

Executar os processos de codificação, manutenção e documentação de aplicativos computacionais para desktop

SENAC PE

Thiago Nogueira

Instrutor de Educação Profissional

JavaScript – Arrays

Arrays

Definição

- Arrays são a maneira do JavaScript criar listas.
- Arrays podem armazenar qualquer tipo de dados (incluindo strings , números e booleanos).
- Assim como as listas, os arrays são ordenados, o que significa que cada item possui uma posição numerada.

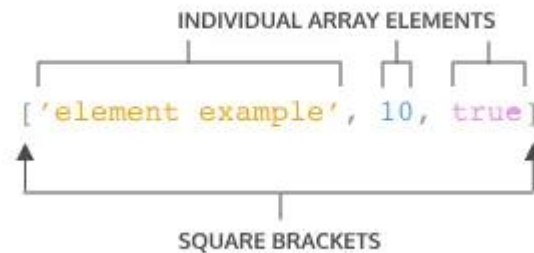


```
let newYearsResolutions = ['Manter um diário', 'Fazer uma aula de dança', 'Aprender a fazer malabarismos'];
```

Arrays

Criando Arrays

- Uma maneira de criar um array é usar um *array literal*.
- Um array literal cria um array colocando os itens entre colchetes `[]`.
- Os arrays podem armazenar qualquer tipo de dados - podemos ter um array que contém todos os mesmos tipos de dados ou um array que contém diferentes tipos de dados.



- Também podemos salvar um array em uma variável

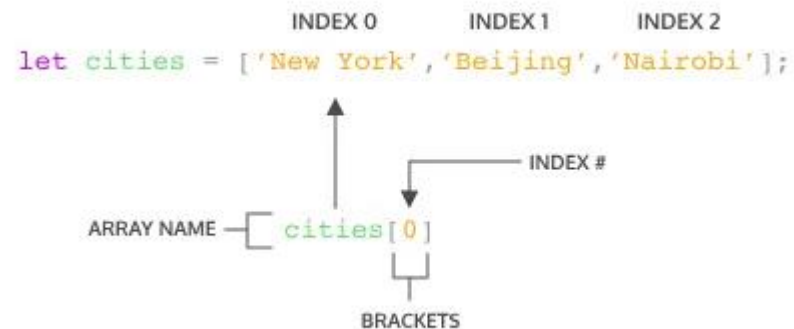


```
let newYearsResolutions = ['Keep a journal', 'Take a falconry class', 'Learn to juggle'];
```

Arrays

Acessando Elementos


- Cada elemento em um array possui uma posição numerada conhecida como *índice*.
- Podemos acessar itens individuais usando seu índice, o que é semelhante a fazer referência a um item em uma lista com base na posição do item.
- Arrays em JavaScript são *indexadas em zero*, o que significa que as posições começam a contar em **0** ao invés de **1**. Portanto, o primeiro item de um array estará na posição **0**.



Arrays

Acessando Elementos

- Também pode-se acessar caracteres individuais em uma string usando a notação de colchetes e o índice.



```
const hello = 'Hello World';  
console.log(hello[6]);  
// Output: W
```

Arrays

Atualizar elementos

- Depois de ter acesso a um elemento em um array, você pode atualizar seu valor.



```
let estacoes = ['Inverno', 'Primavera', 'Verão', 'Outono'];  
  
estacoes[3] = 'Mormaço';  
console.log(estacoes);  
//Output: ['Inverno', 'Primavera', 'Verão', 'Mormaço']
```

Arrays

Arrays com let e const

- Variáveis declaradas com **let** podem ser reatribuídas.
- Variáveis declaradas com a **const** não podem ser reatribuídas. No entanto, os elementos de um array declarado com **const** permanecem mutáveis. O que significa que podemos alterar o conteúdo de um array **const**, mas não podemos reatribuir um novo array ou um valor diferente.

```
// Código Exemplo
let condimentos = ['Ketchup', 'Mostarda', 'Molho de Soja', 'BBK'];

const utensilios = ['Garfo', 'Faca', 'Hachi', 'Garfo'];

condimentos[0] = 'Mayo'
console.log(condimentos)
// Output: ['Mayo', 'Mostarda', 'Molho de Soja', 'BBK']
condimentos = ['Mayo']
console.log(condimentos)
// Output: ['Mayo']

utensilios[3] = 'Colher'
console.log(utensilios)
// Output: ['Garfo', 'Faca', 'Hachi', 'Colher'];
```


Arrays

A propriedade .length

- Uma das propriedades integradas de um array é o **length** que retorna o número de itens no array.
- Acessamos a propriedade **.length** da mesma forma que fazemos com strings.



```
const metasDeAnoNovo = ['Manter um diário', 'Fazer uma aula de dança'];  
  
console.log(metasDeAnoNovo.length);  
//Output: 2
```

Arrays

O método .push()

- Um método `.push()` nos permite adicionar itens ao final de um array.



```
const itemRastreador = ['item 0', 'item 1', 'item 2'];  
  
itemRastreador.push('item 3', 'item 4');  
  
console.log(itemRastreador);  
// Output: ['item 0', 'item 1', 'item 2', 'item 3', 'item 4'];
```

Arrays

O método .pop()

- Outro método de array é o **.pop()** que remove o último item de um array.
 - **.pop()** não aceita nenhum argumento, simplesmente remove o último elemento
 - **.pop()** retorna o valor do último elemento
 - **.pop()** é um método que altera o array inicial

```
const novoItemRastreador= ['item 0', 'item 1', 'item 2'];

const removido = newItemTracker.pop();

console.log(novoItemRastreador);
//Saída: ['item 0', 'item 1']
console.log(removido);
//Saída: item 2
```

Arrays

Outros métodos

- Existem muito mais métodos de array do que apenas `.push()` e `.pop()`. Você pode ler sobre esses métodos de array no documento [JavaScript Array Methods](#).
- `pop()` e `.push()` alteram o array no qual eles são chamados. No entanto, há momentos em que não queremos alterar o array original e podemos usar métodos de array sem fazer mutações.
- Alguns métodos de arrays disponíveis para desenvolvedores JavaScript incluem: `.join()`, `.slice()`, `.splice()`, `.shift()`, `.unshift()` e `.concat()`

Arrays

Arrays aninhados

- Os arrays podem armazenar outros arrays. Quando um array contém outro array, ele é conhecido como *array aninhado*.



```
const nestedArr = [[1], [2, 3]];
```

- Para acessar os arrays aninhados podemos usar a notação de colchetes com o valor do índice, assim como fizemos para acessar qualquer outro elemento:



```
const nestedArr = [[1], [2, 3]];
```

```
console.log(nestedArr[1]); // Output: [2, 3]
```

JavaScript – Estruturas de Repetição

Estruturas de Repetição

Loop for

- Um loop **for** contém três expressões separadas por **;** (ponto e vírgula):
- 1. uma *inicialização*: Inicia o loop e também pode ser usada para declarar a variável iteradora.
- 2. uma *condição de parada*: É a condição com a qual a variável do iterador é avaliada - se a condição for avaliada como verdadeiro, o bloco de código será executado e se for avaliada como falso, o código será interrompido.
- 3. uma *instrução de iteração*: É usada para atualizar a variável do iterador em cada loop.

```
// Exemplo
for (let counter = 0; counter < 4; counter++) {
  console.log(counter);
}
// Output: 0
//         1
//         2
//         3
```

Estruturas de Repetição

Loop reverso

Para executar um loop reverso com **for** , devemos:

- Definir a variável iteradora para o valor desejado mais alto na expressão de inicialização.
- Defina a condição de parada para quando a variável do iterador for menor que o valor desejado.
- O iterador deve diminuir em intervalos após cada iteração.

Estruturas de Repetição

percorrendo arrays

- As estruturas de repetição **for** são muito úteis para iterar estruturas de dados. Por exemplo, podemos usar um loop **for** para **realizar a mesma operação em cada elemento de um array**.
- Para percorrer cada elemento de um array, um loop **for** deve usar a propriedade do array **.length** em sua condição.



```
const animals = ['Grizzly Bear', 'Sloth', 'Sea Lion'];
for (let i = 0; i < animals.length; i++){
  console.log(animals[i]);
}
// Output: Grizzly Bear
//          Sloth
//          Sea Lion
```

Estruturas de Repetição

Loops aninhados

- Quando temos um loop rodando dentro de outro loop, chamamos isso de *loop aninhado*.
- Um uso para um loop **for** aninhado é comparar os elementos em dois arrays. **Para cada iteração do loop **for** externo, o loop **for** interno será executado completamente.**

```
const myArray = [6, 19, 20];
const yourArray = [19, 81, 2];
for (let i = 0; i < myArray.length; i++) {
  for (let j = 0; j < yourArray.length; j++) {
    if (myArray[i] === yourArray[j]) {
      console.log('Both arrays have the number: ' + yourArray[j]);
    }
  }
}
```

Estruturas de Repetição

Loop While

- vamos converter um loop **for** em loop **while**:



```
// Um for que retorna 1, 2, e 3  
for (let counterOne = 1; counterOne < 4; counterOne++){  
    console.log(counterOne);  
}  
  
// Um while que retorna 1, 2, e 3  
let counterTwo = 1;  
while (counterTwo < 4) {  
    console.log(counterTwo);  
    counterTwo++;  
}
```

Estruturas de Repetição

Do While

- Em alguns casos, você deseja que um trecho de código seja executado pelo menos uma vez e, em seguida, faça um loop com base em uma condição específica após sua execução inicial.
- Uma instrução **do...while** diz para executar uma tarefa uma vez e depois continuar a fazê-la até que uma condição especificada não seja mais atendida.

```
let countString = '';  
let i = 0;  
  
do {  
  countString = countString + i;  
  i++;  
} while (i < 5);  
  
console.log(countString);
```

Estruturas de Repetição

break

- Quando queremos impedir que um loop continue a ser executado mesmo que a condição de parada original que escrevemos para nosso loop não tenha sido atendida, podemos usar a palavra-chave **break**.
- O **break** permite que os programas “saiam” do loop a partir do bloco do loop.



```
for (let i = 0; i < 99; i++) {  
  if (i > 2 ) {  
    break;  
  }  
  console.log( 'Banana.' );  
}
```

```
console.log( 'Laranja, você está feliz por eu ter quebrado o ciclo!');
```

JSDoc

JSDoc

Resumo

- Para documentar Javascript, é bom o uso do padrão descrito em JSDoc, cuja página com documentação explicativa é <https://jsdoc.app/>.
- Como instalar:
<https://github.com/jsdoc/jsdoc>
- Como utilizar:
<https://dev.to/cristuker/o-que-e-jsdoc-fdc>

```
/**
 * Esta é uma função de exemplo de uso de JSDoc
 *
 * @example
 *   exemplo(3, 5); // 8
 *
 * @param {Number} obrigatorio   Parametro obrigatório
 * @param {Number} [opcional]    Parametro opcional. Note os '[' ]'
 * @returns {Number}
 */
function exemplo (obrigatorio, opcional) {
  var resultado = 0;
  resultado = obrigatorio + (opcional || 0);
  return resultado;
}
```

Exercícios

Exercícios

Instruções


- Responder os exercícios propostos da aula 11
- Cada questão é resolvida em um arquivo .js separado seguindo o nome modelo:
 - **senac_UC13_aula11_qx.js** onde o x é substituído pelo número da questão
- Documentar o que cada questão pede e como foi resolvida em um arquivo README.md
 - Utilize o padrão JSDoc para comentar cada questão
- Enviar os exercícios para um repositório no GitHub
- Anexar link do repo na atividade do Teams


*“Ensinar é impregnar
de sentido o que
fazemos a cada
instante”*


Paulo Freire

Obrigado!

Thiago Nogueira

 [linkedin.com/tdn](https://www.linkedin.com/tdn)

 thiago.nogueira@pe.senac.br

 (81) 9 9627-0419