Arrow functions

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Temas

1 Declaração e estrutura

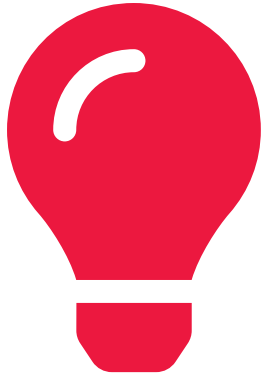
2 Exemplos

1 Declaração e estrutura

“

As funções são os elementos que mais iremos utilizar ao programar em JavaScript.

As **arrow functions** nos permitem digitá-las com uma **sintaxe** mais **compacta**.



”

Estrutura básica

Pensemos em uma função simples que faça a soma de dois números.

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

Agora, vejamos a versão reduzida dessa mesma função, ao transformá-la em uma arrow function.

```
let somar = (a, b) => a + b;
```

Nome de uma arrow function

As arrow functions **são sempre anônimas**, ou seja, não possuem nome como as funções normais.

```
{}  (a, b) => a + b;
```

Se quisermos nomeá-las, é necessário digitá-las como **function expressions**, ou seja, devemos atribuí-las como valores de variáveis.

```
{}  let somar = (a, b) => a + b;
```

A partir de agora podemos chamar nossa função por seu novo nome.

Parâmetros de uma arrow function

Usamos parênteses para indicar os **parâmetros**. Se nossa função não receber parâmetros, devemos deixar os parênteses vazios.

```
let somar = (a, b) => a + b;
```

Uma característica deste tipo de função é que se ela receber apenas um único parâmetro, podemos dispensar o uso dos parênteses.

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

A flecha de uma arrow function

A usamos para indicar ao JavaScript que vamos escrever uma função (substitui a palavra reservada `function`).

```
let somar = (a, b) => a + b;
```

O que está à esquerda da flecha será a entrada da função, ou seja, os parâmetros, e o que está à direita, a lógica (e o retorno possível).

"As arrow functions recebem este nome por conta do operador `=>`. Se analisarmos com um pouco de imaginação, é possível verificar que o operador se parece com uma flecha.

Em Inglês, geralmente é chamado de *fat arrow* (flecha gorda) para diferenciá-lo da flecha simples `->`."

Corpo de uma arrow function

Se a função tem somente uma linha de código, e esta mesma linha é a que vai exibir o **retorno**, não precisamos adicionar chaves, nem a palavra reservada **return**.

```
let somar = (a, b) => a + b;
```

Do contrário, vamos precisar utilizar ambos. Isso normalmente ocorre quando temos mais de uma linha de código em nossa função.

```
let multiplo = (a, b) => {
    let resto = a % b; // Obtemos o resto da divisão.
    return resto == 0; // Se o resto for 0, é múltiplo
};
```

2 | Exemplos

```
let bemVindo = () => 'Olá Mundo!';

let dobro = numero => numero * 2;

let soma = (a, b) => a + b;

let horaAtual = () => {
    let data = new Date();
    return data.getHours() + ':' +
    data.getMinutes();
}
```

Arrow function **sem parâmetros**.

Requer os parênteses para começar.

Por ter uma única linha de código, e essa é a que queremos retornar, o **return** é implícito.

```
let bemVindo = () => 'Olá Mundo!';

let dobro = numero => numero * 2;

let soma = (a, b) => a + b;

let horaAtual = () => {
    let data = new Date();
    return data.getHours() + ':' +
    data.getMinutes();
}
```

Arrow function com um **único parâmetro** (não precisamos dos parênteses para indicá-lo) e com um **return** implícito.

```
let bemVindo = () => 'Olá Mundo!';

let dobro = numero => numero * 2;

let soma = (a, b) => a + b;

let horaAtual = () => {
    let data = new Date();
    return data.getHours() + ':' +
    data.getMinutes();
}
```

Arrow function com **dois parâmetros**.

Necessita dos parênteses e possui um **return** implícito.

```
let bemVindo = () => 'Olá Mundo!';

let dobro = numero => numero * 2;

let soma = (a, b) => a + b;

let horaAtual = () => {
    let data = new Date();
    return data.getHours() + ':' +
    data.getMinutes();
}
```

Arrow function **sem parâmetros** e com um **return explícito**.

Neste caso fazemos uso das **chaves** e do **return**, já que a lógica desta função se desenvolve em mais de uma linha de código.


DigitalHouse>
Coding School