Programação Estruturada

Aula 11 - Arrays: Introdução, Acesso e Métodos

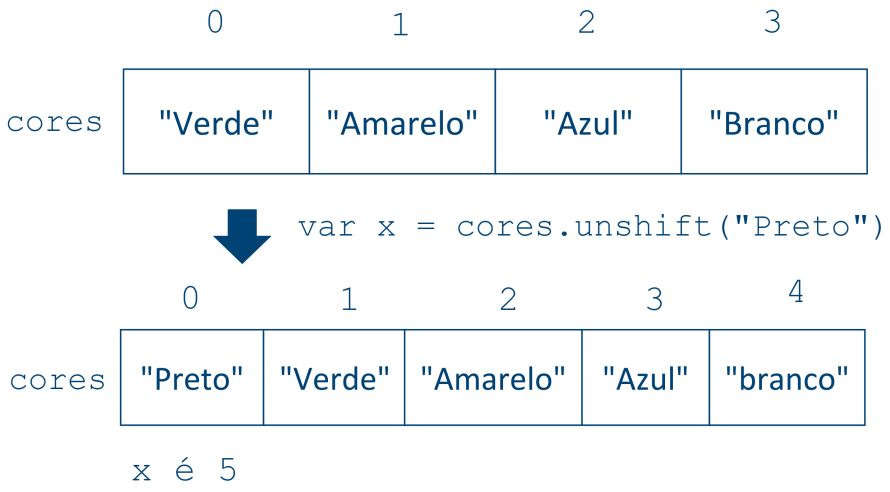
Videoaula 04: Métodos para Arrays (Parte 2)

Videoaula 04: Métodos para Arrays (Parte 2)

Vamos continuar a conhecer os métodos oferecidos por JavaScript, que são bastante úteis para trabalhar com arrays. Nesta videoaula, conversaremos sobre os métodos `unshift`, `shift`, `delete`, `splice`, `concat` e `slice`.

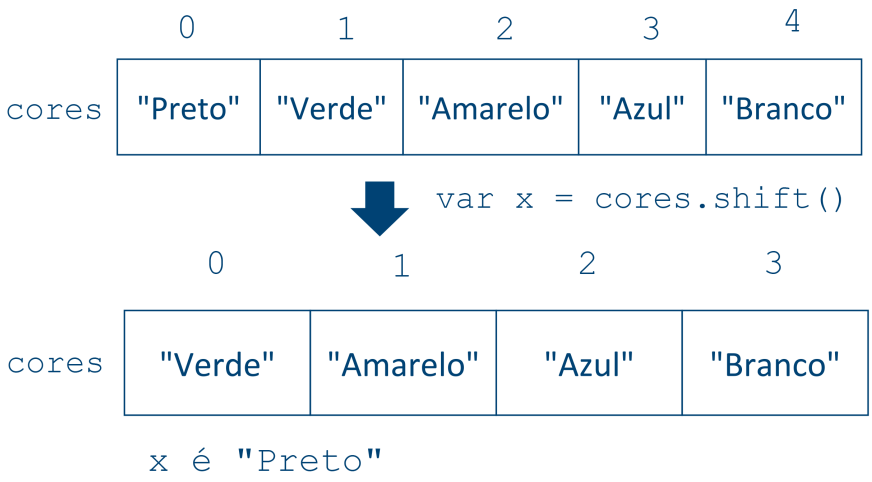
O método `unshift` adiciona o elemento passado no início do array e retorna o tamanho final do array. No exemplo (Figura 1), temos que, se adicionarmos a *string* `"Preto"` ao array `cores` usando o método `unshift` e atribuindo o resultado desse método à variável `x`, a *string* `"Preto"` será acrescentada no início do array `cores` e a variável `x` terá o valor 5, ou seja, o novo tamanho do array.

Figura 1 - `unshift`



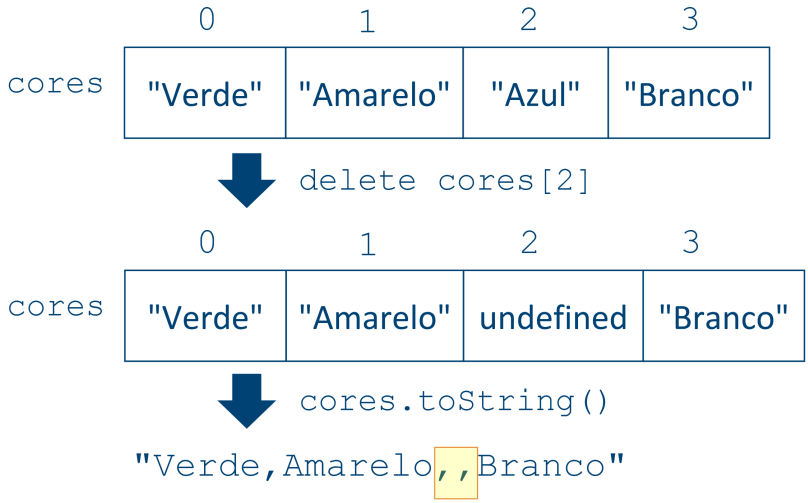
O método `shift` remove o primeiro elemento do array e o retorna. No exemplo (Figura 2), temos que, se aplicarmos `shift` ao array `cores` atribuindo o resultado desse método à variável `x`, a *string* `"Preto"` será removida do início do array `cores` e a variável `x` terá o valor `"Preto"`.

Figura 2 - `shift`



O operador `delete` remove uma propriedade de um objeto. Como arrays JavaScript são objetos, podemos excluir elementos usando esse operador. Ao aplicarmos o `delete` no elemento de índice 2, alteramos o valor desse elemento para `undefined`. Porém, note (Figura 3) que esse comando cria um "buraco" de valor `undefined` no array. Por esse motivo, para remover um elemento, o uso de `pop` ou `shift` é mais recomendado.

Figura 3 - `delete`

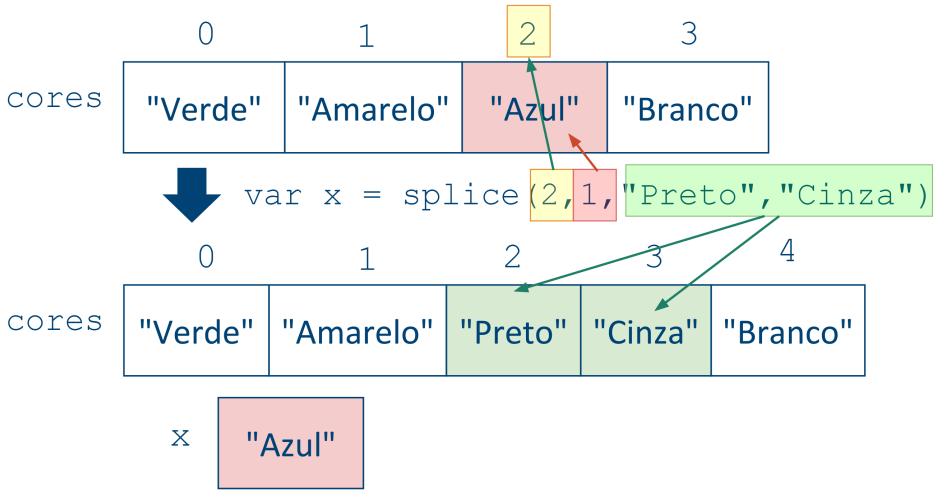


É importante ressaltar que valores `undefined` não são inseridos no resultado da chamada do método `toString` aos arrays. Portanto, a *string* resultante da aplicação do método `toString` possui duas vírgulas seguidas, pois o valor `undefined` não é inserido no resultado.

Veja agora o método `splice`, que também pode ser usado para adicionar e remover elementos do array. A sua principal diferença é que ele permite que isso seja feito dentro dos arrays e não apenas no início e no final. O primeiro parâmetro do método `splice` define a posição onde os novos elementos devem ser adicionados. O segundo parâmetro indica quantos elementos devem ser removidos a partir do índice usado no primeiro parâmetro. Por fim, o restante dos parâmetros define os novos elementos a serem adicionados. Além de transformar o array, o método `splice` retorna um array contendo os elementos removidos.

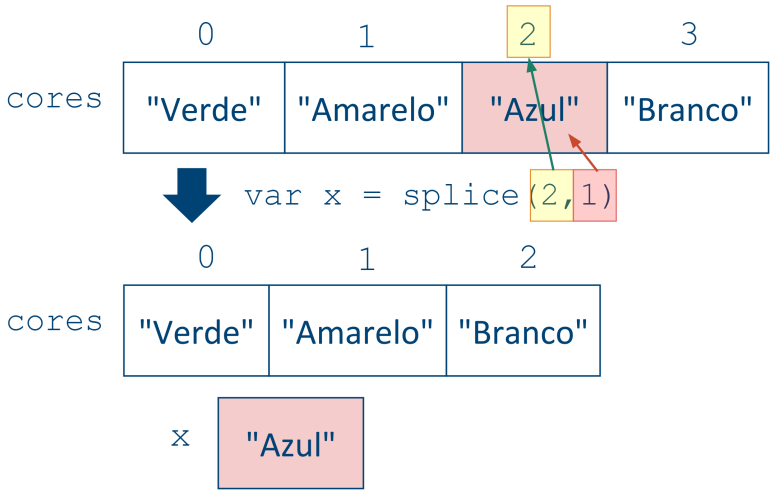
No exemplo que vemos no slide (Figura 4), usamos o 2 como o primeiro parâmetro. Portanto, essa operação terá efeito a partir do elemento de índice 2 do array `cores`. No segundo parâmetro utilizamos o valor 1. Assim sendo, o resultado removerá um elemento a partir do índice 2, ou seja, removeremos o elemento `"Azul"` do resultado. Por fim, também passamos, nessa ordem, os valores `"Preto"` e `"Cinza"`. Note que o resultado inclui esses dois valores no array a partir da posição passada no primeiro parâmetro, ou seja, da posição 2. Note também que o método retorna um array contendo apenas um elemento, `"Azul"`, o qual foi o único valor removido. Esse valor foi atribuído à variável `x`.

Figura 4 - `splice`



Se usado de uma maneira inteligente, o método pode ser utilizado também para remover elementos. Para isso, basta usarmos apenas os dois primeiros parâmetros, ou seja, a posição onde os novos elementos deveriam ser adicionados e quantos elementos devem ser removidos a partir dessa posição. Ao não passarmos mais parâmetros, não adicionaremos nenhum novo elemento. Na prática, isso será o mesmo que apenas remover elementos. Veja no slide (Figura 5).

Figura 5 - `splice` com apenas 2 parâmetros



Nele, estamos novamente usando 2 como o primeiro parâmetro e 1 como segundo parâmetro. Portanto, essa operação terá efeito a partir do elemento de índice 2 do array `cores` e removerá um elemento a partir dessa posição. Portanto, como não passamos mais nenhum parâmetro, nenhum novo valor será inserido. Na prática, ao chamarmos `slice(2,1)` estamos removendo um elemento a partir do índice 2. Ou seja, o arrays `cores` terá, nessa ordem, os elementos `"Verde"`, `"Amarelo"` e `"Branco"`. O método ainda retorna um array contendo apenas um elemento, `"Azul"`, o qual foi o único valor removido. Esse valor foi atribuído à variável `x`.

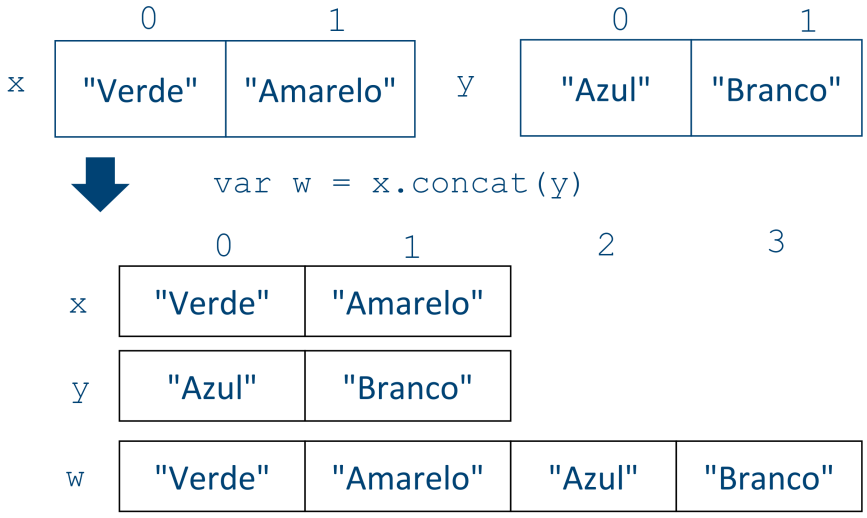
Outro método bastante útil é o `concat`, que concatena os arrays passados como parâmetros. Em outras palavras, esse método une os dois arrays.

Atenção

É importante ressaltar que esse método tem um comportamento ligeiramente diferente dos que já foram apresentados, pois ele não altera os arrays existentes. Ele sempre retornará um novo array.

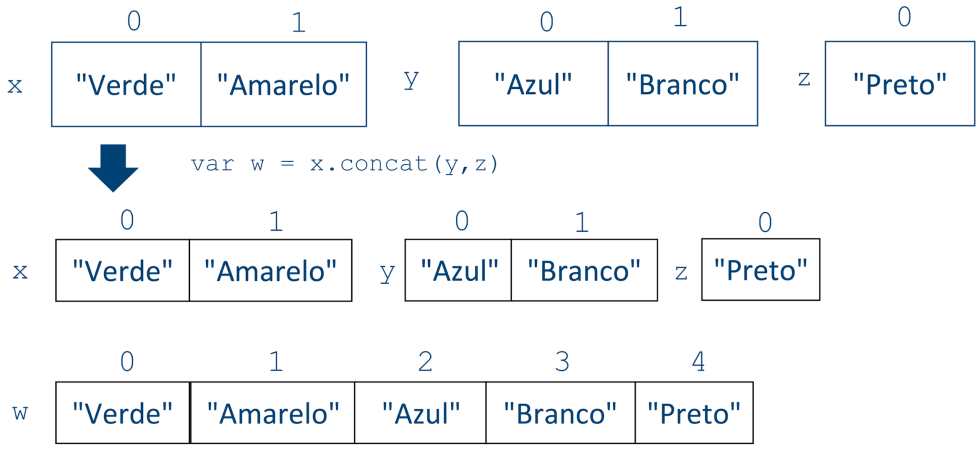
Veja no slide (Figura 6). Nele, temos inicialmente dois arrays. O array `x` possui os elementos `"Verde"` e `"Amarelo"`, enquanto o `y` possui os elementos `"Azul"` e `"Branco"`. Quando atribuímos o valor da expressão `x.concat(y)` à variável `w`, essa variável passa a ter como valor o array com elementos `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"`. No entanto, os valores das variáveis `x` e `y` continuam inalterados.

Figura 6 - `concat`



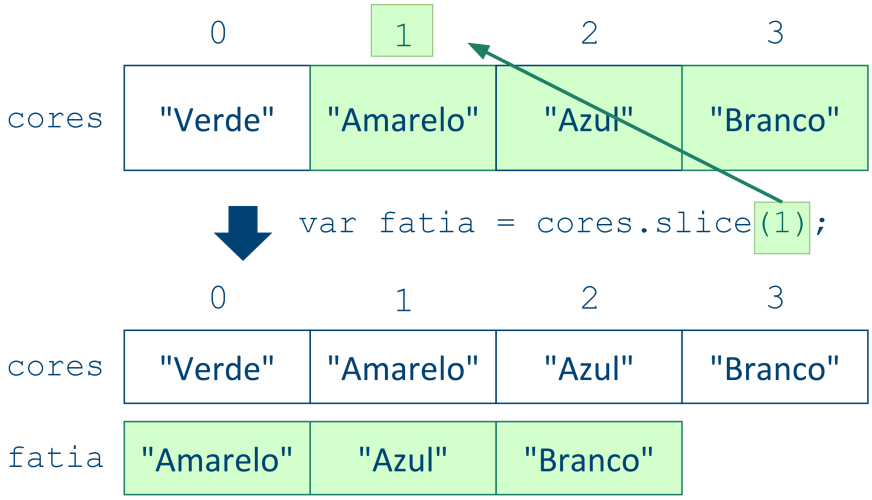
É importante ressaltar que o método `concat` pode receber uma quantidade qualquer de parâmetros. Nesse novo exemplo que podemos ver no slide (Figura 7), temos um terceiro array, `z`, que possui apenas o elemento `"Preto"`. Note que agora estamos aplicando o mesmo método `concat` ao array `x`, mas desta vez passamos os arrays `y` e `z` como parâmetro. Esse uso faz com que a resposta seja o resultado da concatenação de `x` com `y` e `z`. Mais uma vez, os valores das variáveis `x`, `y` e `z` continuam inalterados.

Figura 7 - `concat` com mais de um parâmetro



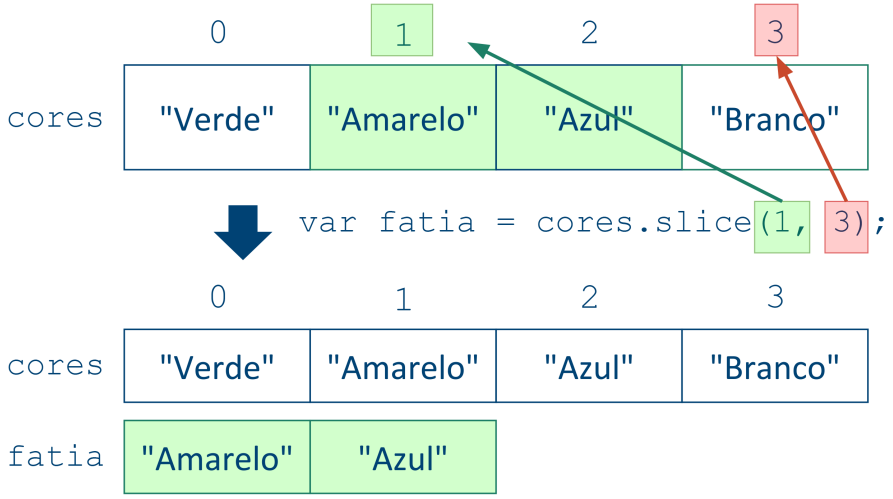
O último método de arrays que apresentarei nesta aula é o `slice`. Esse método retorna uma "fatia" (*slice* em inglês) do array à qual ele é aplicado. Assim como o método `concat`, ele não altera o array original, mas apenas retorna um novo array com o resultado da operação. Caso usemos apenas um parâmetro, a "fatia" retornada pelo método `slice` começa no índice passado como parâmetro. No exemplo que você pode ver no slide (Figura 8), o resultado de aplicarmos `slice(1)` ao array `cores` não altera esse array, mas retorna a fatia do array que começa no índice 1 do array `cores` e vai até o seu final.

Figura 8 - `slice`



O método `slice` também pode receber dois parâmetros. Nesse caso, o método retorna uma "fatia" do array que começa no índice passado como primeiro parâmetro e termina no elemento de índice anterior ao índice passado como segundo parâmetro. No exemplo que você pode ver no slide (Figura 9), o resultado de aplicarmos `slice(1,3)` ao array `cores` não altera esse array, mas retorna a fatia do array que começa no índice 1 do array `cores` e vai até antes do índice 3. Note que o elemento de índice 3 do array cores, `"Branco"`, não faz parte da fatia.

Figura 9 - `slice` com mais de um parâmetro



Para concluir esta videoaula, vamos agora rever os métodos e exemplos que vimos sendo executados na prática. Para isso, você pode utilizar o Código 1. Fique à vontade para editá-lo a fim de fazer vários testes.

Código 1 - 11_9 Métodos para Arrays - Parte 2.html

```html
<html >
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 11</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Métodos para Arrays - Parte 2</h1>
    <!--
    Quantidade de Notas: <input type="number" id="total" value="">
    <button onclick="ler_notas()">INICIAR</button>
    -->
    <p id="resultado"></p>

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

```

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

```

```
4  /* Testando shift */
5  cores.unshift("Preto");
6  var x = cores.shift();
7  resposta = "Novo array ["+cores+"] teve removido "+x;
8
9  /* Testando delete */
10 //delete cores[2];
11 //resposta = cores;
12
13 /* Testando splice */
14 //var x = cores.splice(2,1,"Preto","Cinza");
15 //var x = cores.splice(2,1);
16 //resposta = "Novo array ["+cores+"] teve removido ["+x+"]";
17
18 /* Testando concat */
19 //var x = ["Verde","Amarelo"];
20 //var y = ["Azul","Branco"];
21 //var z = ["Preto"];
22 //var w = x.concat(y,z);
23 //resposta = "x é ["+x+"] <br>";
24 //resposta = resposta + "y é ["+y+"] <br>";
25 //resposta = resposta + "z é ["+z+"] <br>";
26 //resposta = resposta + "w é ["+w+"] <br>";
27
28 /* Testando slice */
29 //var fatia = cores.slice(1);
30 //var fatia = cores.slice(1,3);
31 //resposta = "cores é ["+cores+"] <br>";
32 //resposta = resposta + "fatia é ["+fatia+"] <br>";
33
34
35 document.getElementById("resultado").innerHTML = resposta;
36
```

A página HTML apresentada é bem simples e, basicamente, tem um campo chamado `resultado`. Para escrever nesse campo, estaremos construindo alguma *string* no JavaScript. Então, essa *string*, que será a variável `resposta`, começará vazia, e no final, na linha 35, esse valor será usado para escrever na tela. Declaramos na linha 1 nosso array já utilizado anteriormente nos exemplos, o qual tem os elementos `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"`, e testamos os métodos `shift` e `unshift`.

Começaremos com o método `unshift`. Aplicamos esse método, passando o valor `"Preto"`, e depois, diremos que o nosso texto será `"Novo array"` com o `toString` aplicado a esse novo array `cores`. Veja essa ação na Figura 10.

Figura 10 - Usando `unshift`

```
5  cores.unshift("Preto");
6  resposta = "Novo array ["+cores+"]";
```

Verificando o resultado, ao carregar a página HTML, você verá que de fato, esse novo elemento, a *string* `"Preto"`, foi colocado no começo do array.

Após colocar o elemento `"Preto"` no começo do array, damos um `shift` nesse array e atribuímos o resultado dessa chamada à variável `x`. Além de imprimir o novo array, direi qual foi o valor removido, passando o valor da variável `x`, que recebeu na linha 6 a saída/retorno da aplicação do método `shift` à variável `cores`. Veja esse processo na Figura 11.

Figura 11 - Usando `shift`

```
5  cores.unshift("Preto");
6  var x = cores.shift();
7  resposta = "Novo array ["+cores+"] teve removido "+x;
8
```

Como o elemento `"Preto"` acabou de ser colocado, esperamos, obviamente, que esse valor seja a *string* `"Preto"` e, ao carregar a página HTML, você verá que o array, de fato, agora ficou inalterado. Isso porque, colocamos o `"Preto"` no começo e depois tiramos o primeiro elemento e o elemento removido será a *string* `"Preto"`.

Vamos conhecer o método `delete`. Iremos, simplesmente, aplicar o `delete` ao índice 2 do array `cores`. E aí então, daremos um `toString` no novo array.

Figura 12 - Usando `delete`

```
1  var cores = ["Verde","Amarelo","Azul","Branco"];
2  var resposta = "";
3
4  /* Testando delete */
5  delete cores[2];
6  resposta = cores;
```

Ao carregar a página HTML, aparecerão duas vírgulas sem nada entre elas, porque ao fazer o `delete` do índice 2 do array, esse valor se transformou em `undefined`, e não vai fazer parte da *string* resultante da aplicação do `toString` a esse array. Dessa forma, teremos essas duas vírgulas coladas, porque no meio desse array tem o valor `undefined`. Veja a Figura 13.

Figura 13 - Testando o `delete`

Vamos conhecer o método `splice`. Daremos um `splice` no array `cores` a partir do índice 2. Apagamos um elemento e inserimos os elementos `"Preto"` e `"Cinza"`. Na `resposta`, além de informar qual é o novo array, será dito o que foi removido, passaremos `x`, recebendo o valor do retorno da aplicação do método `splice`. Veja essa ação na Figura 14.

Figura 14 - Usando `splice`

```
4  /* Testando splice */
5  var x = cores.splice(2,1,"Preto","Cinza");
6  //var x = cores.splice(2,1);
7  resposta = "Novo array ["+cores+"] teve removido ["+x+"]";
```

Ao recarregar a página HTML, o novo array será `"Verde"`, `"Amarelo"`, `"Preto"`, `"Cinza"` e `"Branco"`, porque a partir do índice 2 foi removido um elemento, o `"Azul"`, e foram colocados no lugar dele os elementos `"Preto"` e `"Cinza"`. Dessa forma, o `"Azul"` aparecerá como elemento removido.

Agora, usaremos o `splice` sem passar parâmetros. Iremos trabalhar a partir do índice 2 e remover um elemento. Nesse caso, tudo que ele vai fazer é remover o elemento `"Azul"`. E ao carregar a página HTML você verá, de fato, que tudo o que aconteceu foi o elemento `"Azul"` ter sido removido do array. Veja a Figura 15.

Figura 15 - Testando o `splice`

Um outro método, já conhecido, é o `concat`. Faremos o seguinte: declaramos a variável `x` com `"Verde"` e `"Amarelo"`; a variável `y` com `"Azul"` e `"Branco"`; e a variável `z` com `"Preto"`. Vimos que é possível concatenar dois ou três elementos, então diremos que a variável `w` é o resultado da concatenação de `x` com `y` com `z`. Logo em seguida, imprimimos quem é `x`, quem é `y`, quem é `z` e quem é `w`.

Figura 16 - Usando `concat`

```
4   /* Testando concat */
5   var x = ["Verde","Amarelo"];
6   var y = ["Azul","Branco"];
7   var z = ["Preto"];
8   var w = x.concat(y,z);
9   resposta = "x é ["+x+"] <br>";
10  resposta = resposta + "y é ["+y+"] <br>";
11  resposta = resposta + "z é ["+z+"] <br>";
12  resposta = resposta + "w é ["+w+"] <br>";
```

Espera-se que `x`, `y` e `z` permaneçam com esses valores, ou seja, `x` é o array com `"Verde"` e `"Amarelo"`; `y` é o array com `"Azul"` e `"Branco"`; `z` é o array com `"Preto"`, e, por fim, `w` será, de fato, o resultado da concatenação desses três arrays. Ao carregar a página (Figura 17), você verá que `x`, `y` e `z` permaneceram inalterados e `w`, de fato, é a concatenação de `"Verde"` e `"Amarelo"`, que é o `x`, com `"Azul"` e `"Branco"`, que é o `y`, com `"Preto"`, que é o `z`.

Figura 17 - Testando o `concat`

Por fim, vamos conhecer o método `slice`, que é quem pegará `fatia`, e nesse primeiro teste, `fatia` é simplesmente o resultado da aplicação de `slice` a partir do elemento de índice 1. Veja a figura 18.

Figura 18 - Usando `slice`

```
4   /* Testando slice */
5   var fatia = cores.slice(1);
6   //var fatia = cores.slice(1,3);
7   //resposta = "cores é ["+cores+"] <br>";
8   resposta = resposta + "fatia é ["+fatia+"] <br>";
9
10  document.getElementById("resultado").innerHTML = resposta;
```

Lembrando que temos o array `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"`, e ao fazermos `slice` a partir do índice 1, teremos o array `"Amarelo"`, `"Azul"` e `"Branco"`, porque `fatia` começa no elemento de índice 1, que é o `"Amarelo"`.

Poderei dizer, também, onde se encontra o final dessa `fatia`. Então, agora, utilizaremos o `slice` usando os índices 1 e 3. Nesse caso, a fatia começará no índice 1 e terminará antes do índice 3, ou seja, no índice 2. Veja a figura 19.

Figura 19 - Usando `slice` com dois parâmetros

```
4   /* Testando slice */
5   var fatia = cores.slice(1,3);
6   //resposta = "cores é ["+cores+"] <br>";
7   resposta = resposta + "fatia é ["+fatia+"] <br>";
8
9   document.getElementById("resultado").innerHTML = resposta;
```

Ao carregar a página HTML, espera-se que `fatia` seja apenas `"Amarelo"` e `"Azul"`, porque ela começa no índice 1, que é o elemento `"Amarelo"`, e termina no índice 2, que é o `"Azul"`

Ao pedir para ser impresso também o array `cores`, e ao recarregar a página HTML, você verá que ele permanecerá inalterado. Então, assim como o método anterior, esse método `slice`, não altera os arrays originais, ele apenas retorna um novo array.

Agora que você já sabe como declarar arrays, acessar seus elementos e alguns métodos importantes que podem ser utilizados para manipulá-los, podemos nos aprofundar um pouco mais no trabalho com essa importante estrutura de dados.

Na próxima aula, você aprenderá como ordenar arrays, assim como outros métodos muito úteis de iteração sobre os arrays que permitem, por exemplo, que você aplique uma determinada função a todos os elementos de um array. Além disso, na próxima aula, você também aprenderá sobre arrays bidimensionais, também chamados de matrizes. Tchau, tchau!!!