Programação Estruturada

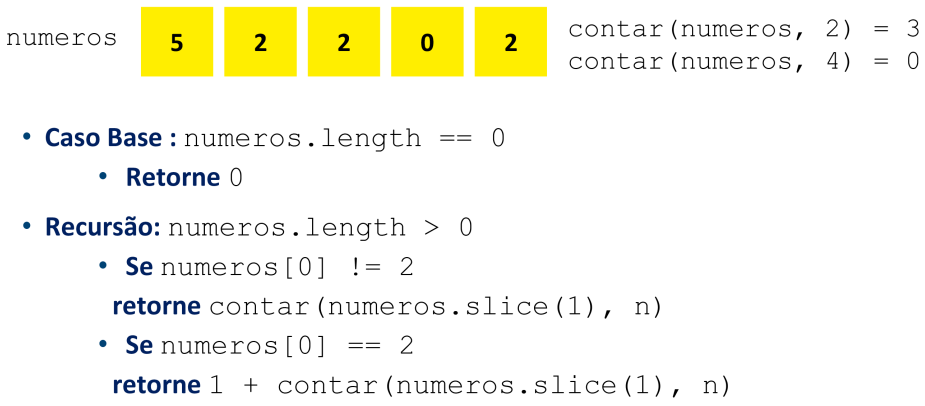
Aula 14 - Recursão: Arrays, Matrizes e Exercícios

Videoaula 02: Contando Elementos em Arrays

Videoaula 02: Contando Elementos em Arrays

Agora, vou apresentar um exemplo de uma função recursiva que retorna a quantidade de vezes que um determinado elemento aparece em um array. Por exemplo, supondo que temos o array `numeros` apresentado no slide (Figura 1), essa função retorna 3 para o valor 2 e 0 para o valor 4, pois não temos o elemento 4 no array.

Figura 1 - Contando ocorrências de elementos



Novamente, tem-se que o **caso base** é quando o array está vazio. Nesse caso, é claro que o resultado tem que ser 0, não acha? Afinal de contas, quantas vezes qualquer valor acontece no array vazio? Nenhuma. Você concorda?

Bem, e no caso recursivo? O retorno vai depender do valor que se tem no início do array. Se ele for diferente do valor `n` procurado, temos que simplesmente retornar a quantidade de vezes que `n` ocorre no resto do array. Caso o valor do primeiro elemento do array seja igual a `n`, adicionamos 1 à quantidade de vezes que `n` ocorre no resto do

array. Lembre-se que já vimos em aulas anteriores que o "resto" do array pode ser obtido usando a função `slice`, ou seja, a expressão `numeros.slice(1)` remove apenas o primeiro elemento do array retornando o "resto" do array. Portanto, para contar a quantidade de vezes que ocorre no "resto" do array, invocamos a recursão `contar(numeros.slice(1), n)`.

Nesse momento, irei apresentar o funcionamento das chamadas recursivas. No primeiro exemplo, onde é chamada a função `contar` com o array `[5, 2, 2, 0, 2]` e o elemento 2, esperamos que o resultado seja 3, pois o valor 2 aparece três vezes nesse array.

Veja como funciona a pilha de chamadas!

Figura 2 - Contando ocorrências de elementos

```
(1)     contar([5, 2, 2, 0, 2], 2)
(2)     = contar(slice([5, 2, 2, 0, 2], 1), 2)
(3)     = contar([2, 2, 0, 2], 2)
(4)     = 1 + contar(slice([2, 2, 0, 2], 1), 2)
(5)     = 1 + contar([2, 0, 2], 2)
(6)     = 1 + (1 + contar(slice([2, 0, 2], 1), 2))
(7)     = 1 + (1 + contar([0, 2], 2))
(8)     = 1 + (1 + contar(slice([0, 2], 1), 2))
(9)     = 1 + (1 + contar([2], 2))
(10)    = 1 + (1 + (1 + contar(slice([2], 1), 2)))
(11)    = 1 + (1 + (1 + contar([], 2)))
(12)    = 1 + (1 + (1 + 0))
(13)    = 1 + (1 + 1)
(14)    = 1 + (1 + 1)
(15)    = 1 + 2
(16)    = 3
(17)    = 3
```

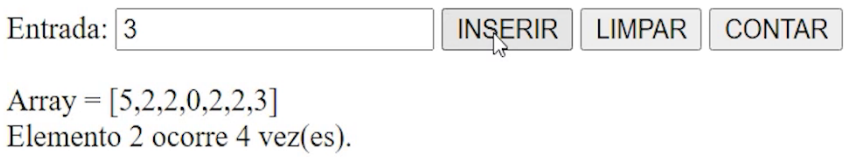
A primeira chamada na linha 1 faz a chamada na linha 2. Note que não foi adicionado o 1, pois o primeiro elemento, o número 5, não é igual a 2. A linha 3 simplesmente substitui a chamada `slice([5, 2, 2, 0, 2],1)` pelo seu resultado, ou seja, o array `[2, 2, 0, 2]`. A chamada na linha 3 faz a chamada na linha 4. Note que agora o número 1 foi adicionado, pois o primeiro elemento do array é igual a 2. O mesmo acontece com a chamada da linha 5, pela mesma razão. O resultado da

chamada da linha 7, porém, não adiciona 1. No entanto, novamente, a chamada da linha 9 adiciona o 1, pois mais uma vez o primeiro elemento é o 2. Por fim, na linha 11, chegamos ao **caso base** e, na linha 12, o número 0 é retornado. A partir desse momento, começamos a desempilhar as chamadas retornando 1 nas linhas 13 e 14, 2 na linha 15 e, finalmente, 3 nas linhas 16 e 17.

Veja como podemos escrever esse programa e, muito importante, como acompanhar essa pilha de chamadas.

Nesse exemplo, temos o botão INSERIR onde é possível colocar os elementos 5, 2, 2, 0, 2 e ele vai inserindo no array. Tem o botão LIMPAR que vai limpar o array, e o botão CONTAR que ao apertá-lo, vou colocar um *prompt* solicitando que digite um valor que desejo localizar, por exemplo vamos colocar o 2. Ao clicar em OK, ele informa: "Elemento 2 ocorre 3 vez(es)" e a partir de agora, ele vai estar sempre contando à medida que for inserindo. Então se eu inserir mais um 2, ele já aumenta a contagem. Entretanto, se eu inserir um 3 nada ocorre com a contagem, mas o array foi alterado, como mostra a figura 3. ok?

Figura 3 - Contando Ocorrências em um Array



Como fica a nossa solução? O nosso HTML é simples, tem uma entrada e três botões, um para inserir, um para limpar e um para contar chamando as funções, respectivamente, `inserir`, `limpar` e `contar_elemento`. E o parágrafo que é onde vai está escrevendo o nosso resultado.

Código 1 - 14_5 Contar Elementos.html

```
1  <html >
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 14</title>
5    </head>
6    <body>
```

```
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Contando Ocorrências em um Array</h1>

    Entrada: <input type="number" id="entrada" value="">
    <button onclick="inserir()">INSERIR</button>
    <button onclick="limpar()">LIMPAR</button>
    <button onclick="contar_elemento()">CONTAR</button>
    <p id="resultado"></p>

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

```
var numeros = [];
var elemento;

function inserir() {
  var x = Number(document.getElementById("entrada").value);
  numeros.push(x);
  imprimir();
}

function limpar() {
  numeros = [];
  imprimir();
}

function contar_elemento() {
  do {
    elemento = prompt("Digite um número:");
  } while (elemento == "" || isNaN(elemento));
  imprimir();
}

function imprimir() {
  var resposta = "Array  = [" + numeros + "] <br>";
  if (!isNaN(elemento)) {
    var contagem = contar(numeros, elemento);
    resposta = resposta + "Elemento " + elemento + " ocorre " + contagem + " vez(es). <br>";
  }
  document.getElementById("resultado").innerHTML = resposta;
}

function contar(a,e) {
  var contagem;

```

```
34    if (a.length == 0) {
35        contagem = 0;
36    } else {
37        if (a[0] != e) {
38            contagem = contar(a.slice(1), e);
39        } else {
40            contagem = 1 + contar(a.slice(1), e);
41        }
42    }
43    return contagem;
44 }
45
```

No JavaScript vai ter, além do array onde armazenamos os números, uma variável chamada `elemento`, que vai armazenar o valor que estamos procurando. Note que esse valor inicialmente é *undefined* porque a gente não definiu um valor para ele.

A função para inserir um elemento, já vimos e não tem mistério, a função `limpar` também e para contar o elemento tudo que vou fazer é um *prompt* que é uma função em JavaScript que abre a janela, e nela aparecerá "Digite um número:". E ao digitar o texto e dar OK, ele passa esse valor que ele pegou para a variável `elemento` através da atribuição da Linha 17.

Então verificamos se esse elemento inserido é de fato um número e que o texto não é vazio, ou seja, verifica se o elemento é vazio ou não é um número usando a função `isNaN`. Se uma dessas condições for satisfeita, ele vai pedir um novo valor. Por exemplo: Se na nossa página apertarmos `contar` e inserirmos vazio, ele pede de novo. Se eu botar um texto ele pede de novo, até que eu coloque um número.

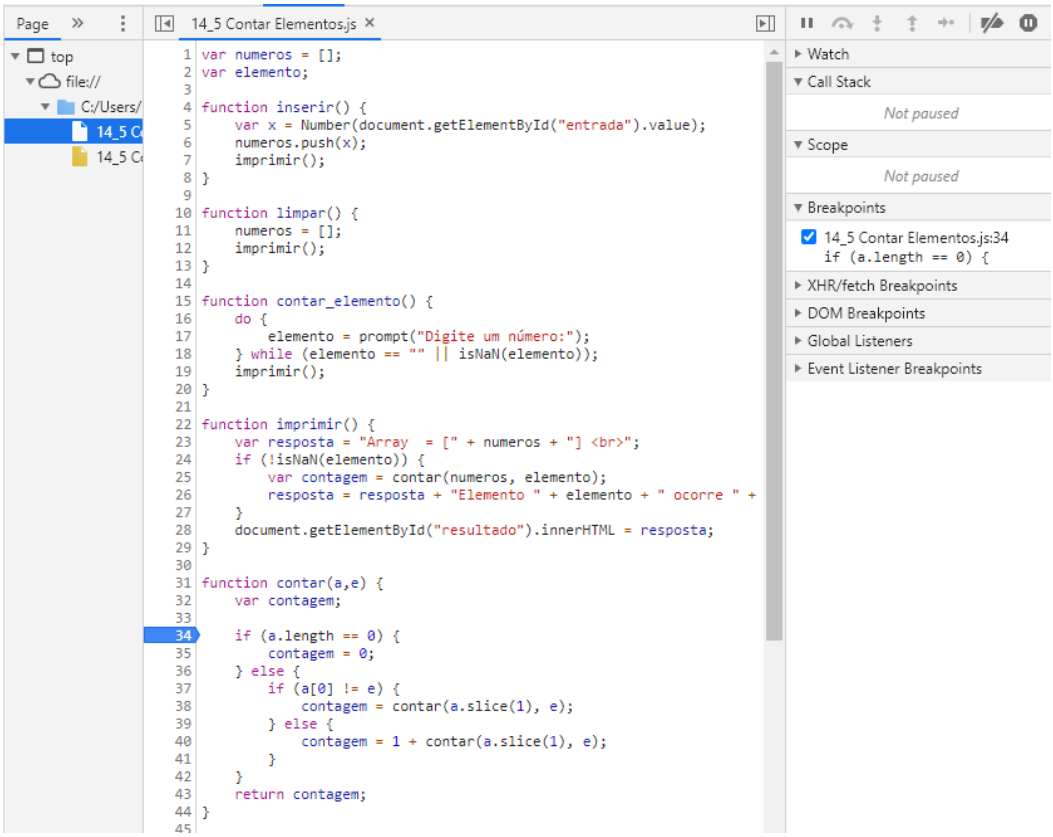
Em seguida, ele chama a função `imprimir`, que além de imprimir um array, imprime a contagem. Essa contagem, na verdade só é feita se o meu elemento tiver definido, e para isso, escolhi usar a comparação usando `isNaN`: não é o caso que o elemento não é um número, logo, ele é um número. O *undefined* não passa por esse teste, então não entraria. Na linha 25 é definida a variável `contagem` com a função `contar` passando o array de números e o elemento. Então é impresso na resposta que diz a quantidade de vezes que o elemento ocorre. E altero novamente a página com essa *string* que construí (linha 28).

Nosso foco neste exemplo é saber com essa função `contagem`, de um determinado array e de um determinado elemento, quantas vezes esse elemento ocorre nesse array, ela é definida. Para isso, começamos declarando a variável `contagem` e se o array chegou no final, ou seja, se o `a.length` é igual a 0, a contagem é 0, porque esse elemento ocorre 0 vezes no array vazio, e esse é o nosso caso base para essa função, ok?

Caso contrário, terei que, se `a[0]` for diferente do `e`, vou continuar contando, mas como é diferente, não vou incrementar o contador que tenho, então simplesmente vou contar no resto. Assim, vou contar esse elemento `e` no `a.slice(1)` que é o resto, a calda desse array, caso contrário, é porque ele é igual e se ele for igual, vou dizer que a minha `contagem` é 1, porque eu contei 1 mais o que eu contar no resto, então é 1 mais `contar(a.slice(1),e)` e retorno a minha `contagem`. Com essa solução conseguimos ter aquele comportamento que vimos na página.

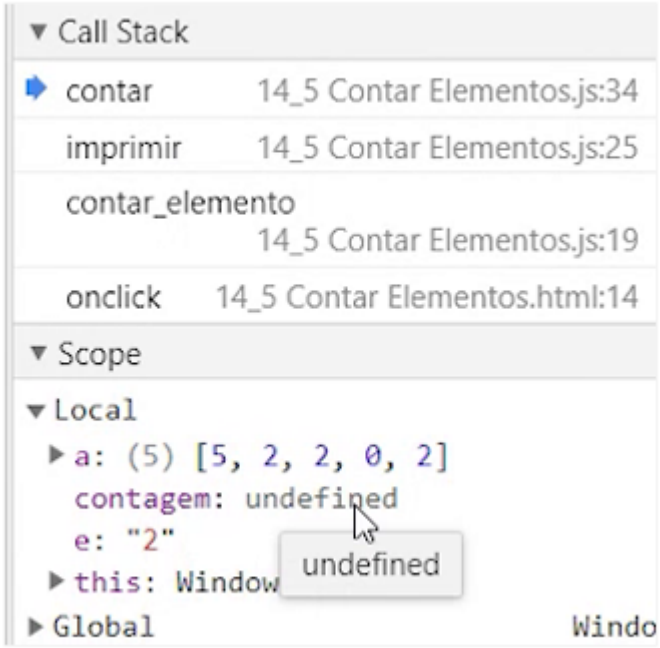
Agora, vamos ver como podemos acompanhar essa pilha de chamadas recursivas através da ferramenta de depuração do Google Chrome.

Figura 4 - Tela da Ferramenta de Depuração do Google Chrome



Iniciando pela página, temos aqui o nosso array [5, 2, 2, 0, 2], então vou chamar a ferramenta de depuração do Google Chrome, teclando F12 que é a tecla de atalho do Google Chrome, e vou criar um *breakpoint* na linha 34 dentro da função `contar`. Em seguida, vamos chamar essa função, passar o elemento 2 e paramos no *breakpoint* na linha 34, onde temos a pilha de chamadas, a *Call Stack*. Na *Call Stack*, *onclick* chamou `contar_elemento`, que chamou a função `imprimir` que chamou a função `contar`. Em `contar`, no contexto um array completo [5, 2, 2, 0, 2] e notem que a contagem está indefinida, vamos calcular o valor de `contagem`.

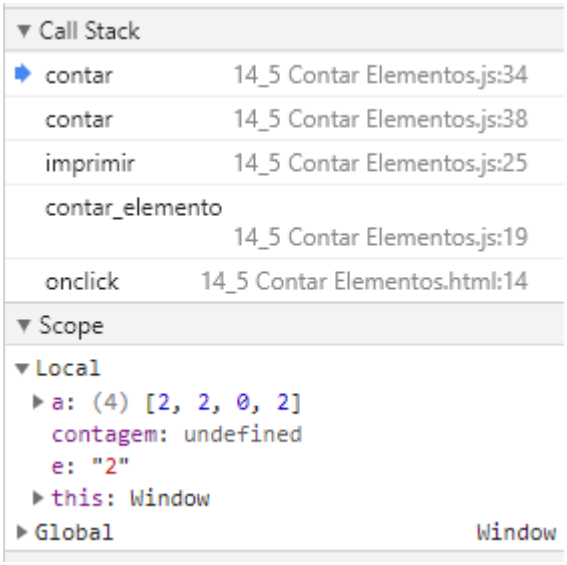
Figura 5 - Pilha de Chamadas e Escopo Iniciais



O primeiro elemento é o 5 o array não é vazio, então pulamos a linha 34, a condição do `if` não entra e vamos tentar no `else`, e o primeiro elemento é o 5 que é diferente do `e`, assim entramos de fato na linha 38 e vamos chamar recursivamente a função `contar` com o resto, com o array [5, 2, 2, 0, 2] tirando o primeiro elemento.

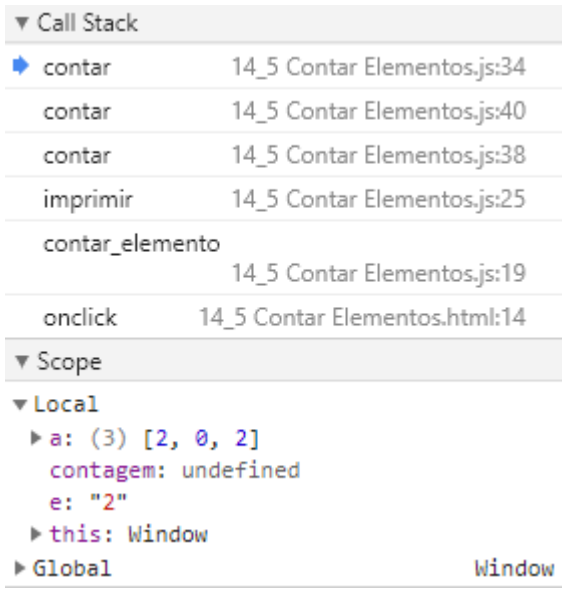
Então se chamarmos a função `contar`, inserimos mais uma chamada na *Call Stack*, só que agora o contexto é [2, 2, 0, 2].

Figura 6 - Atualizando a Pilha de Chamadas e o Escopo



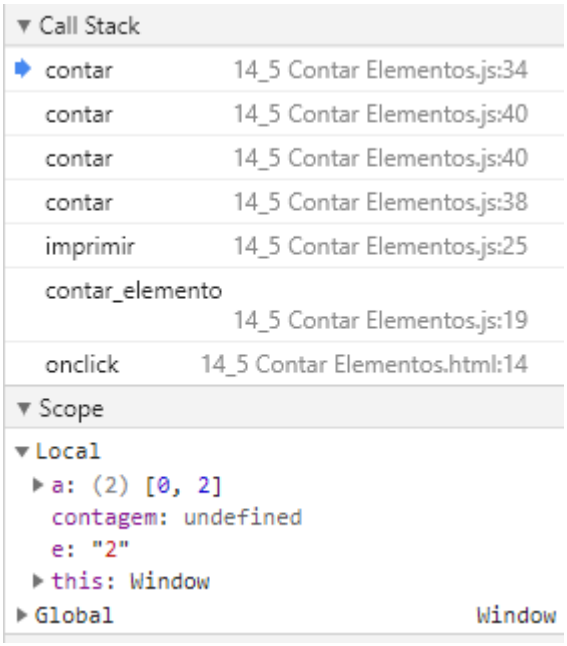
Novamente não é vazio, pulamos a linha 34, e o primeiro elemento é igual ao 2, ou seja, não vamos entrar na linha 38, mas na linha 40 e nela fazemos o quê? "Chamamos `1 +` ", e recursivamente a função `contar` com o resto que vai ser [2, 0, 2], para isso inseri mais uma chamada na *Call Stack* e agora temos o array `[2, 0, 2]`.

Figura 7 - Atualizando a Pilha de Chamadas e o Escopo



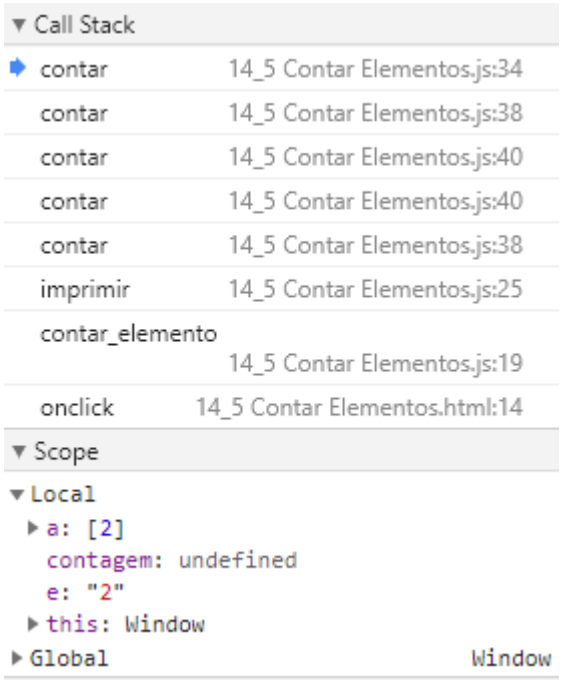
Novamente o array não é vazio, o primeiro elemento é igual ao 2, então entramos na linha 40 onde fazemos novamente `1 + contar` do resto, e vai inserir mais uma chamada na *Call Stack*.

Figura 8 - Atualizando a Pilha de Chamadas e o Escopo



Temos agora o array `[0, 2]` que não é vazio, logo, pulamos a linha 35. Na linha 37, o primeiro elemento é diferente dessa vez, então entramos na linha 38, e a contagem vai ser simplesmente a contagem do resto que vai ser o array [2], então mais uma chamada na *Call Stack*, agora temos 5.

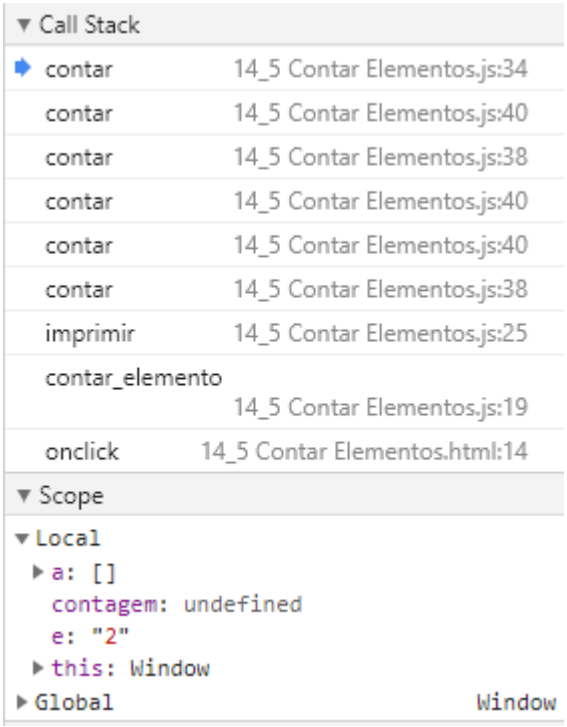
Figura 9 - Atualizando a Pilha de Chamadas e o Escopo



O array tem o elemento 2 apenas, não é vazio e o primeiro elemento que é igual a 2, então novamente pulamos a linha 38 e entramos na linha 40 onde a contagem é `1 +` contar do resto.

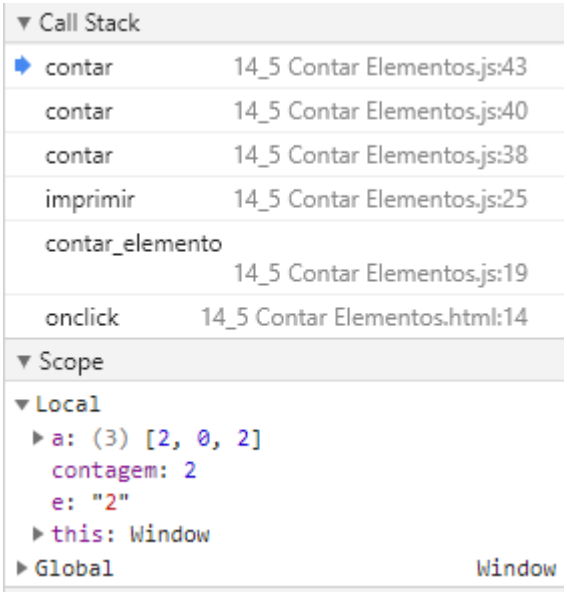
Qual o resto do array [2]? Temos a chamada do contar e o resto é o array vazio, finalmente, chegamos no caso base onde finalmente temos que `a.length==0`, porque nós temos o array vazio. Logo, entramos na linha 35 e a contagem agora vai deixar de ser *undefined* para ser 0, ok? E esse é o retorno da última chamada da pilha que vai retornar 0 e a gente vai desempilhar uma chamada.

Figura 10 - Atualizando a Pilha de Chamadas e o Escopo



Após desempilharmos, como a chamada retornou 0, a gente fica com 1 + 0, que é 1, e esse vai ser o retorno da última chamada do topo da pilha, então também desempilhamos ela e essa chamada retornou 1, a contagem vai passar a ser 0, porque estamos no contexto onde é 0 e 2, então retornou 1, e agora retornamos 1 e estamos no contexto onde o primeiro elemento é o 2 e teremos 1 + 1 = 2 que é o retorno desse topo.

Figura 11 - Atualizando a Pilha de Chamadas e o Escopo



Prosseguimos desempilhando, então mais uma vez retornamos 2, desempilhamos 1 da pilha e vamos retornar 2 novamente, retornamos 3 agora e chegamos finalmente à primeira chamada da função `contar`, onde temos o array completo e o retorno é 3. E esse é o retorno que essa função vai dar para a função `imprimir`, e assim voltamos para a função `imprimir`. Construímos a resposta com 3 e vamos imprimir na tela, então imprimimos "O elemento 2 ocorre 3 vezes". É assim que a gente pode acompanhar a pilha de chamadas recursivas no Google Chrome, ok?

Figura 12 - Contando Ocorrências em um Array

Percebeu que podemos utilizar a ferramenta de depuração não só para identificar erros, mas também para entender melhor a execução do código? Top, não é mesmo?

Fico por aqui nesta videoaula e, na próxima, você verá um exemplo bem bacana de como encontrar um elemento em um array ordenado sem usar o método `indexOf`. Ele será bastante útil em linguagens de programação que não têm esse recurso. Até mais!