Clean Code no Front-End

Irei contar para vocês a história de um jovem empreendedor chamado João...

O código ruim é inversamente proporcional à produtividade, quanto mais código ruim, menor será a produtividade da equipe.

Trecho retirado do livro "Clean Code"

A única maneira de isso não acontecer, é manter o código limpo

Matheus Ruan

Aí vem a pergunta:

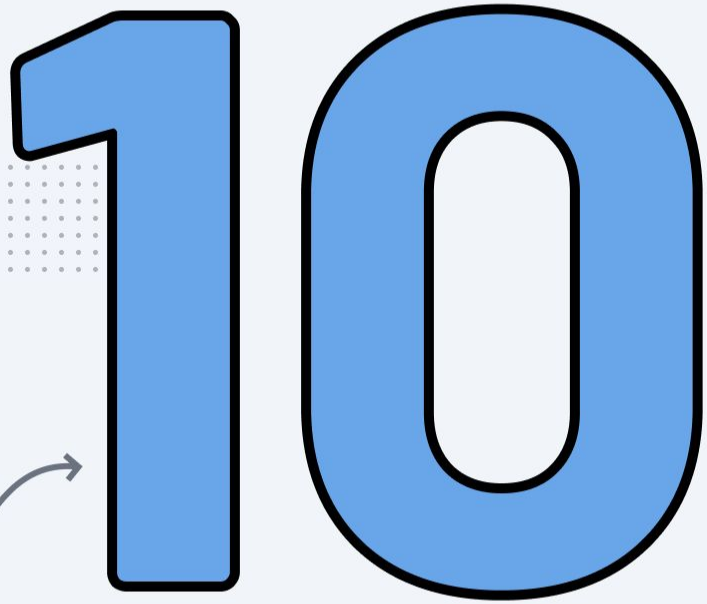
"O que danado é código limpo?"

Definição:

Definição principal: Clean Code ou código limpo se refere a um conjunto de boas práticas na escrita de software que você pode aplicar para **obter** uma **maior legibilidade** e **manutenibilidade** do seu código.

“Um código limpo é um código que foi cuidado por alguém. Alguém que calmamente o manteve simples e organizado.[...] Alguém que se importou”

Michael Feathers



DICAS ARRETADAS DO CLEAN CODE

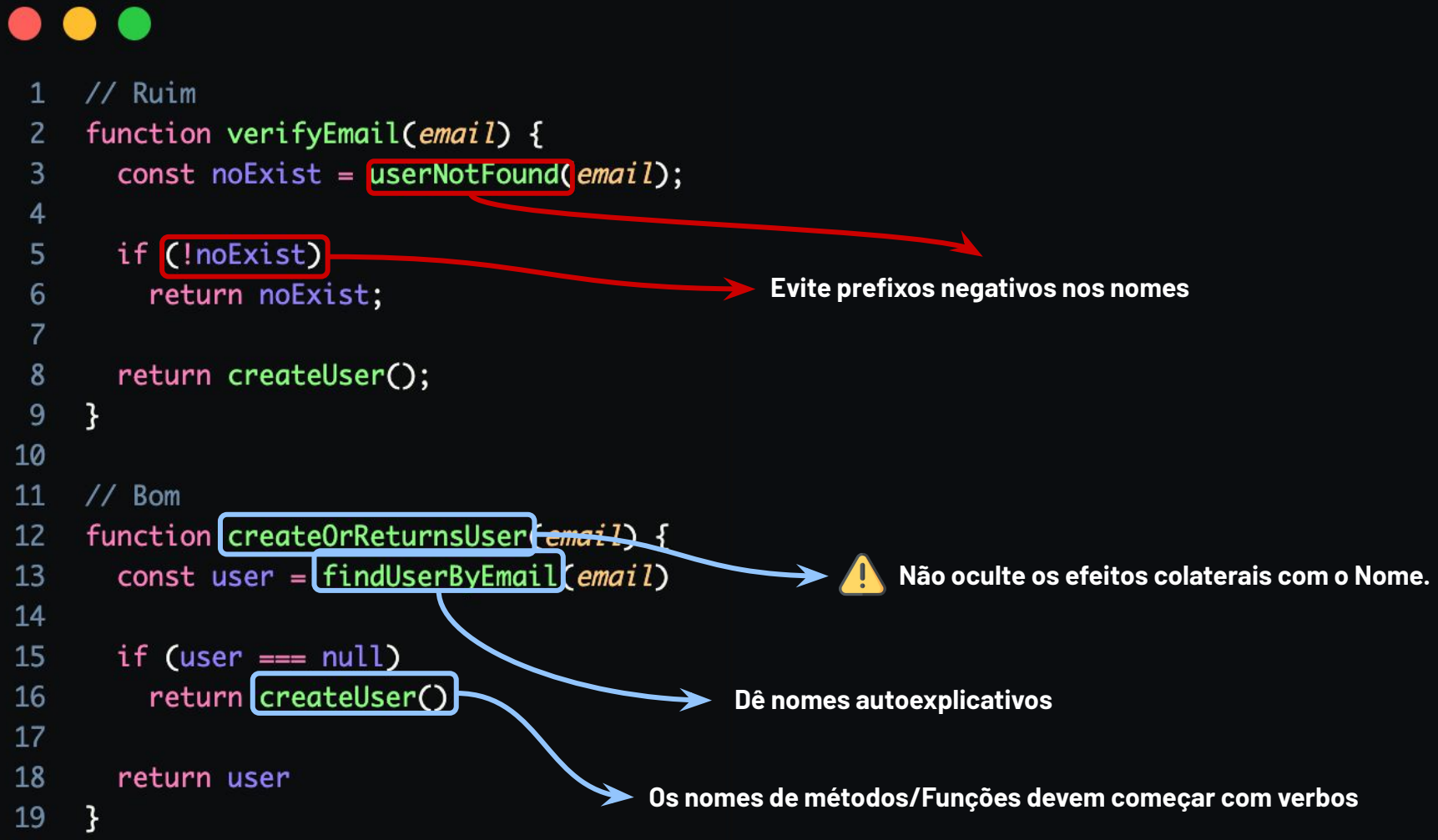
Matheus Fiuza

1. DICA ARRETADA

NOMES

Nomes *em software são* ***90% responsável*** *pela* ***legibilidade do código***

Matheus Pinn


// Ruim
function verifyEmail(email) {
const noExist = userNotFound(email);
if (!noExist)
return noExist;
return createUser();
}
// Bom
function createOrReturnsUser(email) {
const user = findUserByEmail(email)
if (user === null)
return createUser()
return user
}
Evite prefixos negativos nos nomes
Não oculte os efeitos colaterais com o Nome.
Dê nomes autoexplicativos
Os nomes de métodos/Funções devem começar com verbos

2. DICA ARRETADA

FUNÇÕES

As funções devem fazer uma coisa só.

Matheus Fiuza

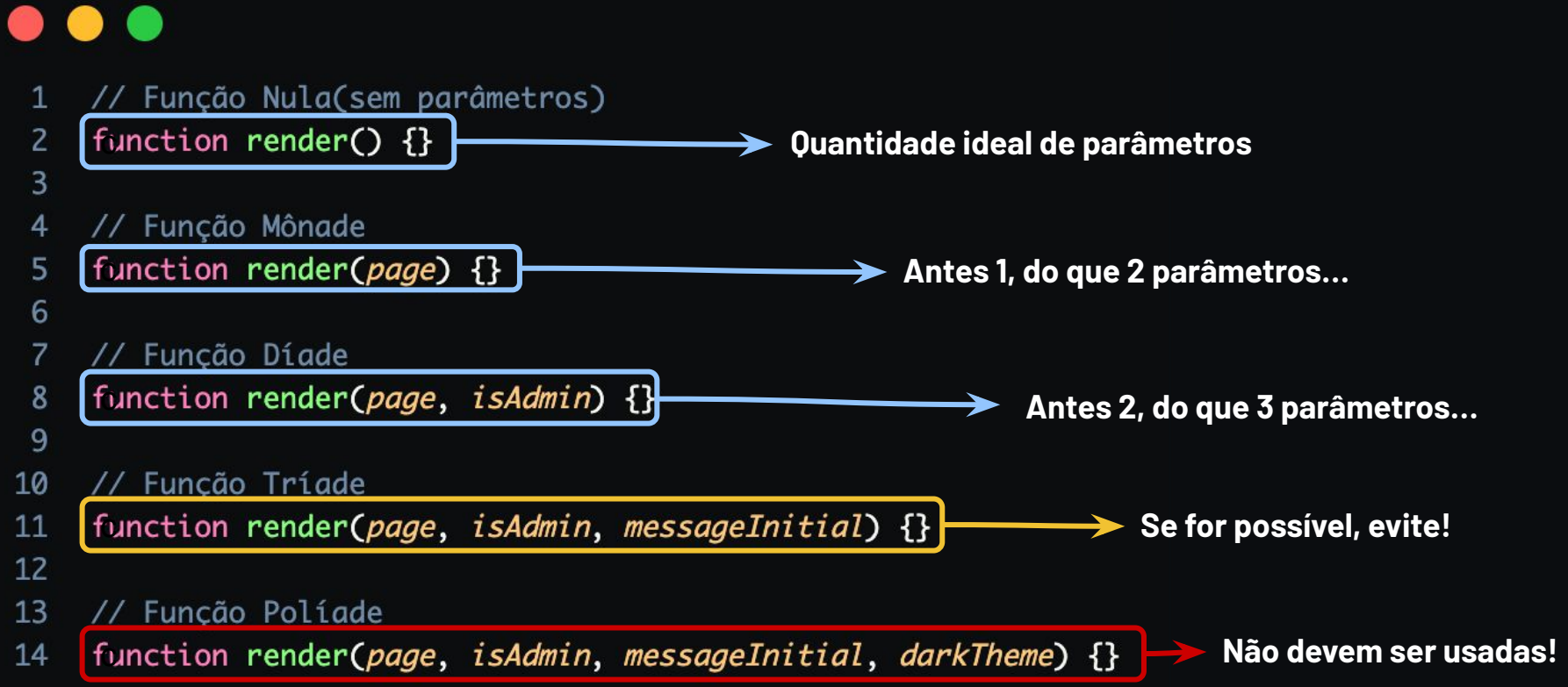

// Bom
function createOrReturnsUser(email) {
const user = findUserByEmail(email)
if (user === null)
return createUser()
return user
}
Crie funções pequenas (máximo 20 linhas)
Blocos dentro de instruções (if, else, while e outros) devem ter apenas uma linha.
Devem fazer uma coisa... apenas ela!!
Devem fazer ou responder algo, mas não ambos. (comando-consulta)

3. DICA ARRETADA

PARÂMETROS

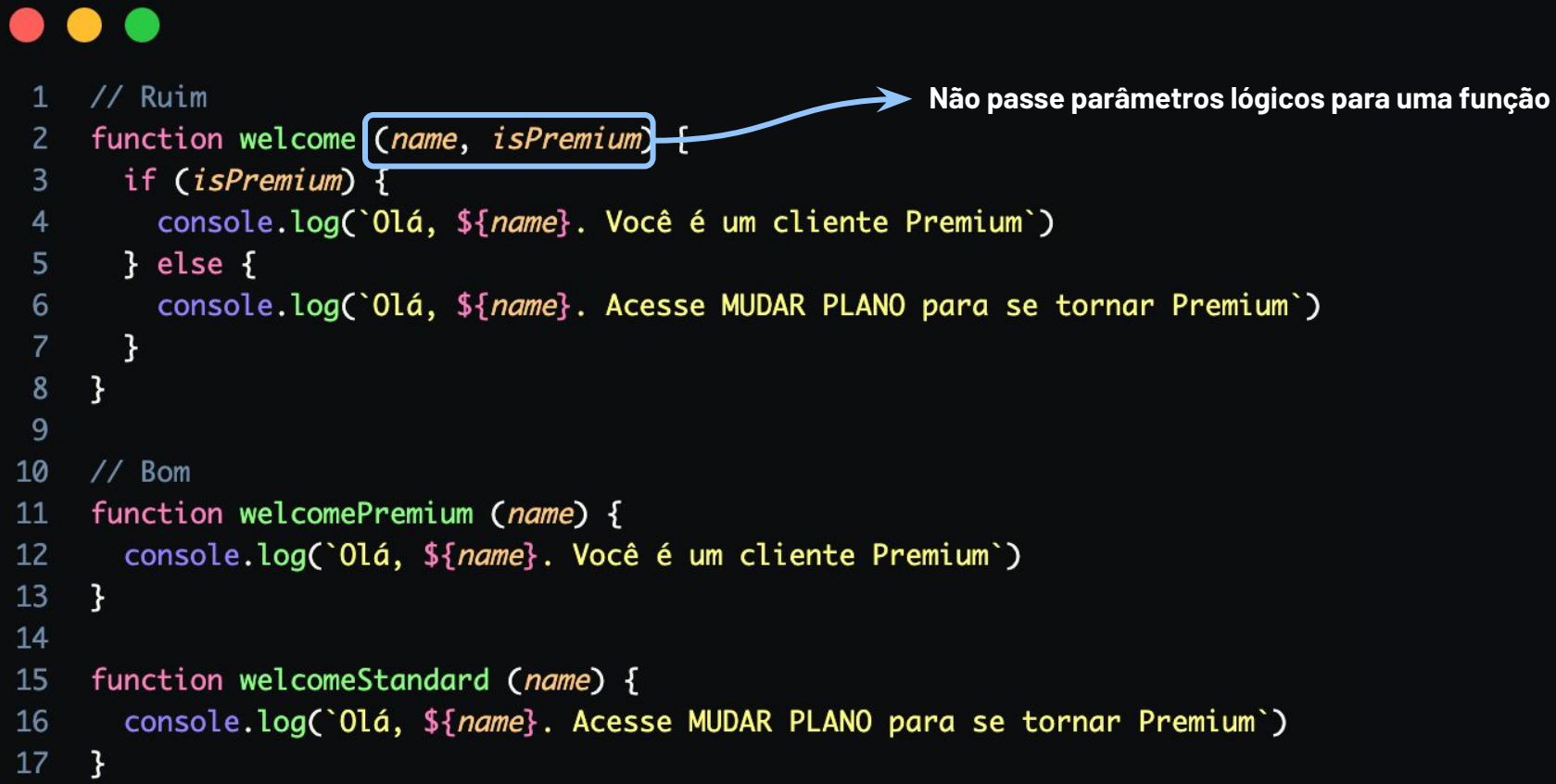
*A **quantidade ideal** de parâmetros*
*para uma função é **ZERO***

Matheus Frura


// Função Nula(sem parâmetros)
function render() {}
Quantidade ideal de parâmetros
// Função Mônade
function render(page) {}
Antes 1, do que 2 parâmetros...
// Função Díade
function render(page, isAdmin) {}
Antes 2, do que 3 parâmetros...
// Função Tríade
function render(page, isAdmin, messageInitial) {}
Se for possível, evite!
// Função Políade
function render(page, isAdmin, messageInitial, darkTheme) {}
Não devem ser usadas!

```
1  // Ruim
2  function calculateAreaM2(width, length) {
3    return width * length;
4  }
5  calculateAreaM2(40, 200)
6
7  // Bom
8  function calculateAreaM2({ width, length }) {
9    return width * length;
10 }
11 calculateAreaM2({
12   width: 40,
13   length: 200
14 })
```

Reduzir o número de parâmetros através da criação de objetos.
(Agrupando parâmetros relacionados)


Não passe parâmetros lógicos para uma função
// Ruim
function welcome (name, isPremium) {
if (isPremium) {
console.log(`Olá, ${name}. Você é um cliente Premium`)
} else {
console.log(`Olá, ${name}. Acesse MUDAR PLANO para se tornar Premium`)
}
}
// Bom
function welcomePremium (name) {
console.log(`Olá, ${name}. Você é um cliente Premium`)
}
function welcomeStandard (name) {
console.log(`Olá, ${name}. Acesse MUDAR PLANO para se tornar Premium`)
}

4. DICA ARRETADA

CONDICIONAIS

Uma das fontes mais comuns *de*
complexidade *em um programa é uma lógica*
condicional complexa

Matheus Fiuza

```
1  // Ruim
2  if (!aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd))
3    charge = quantity * plan.summerRate
4  else
5    charge = quantity * plan.regularRate + plan.regularServiceCharge
6
7  // Bom
8  if (summer())
9    charge = summerCharge()
10 else
11   charge = regularCharge()
12
13 function summer() {
14   return !aDate.isBefore(plan.summerStart) && !aDate.isAfter(plan.summerEnd)
15 }
```

Evite condicionais negativas (mais difícil de entender)

Decomponha as condicionais para torná-las mais fáceis de entender

```
// Ruim
function getPayAmount() {
  let result

  if (isDead) {
    result = deadAmount()
  } else {
    if (isSeparated)
      result = separatedAmount()
    else {
      if (isRetired)
        result = retiredAmount()
      else
        result = normalPayAmount()
    }
  }

  return result
}
```

Evite condicionais aninhadas

```
// Bom
function getPayAmount() {
  if (isDead) return deadAmount()
  if (isSeparated) return separatedAmount()
  if (isRetired) return retiredAmount()

  return normalPayAmount()
}
```

Cláusulas de guarda / retorno antecipado

5. DICA ARRETADA

LAÇOS DE REPETIÇÃO

Dividir um laço pode deixá-lo mais fácil de ser usado

Matheus Pinn

```
// Ruim
let averageAge = 0
let totalSalary = 0

for (const p of people) {
  averageAge += p.age
  totalSalary += p.salary
}

averageAge = averageAge / people.length

// Bom
let averageAge = 0
for (const p of people) {
  averageAge += p.age
}

let totalSalary = 0
for (const p of people) {
  totalSalary += p.salary
}

averageAge = averageAge / people.length
```

Laço fazendo duas tarefas distintas.

Divida o laço

```
// Ruim
const names = []
for (const i of input) {
  if (i.job === "programmer")
    names.push(i.name)
}

// Bom
const names = input
  .filter(i => i.job === "programmer")
  .map(i => i.name)
```

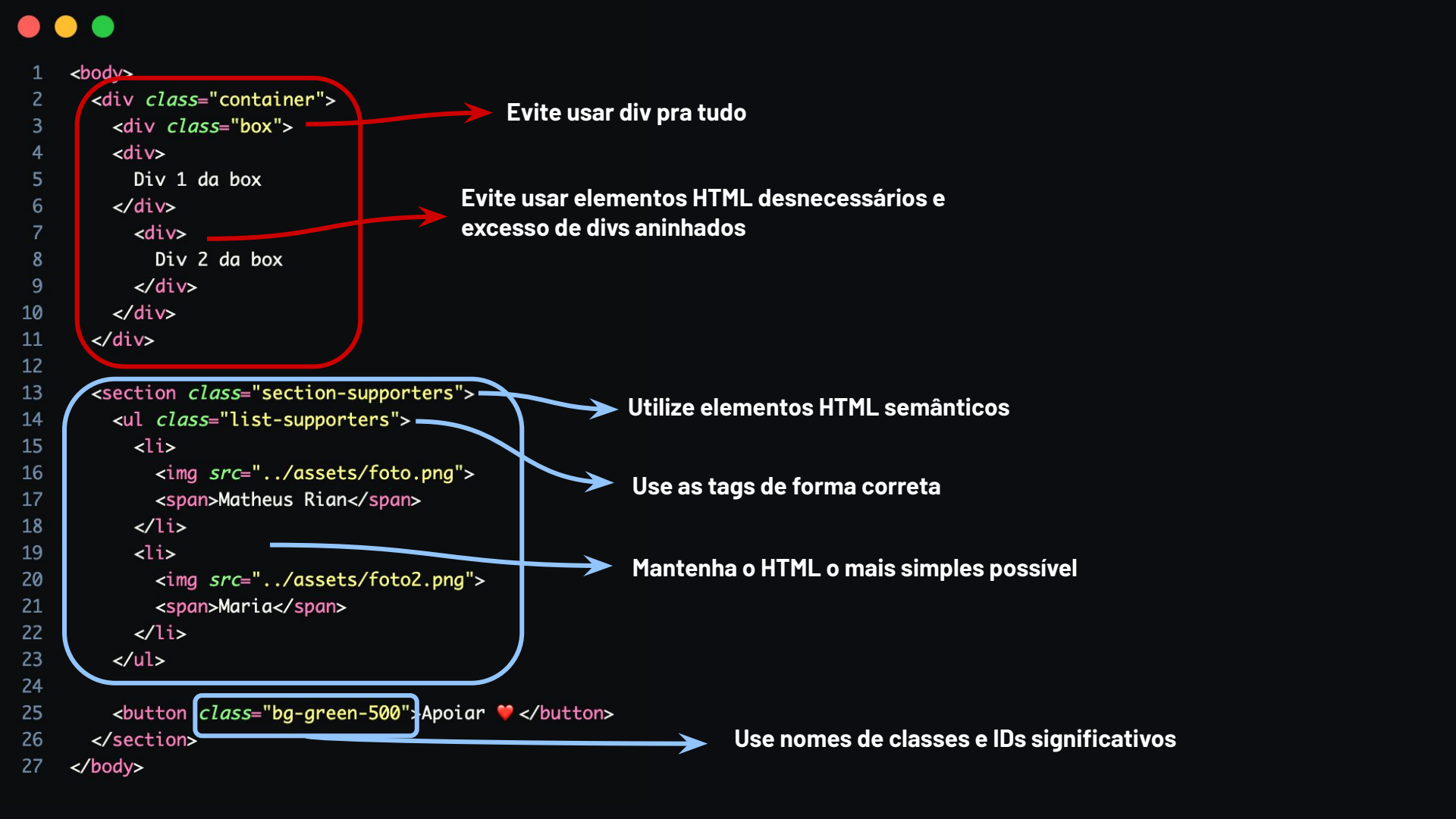
Substituindo laço por pipeline.

6. DICA ARRETADA

CLEAN HTML

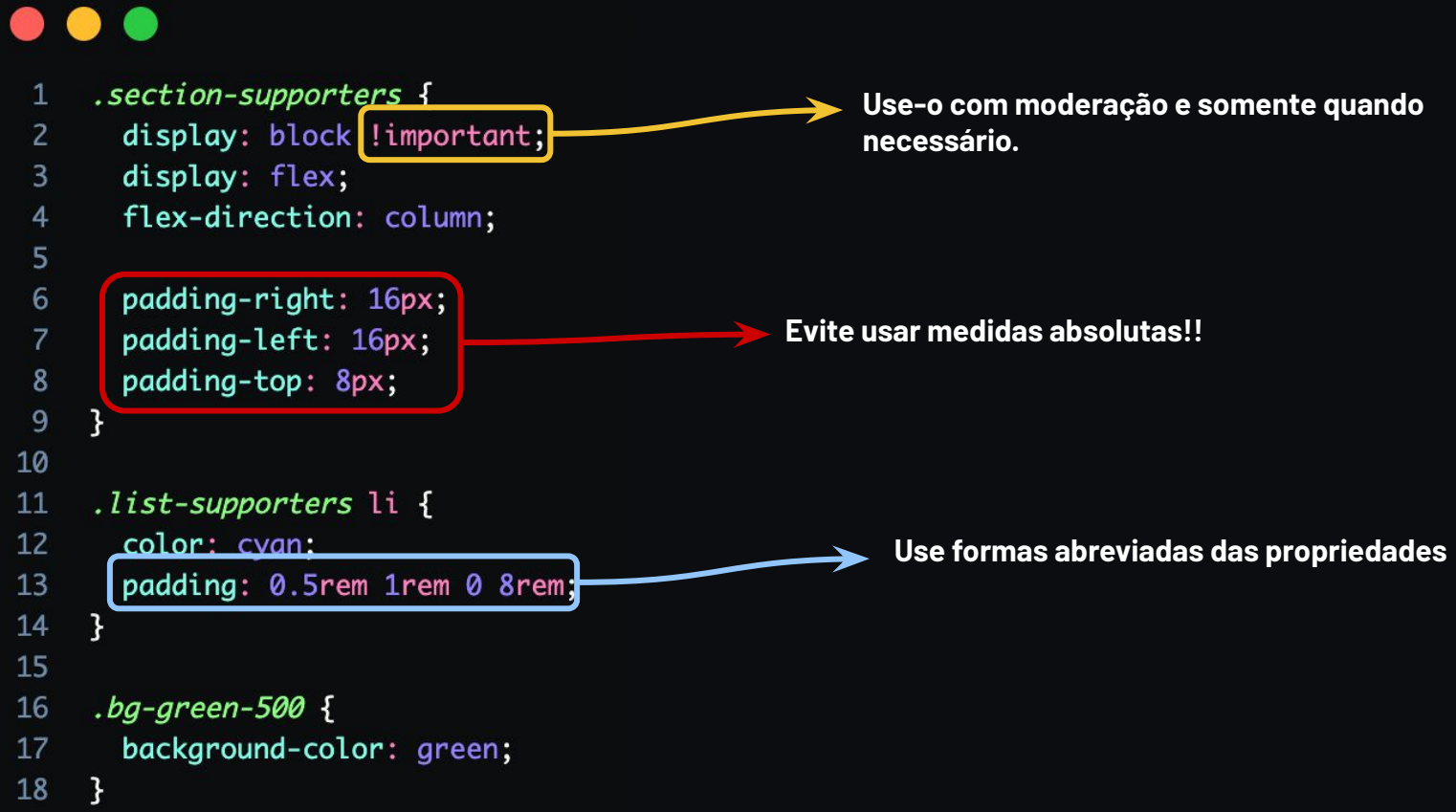
Mantenha o HTML o mais simples possível

Matheus Fiuza


1 <body>
2
3
4
5 Div 1 da box
6
7
8 Div 2 da box
9
10
11
12
13 <section class="section-supporters">
14 <ul class="list-supporters">
15
16 <img src="../assets/foto.png">
17 Matheus Rian
18
19
20 <img src="../assets/foto2.png">
21 Maria
22
23
24
25 <button class="bg-green-500">Apoiar ❤️ </button>
26 </section>
27 </body>
Evite usar div pra tudo
Evite usar elementos HTML desnecessários e excesso de divs aninhados
Utilize elementos HTML semânticos
Use as tags de forma correta
Mantenha o HTML o mais simples possível
Use nomes de classes e IDs significativos

7. DICA ARRETADA

CLEAN CSS


.section-supporters {
display: block !important;
display: flex;
flex-direction: column;
padding-right: 16px;
padding-left: 16px;
padding-top: 8px;
}
.list-supporters li {
color: cyan;
padding: 0.5rem 1rem 0 8rem;
}
.bg-green-500 {
background-color: green;
}
Use-o com moderação e somente quando necessário.
Evite usar medidas absolutas!!
Use formas abreviadas das propriedades

Use pré-processadores

Pré-processadores como Sass ou Less **podem tornar seu código mais modular e fácil de manter**. Eles também **fornecem recursos** como variáveis e mixins que podem **tornar seu código mais eficiente.**

8. DICA ARRETADA

OS 3 PRINCÍPIOS

DRY, KISS e YAGNI

Matheus Fiuza


YAGNI
FAÇA O NECESSÁRIO
DRY
FAÇA UMA VEZ
KISS
FAÇA SIMPLES

9. DICA ARRETADA

REGRA DO ESCOTEIRO

Deixe a área do acampamento mais limpa do que como você a encontrou.

Assim como um escoteiro deve sempre deixar um acampamento melhor do que o encontrou, um **desenvolvedor deve SEMPRE melhorar o código** em que está trabalhando.

10. DICA ARRETADA

OPORTUNIDADE

***REFATORE** COM UM **PROPÓSITO**, evite refatorar apenas por refatorar*

Matheus Puppe

Refatore quando:

- Adicionar novas funcionalidades
- Corrigir um problema
- Precisar entender uma parte do código

Refatore em **pequenos passos** e sempre rodando os testes para garantir que nenhum problema será gerado

Matheus Fiuza

ENQUANTO ISSO, NAS PROFUNDEZAS
da PlayStore/App Store

Me siga no **Linkedin**

Matheus Rian

Desenvolvedor Front-End com mais de **2 anos** de experiência.

accenture

Graduando em Ciência da computação

Matheus Rian