Programação Estruturada

Aula 12 - Arrays: Ordenação, Iteração e Matrizes

Videoaula 02: Iteração sobre Arrays

Videoaula 02: Iteração sobre Arrays

Nesta videoaula, você conhecerá vários métodos de iteração sobre arrays. Esses métodos permitem operar sobre todos os itens dos arrays. Todos eles recebem como um dos argumentos o que chamamos de métodos de *callback*, os quais são utilizados durante a iteração sendo executados em algum momento. Serão apresentados:

1. Métodos para fazer alguma ação com cada um dos elementos do array, como o método `forEach`;
2. Métodos que retornam um novo array baseado no array original, como os métodos `map` e `filter`;
3. Métodos que retornam um único valor baseado nos valores do array original, como os métodos `reduce` e `reduceRight`;
4. Métodos que verificam se os elementos satisfazem determinadas condições, como os métodos `find`, `findIndex`, `every` e `some`

Vamos nessa?

Inicialmente , você conhecerá o método `forEach` ("para cada" em inglês). Esse método chama a função de *callback* passada como argumento uma vez para cada elemento do array. Veja o exemplo.

Figura 1 - `forEach`

```
1   var cores = ["Verde","Amarelo","Azul","Branco"];
2   var resposta = "";
3
4   cores.forEach(adicionar);
5
6   function adicionar(valor, indice, array) {
7     resposta = resposta + "A cor no índice " + indice;
8     resposta = resposta + " é " + valor + "<br>";
9   }
10
11  document.getElementById("resultado").innerHTML = resposta;
```

12_6 ForEach.html e 12_6 ForEach.js

A cor no índice 0 é Verde
A cor no índice 1 é Amarelo
A cor no índice 2 é Azul
A cor no índice 3 é Branco

Nele, adicionamos à resposta que será exibida na tela, uma linha para cada elemento do array de cores. Para isso, na linha 4, invocamos o método `cores.forEach`, passando a função `adicionar` como argumento. Observe que essa função usa três argumentos: o valor do item, o índice do item e o próprio array. Isto faz com que possamos usar esses valores dentro da própria função. No nosso exemplo, usamos o valor do índice na linha 7 e o valor do item na linha 8. Dessa forma, para cada índice `i` e valor `v` desse índice no array, escreveremos a linha "A cor no índice i é v", como podemos ver na saída de execução do programa (Figura 1).

Caso a função de *callback* utilize apenas o valor do item, ela pode ser definida com apenas um argumento, o valor, como podemos ver neste outro exemplo (Figura 2).

Figura 2 - `forEach`

```
var cores = ["Verde","Amarelo","Azul","Branco"];
var resposta = "";

cores.forEach(adicionar);

function adicionar(valor) {
  resposta = resposta + "A cor é " + valor + "<br>";
}

document.getElementById("resultado").innerHTML = resposta;
```

12_6 ForEach.html e 12_6 ForEach.js

A cor é Verde
A cor é Amarelo
A cor é Azul
A cor é Branco

Note que agora a função `adicionar` não utiliza mais o índice. Por esse motivo, podemos defini-la apenas usando o argumento `valor`.

Código 1 - 12_6 ForEach.html e 12_6 ForEach.js

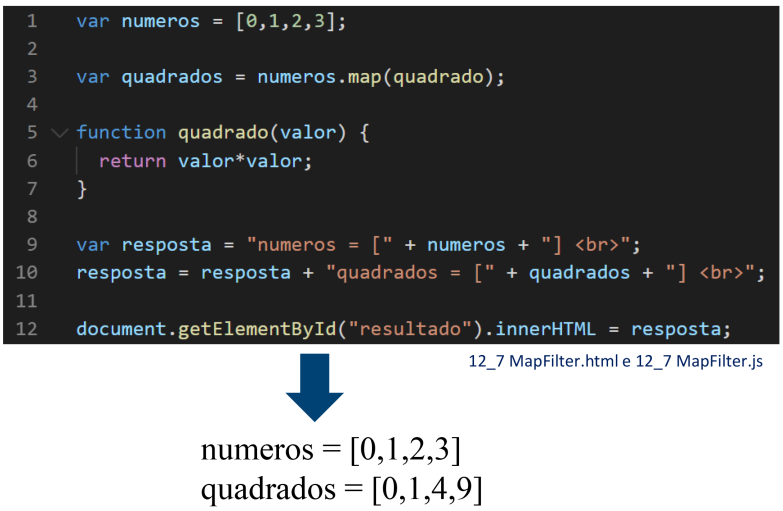
```
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 12</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>ForEach</h1>

    <p id="resultado"></p>

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

```

```
var cores = ["Verde","Amarelo","Azul","Branco"];
var resposta = "";

cores.forEach(adicionar);

function adicionar(valor) {
```

```
  resposta = resposta + "A cor é " + valor + "<br>";
}

document.getElementById("resultado").innerHTML = resposta;

/*
function adicionar(valor, indice, array) {
  resposta = resposta + "A cor no índice " + indice;
  resposta = resposta + " é " + valor + "<br>";
}
*/

```

Agora, você conhecerá os métodos `map` e `filter`, os quais retornam um novo array baseado no array original. O método `map` ("mapear" em inglês) cria um novo array executando a função de *callback* passada a cada elemento do array original, sem alterá-lo. Assim como no método `forEach`, os métodos de *callback* utilizados no `map` também podem ter mais dois argumentos, o índice e o array. Para simplificar o nosso exemplo, porém, não utilizaremos esses argumentos e, por isso, a nossa função `quadrado` possui apenas um argumento, o valor do item do array.

Figura 3 - `map`



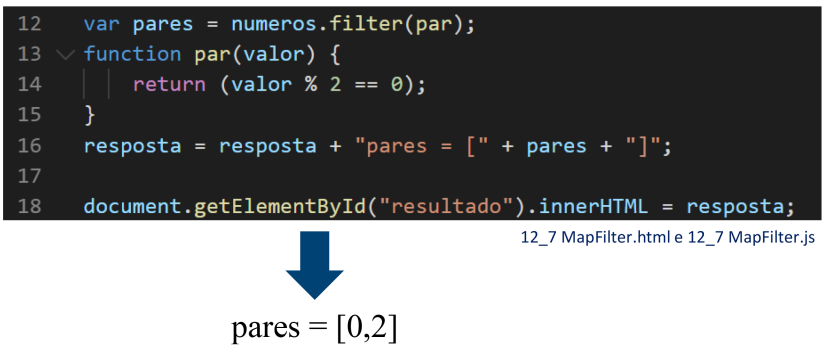
Essa função simplesmente retorna, na linha 6, o valor do item elevado ao quadrado. Dessa forma, ao utilizarmos função `quadrado` como argumento de `numeros.map`, na linha 3, o valor de cada elemento do array `quadrados` será

exatamente igual ao valor do item correspondente no array `numeros` elevado ao quadrado.

Assim, temos que: `quadrados[0]` será igual a `numeros[0]` elevado ao quadrado, ou seja, `0`; `quadrados[1]` será igual a `numeros[1]` elevado ao quadrado, ou seja, 1; `quadrados[2]` será igual a `numeros[2]` elevado ao quadrado, ou seja, `4`; e `quadrados[3]` será igual a `numeros[3]` elevado ao quadrado, ou seja, `9`. Note que o array `numeros` permanece inalterado.

Chegou a hora de conhecer o método `filter` ("filtrar" em inglês). Esse método retorna um novo array contendo apenas elementos que satisfaçam uma determinada condição ou, em outras palavras, que passem em um teste. Para isso, ela utiliza uma função de *callback* que retorna um valor booleano para cada valor do array. A função `filter` filtra o array e mantém apenas índices com retorno verdadeiro. Veja o exemplo (Figura 4).

Figura 4 - `filter`



Nele, temos a definição da função `par` entre as linhas 13 e 15. Esta retorna `true` apenas se o resto da divisão de `valor` por 2 for 0, ou seja, se o valor for par. Dessa forma, se passarmos essa função de *callback* para o método `filter`, o array retornado conterá apenas elementos de `numeros` que forem par. No nosso exemplo, o array `pares` terá os elementos 0 e 2 apenas. Isto porque os elementos 1 e 3 são ímpares e, portanto, não são inseridos no array retornado ao aplicarmos o filtro. Note que o array `numeros` permanece inalterado.

Código 2 - 12_7 MapFilter.html e 12_7 MapFilter.js

```html
<html >
   <head>
      <meta charset="UTF-8" />
      <title>Programação Estruturada - Aula 12</title>
   </head>
   <body>
      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

      <h1>Map e Filter</h1>

      <p id="resultado"></p>

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

```

```javascript
var numeros = [0,1,2,3];

var quadrados = numeros.map(quadrado);

function quadrado(valor) {
  return valor*valor;
}

var resposta = "numeros = [" + numeros + "] <br>";
resposta = resposta + "quadrados = [" + quadrados + "] <br>";

var pares = numeros.filter(par);
function par(valor) {
    return (valor % 2 == 0);
}
resposta = resposta + "pares = [" + pares + "]";

document.getElementById("resultado").innerHTML = resposta;

```

Os métodos `reduce` ("reduzir" em inglês) e `reduceRight` ("reduzir da direita" em inglês) executam uma função em cada elemento da matriz para produzir um único valor. A diferença entre eles é que o primeiro faz essa aplicação da esquerda para a direita e o segundo faz essa aplicação da direita para esquerda. Assim como os métodos anteriores, `reduce` e `reduceRight` também não alteram o array original. Uma diferença importante entre eles e os que vimos até então é que a função de

callback tem quatro argumentos: além do valor do item, do índice do item e do próprio array, a função também recebe um acumulador. Esse acumulador inicialmente recebe um valor inicial passado na chamada do método `reduce` e `reduceRight`, e depois fica recebendo o valor retornado na aplicação da função ao elemento anterior do array.

Atenção

É importante observar que caso o valor inicial do acumulador não seja passado na chamada do método `reduce` e `reduceRight`, ele receberá o valor do primeiro elemento do array. Por isso, chamar `reduce` em um array vazio sem valor inicial para o acumulador, é um erro. O mais seguro, portanto, é fornecer um valor inicial. Existem outras situações peculiares que acontecem com arrays vazios e de tamanho 1, dependendo do fornecimento ou não do valor inicial do acumulador. Veja agora exemplos práticos de uso de `reduce` e `reduceRight`, além dessas situações peculiares.

Código 3 - 12_8 Reduce.html

```html
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 12</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Reduce e ReduceRight</h1>

    <p id="resultado"></p>

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

```

```
var resposta = "";

// Funções de CallBack
```

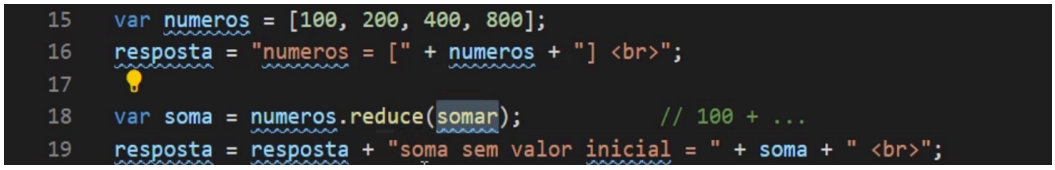
```
4
5  function somar(acumulador, valor) {
6      return acumulador + valor;
7  }
8
9  function subtrair(acumulador, valor) {
10     return acumulador - valor;
11 }
12
13 // Exemplos simples em arrays com mais de um elemento
14
15 var numeros = [100, 200, 400, 800];
16 resposta = "numeros = [" + numeros + "] <br>";
17
18 var soma = numeros.reduce(somar);          // 100 + ...
19 resposta = resposta + "soma sem valor inicial = " + soma + " <br>";
20
21 var somaAC = numeros.reduce(somar, 10000);  // 10000 + 100 +
22 resposta = resposta + "soma com valor inicial = " + somaAC + " <br>";
23
24 var subR = numeros.reduce(subtrair);      // 100 - ...
25 resposta = resposta + "subtracao (reduce) = " + subR + " <br>";
26
27 var subRR = numeros.reduceRight(subtrair); // 800 - ...
28 resposta = resposta + "subtracao (reduceRight) = " + subRR + " <br>";
29
30 // Situações Especiais
31
32 // (1) Chamando reduce em array vazio SEM valor inicial (ERRO)
33 //var numeros = [];
34 //var soma = numeros.reduce(somar);
35
36 // (2) Chamando reduce em array vazio COM valor inicial
37 //var numeros = [];
38 //var soma = numeros.reduce(somar, 10000);
39 //resposta = resposta + "Soma com valor inicial = " + soma + " <br>";
40
41 // (3) Chamando reduce em array de tamanho = 1 COM valor inicial
42 //var numeros = [100];
43 //var soma = numeros.reduce(somar, 10000);
44 //resposta = resposta + "Soma com valor inicial = " + soma + " <br>";
45
46 // (4) Chamando reduce em array de tamanho = 1 SEM valor inicial
47 //var numeros = [100];
48 //var soma = numeros.reduce(somar);
49 //resposta = resposta + "Soma com valor inicial = " + soma + " <br>";
50
51 document.getElementById("resultado").innerHTML = resposta;
```

Nesse exemplo, o HTML é extremamente simples, só tem o campo para escrever, e o interessante vai ser realmente conhecer a utilização dos métodos `reduce` e `reduceRight`.

No JavaScript, temos a variável `resposta`, e vou declarar duas funções de *callback*. A primeira delas é a função `somar` que tem um acumulador e um valor, e o que ele faz é simplesmente somar o valor do acumulador com o valor que foi passado, e é isso que ele retorna. A segunda é a função `subtrair`, que também recebe o acumulador e o valor e retorna a subtração do acumulador pelo valor. Usando essas funções, vamos agora a alguns exemplos de utilização dos métodos `reduce` e `reduceRight`.

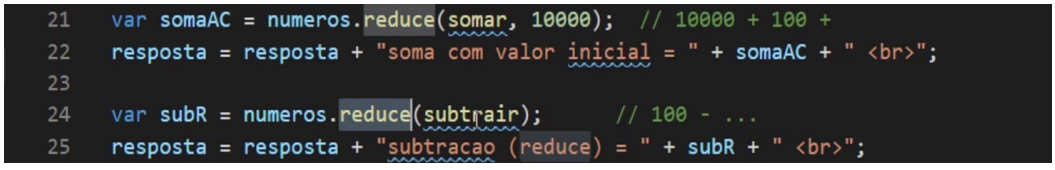
Na linha 15, vamos declarar o array `numeros` contendo os valores 100, 200, 400 e 800, e vamos declarar que a resposta é esse array da linha 16. Ou seja, vai exibir: `numeros` é igual ao array. Depois, a gente vai fazer um `reduce`, passando a função `somar` nesse array.

Figura 5 - `reduce(somar)`



Não estamos passando o valor inicial então, na linha 19, vou dizer: "A soma, sem o valor inicial", e o que ele vai retornar é 100, mais 200, mais 400 e mais 800. Depois, para que vocês vejam que, se eu realmente passar um valor inicial, ele começa com esse valor já no acumulador, eu vou passar o valor 10000 como segundo argumento, ok? Então, quando eu fizer um `reduce` usando 10000 como valor inicial, ele vai começar somando 10000, mais 100, mais 200, mais 400 e mais 800.

Figura 6 - `reduce(somar, 10000)` e `reduce(subtrair)`



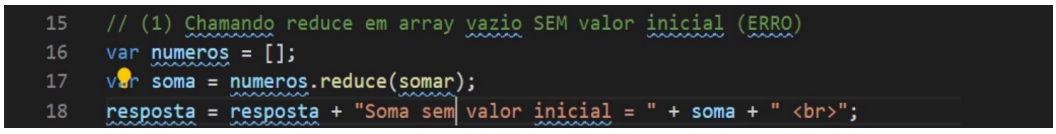
Depois, faço um `reduce` na linha 24, usando `subtrair` sem valor inicial. Você vai ver que o resultado vai ser 100 menos 200, menos 400 e menos 800. Logo, vou fazer um `reduceRight` para que você veja que vai fazer um `reduce` da direita para a esquerda, então o resultado vai ser 800 menos 400, menos 200, menos 100. Assim, na página HTML, vemos qual foi o resultado.

Figura 7 - Saída do Exemplo

Perceba no início o array `numeros`, em seguida a soma sem o valor inicial, que dá 1500, pois é o 100, mais 200, mais 400, mais 800, dá 1500, perfeito! Entretanto, na linha seguinte, ao passar o valor inicial que era 10000, ele vai somar esses 10000 com essa soma de 1500, ou seja, fica 11500. A subtração usando o `reduce` deu 100 menos 200, menos 400 e menos 800, que dá -1300, é o que ele fez. E no `reduceRight`, que fica da direita para a esquerda, temos: menos 800 , menos 400, menos 200, menos 100, que vai dá 100, ou seja, ele foi subtraindo da direita para a esquerda. Ok?

Agora, vejamos aquelas situações especiais que mencionei anteriormente. Primeiro, vamos ver o que acontece se eu chamar um `reduce` em um array vazio sem valor inicial, para isso vou declarar a variável `numeros` e dizer que soma é o `numeros.reduce`, passando a *callback* `somar`, mas com `numeros` vazio, como consta na linha 16 da Figura 8, ok?

Figura 8 - `reduce(somar)` em um Array Vazio



E, na linha 18, vamos colocar "sem valor inicial", ou seja, quero escrever na tela, mas se eu recarregar, não aparece nada. Isso porque a página gerou um erro já que estou chamando `reduce`, passando um array vazio e sem passar um valor inicial.

Agora, se eu passar um valor inicial, realizando um segundo teste, dessa vez passando o 10000 como valor inicial.

Figura 9 - `reduce(somar, 10000)` em um Array Vazio

```
15  // (2) Chamando reduce em array vazio COM valor inicial
16  var numeros = [];
17  var soma = numeros.reduce(somar, 10000);
18  resposta = resposta + "Soma com valor inicial = " + soma + " <br>";
```

E veja o que acontece:

Figura 10 - Saída do Exemplo

Ele considera o valor inicial e, como não tinha com o que somar, retornou apenas o valor inicial. Então, o erro foi removido, por isso que sempre falo da importância de usar um valor inicial.

Agora, vamos fazer outro teste, iremos chamar o `reduce` em um array de tamanho 1 com o valor inicial. Para isso, tenho um array com o valor 100 dentro dele e tenho o 10000.

Figura 11 - `reduce(somar, 10000)` em um Array com Um Elemento

```
15  // (3) Chamando reduce em array de tamanho = 1 COM valor inicial
16  var numeros = [100];
17  var soma = numeros.reduce(somar, 10000);
18  resposta = resposta + "Soma com valor inicial = " + soma + " <br>";
```

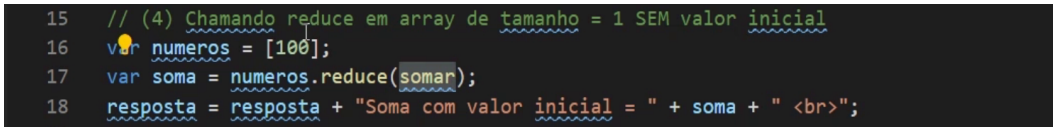
Sabem o que acontece? Ele somou, deu tudo certo e sem nenhum problema.

Figura 12 - Saída do Exemplo

O que não pode é chamar a função `reduce` sem passar um valor inicial em um array vazio.

E, por fim, um último teste, uma última situação especial que eu queria mostrar para você é chamar o `reduce` em um array de tamanho 1, sem o valor inicial. Com isso, tenho o array `numeros` com o valor 100 e chamo o `reduce(somar)` sem passar o valor inicial.

Figura 13 - `reduce(somar)` em um Array com Um Elemento



Então, ao recarregar a página HTML, percebemos que ele só pega o primeiro valor, que é 100, pois considera isso no acumulador. Assim, o que quero dizer com esse exemplo, é que não se pode chamar `reduce` e `reduceRight`, passando um array vazio, mas não passando um valor inicial.

Os quatro últimos métodos desta videoaula são capazes de verificar os elementos de um array com relação a alguma condição. São eles: `every`, `some`, `find` e `findIndex`. Assim como os métodos anteriores, esses métodos não alteram o array original. Além disso, com relação às funções de *callback* que podem ser usadas, esses métodos são idênticos ao método `filter` em dois aspectos:

1. A função de *callback* usa três argumentos: o valor do item, o índice do item e o próprio array. Caso a função de *callback* utilize apenas o valor do item, ela pode ser definida com apenas um argumento, o valor. Por simplicidade, no que se segue, vamos utilizar funções de *callback* com um argumento apenas.

2. A função de *callback* representa condições, ou seja, deve retornar valores booleanos para cada valor do array.

Dessa forma, o método `every` ("todos" em inglês) retorna `true` apenas se todos os elementos do array satisfizerem a condição representada pela função de *callback*. Ou seja, o retorno dessa função deve ser `true` para todos os elementos do array. Por outro lado, o método `some` ("algum" em inglês) retorna `true` apenas se algum

elemento do array satisfizer a condição representada pela função de *callback*. Ou seja, o retorno dessa função deve ser `true` para pelo menos um dos elementos do array. Veja o exemplo de código no slide (Código 4).

Código 4 - 12_9 EverySome.html e 12_9 EverySome.js

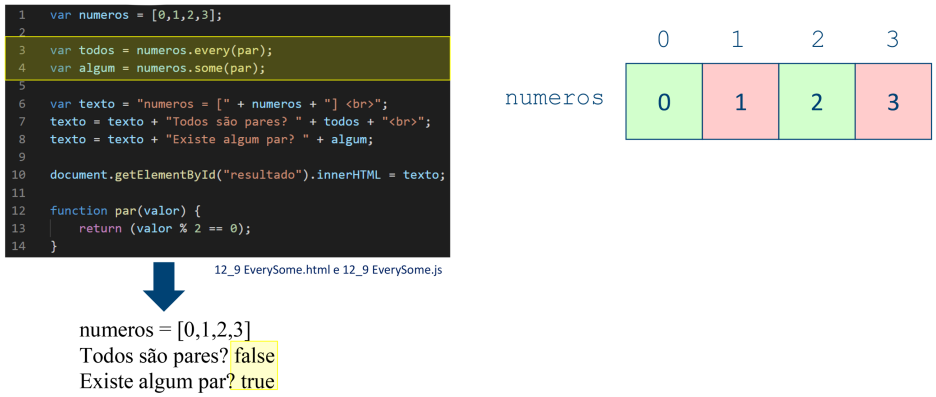
```html
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 12</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Every e Some</h1>

    <p id="resultado"></p>

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

```

```
var numeros = [0,1,2,3];

var todos = numeros.every(par);
var algum = numeros.some(par);

var texto = "numeros = [" + numeros + "] <br>";
texto = texto + "Todos são pares? " + todos + "<br>";
texto = texto + "Existe algum par? " + algum;

document.getElementById("resultado").innerHTML = texto;

function par(valor) {
   return (valor % 2 == 0);
}

```

Nele, temos novamente um array com os números 0, 1, 2 e 3 e, repetimos, nas linhas 12 a 14, a função `par`, a qual verifica se um número dado é par. O destaque aqui vai para as linhas 3 e 4. Na linha 3, usamos o método `every` para atribuir à variável `todos` o resultado da verificação se todos os elementos são pares. Para isso, chamamos o método `every` no array `numeros` e passamos a função `par` como

argumento. De maneira similar, na linha 4, usamos o método `some` para atribuir à variável `algum` o resultado da verificação se algum elemento for par. Como esperado, veja na Figura 14 que a variável `todos` recebe `false`, pois os números 1 e 3 não são pares e a variável `algum` recebe `true`, pois os números 0 e 2 são pares.

Figura 14 - `every` e `some`



IMPORTANTE

Todos os métodos que vimos até agora nesta videoaula, ou seja, `forEach`, `map`, `filter`, `reduce`, `reduceRight`, `every` e `some`, são suportados por todos os navegadores, menos no Internet Explorer 8 e versões anteriores.

Agora, você irá conhecer os métodos `find` e `findIndex`. Ambos procuram um elemento no array que satisfaz a condição representada pela função de *callback*. No entanto, `find` retorna o elemento encontrado e `findIndex` retorna o índice desse elemento. Caso nenhum elemento satisfaça a condição, `find` retorna o valor `undefined` e `findIndex` retorna -1. Veja mais um exemplo (Código 5).

Código 5 - 12_10 Find.html e 12_10 Find.js

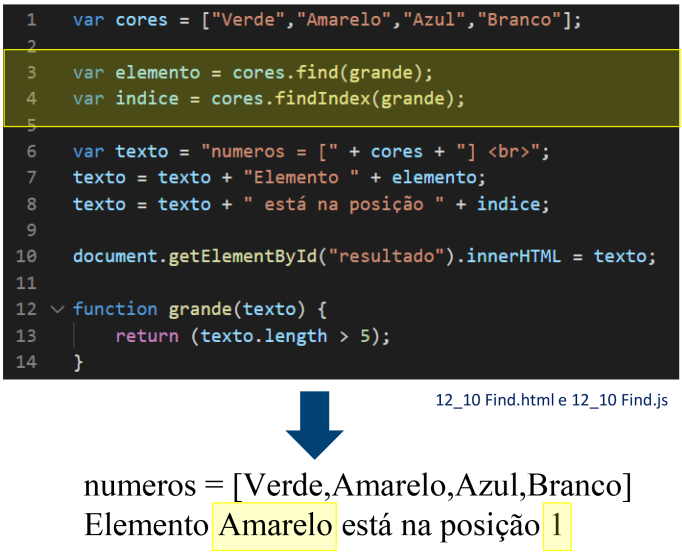
```html
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 12</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Find e FindIndex</h1>

    <p id="resultado"></p>

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

```

```
var cores = ["Verde","Amarelo","Azul","Branco"];

var elemento = cores.find(grande);
var indice = cores.findIndex(grande);

var texto = "numeros = [" + cores + "] <br>";
texto = texto + "Elemento " + elemento;
texto = texto + " está na posição " + indice;

document.getElementById("resultado").innerHTML = texto;

function grande(valor) {
   return (valor.length > 5);
}

```

Desta vez, vamos usar o array de cores `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"` declarado na linha 1. Além disso, nesse exemplo, a função de *callback*, declarada nas linhas 12 a 14, verifica se o tamanho do texto é maior que 5. Veja que ela retorna `true` apenas se o `texto.length`, ou seja, o tamanho do texto, for maior que 5. O destaque aqui vai para as linhas 3 e 4. Na linha 3 usamos o método `find` para atribuir à variável `elemento`, o primeiro elemento do array com tamanho maior do que 5. Para isso, chamamos o método `find` no array `cores` e passamos a função `grande` como argumento. De maneira muito similar, na linha 4, usamos o método `findIndex` para atribuir à variável `indice` o índice do primeiro elemento do array com tamanho

maior do que 5. Como esperado, veja na Figura 15 que a variável `elemento` recebe `Amarelo`, pois esse é o primeiro texto com tamanho maior do que 5 no array e a variável `indice` recebe 1, este é o índice da primeira ocorrência de `Amarelo`.

Figura 15 - `Find` e `FindIndex`



Os métodos `find` e `findIndex` não são suportados em navegadores antigos. As primeiras versões dos principais navegadores com suporte total para esses métodos estão listadas no slide (Figura 16).

Figura 16 - `find` e `findIndex`

Concluímos aqui esta videoaula onde você conheceu vários métodos de iteração sobre arrays. Na próxima videoaula, você aprenderá sobre arrays bidimensionais, também chamados de matrizes. Nos encontramos lá!!!