DigitalHouse >
Coding School

ARROW FUNCTIONS

“

Utilizaremos muito as funções quando estivermos programando em Javascript.

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

ESTRUTURA BÁSICA

Pensemos em uma função simples que poderíamos programar com a forma habitual, uma soma de dois números.

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

Agora, vejamos a versão reduzida da mesma função, transformando-a em uma arrow function.

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

ESTRUTURA BÁSICA

Nome

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

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

Se quisermos nomeá-las, é necessário escrevê-la como uma expressão de função, ou seja, atribuí-la como um valor a uma variável.

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

ESTRUCTURA BÁSICA

Parâmetros

Usamos parênteses para indicar os parâmetros. Se nossa função recebe parâmetros, devemos escrevê-los assim:

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

Uma particularidade desse tipo de função é que, se ela recebe somente um **único parâmetro**, podemos omitir o uso dos parênteses.

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

ESTRUTURA BÁSICA

Operador flecha

O utilizamos para para indicar ao Javascript que vamos escrever uma função (substitui a palavra reservada function).

O que está a esquerda da flecha será a entrada da função (os parâmetros). O que está a direita, a saída (ou retorno)

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

ESTRUTURA BÁSICA

Corpo

Escrevemos a lógica da função. Se a função tem somente uma linha de código e esta mesma linha retorna um resultado, podemos omitir as chaves e a palavra *return*.

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

Caso contrário, vamos precisar tanto das chaves quanto do return.

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

“

As **arrow functions** tem esse nome por causa do operador **=>**

Se olharmos para ele com um pouco de imaginação, perceberemos que ele se parece com uma flecha.

Em inglês, chamamos de **fat arrow** (flecha gorda) para diferenciá-lo de outra combinação parecida: ->

{ código }

```
let saudar = () => 'Olá, mundo!';

let dobroDe = numero => numero * 2;

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

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

{ código }

```
let saudar = () => 'Olá, mundo!';

let dobroDe = numero => numero * 2;

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

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

Arrow function **sem parâmetros**.

Precisa dos parênteses para ser iniciada.

Ao ter somente uma linha de código, e esta mesma seja a que eu quero retornar, o return fica implícito.

{ código }

```
let saudar = () => 'Olá, mundo!';

let dobroDe = numero => numero * 2;

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

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

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

{ código }

```
let saudar = () => 'Olá, mundo!';

let dobroDe = numero => numero * 2;

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

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

Arrow function **com dois parâmetros**.

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

{ código }

```
let saudar = () => 'Olá, mundo!';

let dobroDe = numero => numero * 2;

let somar = (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 precisa de mais de uma linha de código