Teoria complementar: arrow function

Uma **expressão de arrow function** é uma alternativa compacta a uma expressão de uma função tradicional, mas é limitada e não pode ser usada em todas ***as situações.***

Sintaxe

Sintaxe básica

Um parâmetro. Com uma expressão simples você não precisa da palavra reservada **return**:

```
param => expression
```

Vários parâmetros requerem parênteses. Com uma expressão simples você não precisa da palavra reservada **return**:

```
(param1, paramN) => expression
```

Declarações com várias linhas exigem chaves e a palavra reservada **return**:

```
param => {
 let  a =  1;
return  a + b;
}
```

Vários parâmetros requerem parênteses. Declarações com várias linhas exigem chaves e a palavra reservada **return**:

```
(param1, paramN) => {
 let  a =  1;
return a + b;
}
```

Sintaxe avançada

Para retornar uma expressão de objeto literal, os parênteses são necessários em torno da expressão:

```
params -> . ({foo: "a"}) // devolve o objeto {foo: "a"}
```

Os parâmetros rest são suportados:

```
(a, b,... r) => expression
```

Os parâmetros predefinidos são suportados:

```
(a=400, b=20, c) => expression
```

Desestruturação dentro dos parâmetros são suportados:

```
([a, b] = [10, 20]) => a + b;  // o resultado e 30
({ a, b } = { a: 10, b: 20 }) => a + b; // resultado e 30
```

Quebras de linha

Uma *arrow function* não pode conter uma quebra de linha entre seus parâmetros e sua flecha.

```
var func = (a, b, c)
  => 1;
//SyntaxError: Unexpected token '=>'
```

Entretanto, isto pode ser corrigido ao usar parênteses ou colocar a quebra de linha dentro dos argumentos como visto abaixo para garantir que o código permaneça bonito e leve. Você também pode colocar quebras de linha entre os argumentos.

```
var func  = (a, b, c) =>
 1;

var  func  = (a, b, c) => (
 1
);

var  func  = (a, b, c) => {
 return  1
};

var  func  = (
 a,
 b,
 c)
```

```
=> 1;
```

// Você não verá um erro de Sintaxe (SyntaxError)

Ordem de análise

Apesar de a flecha numa *arrow function* não ser um operador, *arrow functions* possuem regras especiais de análise que interagem diferentemente com precedência de operador (operator precedence) comparadas à funções comuns.

```
let callback;

callback = callback || function() {}; // ok

callback = callback || () => {};
// SyntaxError (Erro de sintaxe): argumentos inválidos de arrow-function

callback = callback || (() => {});    // ok
```

Exemplos

```
// Uma arrow function vazia retorna undefined
let empty = () => {};

(() => 'foobar')();
// Retorna "foobar"
// (esta é uma Expressão de Função Invocada Imediatamente (Immediately
Invoked Function Expression)
// veja 'IIFE' no glossário)

var simple = a => a > 15 ? 15 : a;
simple(16); // 15
simple(10); // 10

let  max = (a, b) => a > b ? a : b;

// Mapeamento, filtragem, ... simples de array

var  arr = [5, 6, 13, 0, 1, 18, 23];
```

```
var sum = arr.reduce((a, b) => a + b);
// 66

var even = arr.filter(v => v % 2 == 0);
// [6, 0, 18]

var double = arr.map(v => v * 2);
// [10, 12, 26, 0, 2, 36, 46]

// Cadeias de promessa (promisse chains) mais concisas
promise.then(a => {
  // ...
}).then(b => {
  // ...
});

// Arrow functions sem parâmetros que são visualmente mais fáceis de
analisar
setTimeout( () => {
  console.log('E aconteço antes');
  setTimeout( () => {
     // deeper code
    console.log('Eu aconteço depois');
  }, 1);
}, 1);
```

Referência:

Arrow functions - JavaScript | MDN