



# < Funções >

Stephany Nusch  
Software Engineer @ QuintoAndar

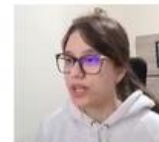


## Objetivo do Curso



- < >
1. Mostrar como declarar funções
  2. Como manipular parâmetros
  3. Apresentar loops e outras declarações
  4. Apresentar o argumento "this"

# Percurso



## Aula 1

Tipos de função

## Aula 2

Parâmetros

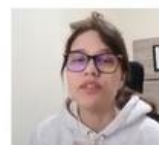
## Aula 3

Loops

## Aula 4

A palavra "this"

# Percurso



## Aula 5

Arrow functions

## Aula 6

Atividade Prática



# < Aula 1: Tipos de função >

## Funções

## Objetivos



- < 1. Apresentar a estrutura de uma função >
2. Alguns outros tipos de funções e como são utilizadas



Aula 1 | Etapa 1:

# Estrutura

## Funções



## Estrutura

Definição comum de uma função



```
function nome(parametros) {  
  // instruções  
}
```

Variáveis criadas  
dentro de uma função  
apenas podem ser  
utilizadas dentro dela.



## Definição comum de uma função

```
function nome(parametros){  
  // instruções  
  return; //valor de retorno  
}
```

Quando invocamos o  
“return”, a **função** para  
de ser executada.



Aula 1 | Etapa 2:

# Função anônima

## Funções

# Função Anônima

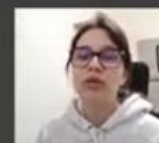
Funções que representam expressões



```
const soma = function (a, b) {  
  return a + b;  
}
```

```
soma(1, 2) // 3  
soma(5, 4) // 9
```

Uma variável pode armazenar uma função.



Aula 1 | Etapa 3:

## Função autoinvocável

Funções



# Função autoinvocável

IIFE (Immediately Invoked Function Expression)



```
(  
  function() {  
    let name = "Digital Innovation One"  
    return name;  
  }  
)();  
  
// Digital Innovation One
```

Uma função anônima entre parênteses, seguida por outro par de parênteses, que representa sua chamada.



# Função autoinvocável

IIFE (Immediately Invoked Function Expression)



```
(  
  function(a, b) {  
    return a + b;  
  }  
) (1, 2);  
  
// 3
```

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

Também pode ser utilizada com parâmetros ou armazenada em uma variável





## Aula 1 | Etapa 4:

# Callbacks

## Funções



# Callbacks



Uma função passada como argumento para outra.

```
const calc = function(operacao, num1, num2){  
  return operacao(num1, num2);  
}  
  
const soma = function(num1, num2) {  
  return num1 + num2;  
}  
  
const sub = function(num1, num2) {  
  return num1 - num2;  
}  
  
const resultSoma = calc(soma, 1, 2);  
const resultSub = calc(sub, 1, 2);  
  
console.log(resultSub); // -1  
console.log(resultSoma); // 3
```

Utilizando callbacks, você tem maior controle da ordem de chamadas.