


Prof. Marcos Nava



Programação de Scripts



Agenda da Aula

- Funções
 - Parâmetros e retornos opcionais
 - Parâmetros variáveis
 - Parâmetro Padrão
 - Arrow Functions
 - Funções anônimas
 - Callbacks

Funções - Parâmetros e Retornos Opcionais

Quando temos funções em Javascript, seus parâmetros e retornos são opcionais. Estas características diferem muito de outras linguagens:

```
function area(largura, altura)
{
    const area = largura * altura;

    if(area > 20)
    {
        console.log(`Valor acima do permitido:
        ${area}m2.`);
    }
    else
    {
        return area;
    }
}
```

Funções - Parâmetros e Retornos Opcionais

Qual seria o resultado para:

```
console.log(area(2, 2));
```

```
console.log(area(2));
```

```
console.log(area());
```

```
console.log(area(2, 3, 17, 22, 6));
```

```
console.log(area(5, 5));
```

Saída:

4

NaN

NaN

6

Funções - Parâmetros Variáveis

Em Javascript o número de parâmetros pode variar:

```
function soma()  
{  
    let soma = 0;  
    for(i in arguments)  
    {  
        soma += arguments[i];  
    }  
    return soma;  
}
```

Funções - Parâmetros Variáveis

Qual seria o resultado?

```
console.log(soma());  
console.log(soma(1));  
console.log(soma(1, 2));  
console.log(soma(1.5, 2.5, 4));  
console.log(soma(1, 2, 3, 'soma'));  
console.log(soma('a', 'b', 'c'));  
console.log(soma(1, 2, '3'));
```

Saída:

```
0  
1  
3  
8  
6soma  
0abc  
33
```

Funções - Parâmetros Padrão

Existem várias maneiras de usarmos parâmetros padrão.

Antes da versão nova do Javascript tínhamos maneiras um tanto quanto complicadas de fazer isso, mas agora é bem mais simples e seguro de usarmos.

Vejamos primeiro as maneiras antigas e compararemos com as mais atuais.

Funções - Parâmetros Padrão

Primeira forma:

```
function soma1(a, b, c)
{
    a = a || 1;
    b = b || 1;
    c = c || 1;
    return a + b + c;
}

console.log(soma1(), soma1(3), soma1(1, 2, 3), soma1(0, 0, 0));
```

Saída:

3 5 6 3

Funções - Parâmetros Padrão

Segunda, terceira e quarta forma:

```
function soma2(a, b, c)
{
    a = a !== undefined ? a : 1;
    b = 1 in arguments ? b : 1;
    c = isNaN(c) ? 1 : c;
    return a + b + c;
}

console.log(soma2(), soma2(3), soma2(1, 2, 3), soma2(0, 0, 0));
```

Saída:

3 5 6 0

Funções - Parâmetros Padrão

A maneira atual, que deve ser usada:

```
function soma3(a = 1, b = 1, c = 1)
{
    return a + b + c;
}
console.log(soma3(), soma3(3), soma3(1, 2, 3), soma3(0, 0, 0));
```

Saída:

3 5 6 0

Funções - Arrow

Arrow functions são funções curtas, anônimas e com retorno implícito.

```
let dobro = function (a)
```

```
{
```

```
  return 2 * a;
```

```
}
```

```
dobro = (a) => {
```

```
  return 2 * a;
```

```
}
```

```
dobro = a => 2 * a;
```

Saída:

3 5 6 0

Funções - Arrow

Existe um problema que as funções Arrow resolvem. É a questão do `this`. Com as funções comuns o `this` pode mudar conforme o contexto e isso gera muita confusão:

```
function Pessoa1()  
{  
  this.idade = 0;  
  setInterval(function() {  
    this.idade++;  
    console.log(this.idade)  
  }, 1000)  
}  
new Pessoa1
```

Saída:

NaN
NaN
NaN
NaN
NaN
...

Funções - Arrow

Para resolver este tipo de problema usamos o bind:

```
function Pessoa1()  
{  
  this.idade = 0;  
  setInterval(function() {  
    this.idade++;  
    console.log(this.idade)  
  }.bind(this), 1000)  
}  
new Pessoa1
```

Saída:

1
2
3
4
5
...

Funções - Arrow

Usando as arrow functions você não tem este problema:

```
function Pessoa2()  
{  
  this.idade = 0;  
  setInterval(() => {  
    this.idade++;  
    console.log(this.idade)  
  }, 1000);  
}  
new Pessoa2
```

Saída:

1
2
3
4
5
...

Funções - Anônimas

As funções anônimas são muito usadas no contexto do Javascript:

```
const soma = function(x, y)
{
    return x + y;
}

const imprimirResultado = function(a, b, operacao = soma)
{
    console.log(operacao(a,b));
}

imprimirResultado(3, 4);
imprimirResultado(3, 4, function (x, y) {
    return x - y;
});
imprimirResultado(3, 4, (x,y) => x * y);
```

Saída:

7
-1
12

Funções - Callback

Funções de Callback são funções que são passadas para que um determinado evento execute:

```
const fabricantes = ['Mercedes', 'Audi', 'BMW'];
```

```
function imprimir(nome, indice)
```

```
{
```

```
  console.log(`${indice + 1}. ${nome}`);
```

```
}
```

```
fabricantes.forEach(imprimir);
```

```
fabricantes.forEach(fabricante => console.log(fabricante));
```

Saída:

1. Mercedes

2. Audi

3. BMW

Mercedes

Audi

BMW

Funções - Callback

Outro exemplo:

```
const notas = [7.7, 6.5, 5.2, 8.9, 3.6, 7.1, 9.0];  
  
// Sem callback  
const notasBaixas = [];  
for(let i in notas)  
{  
  if(notas[i] < 6)  
  {  
    notasBaixas.push(notas[i]);  
  }  
}  
  
console.log(notasBaixas);  
  
const notasBaixas2 = notas.filter(nota => nota < 6);  
console.log(notasBaixas2);
```

Saída:

```
[ 5.2, 3.6 ]  
[ 5.2, 3.6 ]
```