLEXBOX?

- a ideia é simples: os filhos de um elemento com flexbox pode se posicionar em qualquer direção e pode ter dimensões flexíveis para se adaptar;
- você pode posicionar os diversos elementos independente da sua posição na estrutura do HTML;
- um dos problemas do float é a sua dependência com os elementos na estrutura do HTML;
 - estes elementos precisam estar em uma ordem específica, se não o layout não dá certo.
- com o Flexbox essa ordem não importa, isso quer dizer que você pode organizar a informação do seu HTML de beneficiando o SEO e Acessibilidade

ELEMENTOS E VOCABULÁRIO

- Flexbox é uma nova maneira de posicionar elementos do seu *layout* e por isso precisamos de novos nomes para identificar os elementos da estrutura;
- Flex container → é o elemento que envolve sua estrutura. define-se que um elemento é um Flex Container com a propriedade `display` com os valores `flex` ou `inline-flex`;
- Flex Item → são os elementos filho do flex container;
- Eixos ou Axes → são as duas direções básicas que existem em um Flex Container: *main axis*, que seria o eixo horizontal ou o eixo principal e o *cross axis* que seria o eixo vertical.

ELEMENTOS E VOCABULÁRIO

- Directions → determina a origem e o término do fluxo dos ítems. Eles seguem o vetor estabelecido pelo modo tradicional de escrita: esquerda para direita, direita para esquerda etc;
- a propriedade order determina o lugar que os elementos aparecerão;
- a propriedade flex-flow determina a ordem do fluxo em que os elementos aparecerão;
- você pode definir se os elementos serão forçados a ficar em uma mesma linha ou se eles irão quebrar em várias linhas com a propriedade flex-wrap;

GUIA DEFINITIVO

para aprender e realmente entender... só praticando

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

EXEMPLOS E ESTUDOS

alguns estudos sobre flexbox

<https://github.com/diogocezar/dctb-flex>

APRENDA JOGANDO

guie um sapinho com comandos flexbox e aprenda se divertindo

<http://flexboxfroggy.com/>

LIVE CODE

vamos ver algumas coisas ;)

[https://github.com/diogocezar/dctb-utfpr/tree/master/2017/programacao-web-2/
ex10-css-flex](https://github.com/diogocezar/dctb-utfpr/tree/master/2017/programacao-web-2/ex10-css-flex)

CSS GRID

O QUE É CSS GRID?

- faz parte da especificação do CSS3 que promete organizar elementos na página previsivelmente quando o *layout* precisa ser visualizado em diversos tamanhos de tela e em diversos dispositivos;
- nos ajuda a organizar elementos sem a ajuda do float;
- também nos ajudam a resolver problemas de box model → quando acrescentamos, padding, margin e border além da largura do elemento.

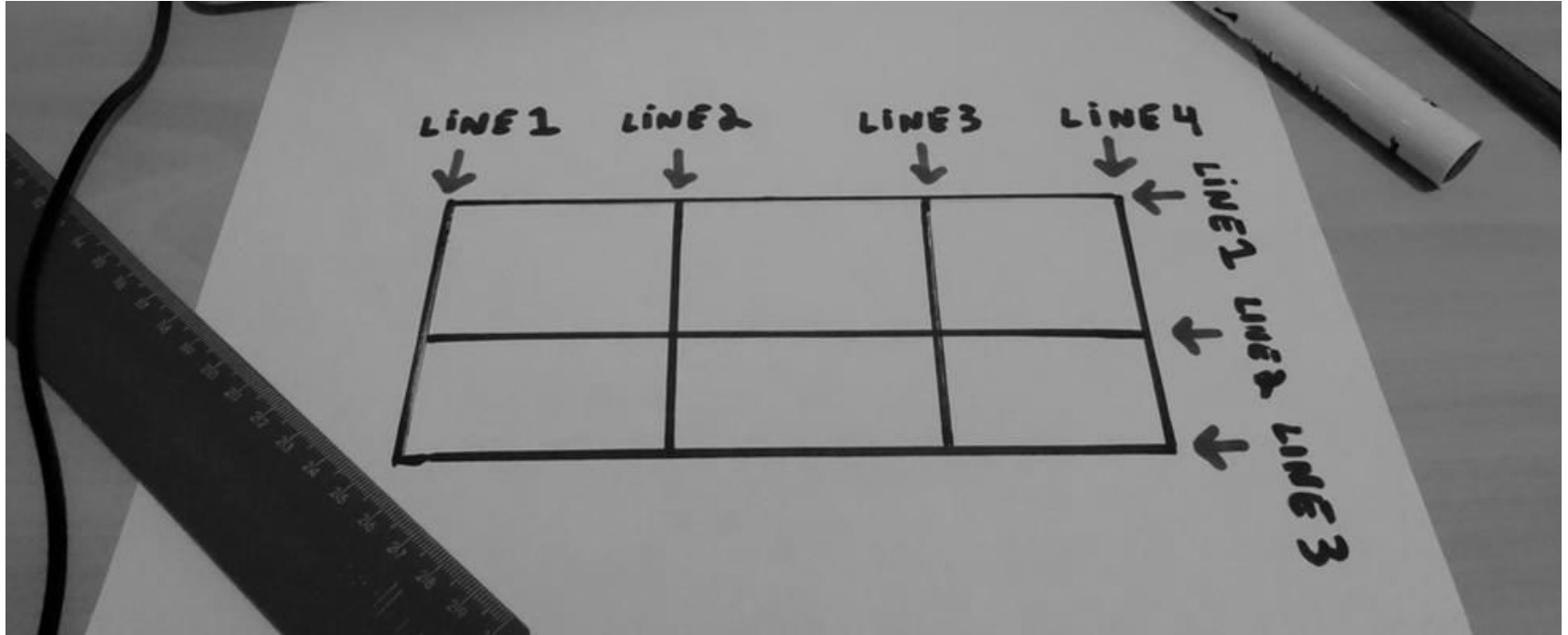
O QUE É CSS GRID?

- utiliza um conceito de 2 dimensões, enquanto o flexbox utiliza conceitos de 1 dimensão;
- já se pode utilizar?
 - <http://caniuse.com/#feat=css-grid>

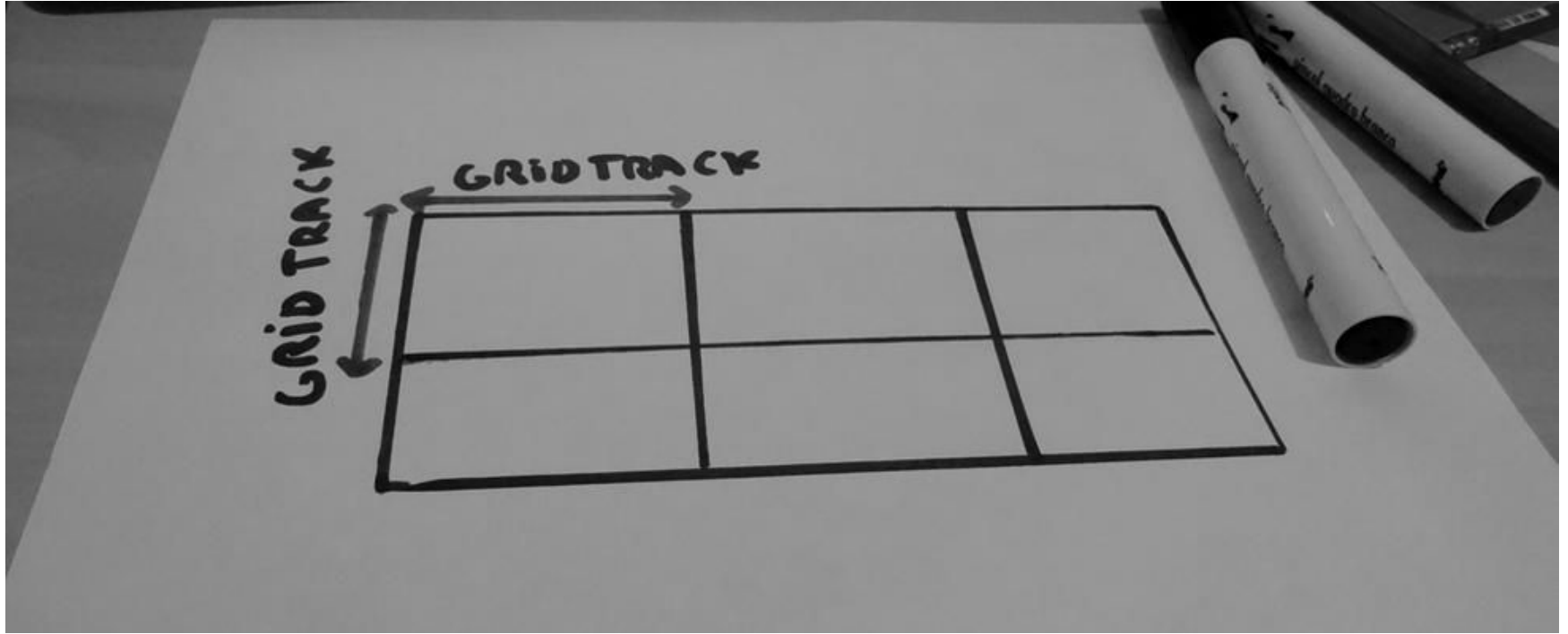
TERMINOLOGIA

- vamos definir alguns termos utilizados com grids:
 - grid lines → são as linhas que definem o grid, elas podem ser distribuídas de forma horizontal ou vertical;
 - grid tracks → é o espaço horizontal ou vertical entre duas grid lines;
 - grid cell ou grid item → é o espaço entre quatro Grid Lines, sendo a menor unidade em nosso grid, conceitualmente podemos fazer uma analogia com uma célula de tabela;
 - grid áreas → é qualquer espaço no Grid usado para exibir um ou mais Grid Cells/Items.

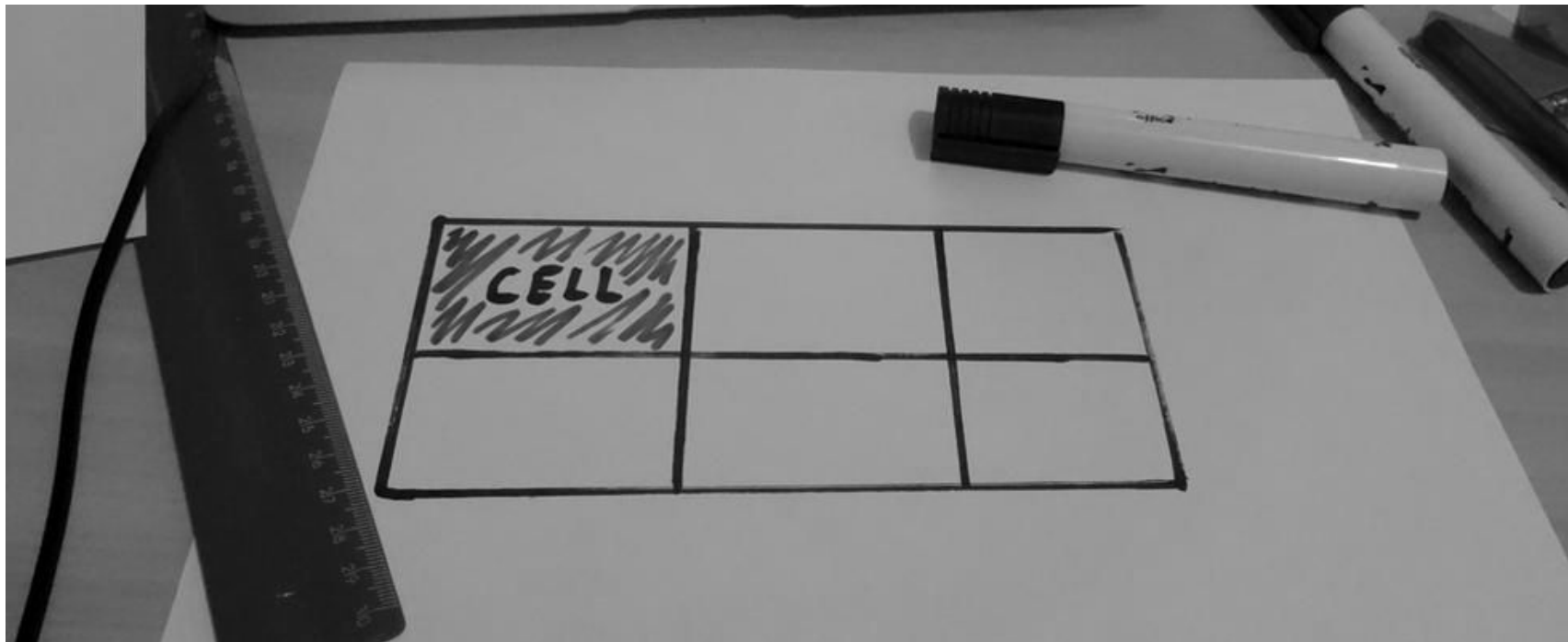
GRID LINE



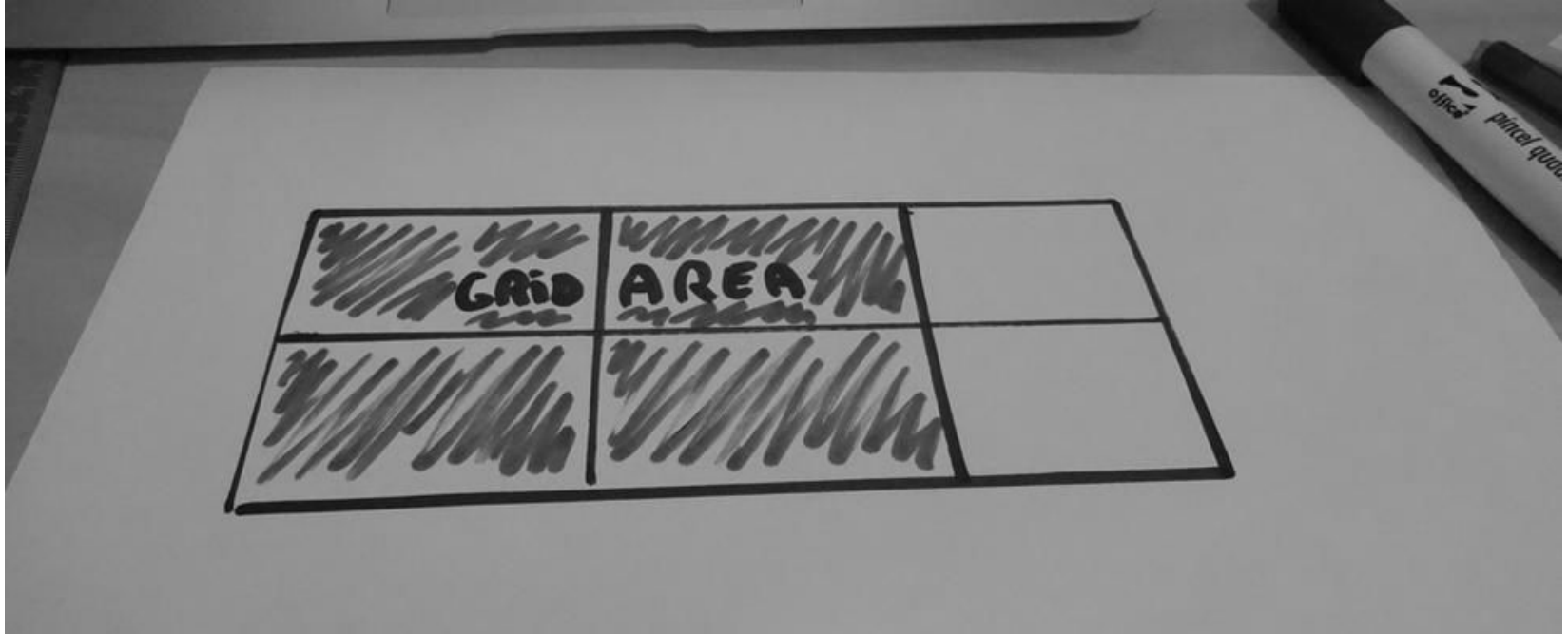
GRID TRACKS



GRID CELL



GRID ÁREAS



TERMINOLOGIA

- vamos definir alguns termos utilizados com grids:
 - grid containers → é o pai direto de todos os itens do grid sendo o elemento que recebe a propriedade `display: grid`;
 - grid items → são os itens que representam o conteúdo do grid, cada filho direto do grid container torna-se um grid item;
 - para definir as colunas e linhas do grid use as propriedades grid-template-columns e grid-template-rows.
 - use grid-column-start e grid-row-start para especificar a grid line onde o item começa e grid-column-end e grid-row-end para especificar a grid line onde o item termina;

TERMINOLOGIA

- vamos definir alguns termos utilizados com grids:
 - area naming → No grid container podemos manipular a posição e comportamento dos grid itens, isso através da propriedade grid-template-areas, onde podemos literalmente montar todo o nosso layout em apenas uma propriedade;
 - para facilitar e atribuir mais semântica ao que estamos fazendo, atribuímos um nome a nossos grid itens através da já conhecida grid-area.

EXEMPLOS E ESTUDOS

alguns estudos sobre grid

<https://github.com/diogocezar/dctb-grid>

GUIA DEFINITIVO

para aprender e realmente entender... só praticando

<https://css-tricks.com/snippets/css/complete-guide-grid/>

APRENDA JOGANDO

aprenda se divertindo no grid garden

<http://cssgridgarden.com/>

LIVE CODE

vamos ver algumas coisas ;)

[https://github.com/diogocezar/dctb-utfpr/tree/master/2017/programacao-web-2/
ex11-css-grid](https://github.com/diogocezar/dctb-utfpr/tree/master/2017/programacao-web-2/ex11-css-grid)

QUAL DEVO USAR?

QUAL DEVO UTILIZAR?

- flexbox → é para layouts unidimensionais; ou seja, qualquer coisa que precise ser disposta em uma linha reta;
- grid → é a solução certa quando você deseja controlar o dimensionamento e o alinhamento em suas dimensões;

MATERIAIS COMPLEMENTARES FLEX E GRID

- <https://goo.gl/hVh29F>
- <https://goo.gl/65YKWC>

GERENCIADORES DE PACOTES

**a base para instalações de
ambientes *web***

GERENCIADORES DE PACOTES

- é comum você inserir códigos de terceiros em seus projetos;
- esses projetos estão em constante evolução, certo?
- problema → como manter tudo atualizado e organizado?
- solução → utilizando empacotadores de código;
- os empacotadores utilizam estruturas pré-definidas para disponibilizar componentes e plugins para sua aplicação;
- podem ser conhecidos também como: gerenciadores de dependências;

GERENCIADORES DE PACOTES

- um dos mais conhecidos é o **npm**;
- é o empacotador oficial da linguagem *NodeJS*;
- com ele é possível instalar milhares de projetos, não somente em *NodeJS*;
- onde encontrar os projetos?
 - <https://www.npmjs.com/>
- como instalar o **npm**? ele vem junto com o *NodeJS*, ou... `apt-get install npm`

GERENCIADORES DE PACOTES

- o **npm** gerencia os seus projetos a partir de um arquivo JSON chamado `package.json`
- é no `package.json` onde estão todas as informações do seu projeto, como:
 - nome;
 - versão;
 - descrição;
 - autor;
 - licença;
 - dependências;
 - outros.

GERENCIADORES DE PACOTES

EXEMPLO DE UM PACKAGE.JSON

```
{
  "name": "minha-api",
  "version": "1.0.0",
  "description": "Api de testes",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "testes",
  ],
  "author": "Diogo Cezar",
  "license": "WTFPL"
}
```

GERENCIADORES DE PACOTES

- o objeto de dependências é o mais importante em seu projeto;
- fica armazenado em uma pasta chamada *node_modules*;
- essa pasta nunca deverá ser enviada para seu repositório git;
 - por que? *para economizar espaço*, visto que, sempre que alguém utilizar a estrutura, poderá baixar novamente as dependências utilizando o comando `npm install` que buscará as informações em *package.json*
 - por isso, não se esqueça de adicioná-la ao `.gitignore`

GERENCIADORES DE PACOTES

- o comando `npm init` inicia um novo projeto;
- o comando `npm install nome_modulo` serve para instalar uma nova dependência;
- o parâmetro `npm install nome_modulo -g` ou `--global` diz qual o modo de instalação:
 - se uma instalação é local ela será utilizada somente em seu projeto;
 - se uma instalação é global ela será instalada e estará disponível em todo seu sistema, criando um comando para seu sistema;
- deve-se utilizar o parâmetro `--save` para adicionar esta dependência na lista de dependências do arquivo `package.json`

GERENCIADORES DE PACOTES

- caso precise instalar uma versão específica de um pacote, pode-se utilizar:
`npm install --save modulo@versão;`
- então possuímos algumas formas diferentes de especificar a versão do nosso módulo, que são:
 - ~versão → equivalente à versão;
 - ^versão → compatível com a versão;
 - versão → precisa ser a versão exata;
 - >versão → precisa ser maior que a versão;
 - >=versão → precisa ser maior ou igual a versão;
 - < versão → precisa ser menor que a versão;
 - <=versão → precisa ser menor ou igual a versão;

GERENCIADORES DE PACOTES

- às vezes será preciso de algumas dependências apenas em modo de desenvolvimento, e não no modo de produção;
- para isso você pode salvar uma dependência específica como dependência de desenvolvimento;
- para isso basta utiliza ao invés do `--save` o `--save-dev`;

GERENCIADORES DE PACOTES

- ou ainda, você poderá instalar uma dependência opcional:
- para isso basta utiliza ao invés do `--save` o `--optional`;

GERENCIADORES DE PACOTES

- o `npm run` é o que executa seu projeto;
- você pode executar *scripts* para automatizar suas tarefas;
- para isso, você precisará configurar a seção `scripts` do seu `package.json`:

```
"scripts": {  
  "roda": "node script.js"  
},  
...
```

- e depois, basta executar o comando: `npm run roda` para executar o script em questão;

GERENCIADORES DE PACOTES

- **Bower** é um gerenciador de pacotes para a web;
- como se fosse um **npm**, mas para o desenvolvimento *web*, ao invés de desenvolvimento para o node;
- o propósito de Bower é para gerenciar os pacotes de um front-end, que podem incluir não apenas os arquivos javascript, mas também HTML, CSS, imagens e arquivos de fontes;
- onde encontrar os pacotes?
 - <https://bower.io/search/>

GERENCIADORES DE PACOTES

- e para instalar o bower, bom... **npm**:
- `npm install bower -global`
- o bower criará toda a estrutura em um objeto separado do *node_modules*;
- seu objeto de dependência é: *bower_components*

AUTOMATIZADORE S



**as tecnologias para compilar
SASS**

O QUE SÃO?

- *task runners* são automatizadores de tarefas;
- ajudam bastante nas tarefas manuais do dia a dia;
- de uma forma bem resumida... um projeto web precisa passar por algumas otimizações antes de ir para produção;
- como por exemplo:
 - minificação do CSS;
 - otimização de imagens;
 - unificação e/ou ofuscação de JavaScripts;

O QUE SÃO?

- essas otimizações são extremamente importantes para obter um melhor ranking de SEO do site em motores de busca;
- entre algumas métricas, a velocidade de carregamento conta muito;
- essas ações ajudam neste processo;

O QUE SÃO?

- problema:
 - é impossível escrever nossos códigos já de forma minificada;
 - não recebe as imagens já otimizadas;
- solução:
 - utilizar alguma ferramenta que faça isso de forma automática por você;

O QUE SÃO?

- outro fator: compilação de arquivos;
- utilização de pré-processadores em CSS, como SASS ou LESS, ou ainda um transpiler de JavaScript como Babel ou CoffeScript;
- nestes casos, você de alguma mágica que transforme os arquivos originais nos arquivos compilados;

GRUNT

- esse foi o primeiro task runner a ganhar destaque na comunidade JavaScript;
- ele é simples e as tarefas são definidas em um arquivo de configuração;
- essa simplicidade pode incomodar quando se precisa de um fluxo mais complexo;
- possui vários plugins → <https://gruntjs.com/plugins>
- pode ser encontrado em → <https://gruntjs.com/>

NPM SCRIPTS

- como já vimos com o NPM podemos criar scripts customizados para realizar determinadas ações;
- podemos utilizar webpack e browserify para otimização de algumas tarefas pelos próprios scripts npm;

OUTRAS ALTERNATIVAS

- <http://broccolijs.com/> e <http://brunch.io/> são algumas dessas opções, além de opções muito recente, que pouquíssimas pessoas usam ou já ouviram falar, como o <https://github.com/lukeed/taskr>.

GULP

- o **gulp** não é uma ferramenta nova, porém ganhou destaque nos últimos anos;
- acabou passando o Grunt, porém não tem a mesma quantidade de plugins;
- existem 2 grandes diferenciais entre o Grunt e o gulp: a definição de suas tasks e como elas são executadas;
 - as tarefas são executadas em modo async e em memória, tornando ele mais rápido que o grunt;
- além disso, as tarefas são definidas como código, o que permite que você programe suas tarefas;

GULP

INSTALANDO O GULP

```
$npm install -g gulp
```

VERIFICAR A VERSÃO INSTALADA

```
$gulp -v
```


GULP

- toda a automação que criaremos ser reunida em um único arquivo chamado *gulpfile.js*
- a localização do arquivo pode variar de acordo com seu projeto, mas você poderá informar as localizações dos arquivos que deseja automatizar;
- normalmente o gulpfile é colocado na raiz do projeto;

GULP

- agora precisamos definir quais serão os plugins a serem utilizados com o gulp;
- você pode buscar os plugins e realizar a instalação como consta na página de instruções;
- e ainda será necessário instalar o próprio gulp no seu projeto;

INSTALANDO PLUGINS PARA O GULP

```
$npm install gulp gulp-sass --save-dev
```

UTILIZANDO SASS

```
Gulp = {
  self: null,
  sass: null,
  init: function(){
    Gulp.self = require('gulp');
    Gulp.sass = require('gulp-sass');
    Gulp.default();
  },
  default: function(){
    Gulp.self.task('default', ['scss']);
  },
  scss: function(){
    Gulp.self.src('./assets/scss/*.scss')
      .pipe(Gulp.sass())
      .pipe(Gulp.self.dest('./assets/css/'));
  }
}; Gulp.init();
```

UTILIZANDO WATCH

com o watch consegue inspecionar permanentemente qualquer alteração em seus arquivos, e a cada nova alteração invocar uma funcionalidade definida no gulp

```
Gulp.self.task('watch', function(){  
  Gulp.self.watch('./assets/scss/**', ['scss']);  
});
```

MATERIAIS COMPLEMENTARES

AUTOMATIZADORES

- <https://goo.gl/qX8L69>
- <https://goo.gl/AtK6pG>

SASS

uffa! agora vamos ao SASS

O QUE É?

- produtividade é a palavra chave;
- pergunta: se você é desenvolvedor web, quantas vezes se pegou copiando e colando códigos CSS com mais de 15 linhas que fazem a mesma coisa?
- pré-processadores CSS vem para suprir essa e outras necessidades;
- os *big players* são:
 - SASS (<http://sass-lang.com>);
 - LESS (<http://lesscss.org>);
- mas existem outros → <https://goo.gl/HQKc1j>

PRÉ-PROCESSAMENTO

- CSS em si pode ser divertido;
- mas códigos grandes e complexos podem ser difíceis de manter;
- é neste momento que um pré-processador pode ajudar;
- permitem que você adicione funcionalidades que não existem no css:
 - variáveis → chega de dar replace de uma cor em todo o documento;
 - aninhamentos (nesting) → propõe uma nova maneira de organizar os seus elementos;
 - mixins → permite a criação de grupos de declarações CSS e sua reutilização, se comportam como funções;
 - partials e imports → inclusão de arquivos;
 - operadores → permite a utilização de operadores para cálculos de medidas;

PRÉ-PROCESSAMENTO

- um pré-processador lê o código que nele é escrito, e o compila em um arquivo CSS normal, que pode ser utilizado em seu site;
- existem várias formas de compilar seu arquivo;
- vimos a ferramenta gulp e suas duas formas de compilação: única e inspecionando o arquivos por modificações;

VARIÁVEIS

- as variáveis são uma forma de se armazenar informações para serem reutilizadas em seu CSS;
- você pode armazenar informações como cores, tipos de fontes, ou qualquer outro valor CSS que você precise reutilizar;
- SASS utiliza o símbolo \$ para transformar algo em uma variável;

VARIÁVEIS

DECLARANDO VARIÁVEIS EM SASS

```
$font-stack: Helvetica, sans-serif;  
$primary-color: #333;  
  
body {  
  font: 100% $font-stack;  
  color: $primary-color;  
}
```

VARIÁVEIS

- quando o arquivos sass é processado, ele obtém os valores definidos nas variáveis e cria um arquivo css normal com os respectivos valores;
- a grande vantagem?
 - se seu projeto mudar a paleta de cores, ou o tamanho das fontes, ou a fonte primária... você só precisará substituir essa instrução na declaração da variável;

VARIÁVEIS

QUANDO COMPILA-SE PARA CSS

```
body {  
  font: 100% Helvetica, sans-serif;  
  color: #333;  
}
```

CONFIGURANDO O AMBIENTE

- modo hard → instalar o nodejs + instalar o npm + instalar o gulp global + criar um diretório + instalar as dependências (gulp e gulp-sass) + criar um gulpfile.js + criar um método para compilar o sass + criar um outro método para inspecionar (watch) + abrir terminal e executar comandos;
 - não se engane, no ambiente de produção... vai ser assim!
- modo easy → <https://www.sassmeister.com/>

PRÁTICA 1 - VARIÁVEIS

- crie três variáveis para cores:
 - \$cor-verde : green;
 - \$cor-vermelho : red;
 - \$cor-cinza: #DDDDDD;
- crie três classes que utilizam essas variáveis:
 - .cor-verde;
 - .cor-vermelho;
 - .cor-cinza;
- altere a cor-vermelho para: blue;
- analise os resultados;

ANINHAMENTO (NESTING)

- quando se escreve em HTML você provavelmente está acostumado com uma estrutura aninhada;
- mas o CSS não permite isso;
- com SASS você poderá aninhar os seus elementos da mesma forma hierárquica que está definida em seu HTML;
- mas tenha cuidado → regras excessivamente aninhadas resultarão em um CSS de difícil manutenção, e isso é considerado má prática.

ANINHAMENTO (NESTING)

UTILIZANDO ANINHAMENTO

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  
  li { display: inline-block; }  
  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
  }  
}
```

ANINHAMENTO (NESTING)

- note que os elementos *ul*, *li* e *a* estão aninhados dentro do seletor *nav*;
- essa é uma ótima forma de organizar o seu CSS tornando-o mais legível;
- seu código fica específico → a regra só será aplicada a um elemento que está dentro de outro;

ANINHAMENTO (NESTING)

QUANDO COMPILA-SE PARA CSS

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
  
nav li {  
  display: inline-block;  
}  
  
nav a {  
  display: block;  
  padding: 6px 12px;  
  text-decoration: none;  
}
```

ANINHAMENTO (NESTING)

- podemos utilizar o operador **&** para indicar o próprio elemento em um aninhamento;
- podem ser utilizados com os pseudo elementos do CSS: after, before, hover, active...

ANINHAMENTO (NESTING)

UTILIZAÇÃO DO OPERADOR &

```
#menu {  
  ul {  
    margin: 0;  
    padding: 0;  
    li{  
      list-style: none;  
      a{  
        color: red;  
      }  
      &:hover{  
        color: green;  
      }  
    }  
  }  
}
```

PRÁTICA 2- ANINHAMENTOS

- crie a seguinte estrutura:
 - header
 - h1 → color: blue;
 - h2 → color: red;
 - h3 → color: green;
 - .box-header
 - p → font-size: 15px;
 - .in → border: 1px solid black;
 - p → color: lime;
 - footer
 - .in → font-size: 10px;
 - .items → text-transform: uppercase;
 - #social → font-size: 15px;

PARTIALS

- outra vantagem interessante do SASS é a possibilidade da criação de partials;
- sim, você já podia fazer isso com CSS `@import url("base.css");`
 - mas não são uma boa prática;
 - gera-se dependências, o que deixa sua velocidade de carregamento comprometida;
- a grande sacada do SASS → ele junta tudo em um único arquivo no momento da compilação!
- uma boa prática é a nomenclatura dos partials iniciando com underscore `_`, por exemplo: `_partial.scss`;
- dessa forma o Sass saberá que este arquivo é um arquivo parcial e não deverá ser compilado;

IMPORT

- é a diretiva para importação de um arquivo partial;
- vamos imaginar que temos os seguintes arquivos: `_reset.scss` e `base.scss`;
- e nós desejamos importar o arquivos `_reset.scss` no arquivo `base.scss`;

IMPORT

_RESET.SCSS

```
html,  
body,  
ul,  
ol {  
  margin: 0;  
  padding: 0;  
}
```

_BASE.SCSS

```
@import 'reset';  
  
body {  
  font: 100% Helvetica, sans-serif;  
  background-color: #efefef;  
}
```

não é necessário incluir a extensão .scss



IMPORT

QUANDO COMPILA-SE PARA CSS

```
html, body, ul, ol {  
  margin: 0;  
  padding: 0;  
}  
  
body {  
  font: 100% Helvetica, sans-serif;  
  background-color: #efefef;  
}
```

DEMONSTRAÇÃO

demonstração da utilização do `partials/import`

MIXINS

- algumas coisas em css são tediosas de se escrever;
- especialmente em CSS3 que prevê uma série de prefixos específicos de cada navegador;
- com mixins permitem que você crie grupos de declarações CSS que você poderá reutilizar em todo seu site;
- e o mais bacana! você ainda pode passar valores para tornar seu mixin mais flexível;
- um bom exemplo para o uso de mixins é a definição dos prefixos de navegadores específicos;

O QUE SÃO PREFIXOS E PORQUE USAR?

- é uma forma de manter a compatibilidade da propriedade em vários navegadores;
- propriedades que ainda não estão totalmente especificadas;
- note que a última chamada é a “oficial” e “sobrescreve” todas as anteriores;

PSEUDO SELETORES

```
div {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  -o-border-radius: 10px;  
  border-radius: 10px;  
}
```

MIXINS

EXEMPLO DE UTILIZAÇÃO PARA BORDER-RADIUS

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
  
.box { @include border-radius(10px); }
```

MIXINS

- para criar um mixin, você precisa utilizar a diretiva `#mixin` e dar um nome a ele;
- no exemplo, nomeamos o mixin de `border-radius`;
- também usamos a variável `$radius` entre parênteses, então podemos passar qualquer raio que desejemos;
- depois que um mixin é declarado, poderá ser utilizado em qualquer trecho do seu código sass começando com `@include` seguido do nome do mixin;

MIXINS

QUANDO COMPILA-SE PARA CSS

```
.box {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  border-radius: 10px;  
}
```


PRÁTICA 3 - MIXINS

- faça um mixin (my-scale) para a propriedade scale;
 - -webkit-transform;
 - -moz-transform;
 - -ms-transform;
 - transform;
- faça um mixin (my-rotate) para a propriedade rotate;
- faça um mixin (my-format) que defina as seguintes propriedades:
 - font-size (variável);
 - color (variável);
 - scale → 1.5 (utilizando o my-scale);

EXTEND

- é uma das funções mais úteis do SASS;
- com a diretiva @extend você será capaz de compartilhar um conjunto de propriedades CSS de um seletor para outro;
- então, chega de ficar copiando e colando!

EXTEND

EXEMPLO DE EXTEND

```
.message {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
  
.success {  
  @extend .message;  
  border-color: green;  
}
```

CONTINUAÇÃO

```
.error {  
  @extend .message;  
  border-color: red;  
}  
  
.warning {  
  @extend .message;  
  border-color: yellow;  
}
```

EXTEND

- neste exemplo nós pegamos todas as propriedades de *.message* e aplicamos a *.success*, *.error* e *.warning*;
- a mágica acontece quando o css é gerado e isso o ajuda a evitar a escrita de múltiplos nomes de classes no seu HTML;

EXTEND

QUANDO COMPILA-SE PARA CSS

```
.message, .success, .error, .warning {  
  border: 1px solid #cccccc;  
  padding: 10px;  
  color: #333;  
}  
  
.success {  
  border-color: green;  
}  
  
.error {  
  border-color: red;  
}  
  
.warning {  
  border-color: yellow;  
}
```

PRÁTICA 4 - EXTEND

- defina uma classe padrão para seus títulos chamada (.my-titles);
 - color: black;
 - font-family: "Arial";
 - border: 1px solid black;
 - padding: 20px;
 - width: 100%;
- crie os títulos de h1 a h4 com tamanhos diferentes (5em, 4em, 2em, 1em) e utilize o extend para importar as propriedades definidas em .my-titles;

OPERADORES

- fazer operações matemáticas no seu CSS pode ser interessante;
- SASS permite que você faça operações com: +, -, *, / e %;
- mas isso também pode ser feito de forma nativa certo?
 - sim! em css puro: `calc(100% - 10px);`
 - a diferença é que o valor final será gerado em seu css e não processado pelo navegador;

OPERADORES

EXEMPLO DE UTILIZAÇÃO DE OPERADORES

```
.container { width: 100%; }

article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}

aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

neste exemplo, criamos um grid simples baseada em uma largura de 960px;

OPERADORES

QUANDO COMPILA-SE PARA CSS

```
.container {  
  width: 100%;  
}  
  
article[role="main"] {  
  float: left;  
  width: 62.5%;  
}  
  
aside[role="complementary"] {  
  float: right;  
  width: 31.25%;  
}
```

PRÁTICA 5 - OPERADORES

- crie uma classe (.class-size-1) que tenha uma largura de $\frac{1}{3}$ de 960px;
- crie uma classe (.class-size-2) que tenha $\frac{1}{2}$ da largura de 1200px;
- crie uma classe (.class-size-3) que tenha $\frac{1}{4}$ da altura da viewport (vh);

CRIE VOCÊ TAMBÉM A SUA STACK

- sempre temos formas de organizar e separar os nossos arquivos css;
- eu criei a minha stack → <https://github.com/diogocezar/dctb-sass>

EXERCÍCIOS COMPLEMENTARES

- <https://github.com/diogocezar/Curso-CSS-SASS/blob/master/Exercises/>

MATERIAIS COMPLEMENTARES SASS

- <https://goo.gl/YbHD5D>
- <https://goo.gl/PmM1jS>
- <https://goo.gl/Aa6R8f>
- <https://goo.gl/Xu2vSj>

THAT'S ALL FOLKS