

SENAC  
Campus Santo Amaro

TADS - Análise Desenvolvimento de Sistemas

*PW - Programação Web*



# Aula #3 JavaScript:Módulo II

Professor: Veríssimo - carlos.hypereira@sp.senac.br

23/08/2022

### **Sobre este documento**

Este documento objetiva deixar registrado o conteúdo abordado em sala de aula pelo professor. Importante destacar que a Nota de Aula serve como guia ao professor, bem como serve aos alunos como um norte, quanto ao conteúdo desenvolvido em sala de aula.

Este documento não tem a pretensão de ser uma única fonte para estudo. Para tal, o aluno deverá assistir às aulas e fazer uso (consulta) à bibliografia recomendada na ementa da disciplina, e à bibliografia complementar, apontada pelo professor.

## Preâmbulo da Aula

Esta aula aborda **Funções Anônimas; Tratamento de Arrays; Tratando fluxo: If, Switch e loops.**

Importante destacar que esta aula possui uma abordagem prática, na qual os elementos conceituais servem de guia para a parte prática da aula.



# Contents

<b>1</b>	<b>Funções Anônimas</b>	<b>1</b>
1.0.1	Funções Anônimas - Sintaxe Básica . . .	2
1.0.2	Funções Anônimas: Passando um só parâmetro	3
<b>2</b>	<b>Arrays</b>	<b>5</b>
2.0.1	Criando um Array . . . . .	6
2.0.2	Manipulando um Array . . . . .	6
<b>3</b>	<b>Tratamento de Fluxo</b>	<b>9</b>
3.0.1	Estruturas Condicionais . . . . .	9
3.0.2	Comando Swich . . . . .	16
3.0.3	Estruturas while e do...while e for . . . .	18
3.0.4	Iterações forEach e Map . . . . .	20



# Chapter 1

## Funções Anônimas

Vimos à aula #2; que funções são *“segmentos de códigos”*, que podem ser utilizados pelo programa, ou por outros programas.

Funções podem receber parâmetros, assim como podem retornar valores, ao seus *“chamadores”*. A listagem 1.1 mostra um exemplo de declaração e utilização de uma função chamada *soma*, que retorna o resultado da soma de dois valores.

```
1  /**-----*
2  *          SENAC - TADS - Programacao Web          *
3  *          Exemplo de Funcao                        *
4  *-----*/
5  //Aqui, declaramos a funcao
6  function somar(parcela1, parcela2){
7      let resultado = parcela1+parcela2;
8      return resultado;
9  }
10 let valor1=3;
11 let valor2=6
12 somaDosValores = somar(valor1, valor2);
13 //vamos chamar a funcao e mostrar o resultado
14 console.log('O resultado da soma de ${valor1} + ${
    valor2} =${somaDosValores} ');
```

Listing 1.1: Exemplo de Função

Iremos ver nesta seção uma outra forma de criação de funções: **Funções Anônimas**, também conhecidas como

"Arrow Function"

### 1.0.1 Funções Anônimas - Sintaxe Básica

Função anônima, ou *Arrow Function* é uma função que **não possui nenhum nome associado a ela**. Neste caso usamos apenas a palavra-chave ***function*** sem o nome da função. A sintaxe para a declaração de uma função anônima é:

() ⇒ {}

Onde:

- () = Lista de parâmetros recebidos pela função
- ⇒ valor fixo, Indicar que se trata de uma "Arrow" function
- {} = Bloco de código que representa o ***corpo da função***

Veja exemplo abaixo, a listagem 1.2, que à exemplo da figura 1.1, efetua a soma entre dois números, porém agora está no formato de "Arrow Function".

Observe que à **linha 8** atribuímos à **variável** *somaDosValores* o resultado do processamento da **Arrow Function** (Lembre-se que **Arrow Function** começa na **lista de parâmetros** (neste caso *parcela1* e *parcela2*), depois temos o símbolo seta, seguida do **corpo da função** dentro abre/fecha chaves **⇒ {}**).



```

1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *  Exemplo de Funcao Anonima (Arrow Function) *
4  *-----*/
5  let valor1=3;
6  let valor2=6
7  // Aqui estamos a definir a Arrow Function
8  let somaDosValores = (parcela1, parcela2) => {
9      let resultado = parcela1+parcela2;
10     return resultado;
11 };
12 // vamos chamar a funcao e mostrar o resultado
13 console.log('O resultado da soma de ${valor1} + ${
    valor2} =${somaDosValores(valor1,valor2)} ');

```

Listing 1.2: Exemplo de Função Arrow Function

## 1.0.2 Funções Anônimas: Passando um só parâmetro

Ao passar **somente um parâmetro**, parênteses são opcionais, conforme demonstrado na listagem 2.1.

Observe que à **linha 8** atribuímos à **variável** *nmCliente* o resultado do processamento da **Arrow Function** (Lembre-se que **Arrow Function** começa na **lista de parâmetros** (neste caso *parmNomeCliente*), depois temos o símbolo seta, seguida do **corpo da função** dentro abre/fecha chaves **⇒{}**).

```

1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *  Exemplo de Funcao anonima com um parametro*
4  *-----*/
5
6  nomeCliente="Joaquim Jos da Silva Xavier";
7  //Defincao de funcao Anonima passando somente um
    parametro
8  let nmCliente = parmNomeCliente => {
9      console.log("Bem vindo Sr."+parmNomeCliente)
10 }

```

```
11 nmCliente(nomeCliente);
```

Listing 1.3: Exemplo de Função Anônima com um parâmetro

Observe aqui à **linha 11** que estamos a chamar a função anônima, passando um único valor como parâmetro (No caso o conteúdo da variável *nomeCliente*)

# Chapter 2

## Arrays

Um Array é uma variável especial que contém mais de um valor, ou seja, é ula lista de vaores.

Por exemplo: Você poderá ter a necessidade de tratar uma lista de produtos vendidos na loja:

Table 2.1: Lista de produtos na Loja

Item	Produto
#1	Camisa manga madura
#2	Chinelos Ava Iana
#3	Galochas happy horse
#4	Metacapacete magntico
#5	Luvas Biônicas

## 2.0.1 Criando um Array

Usar um literal de array é a maneira mais fácil de criar um array em JavaScript. A syntax para declaração de array é:

```
const array_name = [item1, item2, ...];
```

Exemplo:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Exemplo de array                    *
4  *-----*/
5
6  // Delcarando array
7  const listaDeProdutos = ["Camisa manga madura",
8                          "Chinelos Ava Iana",
9                          "Galochas happy horse",
10                         "Metacapacete magntico",
11                         "Luvas Bionicas"
12                         ];
13 //aqui estamos a mostrar o conteudo do array.
14 console.log("Lista de Produtos:\n"+listaDeProdutos
15 );
```

Listing 2.1: Exemplo declaração de Array

Oberve o intervalo que compreende da **linhas #7** a **linha #12**, onde fazemos a declaração da lista (array): A lista contém 5 itens (produtos).

## 2.0.2 Manipulando um Array

Veremos agora as formas básicas de manipulação de array. No capítulo que trataremos de loo de repetição, voltaremos a abordar a manipulação de array.

A listagem 3.12 abaixo, podemos ver as várias formas de manipular array:

```
1  /**-----*
2  *          SENAC - TADS - Programacao Web          *
3  *                      Exemplo de array            *
4  *-----*/
5
6  // Delcarando array
7  const listaDeProdutos = ["Camisa manga madura",
8                           "Chinelos Ava Iana",
9                           "Galochas happy horse",
10                          "Metacapacete magntico",
11                          "Luvas Bionicas"
12                          ];
13 //obtendo o tamanho da lista
14 console.log("Lista de Produtos possui "+
15             listaDeProdutos.length+" itens");
16 //-----
17 //acessando um item especifico
18 console.log("O primeiro item da lista e: "+
19             listaDeProdutos[0]+" e o ultimo eh: "+
20             listaDeProdutos[4]);
21 //acessando um item especifico
22 console.log("Outra forma de chegar no ultimo item
23 : "+listaDeProdutos[listaDeProdutos.length-1]);
24 /*pesquisando um item na lista
25     devolve a posicao naqual se encontra o argumento
26     de pesquisa se nao encontrar o item, devolve
27     -1 (um negativo)
28 */
29 console.log(listaDeProdutos.indexOf("Galochas
30 happy horse"));
31 //
32 //alterando um item (Em nosso exemplo: item "
33     Chinelos Ava Iana" ser alterado para "Chinelo
34     do Havai")
35 listaDeProdutos[1]="Chinelo do Havai";
36 console.log(listaDeProdutos);
37 //
38 //Adicionando um item na lista
39 listaDeProdutos.push("Sacolas de palha de Coco
40 Verde");
41 console.log(listaDeProdutos);
42 console.log("Novo tamanho da lista:  "+
```

```
    listaDeProdutos.length);
33 //
34 //removendo o primeiro item da listaDeProdutos: "
    Camisa manga madura"
35 listaDeProdutos.shift();
36 console.log(listaDeProdutos);
37 //
38 //removendo o ultimo item da listaDeProdutos: "
    Sacolas de palha de Coco Verde"
39 listaDeProdutos.pop();
40 console.log(listaDeProdutos);
41 //
42 /*Remover um elemento de acordo com sua posicao;
    Nosso exemplo "Galochas happy horse" neste
    caso, proceder:
43     1 - Localizar o indice do item desejado
        : metodo "indexOf"
44     2 - usar metodo splice (se indice for
        diferente de -1, eh claro)
45     O metodo splice recebe dois parametros:
46     o primeiro eh a posicao no array e
47     o segundo eh a quantidade de elementos
        a remover
48 */
49 let indiceDoItem=listaDeProdutos.indexOf("Galochas
    happy horse");
50 if (indiceDoItem>-1){
51     listaDeProdutos.splice(indiceDoItem,1);
52     console.log("Item excluido com Sucesso!!!");
53 }
54 console.log(listaDeProdutos);
```

Listing 2.2: Exemplo manipulacao de Array

# Chapter 3

## Tratamento de Fluxo

### 3.0.1 Estruturas Condicionais

As declarações condicionais, em qualquer linguagem de programação, nos permitem representar tomadas de decisão, a partir da escolha que deve ser feita.

Abordaremos aqui as seguintes estruturas condicionais:

- Estrutura de Decisão simples: *if*
- Estrutura de Decisão *if...else*
- Encadeamento de decisões ( *if...else* encadeado )

#### Estrutura de Decisão *if*

Esta estrutura permite avaliar uma condição e, a partir dela, executar um bloco de código, somente se o resultado for **Verdadeiro**.

Conforme demonstrado à figura 3.1 podemos verificar que esta estrutura permite a execução de um bloco de código, de

acordo com o resultado da análise da condição (Comando *if*)

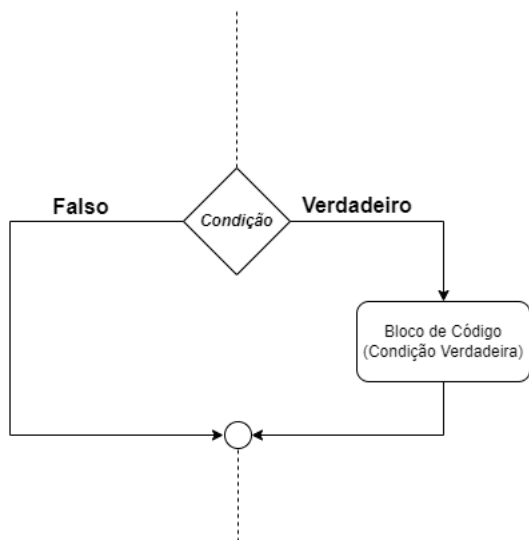


Figure 3.1: Estrutura de decisão if

A **sintaxe** para declarar um comando *if* é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe comando if                  *
4  *-----*/
5  if (condicao){
6  //bloco de codigo: Condicao Verdadeira
7  }
```

Listing 3.1: Sintaxe da estrutura if simples



## Comando *if...else*

*Esta estrutura condicional permite avaliar uma condição e, a partir dela, executar diferentes linhas de código.*

*Conforme demonstrado à figura 3.2 podemos verificar que esta estrutura permite a execução de um bloco de código verdadeiro ou um bloco de código falso, de acordo com o resultado da análise da condição (Comando *if*)*

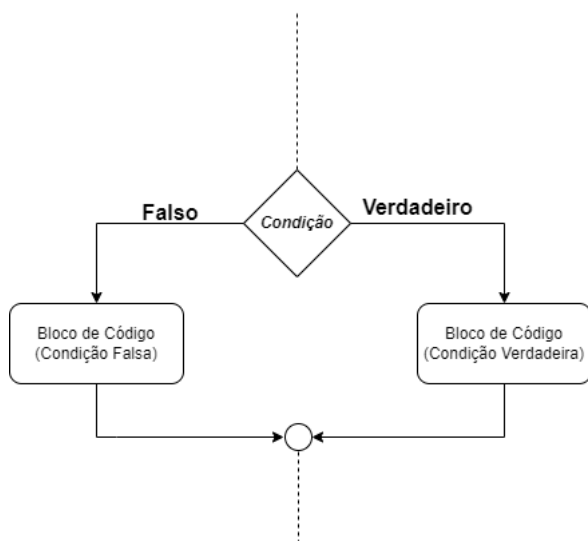


Figure 3.2: Estrutura de decisão *if...else*

A **sintaxe** para declarar um comando *if...else* é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe comando if...else           *
4  *-----*/
5  if (condicao){
6      //bloco de codigo: Condicao Verdadeira
7  } else {
8      //bloco de codigo: Condicao Falsa
9  }
```

Listing 3.2: Sintaxe da estrutura if...else

### Comando *if...else* encadeados

Esta estrutura condicional valiar permite *sucessivas* condições e, a partir dela, executar diferentes blocos de código.

Conforme demonstrado à figura 3.3 podemos verificar que esta estrutura permite a execução de um bloco de código verdadeiro ou um bloco de código falso, de acordo com o resultado da análise da condição (Comando *if*)

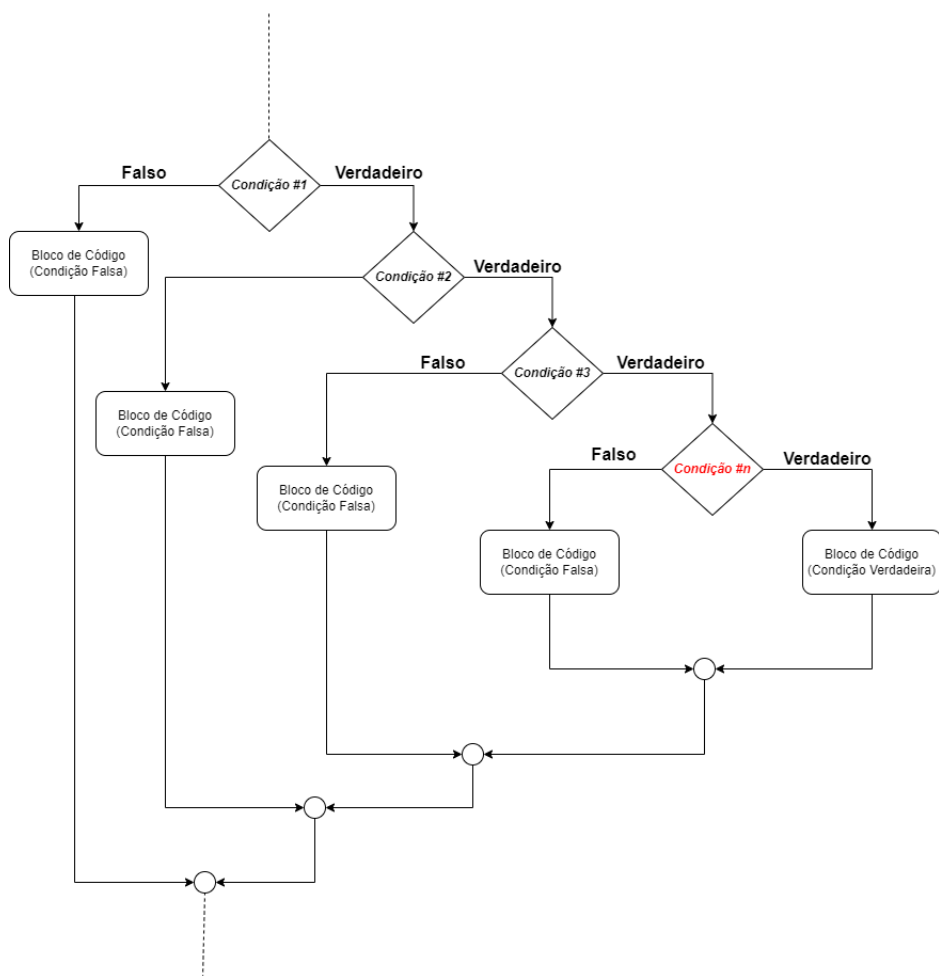


Figure 3.3: Estrutura de if...else encadeados

A *sintaxe* para declarar *if...else encadeados* é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe comando if...else encadeados *
4  *-----*/
5  if (condicao#1){
6      if (condicao#2) {
7          if (condicao#3) {
8              if (condicao#n) {
9                  // bloco#n de codigo: Condicao
10                 verdadeira
11             } else {
12                 //bloco#n de codigo: Condicao
13                 Falsa
14             }
15         } else {
16             //bloco#3 de codigo: Condicao Falsa
17         }
18     } else {
19         //bloco#2 de codigo: Condicao Falsa
20     }
21 }
```

Listing 3.3: Sintaxe da estrutura if...else encadeado

## Operadores Condicionais

O quadro 3.1 mostra os operadores de comparação na linguagem *JavaScript*.

Table 3.1: OPeradores de comparação JavaScript

<i>Operador</i>	<i>Operação</i>	<i>Exemplo</i>
$>$	Maior que	$(a > b)$
$<$	Menor que	$(a < b)$
$\geq$	Maior ou igual a	$(a \geq b)$
$\leq$	Menor ou igual a	$(a \leq b)$
$==$	Igual a	$(a == b)$
$\neq$	Diferente a	$(a \neq b)$
$===$	Idêntico a	$(a === b)$
$\neq$	Não Idêntico a	$(a \neq b)$
$\&\&$	E/AND	$(a\&\&b)$
$\ \ $	OU/OR	$(a\ \ b)$

### 3.0.2 Comando Switch

A instrução `switch` é usada para executar diferentes ações com base em diferentes condições. Este comando é uma alternativa ao comando `if`.

A sintax para declaração de `switch` é:

```
1 switch(expression) {  
2     case x:  
3         // code block  
4         break;  
5     case y:  
6         // code block  
7         break;  
8     default:  
9         // code block  
10 }
```

Listing 3.4: Sintax para declarar Switch

A listagem 3.5 abaixo, podemos ver a utilização do `switch`.

Observer os seguintes pontos:

- **linha 8** inicia-se o comando, utilizando a variável `diaObtido`
- O comando `break` dever ser colocado dentro de cada `case` obtido
- à **linha 30**, o comando `default` serve para identificar que os testes feitos em todos comandos `case`, não foram atendidos.

```
1  /**-----*
2  *          SENAC - TADS - Programacao Web          *
3  *          Exemplo de Switch                        *
4  *-----*/
5
6  let diaObtido=new Date().getDay();
7  diaObtido=5;
8  switch (diaObtido) {
9      case 0:
10         diaDaSemana = "Domingo";
11         break;
12     case 1:
13         diaDaSemana = "Segunda-Feira";
14         break;
15     case 2:
16         diaDaSemana = "Terca-Feira";
17         break;
18     case 3:
19         diaDaSemana = "Quarta-Feira";
20         break;
21     case 4:
22         diaDaSemana = "Quinta-Feira";
23         break;
24     case 5:
25         diaDaSemana = "Sexta-Feira";
26         break;
27     case 6:
28         diaDaSemana = "Sabado";
29         break;
30     default:
31         diaDaSemana = "Dia Invalido";
32         break;
33 }
34 console.log("Hoje eh:"+diaDaSemana)
```

Listing 3.5: Exemplo manipulacao de Switch

### 3.0.3 Estruturas while e do...while e for

*Dos paradigmas da lógica de programa, a repetição de um trecho de código, é uma grande ferramenta para os programadores. As estruturas de repetição `while`; `do...while` e `for` proporcionam implementar este paradigma.*

Para quem quer ter sucesso em qualquer linguagem de programação, é imperativo ter pleno domínio, destas estruturas de repetição.

#### Repetição `while`

A **sintaxe** para declarar `while` é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe da estrutura while          *
4  *-----*/
5  while (condicao) {
6      //Bloco de comandos
7  }
```

Listing 3.6: Sintaxe da estrutura while

#### Repetição `do...while`

A **sintaxe** para declarar `do...while` é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe da estrutura do...while      *
4  *-----*/
5  do {
6      //Bloco de comandos
7  }
8  while (condicao)
```

Listing 3.7: Sintaxe da estrutura do...while



## for

A **sintaxe** para declarar *for* é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe da estrutura for            *
4  *-----*/
5  for (inicializacao; condicao; incremento) {
6      //Bloco de comandos
7  }
```

Listing 3.8: Sintaxe da estrutura for

### 3.0.4 Iterações `forEach` e `Map`

#### `forEach`

O método `forEach()` é um tipo de estrutura de repetição, só que ele é disponível para a manipulação dos elementos de um array. Ele executa uma dada função em cada elemento de um array.

A **sintaxe** para declarar `forEach` é:

```
1  /**-----*
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe da estrutura forEach          *
4  *-----*/
5  array.forEach(funcao_callback(elemento [, indice
    [, array]])[, thisArgument]);
```

Listing 3.9: Sintaxe da estrutura `forEach`

Onde:

- **array:** corresponde ao array de elementos;
- **funcao\_callback:**
  - **elemento:** elemento atual que pertence ao array e é passado como referência para a função callback;
  - **índice:** valor opcional que corresponde ao índice do elemento atual;
  - **array:** alor opcional que representa o nome do array em que o método `forEach()` será aplicado;
- **thisArgument:** valor opcional, que corresponde ao valor “this”.

Veja os três exemplos na listagem abaixo:

```

1  /**-----*
2  *          SENAC - TADS - Programacao Web          *
3  *          Exemplos de utilizacao de  forEach      *
4  *          *                                         */
5  /**-----*
6  *          Exemplo #01                             *
7  *          *                                         */
8  var arrayNumeros = [12,22,33,44,45,667];
9  function verificaPares1(elemento){
10     if (elemento % 2 == 0)
11         console.log('O numero ${elemento} eh par');
12 }
13 arrayNumeros.forEach(verificaPares1);
14 /*sa da em console:
15 O numero 12 eh par
16 O numero 22 eh par
17 O numero 44 eh par
18 */
19 /**-----*
20 *          Exemplo #02                             *
21 *          *                                         */
22 var arrayMarcas = ['Ford', 'BMW', 'Fiat', 'Audi',
23                   'Volkswagen'];
24 arrayMarcas.forEach((marca, indice) => {
25     console.log("A marca " + marca + " corresponde
26                 ao indice: " + indice);
27     if (marca === 'BMW'){
28         let itemExcluido = arrayMarcas.splice(
29             indice,1);
30         console.log("Confira: Excluimos o elemento
31                     " + itemExcluido);
32     }
33 });
34 console.log(arrayMarcas)
35 /* sa da:
36 A marca Ford corresponde ao indice: 0
37 A marca BMW corresponde ao indice: 1
38 Confira: Excluimos o elemento BMW
39 A marca Audi corresponde ao indice: 2

```

```
36 A marca Volkswagen corresponde ao indice: 3
37 [ 'Ford', 'Fiat', 'Audi', 'Volkswagen' ]
38 */
```

Listing 3.10: Dois Exemplos de forEach

## Map

O método [Map](#) é invocado a partir de um array e recebe como parâmetro uma função de **callback**, que é invocada para cada item e retorna o valor do item equivalente no array resultante.

A **sintaxe** para declarar [Map](#) é:

```

1  /**-----*/
2  *      SENAC - TADS - Programacao Web      *
3  *      Sintaxe da estrutura Map            *
4  *-----*/
5  arrayOriginal.map(funcao_callback(elementoAtual,
    ndice ,arrayOriginal),thisArgumento)

```

Listing 3.11: Sintaxe da estrutura Map

**callback** é uma função que será executada para cada elemento no vetor original e retornará uma representação dele com base em alguma lógica, que será o item equivalente no vetor resultante.

Onde:

- **arrayOriginal:** corresponde ao objeto do tipo array que contém os elementos originais;
- **funcao\_callback:**
  - **elementoAtual:** representa o elemento corrente durante o loop de execução do método;
  - **índice:** valor opcional e corresponde à posição do elementoAtual no array;
  - **arrayOriginal:** conteúdo opcional, que representa o array de origem;
- **thisArgumento:** valor opcional, que corresponde ao this utilizado pela função callback.

Exemplos de utilização de [Map](#):

```
1  /**-----*
2  *          SENAC - TADS - Programacao Web          *
3  *          Exemplo de utiliza o de Map            *
4  *-----*/
5  var numerosAleatorios = [2, 4, 6, 8]; //vetor
   original
6
7  var dobroDaLista = numerosAleatorios.map(function(
   itemAtual){
8      let itemAlterado = itemAtual * 2
9      return itemAlterado; //retorna o item original
   multiplicado por 2
10 });
11
12 console.log(dobroDaLista); //imprime [4, 8, 12,
   16]
13 //
14 /**-----*
15 *                      Exemplo #2                      *
16 *-----*/
17 var produtosMercado = [
18     {
19         idProduto : "LT1234",
20         nmProduto : "Leite em Po Vaquinha Feliz",
21         qtdEstoque: "10"
22     },
23     {
24         idProduto : "FR543",
25         nmProduto : "Farinha de Trigo - Trigo Bom"
26         ,
27         qtdEstoque: "101"
28     },
29     {
30         idProduto : "PR6543",
31         nmProduto : "Bacon - Porquinho Feliz",
32         qtdEstoque: "40"
33     },
34     {
35         idProduto : "RC9876",
36         nmProduto : "Rucula - Prato Saboroso",
37         qtdEstoque: "40"
38     }
39 ]
```

```
38  
39 ];  
40  
41 var produtoDaPrateleira = produtosMercado.map(  
42     function(item, indice){  
43         return item.nmProduto;  
44     });  
45 /* mostra em console:  
46 [  
47     'Leite em Po Vaquinha Feliz',  
48     'Farinha de Trigo - Trigo Bom',  
49     'Bacon - Porquinho Feliz',  
50     'Rucula - Prato Saboroso'  
51 ]  
52 */  
53 console.log(produtoDaPrateleira);
```

Listing 3.12: Dois Exemplos de utilização do método Map()