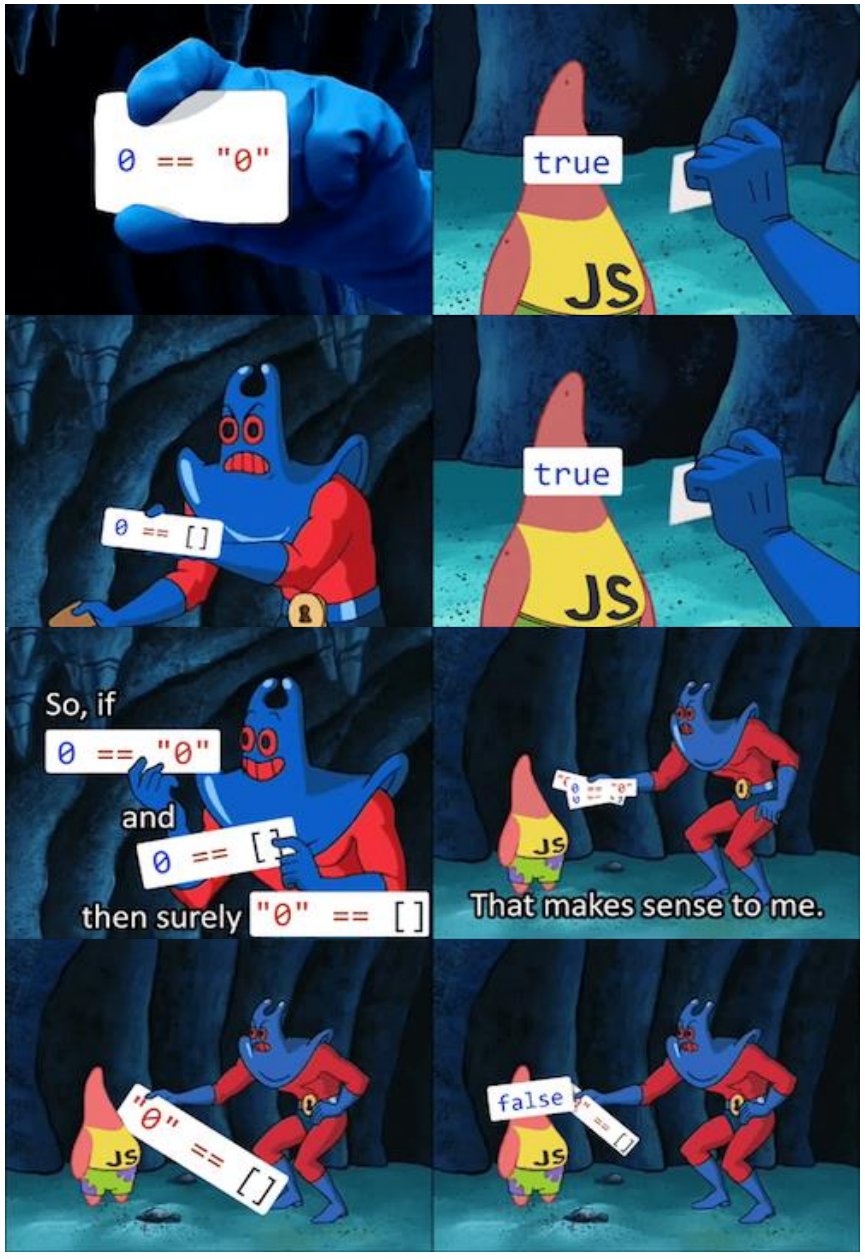
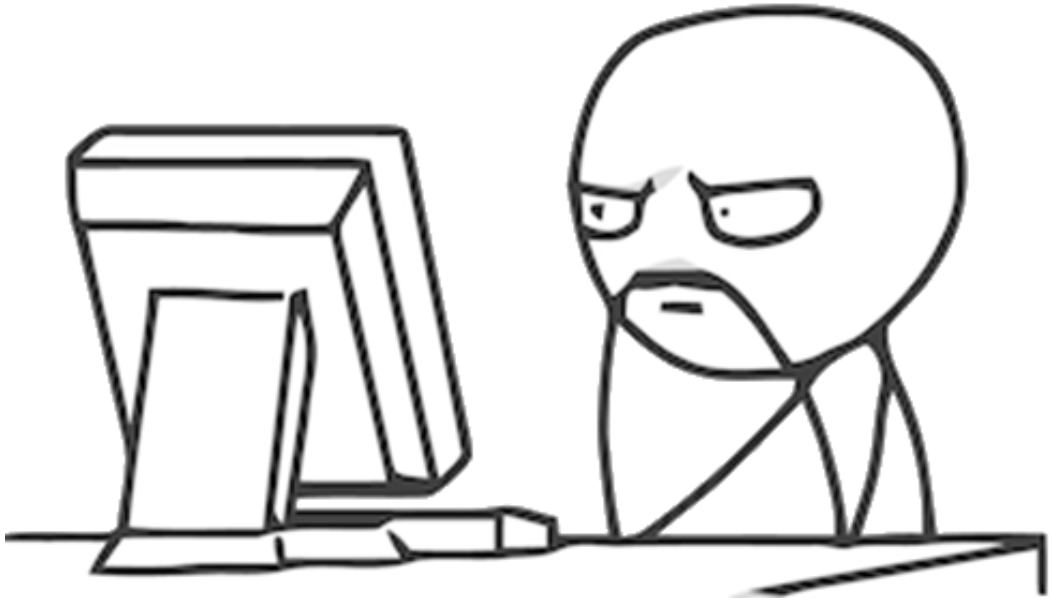
Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end I (front 1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Coerção no JavaScript


0 == "0"
true
JS
0 == []
true
JS
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false
JS

Por que?



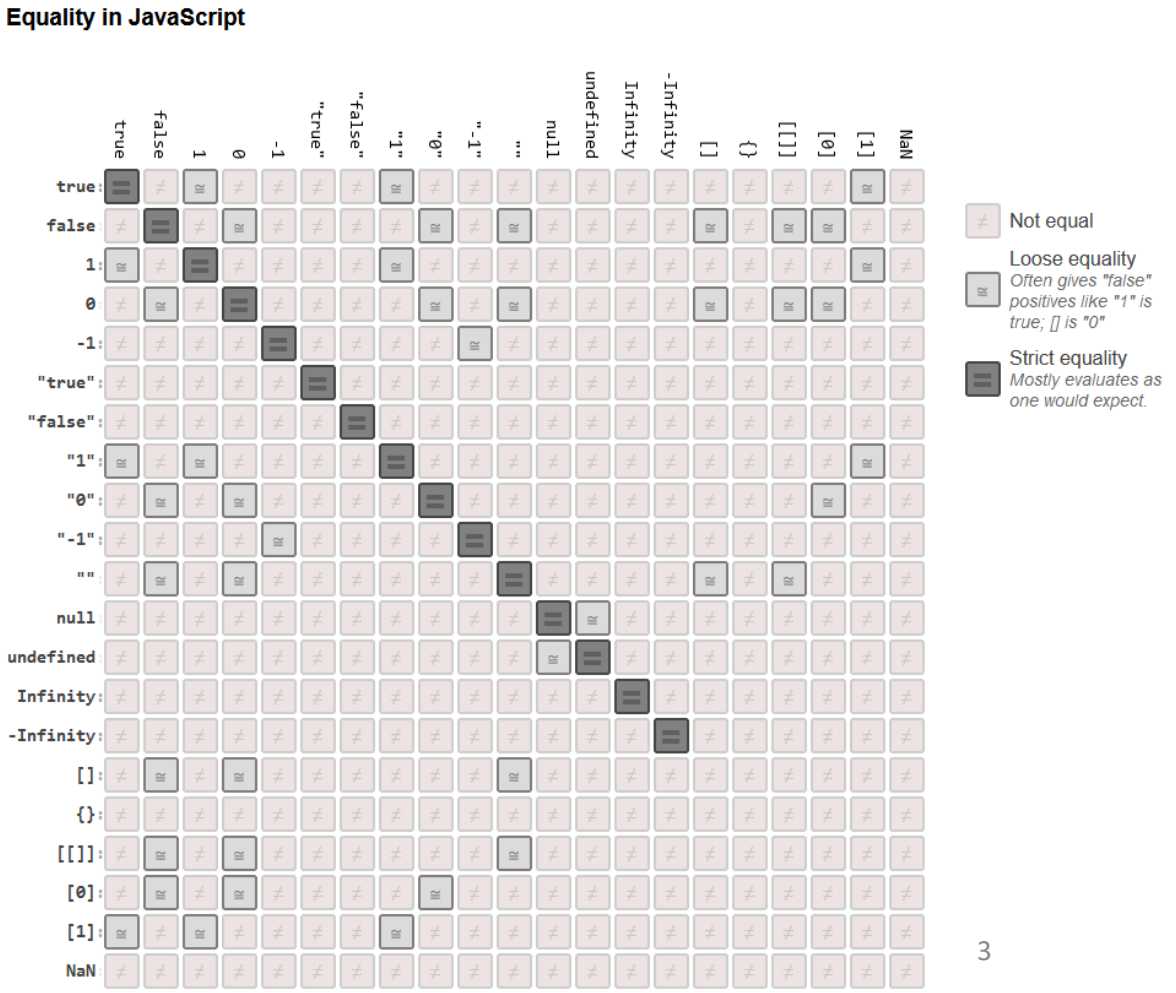
Coerção de tipos em JavaScript

Processo de conversão de um valor de um tipo, para outro

Exemplo: conversão de uma string para um número, um objeto para um booleano e etc.

Qualquer tipo, seja primitivo ou objeto, é um sujeito passível de sofrer coerção de tipo.

Recordando: os primitivos são: `number`, `string`, `booleano`, `null`, `undefined` + `Symbol` (adicionado no ES6).



Coerção Implícita vs Coerção Explícita

Coerção Explícita

Quando um desenvolvedor deseja converter um tipo escrevendo algo como, `Number(valor)`, isso é chamado de **coerção de tipos explícita** ou ***type casting*.**

Coerção Implícita

O JavaScript é uma **linguagem fracamente tipada**, valores também podem ser convertidos entre diferentes tipos automaticamente, e isso é chamado de **coerção de tipos implícita.**

Acontece ao atribuirmos operandos para valores de diferentes tipos. Por exemplo:

```
1 == null

2/'5'

null + new Date()
```

Também pode decorrer do contexto. Por exemplo, ao usar:

```
if (valor) {
        …
}
```

onde **`valor`** será forçado a retornar um **booleano**.

Ao executar `0 == "0"` por que o JavaScript retorna **`true`**?

`0` é um número e `"0"` é uma `string`...

Não deveriam devem ser iguais!

A maioria das linguagens de programação respeita isso.

Por exemplo, em Java ocorreria o seguinte erro:

```
error: incomparable types: int and String
```

No JavaScript, ao comparar dois valores via ==, um dos valores pode sofrer coerção de forma automática.

Ao invés do desenvolvedor converter explicitamente seus tipos, **o JavaScript faz isso por ele, por baixo dos panos**.

Isso é conveniente **se o desenvolvedor estiver fazendo de propósito**. Mas pode **ser potencialmente prejudicial** se ele não tiver conhecimento das implicações.

REGRAS PARA COERÇÃO

REGRA 1

Se **`x`** for **`Number`** e **`y`** for **`String`**, retorne

```
x == toNumber(y)
```

```
0 == "0"        // true
```

Pela regra, o segundo `0` **se torna um número**! Então...

`0 == 0` é **verdadeiro.**

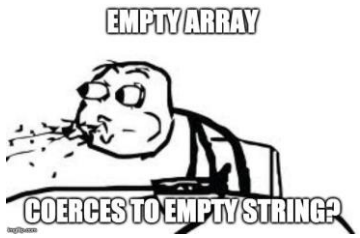
Arrays também sofrem coerção

A coerção **não se limita a primitivos como strings, números ou booleanos**. Exemplo:

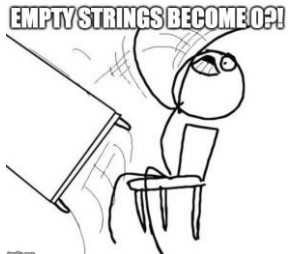
```
0 == [] // true ?????
```

Três pontos devem ser considerados aqui...

1) **Em JavaScript, arrays são objetos...**



2) **Um array vazio, por coerção, torna-se string vazia.**



3) **String vazia então se torna 0.**

REGRA 2

Se x for `String` ou `Number` e y for `Object`,

Então retorne

```
x == toPrimitive(y)
```

Três pontos devem ser considerados aqui...

1) **arrays são objetos...**

2) **Por coerção, um array vazio torna-se string vazia.**

3) **String vazia então se torna 0.**

De acordo com a especificação, **JavaScript primeiro procura o método toString de um objeto para fazer a coerção**.

Para **arrays**, o método **toString** junta todos os seus elementos e os retorna como uma **string**.

```
[1, 2, 3].toString() // "1,2,3"
['hello', 'world'].toString() // "hello,world"
[].toString() // ""
```

Depois de fazer a coerção do array para "", lembremos da primeira regra:

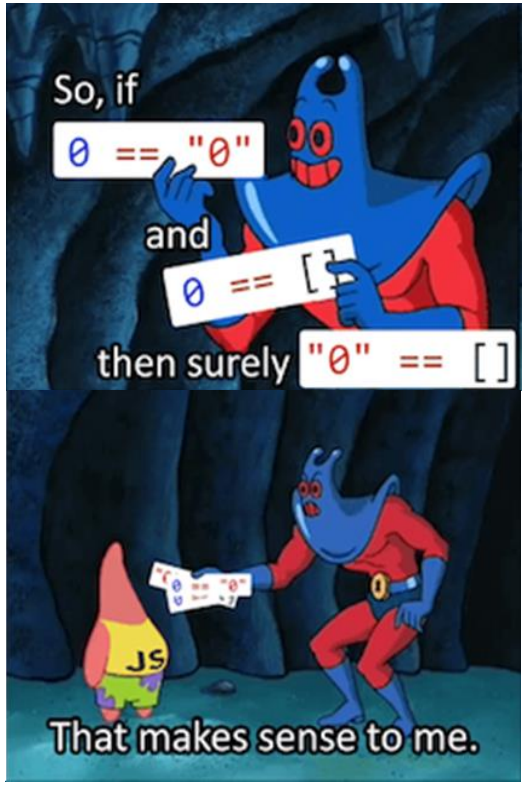
Se x for Number e y for String, retorne x == toNumber(y)

Então para 0 == ""

Como 0 é Number e "" é String, retorne 0 == toNumber("")

toNumber("") retorna **0**.

Portanto, 0 == 0 mais uma vez...




```
0 == "0" // true
```

Porque a coerção transforma a expressão em **`0 == toNumber("0")`**.

```
0 == [] // true
```

Porque a coerção ocorre duas vezes:

a) **`toPrimitive([])`** resulta em **`string`** vazia

b) Então **`toNumber("")`** resulta em **`0`**.

Então, de acordo com essas regras, o que isso deve retornar?

```
"0" == []
```


```
"0" == []
```

Seguindo a especificação mais uma vez:

Se **x** for **String** ou **Number** e **y** for **Object**, retorne **x == toPrimitive(y)**

Significa que:

Como **"0"** é **String** e **[]** é **Object**, retorne **x == toPrimitive([])**

toPrimitive([]) retorna **string** vazia. A comparação agora se tornou

```
"0" == ''
```

"0" e "" são strings, então o JavaScript diz que não é necessária mais coerção.

É por isso que a expressão `"0" == []` resulta em falso .

Use igualdade estrita ===

Use três iguais (igualdade estrita) e durma à noite.

```
0 === "0"     // falso
0 === []      // falso
"0" === []    // falso
```

O uso deste operador **evita totalmente a coerção**!

O benefício de seu uso é o aumento da confiança no próprio código, fazendo com que aquele pressionamento de tecla extra valha totalmente a pena.

Curso Superior de Tecnologia em Sistemas para Internet
Desenvolvimento front-end I (front 1)
Rafael Cunha Cardoso
rafaelcardoso@ifsul.edu.br

Coerção no JavaScript

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Pelotas