Iteradores em JavaScript

Conhecimentos	
Data da aula	@7 de agosto de 2024
Tipo	Atividade em Classe · Aula Expositiva · Resolução de Exercícios Práticos

Serviço Nacional de Aprendizagem Comercial

Departamento Regional de Pernambuco

Habilitação Profissional Técnica em Informática

UC9 - Desenvolver Algoritmos

Thiago Nogueira - Instrutor de Educação Tecnológica

✉️ thiago.nogueira@senac.pe.br

ℹ️

linkedin.com/tdn

📱

(81) 9 9627-0419

📷

@thiagoo._nogueiraa

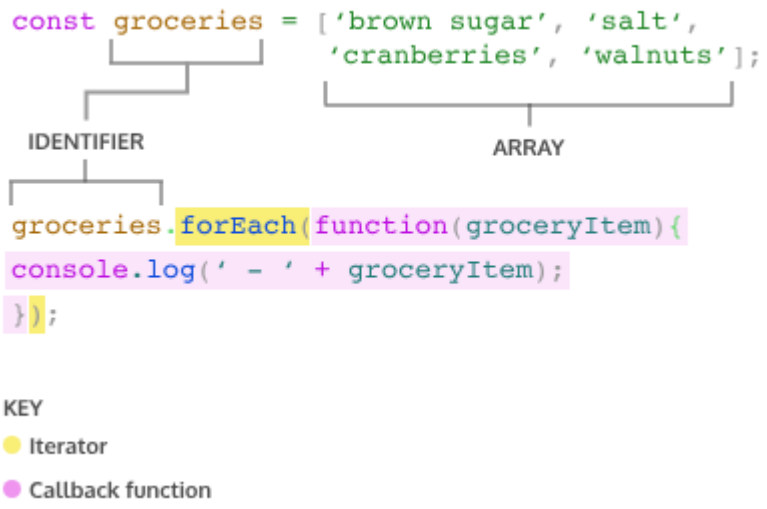
Iteradores

Imagine que você tivesse uma lista de compras e quisesse saber o que era cada item da lista. Você teria que examinar cada linha e verificar o item. Essa tarefa comum é semelhante ao que temos que fazer quando queremos iterar ou percorrer um array. Uma ferramenta à nossa disposição é o `for`. No entanto, também temos acesso a métodos de array integrados que facilitam a iteração.

Os métodos de array JavaScript integrados que nos ajudam a iterar são chamados de *métodos de iteração*, às vezes chamados de *iteradores* . Iteradores são métodos chamados em arrays para manipular elementos e retornar valores.

O método .forEach()

O primeiro método de iteração que aprenderemos é `.forEach()`. Ele executará o mesmo código para cada elemento de um array.



Estrutura do .forEach()

O código acima registrará uma lista bem formatada de mantimentos no console. Vamos explorar a sintaxe de invocação do `.forEach()`.

- `groceries.forEach()` chama o método `forEach` no array `groceries`.
- `.forEach()` recebe um argumento da *callback function*. Uma callback function é uma função passada como argumento para outra função.
- `.forEach()` percorre o array e executa o callback function para cada elemento. Durante cada execução, o elemento atual é passado como argumento para a callback function.

Outra maneira de passar um callback function para o `.forEach()` é usar a sintaxe da arrow function.

```
groceries.forEach(groceryItem => console.log(groceryItem));
```

O método .map()

O segundo iterador que abordaremos é `.map()`. Quando `.map()` é chamado em um array, ele recebe um argumento de uma *callback function* e retorna um novo array! Dê uma olhada em um exemplo de chamada `.map()`:

```
const numbers = [1, 2, 3, 4, 5];

const bigNumbers = numbers.map(number => {
```

```
  return number * 10;
});
```

`.map()` funciona de maneira semelhante a `.forEach()`- a principal diferença é que `.map()` retorna um novo array.

No exemplo acima:

- `numbers` é uma matriz de números.
- `bigNumbers` armazenará o valor de retorno da chamada de `.map()` no array `numbers`.
- `numbers.map` irá iterar através de cada elemento no array `numbers` e passar o elemento para a callback function.
- `return number * 10` é o código que desejamos executar em cada elemento do array. Isso salvará cada valor do array `numbers`, multiplicado por `10`, em um novo array.

Se dermos uma olhada em `numbers` e `bigNumbers`:

```
console.log(numbers); // Output: [1, 2, 3, 4, 5]
console.log(bigNumbers); // Output: [10, 20, 30, 40, 50]
```

Observe que os elementos `numbers` não foram alterados e `bigNumbers` é um novo array.

O método .filter()

Outro método iterador útil é `.filter()`. Como `.map()`, `.filter()` retorna um novo array. No entanto, `.filter()` retorna uma matriz de elementos após filtrar certos elementos da matriz original.

A callback function do método `.filter()` deve retornar `true` ou `false` dependendo do elemento que é passado para ele. Os elementos que fazem com que a função retorne `true` são adicionados à nova matriz. Por exemplo:

```
const words = ['chair', 'music', 'pillow', 'brick', 'pen', 'd

const shortWords = words.filter(word => {
```

```
  return word.length < 6;
});
```

- `words` é uma matriz que contém elementos de string.
- `const shortWords =` declara uma nova variável que armazenará o array retornado da invocação `.filter()`.
- A callback function é uma arrow function que possui um único parâmetro, `word`. Cada elemento do array `words` será passado para esta função como um argumento.
- `word.length < 6;` é a condição na callback function. Qualquer elemento da matriz `words` que tenha menos de `6` caracteres será adicionada à matriz `shortWords`.

Vamos também verificar os valores de `words` e `shortWords`:

```
console.log(words); // Output: ['chair', 'music', 'pillow', '
console.log(shortWords); // Output: ['chair', 'music', 'brick
```

Observe como `words` não foi alterado, e `shortWords` é um novo array.

O método .findIndex()

Às vezes queremos encontrar a localização de um elemento em um array. É aí que método `.findIndex()` entra! Chamar `.findIndex()` em um array retornará o índice do primeiro elemento avaliado `true`.

```
const jumbledNums = [123, 25, 78, 5, 9];

const lessThanTen = jumbledNums.findIndex(num => {
  return num < 10;
});
```

- `jumbledNums` é uma matriz que contém elementos que são números.
- `const lessThanTen =` declara uma nova variável que armazena o número do índice retornado pela invocação `.findIndex()`.
- A callback function é uma arrow function que possui um único parâmetro, `num`. Cada elemento do `jumbledNums` será passado para esta

função como um argumento.

- `num < 10;` é a condição da qual os elementos são verificados. `.findIndex()` retornará o índice do primeiro elemento avaliado `true` para essa condição.

Vamos dar uma olhada no que `lessThanTen` é avaliado:

```
console.log(lessThanTen); // Output: 3
// Se verificarmos qual elemento tem índice de 3:
console.log(jumbledNums[3]); // Output: 5
```

Ótimo, o elemento no índice `3` é o número `5`. Isso faz sentido, pois `5` é o primeiro elemento menor que 10.

Se não houver um único elemento no array que satisfaça a condição , `.findIndex()` retornará `-1`.

O Método .reduce()

Outro método de iteração amplamente utilizado é o `.reduce()`. Ele retorna um único valor após iterar pelos elementos de um array, reduzindo-o. Dê uma olhada no exemplo abaixo:

```
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue)
  return accumulator + currentValue
})

console.log(summedNums) // Output: 17
```

Aqui estão os valores de `accumulator` e `currentValue` conforme iteramos pelo array `numbers`:

Iteração	accumulator	currentValue	valor de retorno
Primeiro	1	2	3
Segundo	3	4	7
Terceiro	7	10	17

Agora vamos examinar o uso `.reduce()` do exemplo:

- `numbers` é uma matriz que contém números.
- `ummedNums` é uma variável que armazena o valor retornado da invocação `.reduce()`.
- `numbers.reduce()` chama o método no array `numbers` e recebe uma callback function como argumento.
- A função de retorno de chamada possui dois parâmetros `accumulator` e `currentValue`. O valor de `accumulator` começa como o valor do primeiro elemento na matriz e `currentValue` começa como o segundo elemento.

O `.reduce()` também pode receber um segundo parâmetro opcional para definir um valor inicial de `accumulator` (lembre-se, o primeiro argumento é a callback function!). Por exemplo:

```
const numbers = [1, 2, 4, 10];

const summedNums = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue
}, 100)  // <- Second argument for .reduce()

console.log(summedNums); // Output: 117
```

Iteração #	accumulator	currentValue	valor de retorno
Primeiro	100	1	101
Segundo	101	2	103
Terceiro	103	4	107
Quarto	107	10	117

Atividades de Aprendizagem

Exercício 1

Crie um array chamado frutas com 4 valores de frutras. Itere sobre o array para registrar 'Eu quero comer um ' mais o nome de cada fruta no console.

Por exemplo: Eu quero comer um morango.

Exercício 2

Crie um array de animais contendo 7 valores (string). Use o método .map() para criar outro array, chamado primeiraLetra que contém o primeiro caractere de cada string do array animais.

Exercício 3

Crie um array de numeros contendo 5 valores inteiros. Use o método .map() para criar outro array, chamado numerosDivididos que contém todos os elementos do array numeros divididos por 100.

Exercício 4

Crie um array com 5 números aleatórios (utilize o método random). Chame o método `.filter()` para retornar valores menores que `250`. Salve-os em um novo array chamado smallNumbers.

Exercício 5

Crie um array chamado animais com 5 elementos strings de animais diversos. Chame `.findIndex()` no array `animais` e retorne o índice do primeiro elemento que começa com `'s'`. Atribua o valor retornado a uma variável chamada `comecaComS`.

Exercício 6

Crie um array de números, contendo 8 valores. utilize o método .reduce() para retornar a soma de todos os valores do array. Para isso, atribua o método no array criado a uma outra variável e chame-a através do console.log(). Após isso imprima os valores somados por iteração.

Atividades Extras

Exercício 1 - Magic 8 Ball

Escreva um programa que, imprima na tela uma pergunta e dada a pergunta, responda aleatoriamente uma das respostas a seguir:

- Sim
- Não sei, só sei que foi assim
- Não conte com isso
- Minhas fontes dizem que não

- Perspectiva não tão boa
- Sinais apontam para que sim
- Definitivamente não
- Difícil prever, tente novamente

Exercício 2 - Dia de corrida

Você foi contratado para escrever um programa que registrará corredores para a corrida e lhes dará instruções no dia da corrida. Veja como funciona nosso cadastro. Existem corredores adultos (maiores de 18 anos) e corredores juvenis (menores de 18 anos). Eles podem se registrar mais cedo ou mais tarde. Os corredores recebem um número de corrida e horário de início com base em sua idade e inscrição.

Número da corrida:

- Os primeiros adultos recebem um número de corrida igual ou superior a 1000.
- Todos os outros recebem um número abaixo de 1000.

Hora de início:

- Os inscritos adultos correm às 9h30 ou 11h00.
    - Os primeiros adultos correm às 9h30.
    - Adultos atrasados correm às 11h.
- Os inscritos jovens correm às 12h30 (independente de inscrição).

Dica: crie três variáveis para armazenar número de corrida (tente utilizar o método random), se foi inscrito e idade do corredor.