Programação Estruturada

Aula 09 - Comandos de Seleção

Videoaula 03: Coerção de Tipos e Curto-Circuito

Videoaula 03: Coerção de Tipos e Curto-Circuito

Devido a essa característica de JavaScript, recomenda-se usar os operadores de igualdade estrita, `===`, e desigualdade estrita, `!==`, para comparar se dois valores são iguais ou diferentes. Isso porque esses operadores retornam menos valores inesperados do que o `==` e o `!=`. Mas isso não impede, é claro, que você também use tais operadores. De qualquer maneira, existem diversas peculiaridades no uso desses operadores quando envolvemos a coerção de tipos. Vamos ver?

Nesse exemplo (ver Código 1), faremos várias comparações e coerções de tipo usando os operadores de igualdade e de igualdade estrita. Essa página HTML é muito simples, com um campo `resultado` novamente, no qual você escreverá um valor booleano gerado ao longo do exemplo.

Código 1 - 09_8 Coerção de Tipos.html

```html
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 09</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Coerção de Tipos</h1>

    <p id="resultado"></p>

    <script src="script.js"></script>
  </body>
</html>

```

```
// Note que usando == temos que false, 0 e '' são iguais,
// mas isso não acontece na igualdade estrita ===

var b = (false == 0);
//var b = (false === 0);
```

```
6
7  //var b = (false == '');
8  //var b = (false === '');
9
10 //var b = (0 == '');
11 //var b = (0 === '');
12
13 // null e undefined são ambos falsos, mas eles são iguais apenas
14 // a eles mesmos
15
16 //var b = (undefined == null);
17 //var b = (null == false);
18 //var b = (undefined == false);
19 //var b = (null == 0);
20 //var b = (undefined == 0);
21 //var b = (undefined === null);
22
23 // Apesar de Nan ser falso, ele não é igual nem a ele mesmo
24
25 //var b = (NaN == null);
26 //var b = (NaN == NaN);
27
28 document.getElementById("resultado").innerHTML = b;
29
```

Veja o que acontece se compararmos `false` com o valor 0. Será que o resultado é `true` ou é `false`? Será que o `false` é `==` a 0? Note que sim, `false` é `==` a 0. Porém, `false` não é `===` 0. Por quê?

Lembre-se, como já foi dito, que o `==` verifica valores e o `===`, a igualdade estrita, compara também o tipo. E, obviamente, temos, `false`, um tipo booleano e 0 um tipo numérico, então essa igualdade estrita é `false`, ok?

Outro exemplo é o `false` com a *string* vazia. Então, `false` é `==` a *string* vazia? Se carregarmos a página, veremos que dará "true". Porém, da mesma forma, se for feita essa comparação usando a igualdade estrita, teremos `false` `===` a *string* vazia, o valor é "false", porque estamos comparando um valor booleano com uma *string*.

Compare agora o valor 0 com a *string* vazia. O 0 é `==` a *string* vazia? Ao carregar a página, o resultado será "true" porque eles são igualmente convertidos em `false`. Porém, se usarmos a igualdade estrita `===`, e recarregarmos a página, veremos que o

resultado é "false" porque temos tipos diferentes: 0 é um número e a *string* vazia é uma *string*.

Veja com relação ao `null` e ao `undefined` como funcionam esses valores. Vamos compará-los e ver como isso acontece usando igualdade e igualdade estrita do `null`, do `undefined` e do `false`. Se compararmos o `undefined` com o `null` usando a igualdade simples, o `==`, veremos que o resultado dará "true", e eles são, de fato, iguais.

Porém, são iguais apenas entre eles mesmos. Veja que no JavaScript se você comparar o `null` com o `false` usando o `==`, ao carregar a página obtém como resultado "false". O `null == false` é "false", assim como o `undefined == false` também dará "false". Então o `null` e o `undefined` são iguais usando o `==` e o `null == undefined` dará "true", porém a comparação deles com `false` dará "false".

O que acontece se compararmos `null` com o 0? `null == 0`? O resultado será "false". O `null` não é `==` a 0. E o `undefined` seria `==` a 0? Da mesma forma, também será "false". Isso reforça aquela informação que apenas `null == undefined` será "true". Usando a igualdade simples, `==` temos "true", será que usando a igualdade estrita nós teremos também como resultado "true"? Ao recarregar a página HTML, temos o resultado "false". Por quê? Porque o `undefined` é um valor `undefined` e `null` é do tipo objeto, eles têm tipos diferentes, então apenas a igualdade simples dará "true" entre eles.

Ao falar sobre o `NaN`, *Not a Number*, apesar de ser "falso", ele não é igual a nada, nem a ele mesmo. Se compararmos o `NaN` com o `null` usando igualdade simples, o resultado será "false", como já dito, ele não é igual a nada nem, repetindo, a ele mesmo.

Então se voltarmos ao exemplo e compararmos `NaN == NaN`, a resposta não será "true", ao carregar a página a resposta continua sendo "false", ou seja, `NaN` não é igual nem a ele mesmo.

Assim como em outras linguagens de programação, os operadores lógicos são processados da esquerda para a direita. Assim que eles têm um valor definido, eles param de ser avaliados retornando o valor que parou o processamento. Essa característica interessante na definição dos valores booleanos das condições é chamada de curto-circuito. O valor retornado no processamento pode ser tratado como um valor verdadeiro ou falso, mas também pode ser tratado, simplesmente, como o valor retornado. É possível usar essa característica para fazer atribuições bem criativas como veremos no exemplo a seguir. Ademais, podemos também fazer isso para verificar a existência de elementos na página, como fizemos no último exemplo.

Por fim, nas condições com ou-lógico, recomendamos colocar primeiro as condições com mais probabilidade de ser `true` e com menos necessidade de processamento. Caso isso aconteça, o resto não precisará ser avaliado, visto que `true || P` é `true`, qualquer que seja o `P`, causando um curto-circuito e evitando a avaliação de `P`. O mesmo acontece com o e-lógico. Nesse caso, porém, recomendamos colocar primeiro as condições com mais probabilidade de ser `false` e com menos necessidade de processamento. Se isso acontecer, o resto não precisará ser avaliado, pois `false && P` é `false`, qualquer que seja o `P`, causando um curto-circuito e evitando a avaliação de `P`. Vamos ver exemplos disso?

Neste exemplo (ver Código 2), usaremos novamente uma HTML muito simples, na linha 11 vamos declarar o `resultado` e ficar escrevendo textos nesse `resultado` pra ver o comportamento do curto-circuito.

Código 2 - 09_9 Curto-Circuito.html

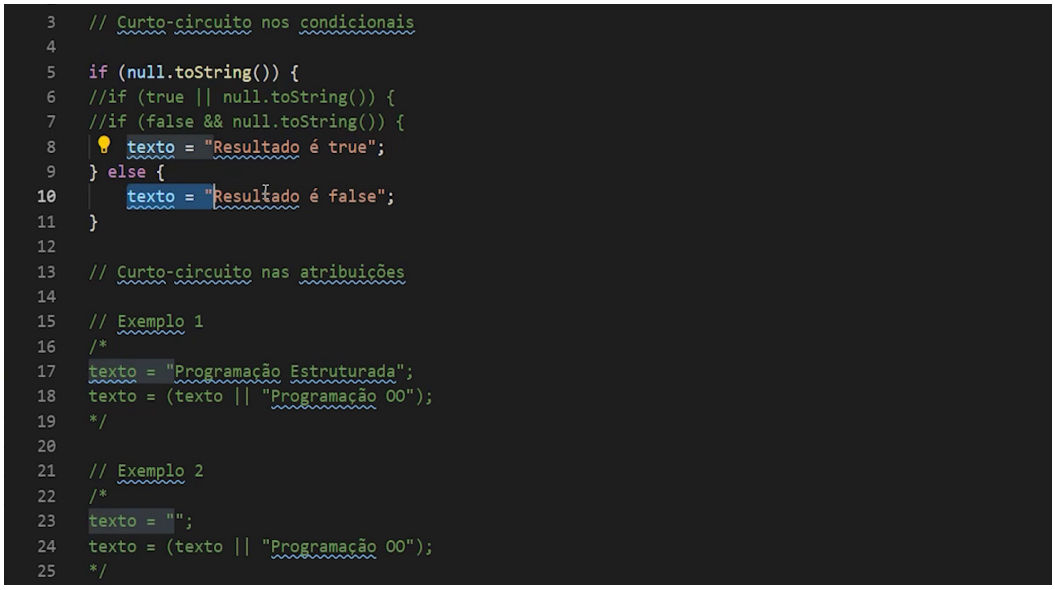
```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 09</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Curto-Circuito</h1>
10
11     <p id="resultado"></p>
12
```

```
    <script src="script.js"></script>
  </body>
</html>
```

```
var texto;

// Curto-circuito nos condicionais

if (null.toString()) {
//if (true || null.toString()) {
//if (false && null.toString()) {
  texto = "Resultado é true";
} else {
  texto = "Resultado é false";
}

// Curto-circuito nas atribuições

// Exemplo 1
/*
texto = "Programação Estruturada";
texto = (texto || "Programação OO");
*/

// Exemplo 2
/*
texto = "";
texto = (texto || "Programação OO");
*/

// Exemplo 3
/*
var x = 0;
var y = 1;
var z = 2;
if (x || y || z) {
  texto = "Resultado é true";
} else {
  texto = "Resultado é false";
}
*/

document.getElementById("resultado").innerHTML = texto;
```

Inicialmente, temos um `if-else` da linha 5 à linha 11 onde diremos que, se `if (null.toString())` se isso retornar uma *string*, nós atribuimos ao texto: "Resultado é true", caso contrário, "Resultado é false". Veja Figura 1.

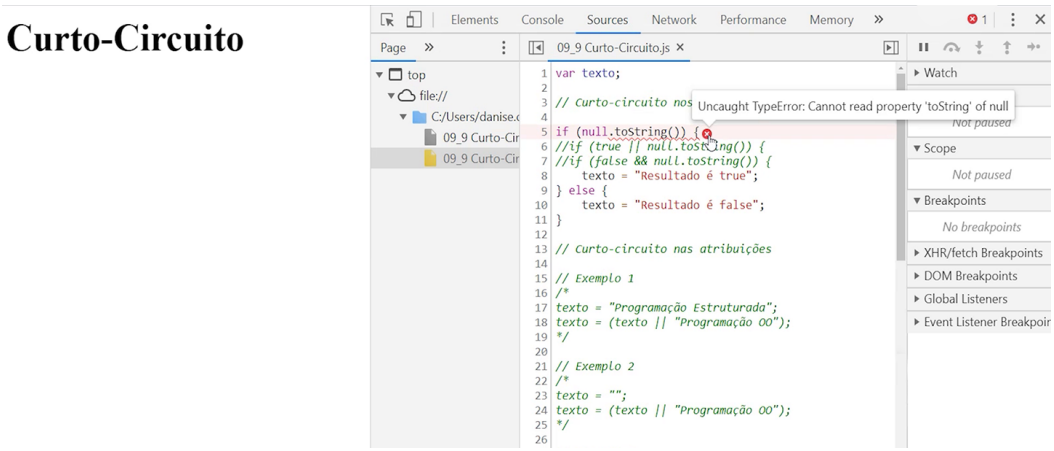
Figura 1 - Testando Curto-Circuito



E na linha 39, o que fazemos simplesmente é atribuir esse texto ao `innerHTML` daquele parágrafo. Então, na prática, escreveremos na página HTML o texto "Resultado é true" se `null.toString()`; caso contrário, "Resultado é false".

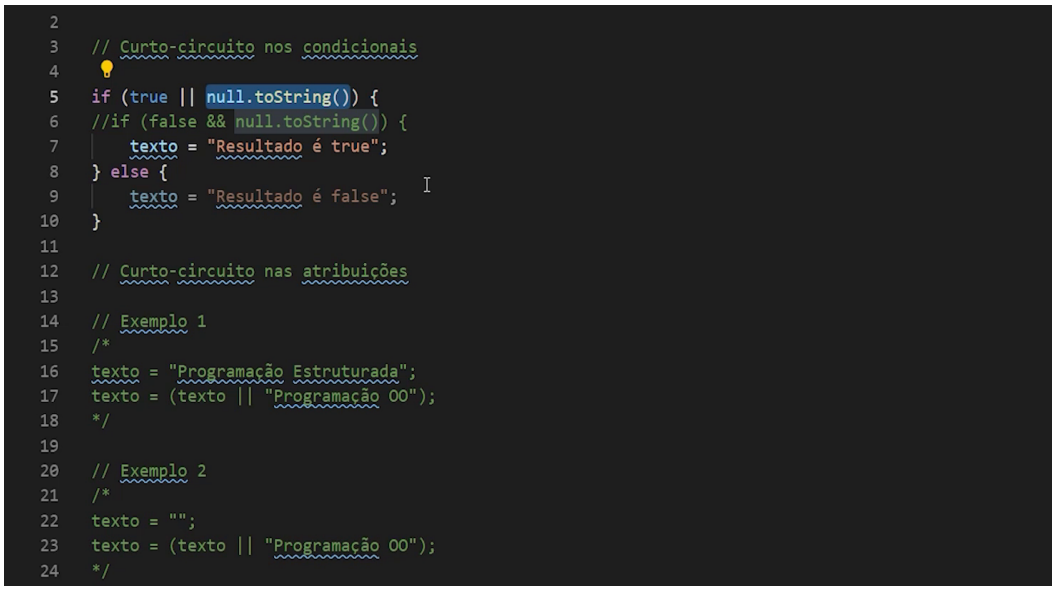
Um aspecto interessante em orientação a objetos, apesar de não ser o foco desta disciplina, mas vou adiantar, é que não podemos chamar métodos num objeto `null`. Se carregarmos a página, não será exibido nada, e se de fato entrarmos no depurador do Chrome temos um erro de tipo pois não é possível ler a propriedade `toString` do `null` (Ver a Figura 2).

Figura 2 - Identificando Erros na Ferramenta de Depuração



Temos um erro nessa página e isso foi causado porque chamei o método `toString` no valor `null`. Vamos ver o que acontece com o curto-circuito. Ao colocar o `if` na linha 5, temos `if (true || null.toString())`, como exposto na Figura 3, erro permanece na página. Continuamos chamando a propriedade `toString` no valor nulo. Mas estou usando isso dentro de um condicional onde a primeira parte é `true`. E como já foi dito na aula, true ou qualquer coisa, é sempre `true`. Então, teremos um curto-circuito. Quando ele verificar que tem um `true` e um `||`,ele vai parar a avaliação daquela expressão e vai executar de fato o “Resultado é true”. O erro simplesmente não vai aparecer por causa do curto-circuito. E isso é um bom exemplo de como ele funciona. Ao voltar à página HTML e carregá-la, aparecerá a mensagem o “Resultado é true”.

Figura 3 - Curto-circuito com o ou-lógico

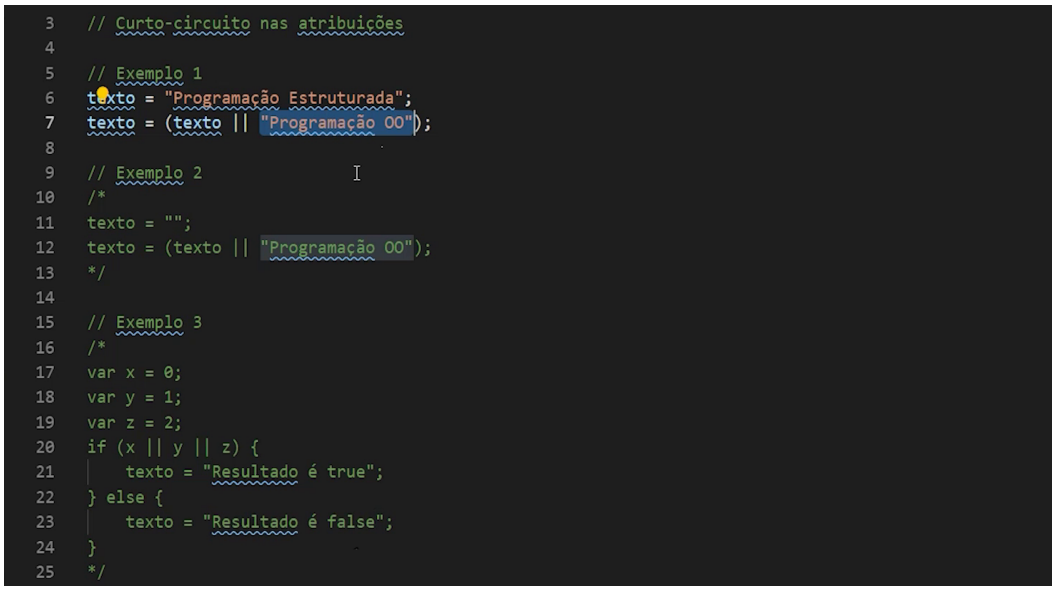


O erro que é causado ao chamar o `toString` no valor `null`, ele não acontece. Da mesma forma, teremos um curto-circuito se fizermos `if (false && null.toString())`, o erro continua, só que temos um `&&` lógico com `false`. E `false` com qualquer coisa é sempre `false`.

Novamente, temos um curto-circuito, só que dessa vez, ele nos leva ao `false`, então o que vai ser executado é o trecho do código do `else`, e ao recarregarmos a página, teremos a mensagem "Resultado é false" e aquele erro não será exibido.

Como é que funciona o curto-circuito nas atribuições? Eu tenho um `texto` que recebe o valor `"Programação Estruturada"` que é o nome da nossa disciplina. Faremos o seguinte, diremos que `texto = (texto || "Programação OO")`. Veja a Figura 4.

Figura 4 - Curto Circuito com *string* não vazia



```
3   // Curto-circuito nas atribuições
4
5   // Exemplo 1
6   texto = "Programação Estruturada";
7   texto = (texto || "Programação OO");
8
9   // Exemplo 2
10  /*
11  texto = "";
12  texto = (texto || "Programação OO");
13  */
14
15  // Exemplo 3
16  /*
17  var x = 0;
18  var y = 1;
19  var z = 2;
20  if (x || y || z) {
21      texto = "Resultado é true";
22  } else {
23      texto = "Resultado é false";
24  }
25  */
```

Vimos nesta aula que qualquer texto que tenha conteúdo é `true`. Então, o que é que acontece? O texto vai ser interpretado com o valor `true` e teremos `true || "Programação OO"`, um curto-circuito vai acontecer e permaneceremos com o texto `"Programação Estruturada"`, que é o texto retornado na avaliação da tentativa de converter esse texto num booleano, ele retorna o valor `"Programação Estruturada"`. Ao recarregar a página HTML, será exibido "Programação Estruturada".

Agora, se tivéssemos o texto vazio e tentássemos fazer essa atribuição, o texto vazio tem o valor `false`, ele avaliaria a outra expressão, `"Programação OO"`, que daria `true`, então o todo daria `true`, mas o valor de `"Programação OO"` quando fôssemos avaliar, ele retornaria o valor `"Programação OO"`, que é a nossa disciplina de Programação Orientada a Objetos. Isso foi um exemplo de curto-circuito nas atribuições. Veja a Figura 5.

Figura 5 - Curto Circuito com *string* vazia

Por fim, temos três variáveis, `x = 0`, `y = 1` e `z = 2`. Faremos uma condição no `if` que faz `x || y || Z`, como exposto na Figura 6, ele vai tentar transformar cada um dos valores que foram atribuídos às variáveis num valor booleano. E, ao carregar a página HTML, o resultado esperado será "true", porque apesar de `x` ser `false`, ele continua e avalia o `y`. Por isso o resultado será "true", porque será um valor numérico diferente de 0. Nesse momento, ele já fez um curto-circuito e não chegou nem a avaliar a variável `z`. Observe que se deixarmos só `x` aparecerá "Resultado é false".

Figura 6 - Curto-Circuito nas Condicionais

Esses foram alguns exemplos de curto-circuito tanto nas condicionais quanto nas atribuições.