

# REVISÃO DE FUNÇÕES

“

Uma função é um **bloco de código** que podemos invocar quantas vezes forem necessárias.

Pode realizar uma **tarefa específica** e **retornar** um valor.

Nos permite **agrupar** o **código** que vamos **utilizar muitas vezes**.



A thick, solid green diagonal stripe runs from the top right corner towards the bottom left, separating the white background from a solid green area on the right.

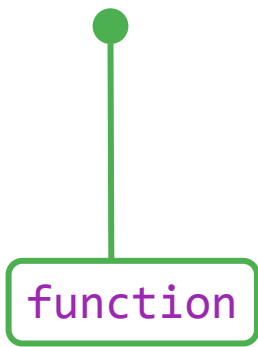
1.

# DECLARAÇÃO E ESTRUTURA

# ESTRUTURA BÁSICA

## Palavra reservada

Usamos a palavra **function** para informar ao Javascript que vamos escrever uma função.

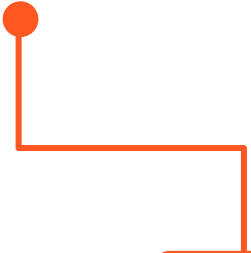


```
function somar (a,b) {  
    return a + b;  
}
```

# ESTRUTURA BÁSICA

## Nome

Definimos um **nome** para nos referirmos à nossa função quando quisermos invocá-la.




```
function somar(a,b) {  
    return a + b;  
}
```

# ESTRUTURA BÁSICA

## Parâmetros

Escrevemos os parênteses e dentro deles os parâmetros da função. Se tiver mais de um, nós os separamos usando vírgulas ,.

Se a função não tem parâmetros, nós escrevemos os parênteses sem nada dentro ().

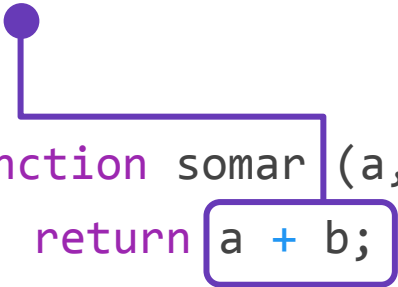


```
function somar (a, b) {  
    return a + b;  
}
```

# ESTRUTURA BÁSICA

## Parâmetros

Dentro de nossa função, podemos acessar os parâmetros como se fossem variáveis. Ou seja, só escrevendo os nomes dos parâmetros, podemos trabalhar com eles.




```
function somar(a, b) {  
    return a + b;  
}
```

The diagram illustrates the function definition. A purple dot is connected by a line to a box that highlights the parameters `a` and `b` in the function signature `function somar(a, b) {`. The return statement `return a + b;` is also highlighted with a purple box.

# ESTRUTURA BÁSICA

## Corpo

Entre as chaves de abertura e fechamento escrevemos a lógica da nossa função, ou seja, o código que queremos que seja executado toda vez que o invocamos.



```
function somar (a,b) {  
    return a + b;  
}
```




# ESTRUTURA BÁSICA

## O retorno

É muito comum ao escrevermos uma função, que tenhamos como objetivo retornar ao exterior o resultado do processo que estamos fazendo dentro dela.

Para isso, usamos a palavra reservada **return** seguida do que quisermos devolver.



```
function somar (a,b) {  
    return a + b;  
}
```

# FUNÇÕES DECLARADAS

São aquelas que são declaradas usando a **estrutura básica**.  
Recebem um **nome formal** pelo qual nós o invocaremos.

```
{}
```

```
function fazerSorvete(quantidade) {  
    return '🍦'.repeat(quantidade)  
}
```



Eles carregam **antes** de qualquer código ser executado.

# FUNÇÕES EXPRESADAS

São aqueles que **são atribuídos como um valor** a uma variável. O nome da função será o **nome** da **variável** que declaramos.

```
let fazerSushi = function(quantidade) {  
  return '🍣'.repeat(quantidade)  
}
```



São carregadas quando o intérprete *chega à linha de código* onde o se encontra a *função*.

A thick, solid green diagonal stripe runs from the top right corner towards the bottom left, separating the white background on the left from the solid green background on the right.

2.

INVOCACÃO

# INVOCANDO UMA FUNÇÃO

A forma de **invocar** (executar) uma função é escrevendo o seu nome seguido de abrir e fechar parênteses.

```
nomeFunção();
```

Caso você queira ver ou salvar os dados que **retornam**, será necessário armazená-los em uma variável, ou fazer um `console.log` da execução.

```
let resultado = nomeFunção();  
console.log(nomeFunção());
```

# INVOCANDO UMA FUNÇÃO

Se a função espera argumentos, podemos passá-los dentro dos parênteses.

É importante respeitar a ordem se houver mais de um parâmetro, pois o Javascript irá atribuí-los na ordem em que chegam.

```
function saudacao(nome, sobrenome) {  
    return 'Olá ' + nome + ' ' + sobrenome;  
}  
  
saudacao('Rafael', 'Silva');  
// retorna 'Olá Rafael Silva'
```

{ }

# INVOCANDO UMA FUNÇÃO

Também é importante notar que quando temos parâmetros em nossa função, o Javascript espera que os utilizemos como argumentos quando o executamos.

```
function saudacao(nome, sobrenome) {  
    return 'Olá ' + nome + ' ' + sobrenome;  
}  
saudacao(); // retorna 'Olá undefined undefined'
```



Sem ter recebido o argumento que precisava, o Javascript atribui o tipo de dados **undefined** para as variáveis do nome e sobrenome.

# INVOCANDO UMA FUNÇÃO

Para este tipo de casos, o Javascript nos permite definir os **valores padrão**.

Se adicionarmos um igual = após o parâmetro, podemos especificar seu valor caso não chegue nenhum.

```
function saudacao(nome = 'visitante',  
  sobrenome = 'anônimo') {  
  return 'Olá ' + nome + ' ' + sobrenome;  
}  
saudacao(); // retorna 'Olá visitante anônimo'
```

{ }



Os **parâmetros** são as **variáveis** que escrevemos quando **definimos** a função.

Os **argumentos** são os **valores** que enviamos quando **invocamos** a função.

