Programação Estruturada

Aula 15 - Tratamento de Erros, Estilo de Código e Boas Práticas de Programação

Videoaula 02: Estilo de Código

Material Didático do Instituto Metrópole Digital - IMD
Versão 5.3 - Todos os Direitos reservados

Videoaula 02: Estilo de Código

Olá, seja bem-vindo de volta! Gostaria de começar esta videoaula perguntando a sua opinião sobre o código que está no slide (Figura 1). O que ele faz? Difícil dizer, não é mesmo?

Figura 1 - Estilo de Código

```
function situacao(entrada1, entrada2, saida) {
var x =
Number(
document.getElementById(entrada1).value
); var
y =
Number(document.getElementById(entrada2).value);
var r=resultado(media(x,y));
document.getElementById(saida).innerHTML
= "O resultado é " + r;}function media(x,y){return (x + y)/2;}
function resultado(m) {
var resultado;
if (m >= 7) {resultado = "Aprovado";} else if (m >= 5) {
resultado = "Recuperação";} else {resultado = "Reprovado";} return resultado;}
```

A verdade é que esse código está uma bagunça. Infelizmente, de vez em quando nos deparamos com códigos como esse. Por isso, precisamos trabalhar juntos para que você possa escrever programas que sigam um determinado estilo de organização a fim de que ele seja o mais limpo e fácil de ser lido.

Veja esta versão do mesmo código fonte.

Código 1 - 15_4 Media.js

```
<!-- 09_5 Media de Notas.html -->
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 09</title>
  </head>
  <body>
```

```
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Situação do Aluno</h1>

    N1: <input type="number" id="N1" value="">
    N2: <input type="number" id="N2" value="">
    <button onclick="situacao('N1', 'N2', 'resultado')">OK</button>
    <p id="resultado"></p>

    <script src="script.js"></script>
  </body>
</html>
```

```
function situacao(entrada1, entrada2, saida) {
  var x = Number(document.getElementById(entrada1).value);
  var y = Number(document.getElementById(entrada2).value);

  var r =  resultado( media(x, y) );

  document.getElementById(saida).innerHTML = "O resultado é " + r;
}

function media(x, y) {
  if (y === undefined) {
    y = 0;
  }
  return (x + y) / 2;
}

function resultado(m) {
  var resultado;

  if (m >= 7) {
    resultado = "Aprovado";
  } else if (m >= 5) {
    resultado = "Recuperação";
  } else {
    resultado = "Reprovado";
  }


  return resultado;
}

/*
function resultado(m) {
  if (!(m < 7)) {
```

```
35        return "Aprovado";
36    } else if (m >= 5) {
37        return "Recuperação";
38    } else {
39        return "Reprovado";
40    }
41 }
42 */
43
```

Muito melhor agora, não é mesmo? Aposto que você finalmente consegue dizer o que esse programa faz. Lembra dele? Usamos esse programa como exemplo em aulas anteriores e nesta estamos disponibilizando-o com o nome **15_4 Media.js**.

Na verdade, tornar tarefas complexas em programas que sejam corretos e legíveis a humanos, está na essência da programação. É por isso que, nessa videoaula, vou apresentar, através de regras de codificação, um estilo de código de minha preferência. Um detalhamento completo dele você poderá encontrar neste QR code (Figura 2).

Figura 2 - Estilo de Código

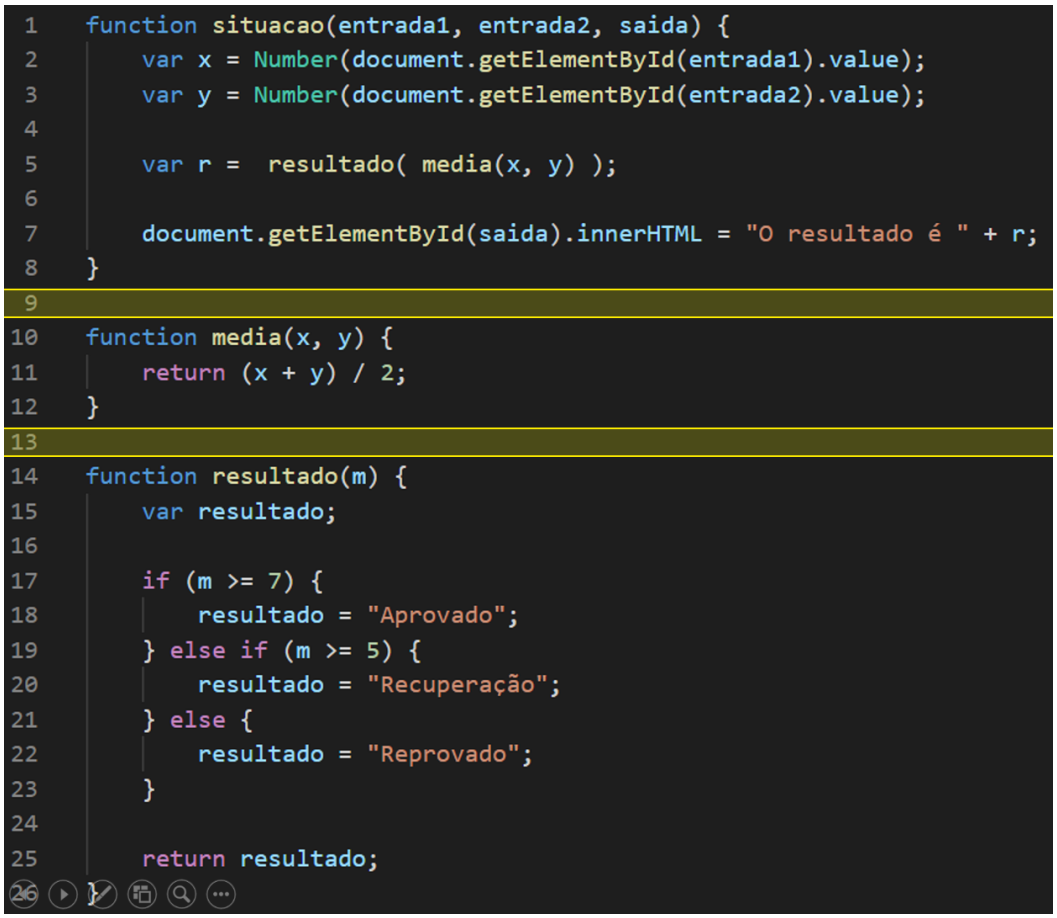
Guias de estilo de código, como esse do QR code acima, contêm regras gerais sobre como escrever o código, por exemplo, que aspas usar, quantos espaços recuar, o comprimento máximo da linha, quando quebrar a linha, entre outros. Este que vou apresentar nesta aula tem apenas 12 regras e dará a você a noção do que um guia de estilo de código deve conter.

É importante frisar que você vai se deparar com estilos de códigos diferentes do que verá aqui. Você pode adotá-los, se desejar. Afinal de contas, estilos de código são preferências, não dogmas religiosos. O importante é que o objetivo seja atingido: um código fácil de ler.

Ao se integrar a uma equipe de desenvolvedores de uma empresa, provavelmente você terá que se adaptar ao estilo de código adotado nela. Afinal de contas, quando todos os membros de uma equipe usam o mesmo estilo de código, existirá na empresa uma uniformidade de estilo de código, independentemente de qual membro da equipe o escreveu. Vamos às regras?

A primeira regra diz que **você deve inserir uma linha em branco entre blocos lógicos**.

Figura 3 - Linha em Branco Entre Funções



No slide (Figura 3), perceba que inserimos as linhas em branco 9 e 13 para separar as funções. No entanto, dentro de uma mesma função, podemos ter blocos lógicos diferentes. Por exemplo, na função `situacao`, temos a declaração das variáveis com os valores de entrada, a declaração da variável de saída com o cálculo de seu valor e a escrita da saída.

Note que, para separar esses três blocos, inserimos as linhas em branco 4 e 6.

Figura 4 - Linha em Branco Entre Trechos da Função

```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

De maneira similar, na função `resultado` inserimos as linhas em branco 16 e 24, para separar a declaração das variáveis, o cálculo do retorno e o retorno propriamente dito.

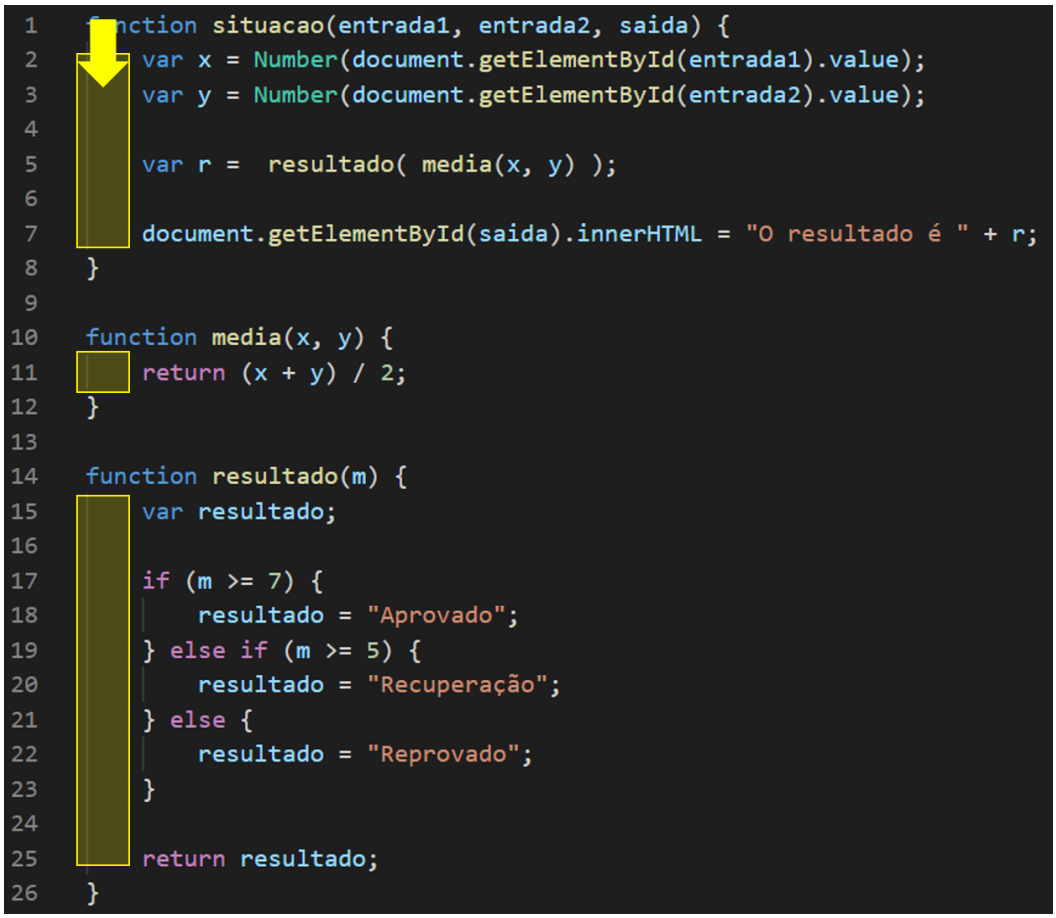
Figura 5 - Linha em Branco Entre Trechos da Função

```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

A regra 2 também é extremamente importante e diz que **você deve recuar as linhas de um bloco de código com relação à declaração do bloco**. Esse recuo, também chamado de indentação, pode ser de dois espaços em branco, de quatro espaços em branco ou de um *tab*. O importante é escolher um espaçamento e usá-lo ao longo de todo o seu código. Eu, particularmente, prefiro o *tab* porque digitamos apenas uma vez para obter o espaçamento, mas admito que será muito fácil encontrar alguém com a opinião diferente da minha 😉.

No slide, note que todo corpo de função tem um espaçamento, no meu caso um *tab*, com relação à declaração das funções. Por exemplo, as linhas 2 a 7 têm um espaçamento com relação à linha 1, que declara a função `situacao`. O mesmo acontece com as declarações das outras duas funções.

Figura 6 - Identação



Note, porém, que isso também acontece nos blocos de código do `if-else` nas linhas 18, 20 e 22, pois elas fazem parte dos blocos de código de cada uma das condições do `if-else`. Mais espaçamento pode ser adicionado caso tenhamos novos blocos dentro desses blocos do `if-else`.

Figura 7 - Identação no `if-else`

```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

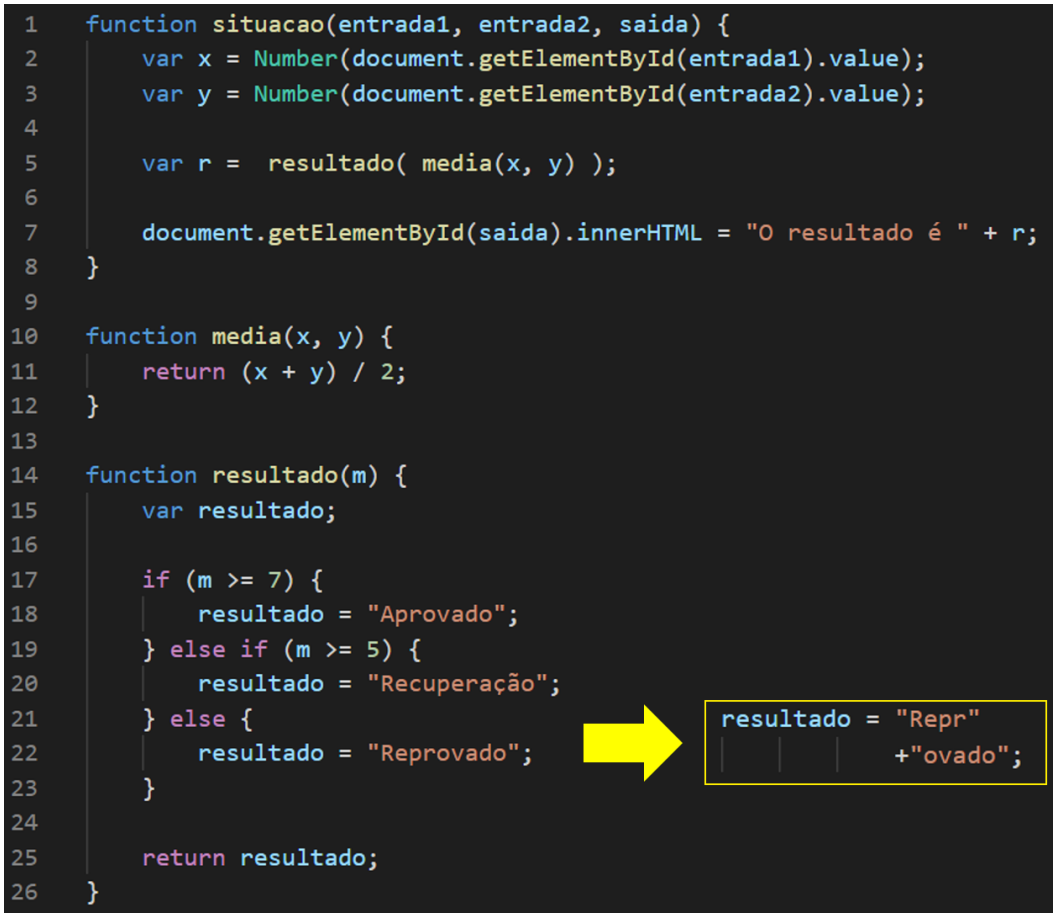
A próxima regra é bastante simples: **sempre use o ponto-e-vírgula para concluir comandos**. Como sabemos, o seu uso é opcional em JavaScript, mas vimos que isso pode causar problemas de interpretação como, por exemplo, no retorno de funções. Portanto, sempre use o ponto-e-vírgula para concluir comandos. Observe que no nosso código essa regra está sendo obedecida.

A regra 4 também é bastante simples: **não escreva linhas muito longas**. O comprimento máximo da linha é, obviamente, uma definição da equipe de programação, mas geralmente, esse valor é entre 80 e 120 caracteres.

No nosso exemplo, a linha mais longa é a linha 7, com 68 caracteres. No caso de quebra de linhas de *strings*, lembre-se que o JavaScript não permite a quebra de linhas dentro de *strings* se usarmos aspas duplas ou simples. Porém, se você usar a crase (`),

essa quebra é permitida, mas um espaço em branco é inserido à *string*. Por isso, prefiro usar a concatenação para permitir a quebra de linha em *strings*, como exemplifico na quebra da *string* `"Reprovado"` no slide (Figura 8).

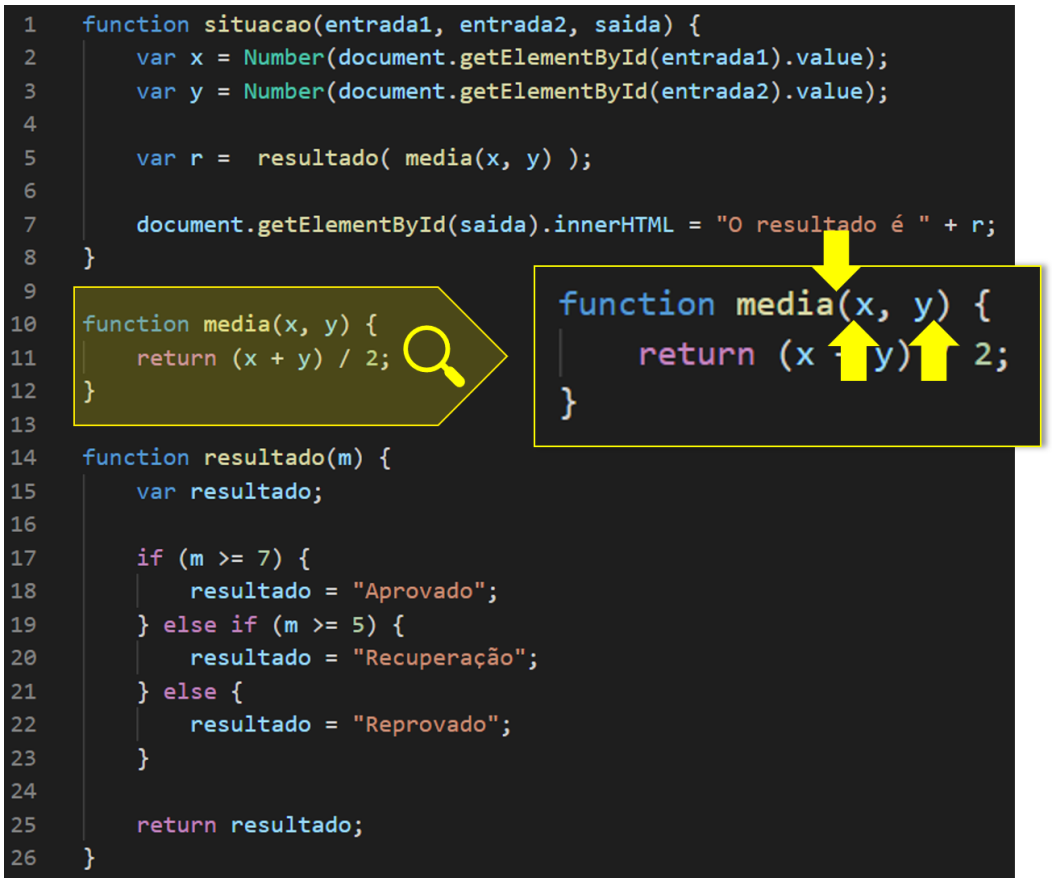
Figura 8 - Quebrando Linhas Dentro de Strings



A quinta regra nos diz que **não devemos inserir espaço em branco entre o nome das funções e os parênteses e entre os parênteses e os parâmetros**.

Por exemplo, veja no slide (Figura 9) a declaração da função `media` em destaque. Observe que não temos espaços em branco entre o nome da função e o parêntese da esquerda. Da mesma forma, não temos espaços em branco entre esse parêntese e o `x` e entre o `y` e o parêntese da direita. Observe que essa regra também é respeitada nas declarações das outras duas funções.

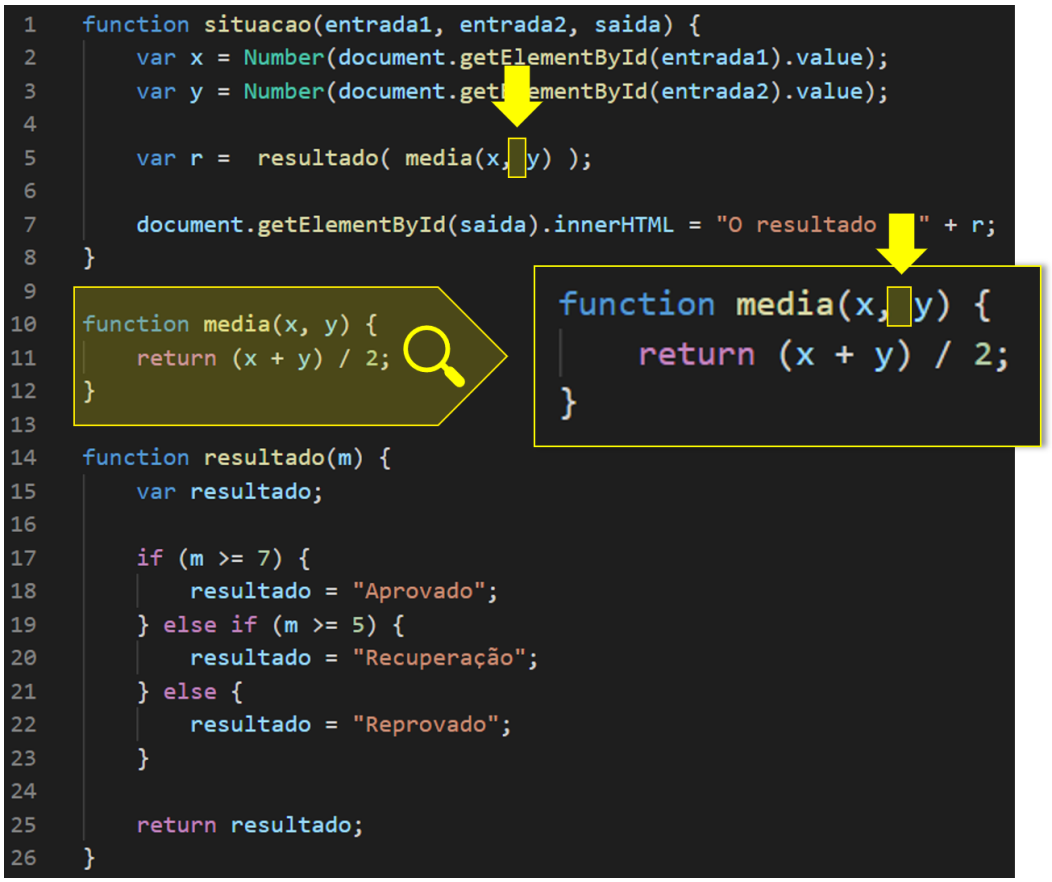
Figura 9 - Espaço em Branco e Argumentos de Função



Por outro lado, a regra 6 nos diz que **devemos inserir espaço em branco entre os parâmetros tanto da declaração das funções quanto nas chamadas delas**.

Por exemplo, veja no slide (Figura 10) a declaração da função `media` em destaque. Observe que temos um espaço em branco entre a vírgula que temos depois do `x` e o `y`. Observe também que, na chamada dessa função na linha 5, isso também acontece. Essa regra é respeitada em todas as declarações e chamadas de função que temos nesse exemplo.

Figura 10 - Espaço em Branco entre Argumentos de Função



A regra 7 determina a **inserção de espaço em branco depois de palavras-chave** como `if`, `else`, `for`, `do`, `while`, `switch` e antes de palavras-chave que dão continuidade a esses blocos, como no caso do `else` e do `while` no laço `do-while`.

Por exemplo, observe no slide (Figura 11) os espaços em branco após o `if` nas linhas 17 e 19, após o `else` na linha 21 e antes do `else` nas linhas 19 e 21.

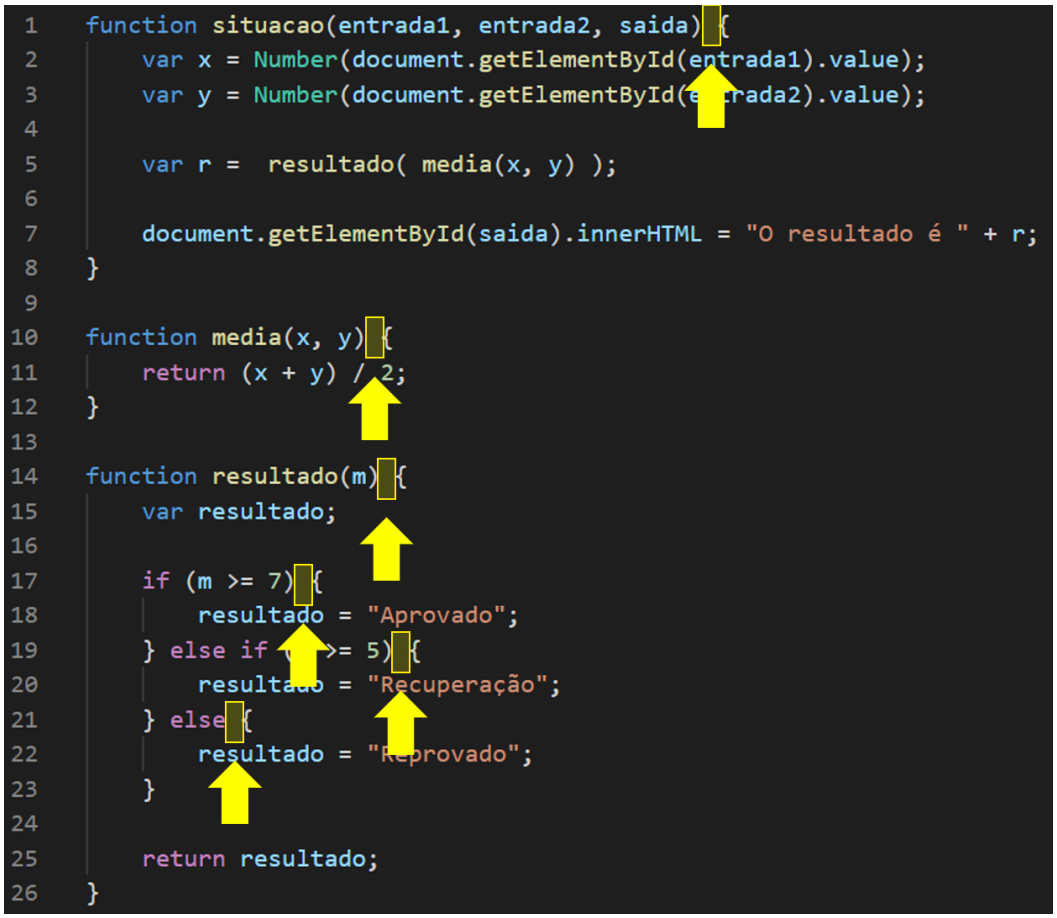
Figura 11 - Espaço em Branco e Palavras-chave

```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

A oitava regra nos diz que **chaves de abertura de blocos devem ser colocadas na mesma linha da declaração do bloco e após um espaço em branco**.

Isto se aplica tanto a blocos de funções, como no caso das declarações das funções `situacao`, `media` e `resultado`, nas linhas 1, 10 e 14, respectivamente, quanto na abertura de blocos de fluxo de controle como, por exemplo, as chaves que abrem os blocos no comando `if-else` do slide (Figura 12) nas linhas 17, 19 e 21.

Figura 12 - Chaves em Aberturas de Blocos

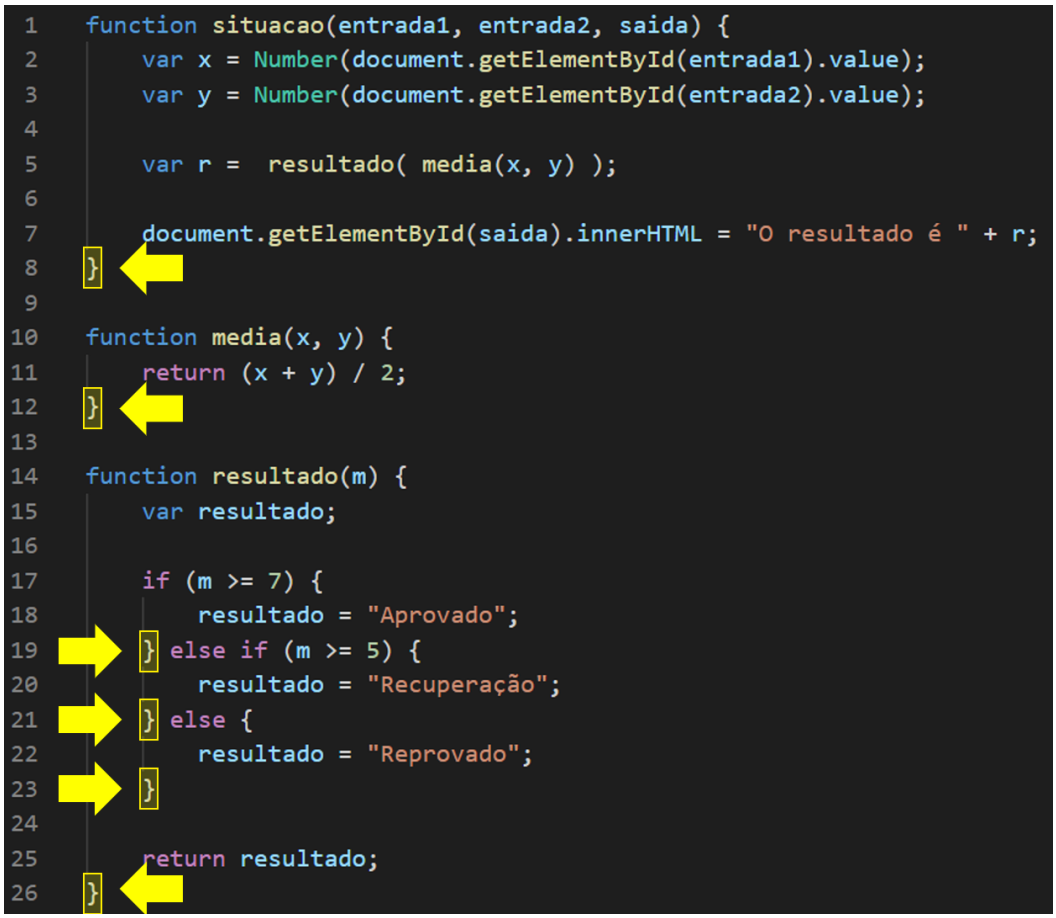


```
1  function situacao(entrada1, entrada2, saida) {
2      var x = Number(document.getElementById(entrada1).value);
3      var y = Number(document.getElementById(entrada2).value);
4
5      var r =  resultado( media(x, y) );
6
7      document.getElementById(saida).innerHTML = "O resultado é " + r;
8  }
9
10 function media(x, y) {
11     return (x + y) / 2;
12 }
13
14 function resultado(m) {
15     var resultado;
16
17     if (m >= 7) {
18         resultado = "Aprovado";
19     } else if (m >= 5) {
20         resultado = "Recuperação";
21     } else {
22         resultado = "Reprovado";
23     }
24
25     return resultado;
26 }
```

Por outro lado, a próxima regra nos diz que **chaves de fechamento de blocos devem ser colocadas na linha seguinte ao final do bloco e alinhadas com a declaração do bloco**.

Isto se aplica tanto a blocos de funções, como no caso do final das funções `situacao`, `media` e `resultado`, nas linhas 8, 12 e 26, respectivamente, quanto no fechamento de blocos de fluxo de controle, como, por exemplo, as chaves que fecham os blocos nos comandos `if-else` do slide (Figura 13), nas linhas 19, 21 e 23.

Figura 13 - Chaves em Fechamento de Blocos



```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

De acordo com a regra 10, **você não deve quebrar linha quando for usar `else`, `else if`, ou o `while` do laço `do-while`**.

No nosso exemplo, observe que isto é o que fizemos nas linhas 19 e 21.

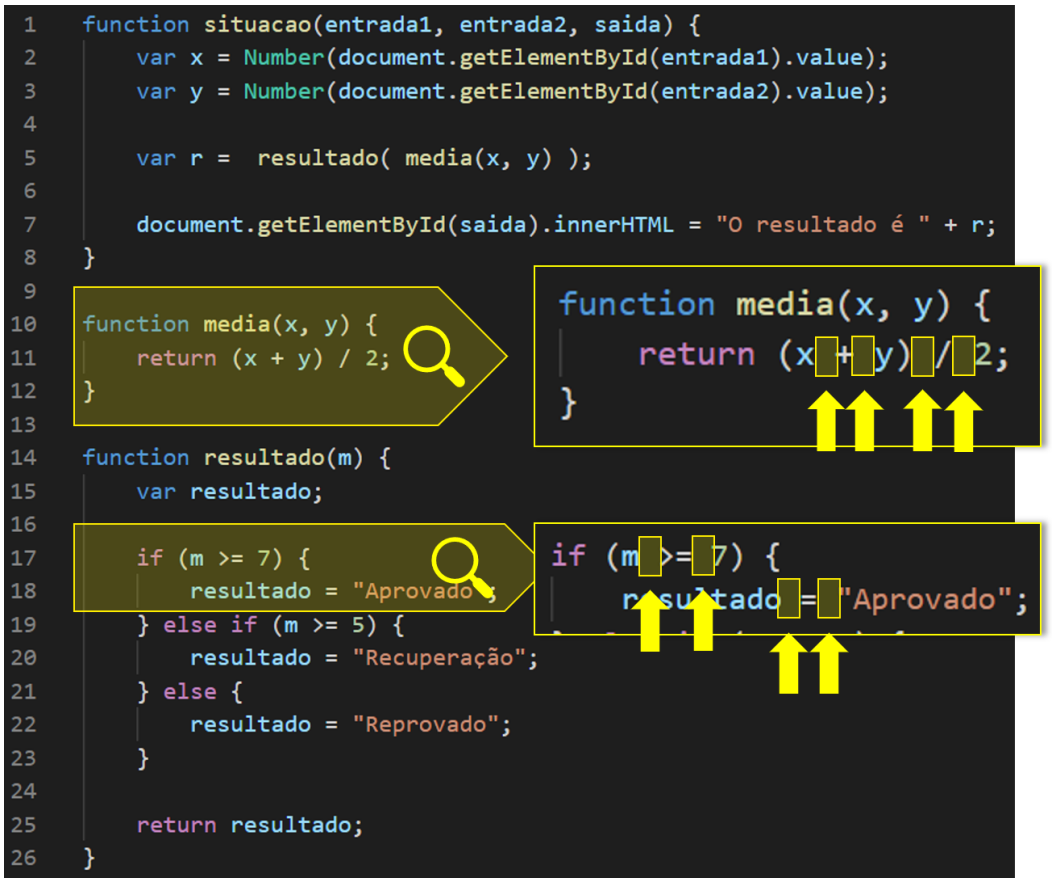
Figura 14 - Quebras de Linha e Palavras-chave

```
1   function situacao(entrada1, entrada2, saida) {
2       var x = Number(document.getElementById(entrada1).value);
3       var y = Number(document.getElementById(entrada2).value);
4
5       var r =  resultado( media(x, y) );
6
7       document.getElementById(saida).innerHTML = "O resultado é " + r;
8   }
9
10  function media(x, y) {
11      return (x + y) / 2;
12  }
13
14  function resultado(m) {
15      var resultado;
16
17      if (m >= 7) {
18          resultado = "Aprovado";
19      } else if (m >= 5) {
20          resultado = "Recuperação";
21      } else {
22          resultado = "Reprovado";
23      }
24
25      return resultado;
26  }
```

A regra 11 nos **sugere colocar um espaço em branco antes e depois dos operadores**.

No slide (Figura 15), observe os trechos dos códigos destacados.

Figura 15 - Espaços em Branco e Operadores



Você notou que inserimos espaços em branco antes e depois dos operadores de soma e divisão no retorno da média? Além disso, também inserimos espaços em branco antes e depois das comparações feitas nas condições do bloco `if-else` como, por exemplo, antes e depois do maior ou igual no trecho destacado do slide. Veja também que fazemos o mesmo antes e depois do operador de atribuição nas linhas 18, 20 e 22. De fato, isso foi feito em todo o código do exemplo. Observe!

A última regra que será apresentada nesta videoaula é **a de colocar espaço em branco antes e depois de chamadas aninhadas de funções**.

Neste exemplo, temos uma chamada de função aninhada na linha 5, onde a chamada à função `media` acontece dentro da chamada à função `resultado`. Nesse caso, note que inserimos espaços em branco antes e depois da chamada à função `media` a fim de separá-la dos parênteses usados na chamada da função `resultado`.

Figura 16 - Espaços em Branco e Chamadas Aninhadas de Funções

Antes de concluir esta videoaula, gostaria de comentar dois pontos importantes com você. Primeiramente, lembra que no início falei que você encontrará estilos de códigos diferentes do que acabamos de ver? Alguns estilos conhecidos e muito utilizados são:

- O Guia de estilo de código JavaScript do Google
- O Guia de estilo de código JavaScript do Airbnb
- O Idiomatic.JS e
- O StandardJS

Acesse os QR codes que estão no slide (Figura 17) para conhecer cada um deles.

Figura 17 - Guias de Estilo de Código

Por fim, é importante que você saiba que existem diversas ferramentas que podem verificar automaticamente o estilo do seu código e fazer sugestões de melhoria. Elas são chamadas de *Linters*. O bacana dessas ferramentas é que elas também podem encontrar alguns erros como, por exemplo, erros de digitação em nomes de variáveis ou funções. Recomendo bastante que você utilize um *Linter*.

A ferramenta de edição que usamos nesta disciplina, a *Visual Studio Code*, não possui um *Linter* JavaScript embutido. Porém, existem muitas extensões de *Linter* JavaScript disponíveis no *VS Marketplace*, a loja de extensões do *Visual Studio Code*. Por exemplo, lá você poderá encontrar o *ES Lint* (provavelmente, o *Lint* mais recente de JavaScript), o *JS Hint* e o *Standard JS*. Acesse o QR code do slide (Figura 18) para buscar essas e outras ferramentas.

Figura 18 - `Linters`

Até a próxima videoaula!!! Tchau!