Programação Estruturada

Aula 11 - Arrays: Introdução, Acesso e Métodos

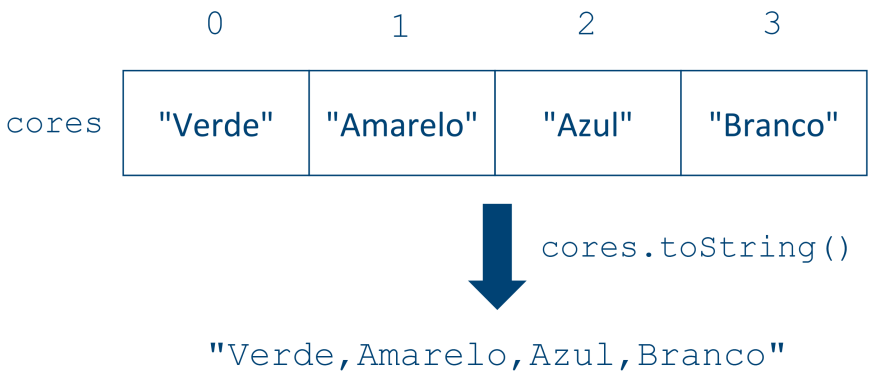
Videoaula 03: Métodos para Arrays (Parte 1)

Videoaula 03: Métodos para Arrays (Parte 1)

Agora que você já aprendeu a definir arrays e acessar os seus elementos, chegou a hora de conhecer alguns métodos oferecidos por JavaScript que são bastante úteis para trabalhar com arrays, como, por exemplo, métodos para inserir e remover elementos e métodos para concatenar, ou seja, unir dois arrays. São eles: `toString`, `join`, `split`, `push`, `pop`, `unshift`, `shift`, `delete`, `splice`, `concat` e `slice`. Nesta videoaula, conversaremos sobre os cinco primeiros métodos e deixaremos os demais para a próxima videoaula. Vamos lá?!

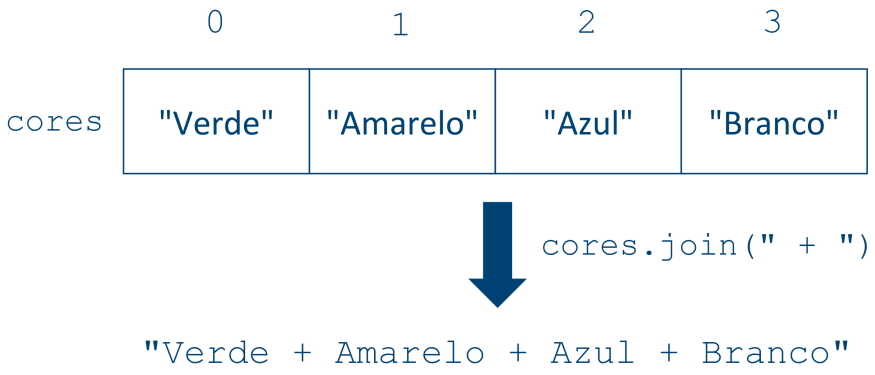
Os métodos `toString`, `join` e `split` nos permitem trabalhar com arrays e *strings*. Para começar, você já conheceu o comportamento do método `toString`, o qual, quando aplicado a um array, retorna uma *string* contendo os elementos do array separados por vírgula. Note (Figura 1) que usando esse método você não tem a opção do separador, pois ele sempre usará a vírgula.

Figura 1 - `toString`



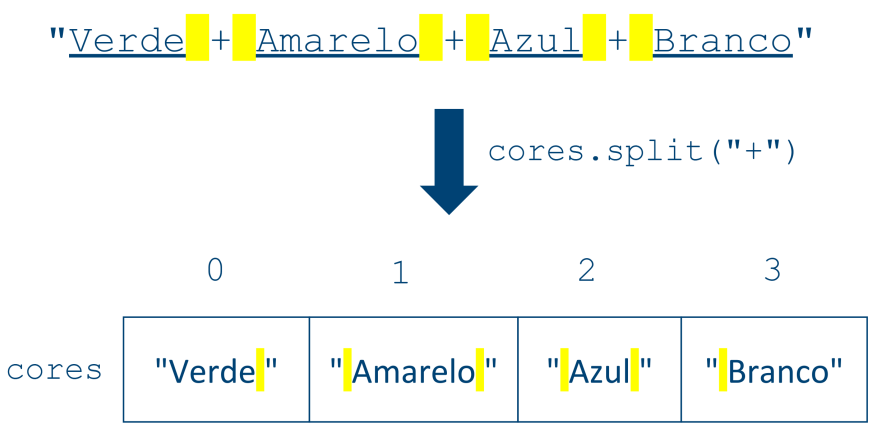
Caso você deseje escolher o separador, deve usar o método `join`, o qual tem o mesmo comportamento do método `toString`, mas utiliza o separador passado por você para separar os elementos do arrays na *string* resultante. Veja no slide (Figura 2) que, usando esse método, escolhemos o símbolo de adição para separar os elementos. Note, porém, que também acrescentamos espaços em branco antes e depois dos elementos.

Figura 2 - `join`



JavaScript também oferece um método para fazer o caminho contrário, ou seja, transformar uma *string* em um array. O método `split` recebe uma *string* que indica quem é o separador dos elementos e retorna um array contendo os elementos que estão separados pelo separador indicado pelo programador. No slide (Figura 3), vemos um exemplo de aplicação desse método ao resultado da *string* que criamos no slide anterior. Note que cada elemento do array corresponde exatamente aos textos que estão entre os símbolos de adição. Note, porém, que os espaços em branco permaneceram nos elementos do array. Para facilitar a visualização, no slide (Figura 3), pintamos de amarelo esses espaços em branco no texto original.

Figura 3 - `split`



A fim de remover esses espaços em branco dos elementos da resposta, temos duas opções: na primeira opção, podemos simplesmente usar o separador com espaços em branco que foi utilizado para gerar a *string*.

Na segunda opção, precisamos aplicar o método `replace` à *string* antes de chamarmos o método `split`. Nesta chamada, utilizamos a expressão regular `/\s/g`, a qual representa todos os espaços em branco da *string*. Dessa forma, estamos substituindo todos os espaços em branco da *string* pela *string* vazia, ou seja, estamos removendo esses espaços em branco. Conforme você viu na aula sobre *strings*, uma expressão regular é um objeto que descreve um padrão de caracteres e que pode ser usado para fazer o que chamamos de casamento de padrão.

#FicaDica

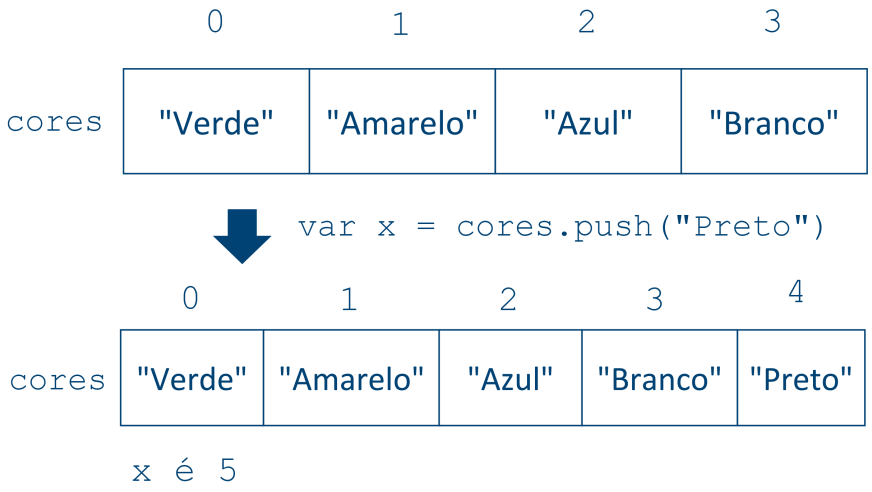
Caso deseje saber mais sobre padrões, você pode acessar um desses links:

JavaScript RegExp Reference

https://www.ecma-international.org/ecma-262/9.0/index.html#sec-regexp-regular-expression-objects

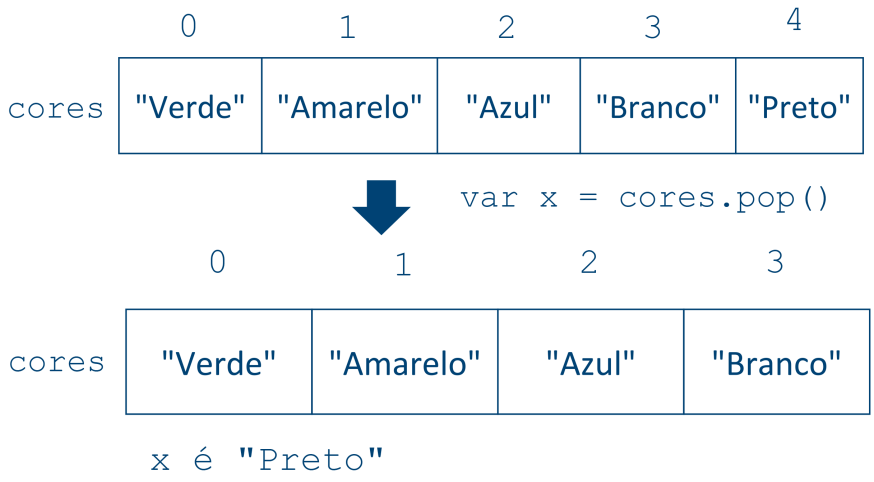
Agora, conheça dois métodos que nos permitem adicionar e remover elementos no final do array. Primeiramente, o método `push` adiciona o elemento passado no final do array e retorna o tamanho final do array. No nosso exemplo (Figura 4), se adicionarmos a *string* `"Preto"` ao array `cores`, usando o método `push`, atribuindo o resultado desse método à variável `x`, a *string* `"Preto"` será acrescentada no final do array `cores` e a variável `x` terá o valor 5, ou seja, o novo tamanho do array. Essa maneira de adicionar elementos a um array é mais segura do que a que vimos anteriormente, pois evita a inserção de "buracos" de valores `undefined` no array.

Figura 4 - `push`



O método `pop` remove o último elemento do array e retorna esse elemento. No nosso exemplo (Figura 5), temos que, se aplicarmos `pop` ao array `cores` atribuindo o resultado desse método à variável `x`, a *string* `"Preto"` será removida do final do array `cores` e a variável `x` terá o valor `"Preto"`.

Figura 5 - `pop`



Juntos, os métodos `push` e `pop` nos oferecem a oportunidade de usar arrays como pilhas, outra estrutura de dados muito utilizada em programas que se baseia no princípio *Last In First Out* (LIFO); ou seja, "o último que entra é o primeiro que sai" e caracteriza um empilhamento de dados.

Conforme vimos, esses métodos trabalham no final do array. No entanto, JavaScript oferece métodos que têm o mesmo comportamento do `push` e do `pop`, mas que trabalham no ínicio do array. Conheceremos esses métodos na próxima videoaula.

Para concluir esta videoaula, vamos agora observar os métodos que vimos sendo executados, você pode utilizar o Código 1 para verificar o comportamento deles na prática. Nossa página HTML é muito simples, tem só um campo, e iremos basicamente trabalhar no arquivo JavaScript.

Código 1 - 11_8 Métodos para Arrays - Parte 1.html e 11_8 Métodos para Arrays.js

```
1  <html >
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 11</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Métodos para Arrays - Parte 1</h1>
```

```
10
11        <!--
12       Quantidade de Notas: <input type="number" id="total" value="">
13       <button onclick="ler_notas()">INICIAR</button>
14       -->
15       <p id="resultado"></p>
16
17       <script src="script.js"></script>
18    </body>
19 </html>
20
```

```
1  var cores = ["Verde","Amarelo","Azul","Branco"];
2  var resposta = "";
3
4  /* Testando toString */
5  resposta = cores.toString();
6
7  /* Testando join e split */
8  //resposta = cores.join(" + ");
9  //resposta = resposta.split("+");
10 //resposta = resposta.replace(/\s/g,"").split("+");
11 //resposta = resposta.split(" + ");
12
13 /* Testando push */
14 //var x = cores.push("Preto");
15 //resposta = "Novo array ["+cores+"] tem tamanho "+x;
16
17 /* Testando pop */
18 //cores.push("Preto");
19 //var x = cores.pop();
20 //resposta = "Novo array ["+cores+"] teve removido "+x;
21
22
23 document.getElementById("resultado").innerHTML = resposta;
24
```

No JavaScript, declaramos a variável `cores`, que é o nosso array, visto no exemplo que tem `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"`. Declaramos também a variável `resposta`, que é um texto. E essa `resposta`, veja no final da linha 23, será utilizada para escrevermos na página.

Inicialmente, testamos o `toString`. Veja na Figura 6, que `resposta` será o `toString` aplicado ao array `cores`. E ao verificarmos na página HTML, veremos que simplesmente serão exibidos os elementos Verde, Amarelo, Azul e Branco, com a vírgula entre eles. Isso acontece porque foram pegos os elementos do array `"Verde"`, `"Amarelo"`, `"Azul"` e `"Branco"`, e acrescentada uma vírgula entre eles, e essa foi a *string* resultante.

Figura 6 - Usando `toString`

Iremos testar o `join` e o `split`. Primeiro, testamos o `join`. O que acontece se pegarmos o array `cores` e aplicarmos o `join`, usando essa *string* da Figura 7: espaço em branco + espaço em branco?

Figura 7 - Usando `join`

Ao carregar a página, será colocado, de fato, " + ", entre os elementos do array na *string* gerada por ele. Caso queira dar um `split`, então `resposta` pegará a *string* que ele gerou e dará um `split` usando o +. Veja que não coloquei o espaço em branco. Recarregue a página, e observe que o + foi substituído por vírgula. Ou seja, a *string* resultante é o método `toString` aplicado ao novo array, que é `"Verde"`, espaço em branco com espaço em branco, `"Amarelo"`, espaço em branco com espaço em branco, e assim por diante.

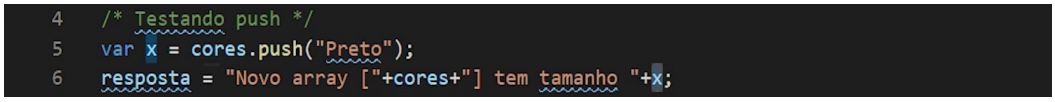
Se quisermos, efetivamente, remover os espaços em branco, há duas maneiras de fazer isso, como já foi dito. Ou damos um `replace` nessa *string*, usando a expressão regular `/\s/g` que, como já vimos, remove todos os espaços em branco e, em seguida, damos um `split` no +. Veja essa ação na linha 06 da Figura 9. Ao carregar a página HTML, você verá que teremos a *string* que vimos no começo deste exemplo: Verde, Amarelo, Azul, Branco.

Figura 8 - Usando `replace` e `split`

Outra maneira de fazer isso é usar a mesma *string* que fizemos o `join`, no `split`, ou seja, " + ". E ao recarregar a página, você terá o mesmo resultado.

Veremos como funciona o `push` e o `pop`. Primeiro, testamos o `push`. Diremos que a variável `x` recebe `cores.push` de `"Preto"`. Em seguida, imprimimos na tela "Novo array", escrevendo o novo array e informando o tamanho dele, como exposto na linha 6 da Figura 9. Para escrever o tamanho dele, use `x`, que recebeu o retorno da chamada do método `push` passando `"Preto"` no array `cores`.

Figura 9 - Usando `push`



E ao carregar a página HTML, veja na Figura 10 que o novo array, de fato, agora tem o `"Preto"` como último elemento, e foi retornado no método o novo tamanho do novo array, e como temos agora cinco elementos, o tamanho 5. Certo?

Figura 10 - Testando o `push`

Agora, testamos o `pop`. Usamos o `push` para colocar `"Preto"` no array, depois damos um `pop` nele e atribuímos isso a `x`. Diremos que `resposta` é o novo array e também qual foi o elemento removido usando, exatamente, o valor da variável `x` que recebeu na linha 6 o retorno da chamada do método `pop`. Veja essa ação na Figura 11.

Figura 11 - Usando `pop`

```
4   /* Testando pop */
5   cores.push("Preto");
6   var x = cores.pop();
7   resposta = "Novo array ["+cores+"] teve removido "+x;
8
```

Ao carregar a página HTML, você verá que novo array, de fato, não terá mais o elemento "`Preto`", e ele emitirá a mensagem "Novo array [Verde, Amarelo, Azul, Branco] teve removido Preto". Tá certo?

Esses são pequenos exemplos, uma oportunidade de colocar em prática os exemplos que vimos nesta videoaula.