let aula = “Funçõ̂es”

```
const professor = "Danilo Santos"
```

Temas

1 Declaração e estrutura

2 Invocar funções

3 Escopo (scope)

0 | O que é uma função?

Imagine uma tarefa do seu dia a dia?

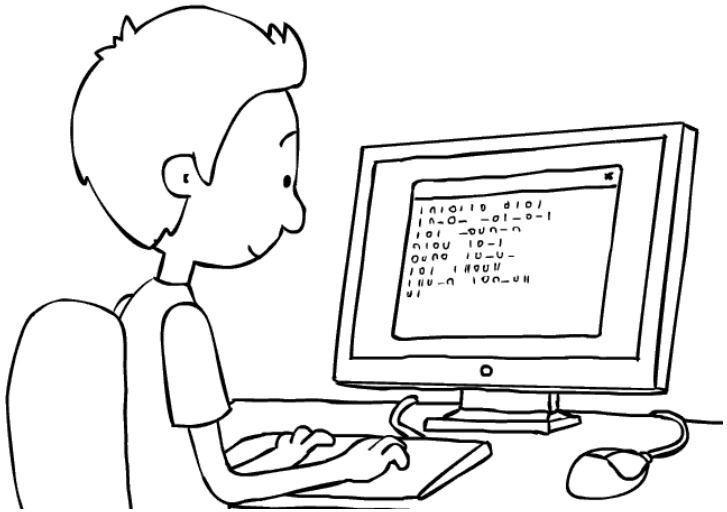
E se você pudesse delegar esta tarefa a alguém?

- Qual tarefa gostaria que alguém fizesse por você?

Podemos dizer que...

Todas estas tarefas são funções.

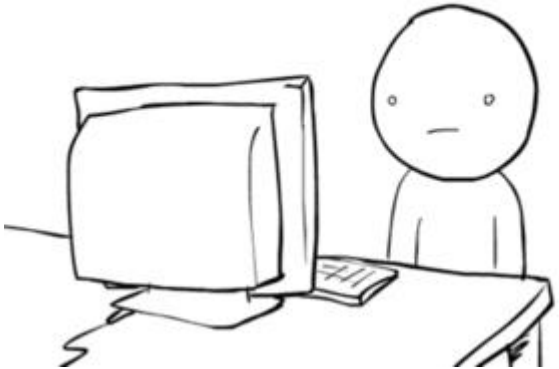
Na programação: Funções são tarefas que criamos para o computador realizar.



Mas e se eu apenas atribuir a função?

A pessoa saberia o que fazer?

- Em qual mercado devo ir?
- O que devo comprar?
- Como devo pagar?



Essas informações são chamadas de Parâmetros em funções.

Funções também retornam?

E se eu precisar de um retorno sobre determinada Tarefa?

Imagine que sua mãe te peça para fazer a
soma dos itens que coloca no carrinho de compras?

- Ela pega o produto e te fala o valor;
- Você calcula e depois diz o resultado.

Essas informações são chamadas de Retorno em funções.

Conclusão

Funções são **ações** executadas assim que são **chamadas** ou em decorrência de um **evento**.

1 | Declaração e estrutura

Estrutura básica

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

Palavra reservada

Usamos a palavra **function** para indicar ao JavaScript que vamos declarar uma função.

Estrutura básica

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

Nome da função

Definimos um nome para referenciarmos nossa função no momento que queremos **invocá-la**.

Estrutura básica

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

Parâmetros

Escrevemos os parênteses e, dentro deles, colocamos os parâmetros da função. Se houver mais de um, os separamos por vírgulas `,`.

Se na função não houver parâmetros, devemos escrever os parênteses sem nada dentro `()`.

Estrutura básica

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

Parâmetros

Dentro de nossa função, poderemos acessar os parâmetros como se fossem variáveis, ou seja, ao escrever os nomes dos parâmetros, podemos trabalhar com eles.

Estrutura básica

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

Corpo

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

Estrutura básica

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

O return

É muito comum, na hora de escrever uma função, que você queira devolver ao exterior o resultado do processo que estamos executando.

Para isso, utilizamos a palavra reservada **return** seguida do que queremos retornar.

Funções nomeadas (declaradas)

São aquelas que são declaradas usando a **estrutura básica**. Podem receber um **nome** escrito após a palavra reservada **function**, por meio do qual podemos invocá-la.

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

Function expression (função expressa)

São aquelas que **são atribuídas como valor** de uma variável. Neste caso, a função em si não tem nome, é uma **função anônima**.

Para invocá-la, podemos usar o nome da variável que declaramos.

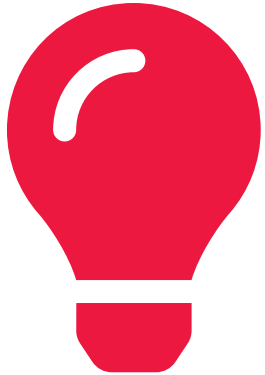
```
let fazerSushi = function (quantidade) {
        return '🍣'.repeat(quantidade);
}
```

2 | Invocar funções

“

Podemos imaginar as funções como se fossem máquinas.

Durante a **declaração**, cuidamos de **construir** a máquina e, durante a **invocação**, a colocamos para **funcionar**.



”

Invocando uma função

Antes de invocar uma função, devemos declará-la. Então, vamos declarar uma função:

```
function fazerSorvete() {
        return '🍦';
}
```

A maneira de **invocar** (também conhecida como **chamar** ou **executar**) uma função é escrevendo seu nome, seguido da abertura e fechamento de parênteses.

```
fazerSorvete(); // Retornará '🍦'
```

Invocando uma função

Se a função tiver parâmetros, podemos passá-los entre parênteses ao invocá-la. **É importante respeitar a ordem**, pois o JavaScript atribuirá os valores na ordem em que chegarem.

{}

```
function bemVindo(nome, sobrenome) {
        return 'Olá, ' + nome + ' ' + sobrenome;
}

bemVindo('João', 'da Silva');
// retornará 'Olá, João da Silva'
```

Invocando uma função

Também é importante ter em mente que, quando temos parâmetros em nossa função, o JavaScript espera que os indiquemos ao executá-la.

```
function bemVindo(nome, sobrenome) {
        return 'Olá, ' + nome + ' ' + sobrenome;
}

bemVindo(); // retornará 'Olá undefined undefined'
```

Neste caso, não tendo recebido o argumento necessário, o JavaScript atribui o tipo de dado **undefined** aos parâmetros *nome* e *sobrenome*.

Invocando uma função

Para casos como o anterior, podemos definir **valores por padrão**.

Se adicionarmos o sinal de igual = depois de um parâmetro, podemos especificar seu valor, caso nenhum chegue.

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

bemVindo(); // retornará 'Olá, visitante anônimo'
```

Armazenando os resultados

No caso de querermos guardar o valor retornado pela função, será necessário armazená-lo em uma variável.

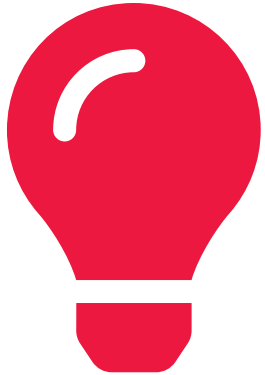
```
function fazerSorvete(quantidade) {
        return '🍦'.repeat(quantidade);
}
{}

let meusSorvetes = fazerSorvetes(3);
console.log(meusSorvetes); // Exibirá no console
'🍦🍦🍦'
```

“

Chamamos **parâmetros** as **variáveis** que escrevemos quando **definimos** a função.

Chamamos **argumentos** os **valores** que enviamos quando **invocamos** a função.

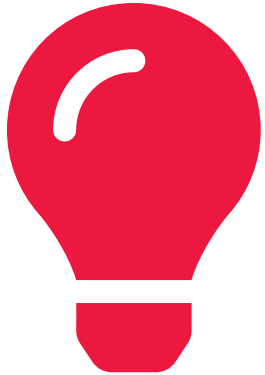
”



3 | Escopo (Scope)

O **escopo (scope)** se refere ao alcance que uma variável tem, ou seja, de onde podemos acessá-la.

No JavaScript, os escopos são definidos principalmente pelas funções.



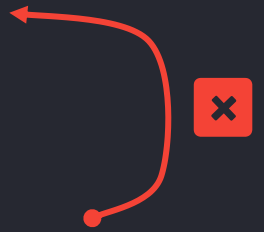
Escopo local

No momento que declaramos uma variável dentro de uma função, esta passa a ter alcance local. Ou seja, essa variável existe unicamente dentro desta função.

Se quisermos fazer uso desta variável fora da função, não poderemos, uma vez que fora do **escopo** onde foi declarada, essa variável não existe. O escopo é delimitado pelas chaves de abertura e fechamento da função.

{}

```
function bemVindo() {
    // todo o código que escrevemos dentro
    // de nossa função, tem escopo local
}
// Não poderemos acessar esse escopo de fora
```



{código}

```
function ola() {
    let bemvindo = 'Olá, como vai?';
    return bemvindo;
}

console.log(bemvindo);
```

Definimos a variável bemvindo **dentro** da função *ola()*, portanto seu **escopo** é **local**.

Somente dentro desta função podemos acessá-la.

{código}

```
function ola() {
    let bemvindo = 'Olá, como vai?';
    return bemvindo;
}

console.log(bemvindo); // bemvindo is not defined
```

Ao querer fazer uso da variável **bemvindo** fora da função, o JavaScript não a encontrará e nos devolve o seguinte erro:

Uncaught ReferenceError: bemvindo is not defined

Escopo global

No momento que declaramos uma variável **fora** de qualquer função, ela passa a ter **escopo global**.

Ou seja, podemos fazer uso dela de qualquer lugar do código em que nos encontremos, inclusive dentro de uma função, e acessar seu valor.

```
{}
// todo código que escrevemos fora
// das funções é global
function minhaFuncao() {
        // Dentro das funções
    // temos acesso às variáveis globais
}
```

{código}

```
var bemvindo = 'Olá, como vai?';

function ola() {
    return bemvindo;
}

console.log(bemvindo);
```

Declaramos a variável **bemvindo** fora de nossa função, portanto seu **escopo** é **global**.

Podemos fazer uso dela de qualquer lugar do código.

{código}

```
var bemvindo = 'Olá, como vai?';

function ola() {
   return bemvindo;
}

console.log(bemvindo); // 'Olá, como vai?'
```

Dentro da função *ola()* chamamos a variável **bemvindo**.

Seu alcance é **global**, portanto, o JavaScript sabe a qual variável estou me referindo e executa a função com êxito.

Bora codar?