

FUNÇÕES (PARTE 2)

ARROW FUNCTIONS

(AULA 7)

CURSO BÁSICO DE PROGRAMAÇÃO COM JAVASCRIPT

MAYARA MARQUES

mmrosatab@gmail.com

SUMÁRIO

- Arrow functions
- Sintaxe
- Outras formas de sintaxe
- Arrow functions vs funções regulares
- Mão na massa

ARROW FUNCTIONS

Além das funções regulares (regular function) e as funções anônimas (anonymous functions), o Javascript dispõe de outro recurso para criar funções, as **arrow functions** ou **funções de seta**.

SINTAXE

```
const hello = () => {console.log( "Hello" )}
```

São **funções anônimas**.

Não utilizam a palavra reservada **function** na sua criação.

Devem **ser atribuídas a uma variável** para serem invocadas.



OUTRAS FORMAS DE SINTAXE

```
const hello = () => {console.log("Hello")}  
hello() // Hello
```

Pode-se omitir **as chaves** em caso de **passagem de uma única instrução**

```
const hello = () => console.log("Hello")  
hello() // Hello
```

```
const add = (num1, num2) => {return num1 + num2}  
console.log(add(1,2)) // 3
```

Pode-se omitir **o return (retorno implícito)** e **as chaves** em caso de **passagem de uma única instrução**

```
const add = (num1, num2) => num1 + num2  
console.log(add(1,2)) // 3
```

Em caso de um **único parâmetro na função**, os **parênteses** podem ser omitidos.

```
const increment = num => num + 1  
console.log(increment(1)) // 2
```

ARROW FUNCTION VS FUNÇÕES REGULARES

Arrows functions **NÃO** são somente uma forma mais enxuta de se escrever funções. Essas possuem algumas diferenças em relação às funções regulares além da sintaxe.

- Diferenças mais relevantes são:
 - Uso da palavra reservada `this`
 - Uso do `arguments` (próxima aula)



ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - A palavra reservada **this** faz referência ao contexto de execução no qual uma função está sendo chamada.
 - É usada para se referir ao objeto atual que está executando uma função.
 - O valor de **this** pode mudar **dependendo do contexto em que a função é chamada**.

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - O que é um contexto em javascript?
 - O contexto refere-se ao valor de **this** em uma determinada parte do código, ou seja, para quem o **this** faz referência (aponta).

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - O que é um contexto em javascript?
 - O contexto refere-se ao valor de **this** em uma determinada parte do código, ou seja, para quem o **this** faz referência (aponta).

*Não confundir **contexto** com **escopo**.*

Escopo é o local onde uma variável é acessível.



ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Há quatro principais contextos nos quais o this pode ser usado em JavaScript.
 - Global
 - Função
 - Método
 - Construtor

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto global
 - Quando o **this** é usado no contexto global, ele se refere ao objeto global do ambiente em que o código está sendo executado. No navegador, o objeto global é o "window", enquanto no Node.js, o objeto global é o "global".

ARROW FUNCTION VS FUNÇÕES REGULARES

- this

```
console.log(this) // retorna 'window' no navegador e 'global' no Node.js
```

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto função
 - Quando o this é usado em uma função normal, ele se refere ao objeto que chama a função. Por exemplo:

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto função

```
function myFunction () {  
  console.log(this)  
}
```

```
myFunction() // Window {0: Window, window: Window,  
self: Window, document: document, name: '',  
location: Location, ...}
```

```
// retorna o objeto global
```

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto método
 - Quando o **this** é usado dentro de um método de um objeto, ele se refere ao próprio objeto.

ARROW FUNCTION VS FUNÇÕES REGULARES

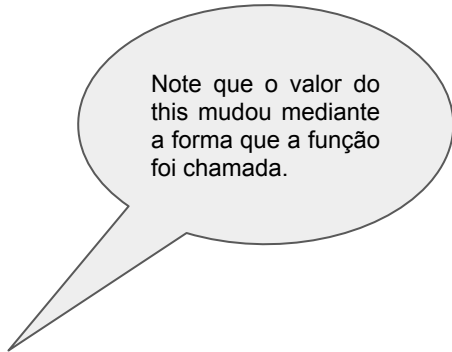
- this
 - Contexto método

```
function myFunction () {  
  console.log(this)  
}
```

```
const obj = {  
  objFunction: myFunction  
}
```

```
obj.objFunction() // Output: {objFunction: f}
```

```
// retorna o objeto obj
```



Note que o valor do this mudou mediante a forma que a função foi chamada.

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto método

```
const person = {  
  name: "Beyonce",  
  sayMyName: function() {  
    console.log(this)  
    console.log("My name is " + this.name)  
  }  
}
```

```
person.sayMyName() // {name: 'Beyonce', sayMyName: f}  
                  // "My name is Beyonce"
```

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto do construtor
 - Quando o **this** é usado dentro de um construtor de uma classe, ele se refere à instância atual da classe que está sendo criada.

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Contexto do construtor

```
class Person {  
  constructor(name) {  
    this.name = name  
  }  
  
  sayMyName() {  
    console.log(this)  
    console.log("My name is " + this.name)  
  }  
}  
  
const person = new Person("Beyonce")  
person.sayMyName() // Person {name: 'Beyonce'}  
                  // "My name is Beyonce"
```

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - É importante entender que o valor de `this`, com funções regulares, é determinado **no momento em que uma função é chamada**.

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Se uma função regular é chamada como um método de um objeto, o valor de **this** será o objeto em que a função é chamada. Se uma função regular é chamada sem nenhum objeto, o valor de **this** será o objeto global (ou undefined no modo estrito).
 - Em funções regulares, o valor de **this** é determinado pela forma como a função é chamada, enquanto nas arrow functions, o valor de **this** é determinado pelo **escopo** em que a função é definida.

ARROW FUNCTION VS FUNÇÕES REGULARES

- this
 - Arrow functions **NÃO** possuem o seu próprio **this**.

ARROW FUNCTION VS FUNÇÕES REGULARES

```
const person = {  
  age: 30,  
  sayMyAge: function() {  
    return "My age is: " + this.age  
  },  
  sayMyAge2: () => {  
    return "My age is: " + this.age  
  }  
}  
  
console.log(person.sayMyAge())  
console.log(person.sayMyAge2())
```

Como o **this** se comporta com arrow functions e funções regulares?

| | |
|----------------------|-------------------------------|
| My age is: 30 | script.js:141 |
| My age is: undefined | script.js:142 |
| Live reload enabled. | index.html:92 |
| > | |
| > | |

ARROW FUNCTION VS FUNÇÕES REGULARES

```
const person = {  
  age: 30,  
  sayMyAge: function() {  
    return "My age is: " + this.age  
  },  
  sayMyAge2: () => {  
    return "My age is: " + this.age  
  }  
}  
  
console.log(person.sayMyAge())  
console.log(person.sayMyAge2())
```

Como o **this** se comporta com arrow functions e funções regulares?

No exemplo ao lado, a arrow function não acessa a propriedade "age" do objeto porque o valor de "this" não está vinculado ao objeto.

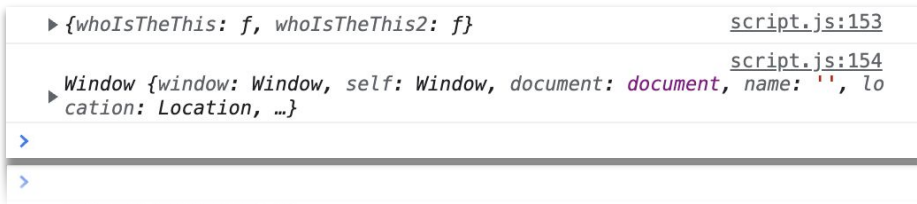
| | |
|----------------------|-------------------------------|
| My age is: 30 | script.js:141 |
| My age is: undefined | script.js:142 |
| Live reload enabled. | index.html:92 |
| > | |
| > | |

ARROW FUNCTION VS FUNÇÕES REGULARES

```
const who = {  
  whoIsTheThis: function() {  
    return this  
  },  
  whoIsTheThis2: () => {  
    return this  
  }  
}  
  
console.log(who.whoIsTheThis())  
console.log(who.whoIsTheThis2())
```

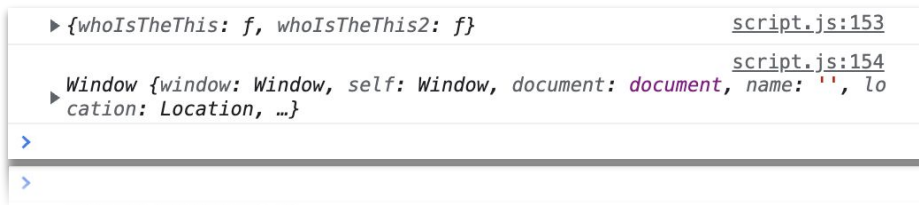
Como o **this** se comporta com arrow functions e funções regulares?

No exemplo ao lado, a arrow function não acessa o contexto do objeto, e sim o contexto de onde o objeto foi definido, neste caso o global. Assim, ela retorna o window, nosso objeto global no navegador.



ARROW FUNCTION VS FUNÇÕES REGULARES

```
const who = {  
  whoIsTheThis: function() {  
    return this  
  },  
  whoIsTheThis2: () => {  
    return this  
  }  
}  
  
console.log(who.whoIsTheThis())  
console.log(who.whoIsTheThis2())
```



```
const button = document.getElementById('btn')
```

```
button.addEventListener('click', () => {  
  console.log(this)  
})
```

Aqui está um pouco confuso de saber o escopo da arrow function para então saber quem é o this!

```
const button = document.getElementById('btn')
```

```
const arrowFn = () => {  
  console.log(this) // Escopo global: `this` é `window`  
}
```

```
button.addEventListener('click', arrowFn)
```

Mas e se passássemos a arrow function para uma variável antes de passá-la como argumento de uma função??



MÃO NA MASSA

MÃO NA MASSA



1. Crie uma arrow function chamada dobro que recebe um número e retorna o dobro dele.
2. Crie uma arrow function chamada soma que recebe dois números e retorna a soma deles.
3. Reescreva a função abaixo usando retorno implícito:

```
function triplo(n) {  
  return n * 3  
}
```

4. Reescreva a função abaixo usando retorno implícito:

```
function mensagem() {  
  return "Seja bem-vindo!"  
}
```

5. Reescreva as funções do mão na massa da aula anterior utilizando arrow functions.

REFERÊNCIAS

- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/this>
- <https://indepth.dev/posts/1357/getting-started-with-modern-javascript-variables-and-scope>
- https://www.w3schools.com/js/js_this.asp
- https://www.w3schools.com/js/js_scope.asp