

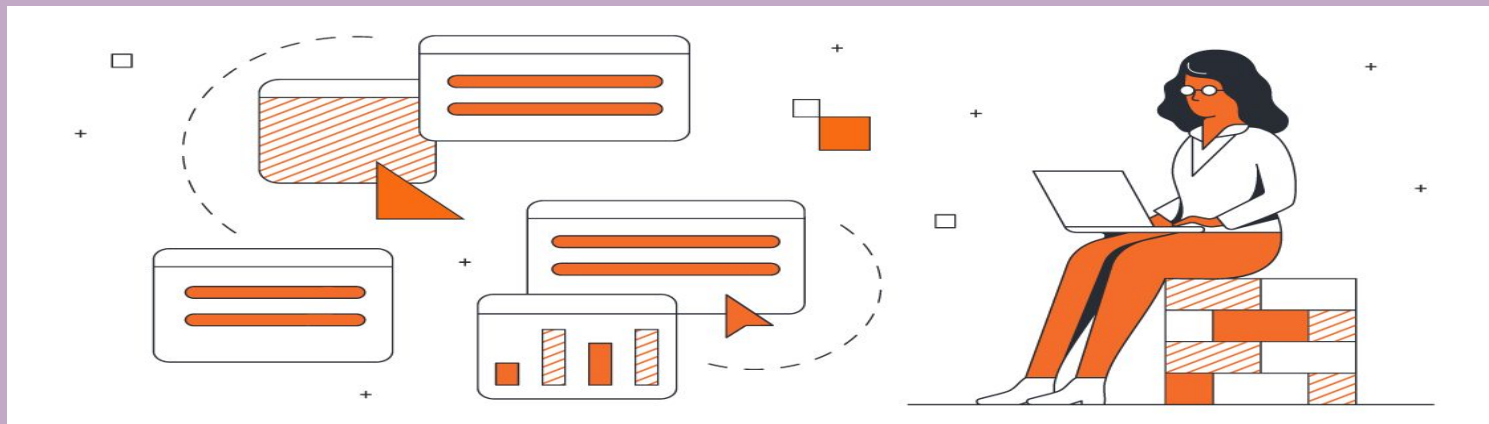
# *Bom dia Descodificadas!!*



# Funções

Uma função é um "bloco de código" que resolve um dado problema específico, que possui um nome associado e que pode ser chamado quando necessário e quantas vezes for preciso.

Pode ser reutilizada.



# *Modularização*

- ★ A modularização (ou componentização) é o mecanismo que permite que um sistema de software seja dividido em partes que interagem entre si. Tais partes são chamadas de módulos.
- ★ A modularização nos permite escrever programas mais legíveis, mais fáceis de manter e reusar e, muitas vezes, com melhor desempenho.

# *Modularização*

- ★ Em JavaScript, a modularização é feita geralmente através da criação e do uso de funções, pois são estruturas de código que permitem que o usuário organize seus programas em partes menores e mais simples.

# JS



Uma função é um conjunto de instruções que recebem entradas, fazem alguns cálculos específicos e produzem saída. Basicamente, uma função é um conjunto de instruções que executam algumas tarefas ou alguns cálculos e, em seguida, retornam o resultado ao usuário.

## *Por que usar funções?*

- ★ Reaproveitar o código;
- ★ Modularizar um programa em partes menores;
- ★ Executar uma tarefa que é frequentemente solicitada;
- ★ Aumentar a legibilidade e manutenibilidade do programa;
- ★ Quebrar grandes problemas em pequenos problemas;
- ★ Organização: o código fica melhor organizado e portanto mais fácil de ser mantido.

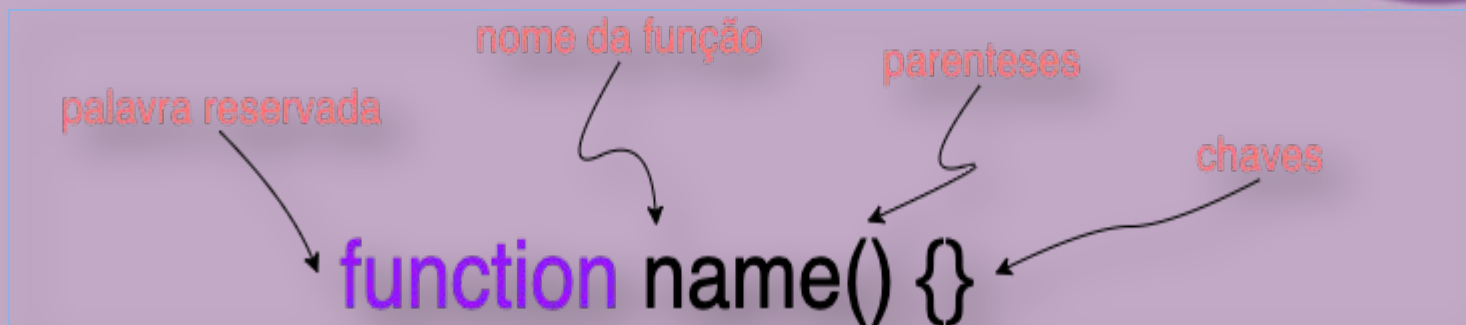
# *Declaração de Funções*

**A declaração de uma função consiste em:**

- ★ A palavra-chave function;
- ★ O nome da função, ou seu identificador, seguido de parênteses;
- ★ O corpo da função, ou o bloco de instruções necessárias para desempenhar uma tarefa, é delimitado por chaves.

# Declaração de Funções (Function Declaration)

A maneira mais primitiva é usando declarações a partir da palavra-chave function.



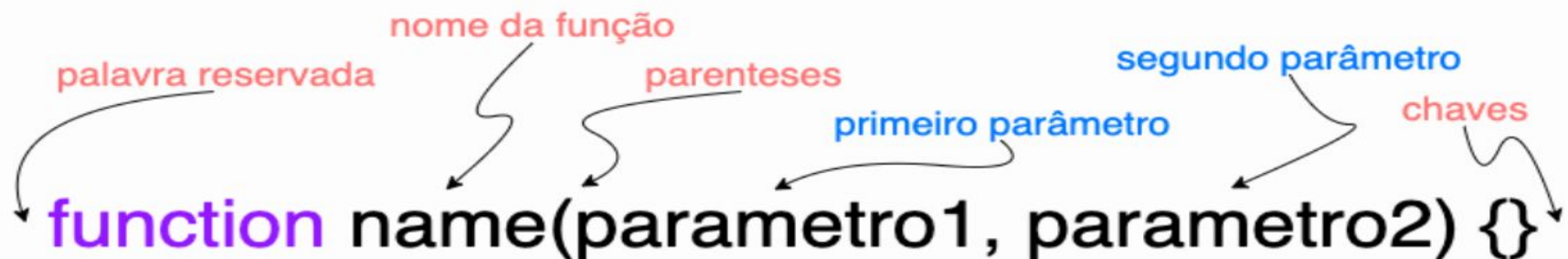
**Exemplo:**

```
function ola() {  
    console.log('Olá')  
}  
ola()
```



# Declaração de Funções (Function Declaration)

Também podemos definir parâmetros opcionais separados por vírgula:



The diagram illustrates the syntax of a function declaration with the following components and labels:

- palavra reservada** (reserved word): points to the word `function`.
- nome da função** (function name): points to the word `name`.
- parenteses** (parentheses): points to the opening parenthesis `(`.
- primeiro parâmetro** (first parameter): points to the text `parametro1`.
- segundo parâmetro** (second parameter): points to the text `parametro2`.
- chaves** (brackets): points to the closing curly brace `}`.

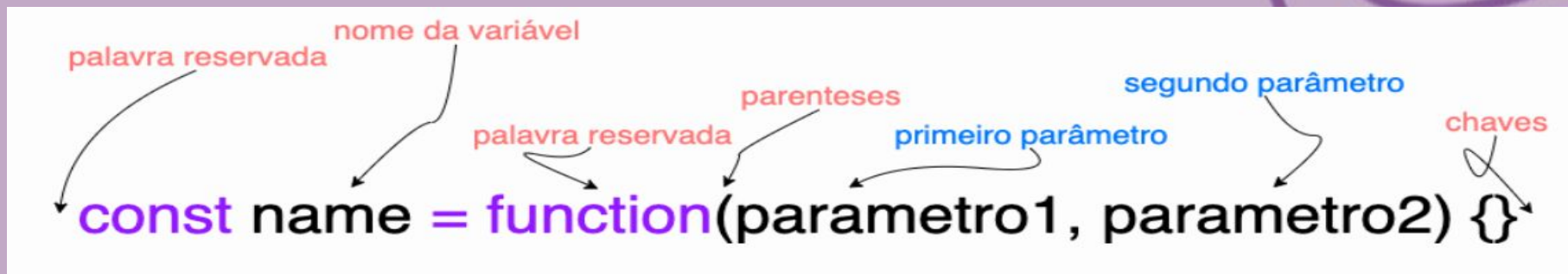
The complete declaration shown is: `function name(parametro1, parametro2) {}`

**Exemplo:**

```
function ola(nome) {  
    console.log('Olá', nome)  
}  
ola('Matheus')
```

# Função de expressão (Function Expression)

As expression e declaration são muito parecidas, a diferença é que uma função de expressão pode ser usada como qualquer expressão em JavaScript, por exemplo:



Uma das sùtis diferenças é que ela está sendo atribuída para uma variável, onde não definimos o nome da função e sim o nome da variável que irá referenciar a mesma.

## Exemplo:

```
const ola = function() {  
    console.log('Olá')  
}  
ola()
```

# Função de seta (Arrow Function)

A criação das funções de flecha é facilitar a criação e utilização de funções em JavaScript, ou seja, elas permitem a criação de funções de maneira resumida.



The diagram illustrates the syntax of an arrow function with the following components and labels:

- palavra reservada** (reserved word): points to `const`
- nome da variável** (variable name): points to `name`
- parenteses** (parentheses): points to the opening parenthesis `(`
- primeiro parâmetro** (first parameter): points to `parametro1`
- segundo parâmetro** (second parameter): points to `parametro2`
- arrow simbolo** (arrow symbol): points to the arrow `=>`
- chaves** (braces): points to the closing curly brace `}`

The complete syntax shown is: `const name = (parametro1, parametro2) => {}`

Em outras palavras, as arrow functions são simplificações para as functions expression.

# Função de seta (Arrow Function)

- ★ Caso o corpo da arrow function tenha apenas uma linha, podemos omitir a declaração das chaves.
- ★ Seguindo o princípio que o corpo tem apenas uma linha, também não precisamos utilizar o return, podemos removê-lo, pois a primeira linha será executada e retornada automaticamente.

## Exemplo:

```
const numeroAleatorio = function() {  
  return Math.random()  
}
```



```
const numeroAleatorio = () => Math.random()
```

```
function nomeCompleto(nome, sobrenome) {  
  return `${nome} ${sobrenome}`  
}
```



```
const nomeCompleto = (nome, sobrenome) => `${nome} ${sobrenome}`
```

# *Função de seta (Arrow Function)*

Comparando a Arrow Function com as outras funções, podemos perceber vários benefícios:

- ★ Ela **pode** ser escrita em apenas em 1 linha de código.
- ★ Sem a palavra-chave function.
- ★ Sem a palavra-chave return.
- ★ Sem o uso de chaves { } (quando tiver apenas uma linha de comando)



# *Função construtora (Function Constructor)*

As funções construtoras são declaradas e definidas como qualquer outra expression ou declaration, o jeito de se usar é o mesmo, a diferença está mais no caso de uso e o que ela retorna.

Uma pequena observação é que normalmente o nome de funções construtoras começa com a primeira letra maiúscula.

As funções construtoras precisam ser invocadas com a palavra reservada new:

## **Exemplo:**

```
function Pessoa(nome) {  
    this.nome = nome  
}  
const p = new Pessoa('Matheus') // { nome: 'Matheus' }
```



# *Função construtora (Function Constructor)*

No exemplo estamos criando um novo objeto Pessoa com a propriedade nome.

Quando utilizamos a palavra reservada new para invocar uma função, o JavaScript por baixo dos panos cria automaticamente um novo objeto para nós, esse objeto pode ser referenciado através do this.

Quando realizamos this.nome = nome, estamos adicionando uma nova propriedade chamada nome para o objeto recém criado, onde o valor da propriedade será o valor informado no parâmetro da função.

```
function Pessoa(nome) {  
    this.nome = nome  
}  
  
const p = new Pessoa('Matheus') // { nome: 'Matheus' }
```

# Chamada de Funções

Para que o código dentro de uma função seja executado, é necessário realizar o processo de **chamar a função**.

**Exemplo:**

```
> // declaração
function saudacao(){
    console.log("Olá!");
}

// chamada
saudacao()

Olá!
```



# Chamada de Funções

Uma característica interessante do Javascript é a possibilidade de realizar chamadas de função antes da efetiva declaração dessa mesma função.

**Exemplo:**

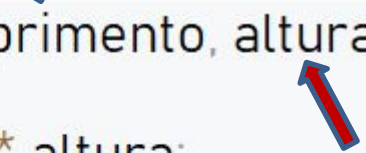


```
olaMundo();  
  
function olaMundo() {  
    console.log('Olá Mundo!');  
}
```

# Parâmetros

- ★ Quando estamos declarando uma função, podemos especificar **parâmetros**.
- ★ Parâmetros permitem que funções aceitem entradas e manipulem essas entradas para desempenhar alguma tarefa.
- ★ Usamos parâmetros para definir qual a informação deve ser passada para a função quando ela é chamada.

**Exemplo:**



```
function calcularArea(comprimento, altura) {  
    var area = comprimento * altura;  
    console.log(area);  
}
```

# Argumentos

- ★ Os valores que são passados para uma função quando ela é chamada são chamados de **argumentos**. Argumentos podem ser passados como valores ou como variáveis.

## Exemplo:

```
function calcularArea(comprimento, altura) {
```

```
    var area = comprimento * altura;
```

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

```
}
```



```
calcularArea(10, 6); // chamando a função com valores como argumentos
```

```
var comprimento_retangulo = 5;
```

```
var altura_retangulo = 6;
```




```
calcularArea(comprimento_retangulo, altura_retangulo); // chamando a  
função com variáveis como argumentos
```

# Parâmetro Padrão ou Pré-definidos

- ★ Permitem que parâmetros regulares sejam inicializados com valores iniciais caso undefined ou nenhum valor seja passado.

## Exemplo:



```
function ola(nome = 'Estranho') {  
    console.log('Olá, ' + nome + '!');  
}  
  
ola('Maria'); // retorna "Olá Maria!"  
ola(); // retorna "Olá Estranho!"
```

# *return*

- ★ A **declaração return** finaliza a execução de uma função e especifica os valores que devem ser retornados para onde a função foi chamada.


## Exemplo:

```
function calcularArea(comprimento, altura) {  
    var area = comprimento * altura;  
    return area; // retorna o valor atribuído a variável 'area' e encerra a  
    execução da função.  
}  
  
console.log(calcularArea(10, 6));
```

# Funções Auxiliares

- ★ São funções que são chamadas dentro de outras funções.
- ★ Podemos usar o valor de retorno de uma função dentro de outra função.
- ★ Funções auxiliares podem ajudar a quebrar tarefas grandes e complexas em tarefas menores e mais simples.

## Exemplo:



```
// Fórmula para conversão:  
// fahrenheit = 9/5*(celsius) + 32  
  
function multiplicaPorNoveQuintos(numero) {  
    return numero * (9 / 5);  
}  
  
function converteParaFahrenheit(celsius) {  
    return multiplicaPorNoveQuintos(celsius) + 32; // invocação da nossa  
    função auxiliar!  
}  
  
console.log(converteParaFahrenheit(15)); // Retorna 59
```



# Expressão de função

- ★ Permite criar uma **função** anônima que não tem nenhum nome de **função**, que é a principal diferença entre **Expressão de Função** e Declaração de **Função**.
- ★ Para invocar uma **expressão de função**, basta escrever o nome da variável na qual a função foi armazenada, seguido dos argumentos envoltos de parênteses

Exemplo:



```
var eFimDeSemana = function(dia) {  
    if (dia === 'Sábado' || dia === "Domingo") {  
        return true;  
    } else {  
        return false;  
    }  
}  
  
console.log(eFimDeSemana('Segunda-Feira'))
```

# Exemplos

## *Tipos de declarações*

```
// declaração da função  
function sayHelloWorld() {  
  return 'Hello World!'  
}
```

```
// declaração da função como expressão  
const sayHelloWorld2 = function() {  
  return 'Hello World!'  
}
```

```
// declaração da arrow function  
const sayHelloWorld3 = () => 'Hello World!'
```

## Chamadas

```
// chamado da função  
console.log(sayHelloWorld())  
console.log(sayHelloWorld2())  
console.log(sayHelloWorld3())
```

## Saídas

```
// saída:  
// Hello World!  
// Hello World!  
// Hello World!
```



# *Saiba mais:*

- [https://www.w3schools.com/js/js\\_functions.asp](https://www.w3schools.com/js/js_functions.asp)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)
- <https://www.youtube.com/watch?v=mc3TKp2Xzhl&t=1490s>
- <https://www.youtube.com/watch?v=ltzRdMj1lzw>
- [https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Default\\_parameters](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Default_parameters)
- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions>
- <https://youtu.be/YgHQRdGZw3w>
- <https://youtu.be/S5Mn0qQzJ-0>



Você encontrará todas as orientações e conteúdos respectivos a cada semana na nossa plataforma!



Bons estudos!!

# < descodificadas />

Sigam nossas redes  
sociais!!!

