INSTITUTO FEDERAL DE SANTA CATARINA - *CHAPECÓ*

Curso: Técnico em Informática Integrado ao Ensino Médio
Matéria: Tópicos Especiais
Docente: Alexandre Anderson dos Santos

SANDRO DE CASTRO, MATEUS TOMCZAK FAGUNDES, JOÃO VICTOR GILIOLI, RAFAEL ROSSA E VICTOR AZAMBUJA

Tópicos Especiais

16 de outubro de 2025

Boas práticas com CSS

Seletores

Evite seletores que não sejam necessários.
Ex: Seletor ID para a tag *body*.

```
<body id="opa">
```

Isso pode causar confusão no seu CSS, caso você esqueça para que serve esse seletor, então é muito mais intuitivo utilizar a própria tag.
No caso de *divs*, que geralmente existem em maior quantidade no código, é mais interessante utilizar **class**.

Id ou Class?

Prefira **class**. Sempre.

Isso torna seu código mais fácil de ser atualizado e reutilizado.
ID é algo muito engessado. Você não pode ter mais de um e não pode repeti-lo, a **class** você pode definir agora para um elemento e depois adicionar em outro elemento que precise.
Você pode criar uma **class** que tenha um CSS muito utilizado em vários elementos e depois utilizar outras classes para especificar outras coisas desse elemento.

Por exemplo:
Você tem dois elementos na página (**.menu** e **.noticias**) que têm funções e visuais diferentes. Mas os dois seguem a base visual do site que é ter fundo cinza, com borda redonda e uma sombra. Ao invés de copiar as propriedades, crie uma nova classe base pra esses elementos:

```
<div class="box menu">...</div>
<div class="box noticias"></div>

.box {
   background: #ccc;
   border-radius: 5px;
   box-shadow: 2px 2px 2px black;
}
```

Prefira criar essa classe nova ***.box*** que escrever o CSS usando as classes dos outros dois elementos **.menu**, **.noticias**. O dia que um terceiro box surgir, é só aplicar a classe, sem mexer no CSS.

Outro motivo pra você evitar **IDs** no CSS é que você não pode sobrescrever suas propriedades com suas classes genéricas reaproveitáveis. Precisaria de outro **ID** pra sobrescrever um seletor de **ID**.

Não brigue com a especificidade. Mantenha a especificidade dos seus seletores no mesmo nível usando classes sempre pra estilizar. São mais fáceis de compor.

Redundância.

É muito comum a gente se pegar escrevendo código redundante.
explicação:

Você tem uma div e dentro dessa div você tem um button.

```
<div>
 <button>
</div>
```

Aí no seu CSS você tem:

```
div: {
 color: red
}
button: {
 color: red
}
```

No CSS, todos os elementos filhos herdam atributos dos seus elementos pais, ou seja: ***O button já tem color:red de nascença!***
Utilize a herança para deixar seu código limpo e nada redundante.

Hexadecimais.

Sempre que você for utilizar um hexadecimal que tiver repetição de letras, utilize seu atalho.
Ex: **#FFFFFF -> #FFF.**

!important.

Evite utilizar o **!important**.
Uma vez alguém me disse:
“Se você precisa utilizar isso, seu código ta errado.”

Especificidade.

Quando necessário, utilize os seletores para estilos específicos de elementos muito repetidos. Assim você evita com que vaze para outros elementos iguais. Ex:

```
div.caixa> img
OU
div.caixa p:
```

CSS Inline.

NÃO.

Ordem 2.

Se possível, mantenha uma ordem nas suas declarações. Isso deixa as coisas mais organizadas.
A Locaweb utiliza dessa maneira:

1. Posicionamento
2. Box Model
3. Tipografia
4. Visual

Ordem + Organização + Agrupamento.

Se possível faça com que seu **CSS** siga a ordem das tags no arquivo **HTML**.

E tente manter sempre uma organização, como por exemplo:
Das linhas 1 a 50 eu defini CSS dos elementos do cabeçalho, das linhas 53 à 211 eu defini CSS dos elementos do menu e etc.

Comentários.

Evite comentários, um código limpo e organizado é autoexplicativo;
A não ser que seja muito necessário.

Não sobrescreva regras (E mobile-first).

Com toda a certeza você já precisou limpar um *float:left* com um *float:none*, ou reescrever um *width:auto*. **Isso não é legal.**

além de você escrever a mesma propriedade pro mesmo elemento duas vezes, você acaba ignorando os padrões no navegador (*float*, por exemplo, já é **none por padrão).** Inverta seus seletores: utilize o *float:left* em elementos que REALMENTE precisam flutar e os demais herdam o valor padrão.

Geralmente isso acontece muito ao adaptar sites para mobile com media queries. Na tela grande você flutua os elementos para todos os lados, mas no celular decide empilhar e tirar o *float* e fica assim:

```
/* desktop */
.botao-magico {
 float: right;
}

/* mobile */
@media (max-width: 600px) {
 .botao-magico {
    float: none;
 }
}
```

Nesse caso, aliás, isso só aconteceu pois usamos a estratégia de escrever os estilos do desktop primeiro e depois sobrescrever com os estilos mobile. Chamamos essa estratégia de **“desktop-first”** e esse é um ótimo argumento para você passar a usar o **“mobile-first”**. Exemplo certo:

```
/* mobile */
/* nada aqui! o padrão é float:none */

/* desktop */
@media (min-width: 600px) {
 .botao-magico {
    float: right;
 }
}
```

Números mágicos não são tão mágicos assim.

Toda vez que você escreve um *margin-top: 37px* ou um *width: 381px* um **bebê foca morre.**

Evite números mágicos no seu CSS que são calculados arbitrariamente.

Pior ainda aquele número mágico pra alinhar algo **(tipo um top:-1px)** que certamente está levando em conta a renderização num certo browser e vai quebrar em outro navegado.

Isso faz seu **CSS** ser pouco reaproveitável e exige manutenção constante. Quer alinhar um ícone com texto? Aprenda a usar o *vertical-align:middle*.

mas e como eu resolvo o *width*? Isso leva ao próximo tópico:

Use unidades flexíveis.

Números mágicos podem ser evitados em muitos casos usando unidades relativas como porcentagens. Se você tem uma página de *940px* de largura e precisa dividir em 5 colunas, não escreva *width:188px* porque você nunca vai lembrar de onde saiu esse valor. Prefira *width:20%* que mostra de maneira mais óbvia que é 1/5 da página.

E, claro, se puder, faça todas as suas unidades de layout com porcentagens. Isso vai fazer seu design ser flexível e não depender do tamanho do navegador. **A Web é uma mídia elástica e confinar sua página a pixels estáticos é transformar a Web em uma mídia mais limitadas como a impressa**.

Para elementos tipográficos ou afetados pela tipografia, use em como medida flexível.

Nomear alguma coisa é a parte mais difícil de programar.

Como boa parte das boas práticas citadas aqui envolvem classes, é bom saber nomeá-las direito.

A regra óbvia, é criar **nomes legíveis e fáceis de entender**: Use *.painel-principal* ao invés de *.pnlPri*.

Mas não é só isso. Precisamos entender, aqui, que nossas classes não devem ter semântica de visual e sim de conteúdo.
"Mas Gabriel..."
Calma coração, calma que eu explico.

Se as suas classes forem *"box-laranja"* ou *"menu-lateral"* e algum dia esses elementos não forem mais laranja ou lateral, o nome da classe perde significado. Seja simples e preciso. Utilize *"box"*, *"menu"*, *"painel"* e por aí vai.

Documente bem as exceções.

Lembra que eu falei lá em cima que iríamos conversar melhor depois? Pois é, é chegada a hora!

Toda regra tem sua exceção.

As vezes você vai precisar de um **ID** no seu **CSS**, ou até um *!important*. Ou, quem sabe, aquele número mágico que você não vai fazer ideia de onde veio amanhã.

Lembre-se, isso é exceção, não abuse. De preferência evite usar.

Mas, caso você faça, documente direitinho no CSS o porquê da sua escolha.
Ex:

```
/* usando ID porque o widget do facebook obriga */

#facebook-like iframe {
  width: 100% !important;
  /* important pra sobrescrever css inline do widget */
}
```