Programação Estruturada

Aula 05 - Entendendo o seu Código: depuração

Videoaula 04: Depuração no Google Chrome

Material Didático do Instituto Metrópole Digital - IMD
Versão 5.3 - Todos os Direitos reservados

Videoaula 04: Depuração no Google Chrome

Nesta videoaula, você vai aprender a aplicar os principais conceitos de depuração no Google Chrome. Para isso, vamos utilizar o exemplo de uma página que calcula a média de dois números dados como entrada.

Agora, vamos identificar o mesmo erro, mas usando as ferramentas de depuração do Chrome. Veja aqui nossa página HTML, vamos exibir o código fonte dela, note que ela está limpa, tem só os dois campos de entrada e o campo de saída, e o nosso JavaScript também, veja que ele não tem nenhum registro de `document.write` ou de `console.log`, ou de `innerHTML`, é simplesmente o código fonte dele, ok?

Código 1 - 05_3 Media.html

```html
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Programação Estruturada - Aula 04</title>
  </head>
  <body>
    <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

    <h1>Situação do Aluno</h1>

    N1: <input type="number" id="N1" value="">
    N2: <input type="number" id="N2" value="">
    <button onclick='situacao()'>OK</button>
    <p id="resultado"></p>

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

```

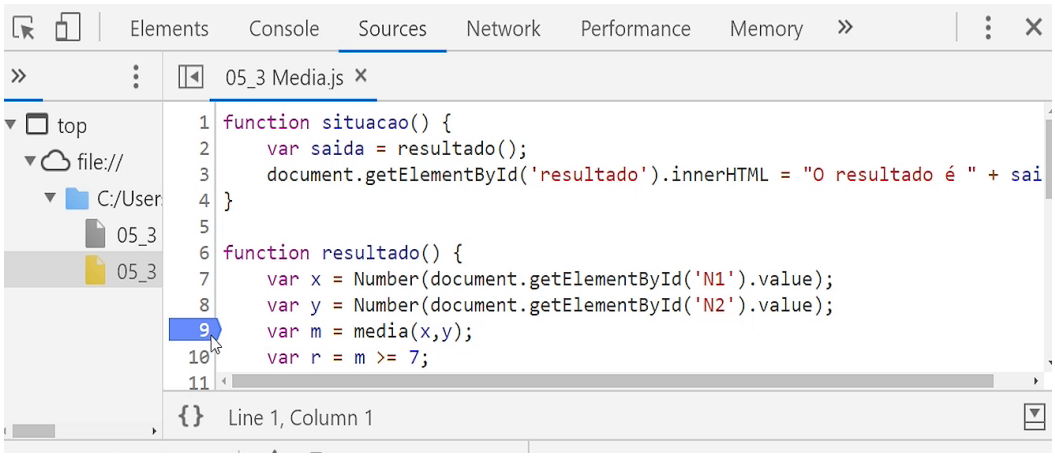
```
function situacao() {
  var saida = resultado();
  document.getElementById('resultado').innerHTML = "O resultado é " + saida;
}

```

```
6  function resultado() {
7     var x = Number(document.getElementById('N1').value);
8     var y = Number(document.getElementById('N2').value);
9     var m = media(x,y);
10    var r = m >= 7;
11
12    return r;
13 }
14
15 function media(a, b) {
16    var c = a + b/2;
17    return c;
18 }
19
```

E o erro permanece na página HTML, se eu colocar 5 e 6 nos campos, note que o resultado que aparece é “true”. A questão agora é identificar esse erro, vamos abrir a ferramenta de depuração do Chrome, apertando F12, e agora, ao invés de usar o `console` para escrever as saídas, iremos usar os `sources`.

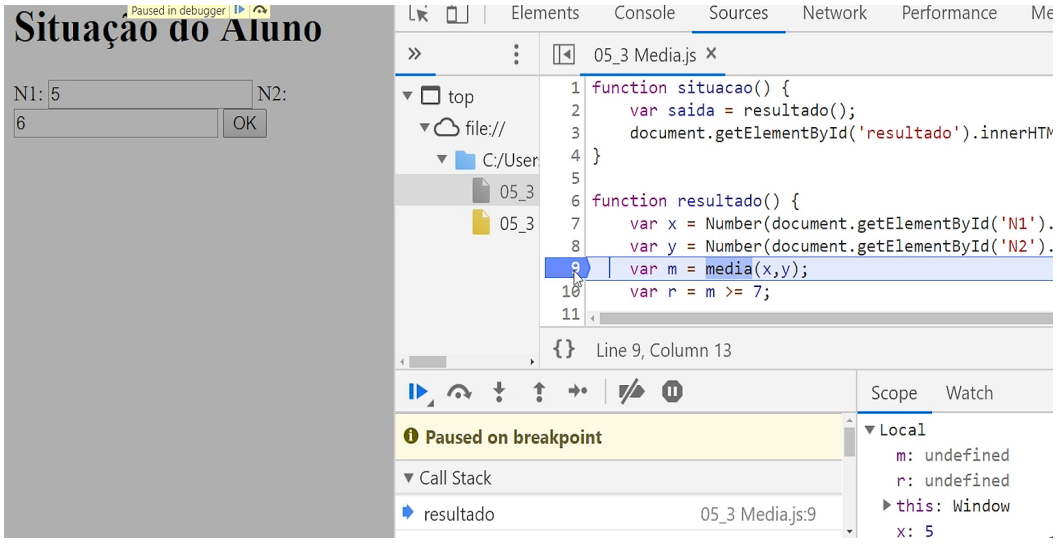
No `sources` temos `media.js`. A primeira coisa que a gente precisa fazer é criar breakpoints, então se eu clicar no número 9, acabei de criar um breakpoint na linha 9, e isso é indicado por essa essa seta azul cobrindo o número 9.

Figura 1 - Indicação de Breakpoint



Ao recarregar a nossa página, insiro os mesmos dados (5 e 6) e, ao clicar em OK, a execução está pausada na linha 9.

Figura 2 - Execução pausada na linha 9



E temos também a janela de escopo local dessa função que estamos atualmente, que é a função resultado. Temos o nosso `m` que é a nossa variável e o nosso `r`, ambos sem valores definidos porque nós não chamamos a função `media` ainda. O `r` também está indefinido porque ele só é definido na próxima linha, mas o `x` tem o valor 5 e o `y` também tem o valor 6. Veja que não foi preciso imprimir na tela, ou em qualquer outro lugar, para saber quais são os valores de `x` e `y`, já sabemos que temos os valores 5 e 6 para esses dois.

Na janela do *Watch* podemos adicionar valores, note que eu posso escrever expressões clicando no +, eu posso escrever, por exemplo: `typeof x`, ele me mostrará o tipo de `x` que, por ser um 5, ele é um número. Ou eu posso buscar o resultado de `(x + y) / 2`, então qualquer expressão que eu escrever aí ele vai mostrar, ok?

Figura 3 - Exibindo o valor de uma expressão

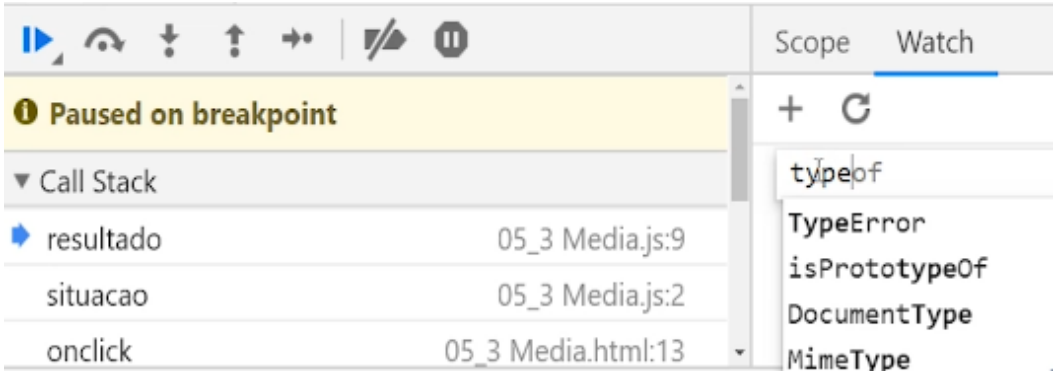
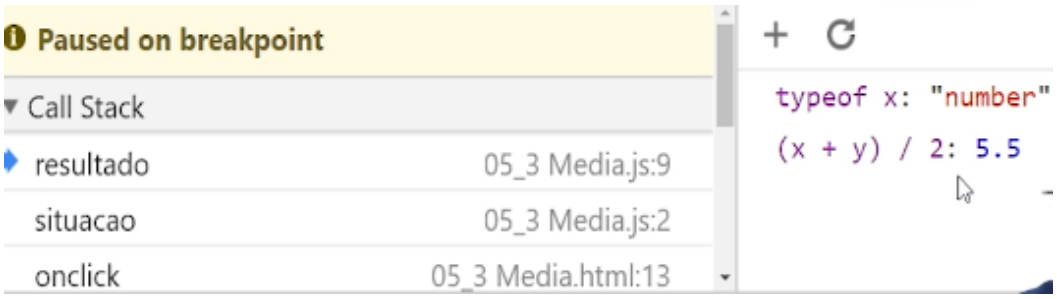
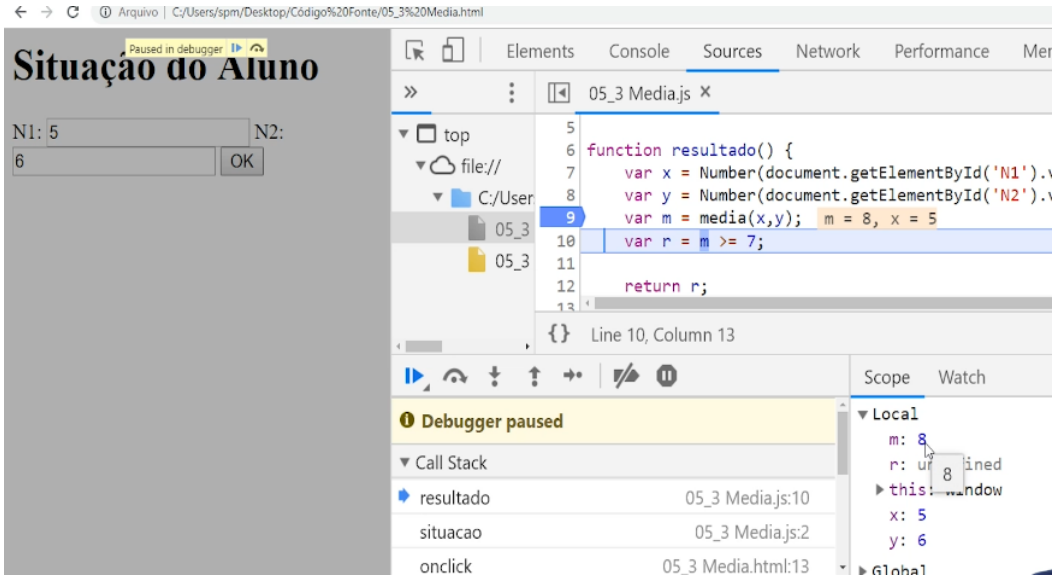


Figura 4 - Exibindo os valores de nossos exemplos



Então, o que eu posso fazer aqui é dar um *step over*, veja que eu tenho a seta, essa seta passa por cima do ponto, é o *step over*. Se eu der um *step over*, ele vai executar a função `media` e atribuir ao `n`, então eu aperto o *step over* e ele executou, foi para a próxima linha, e a gente vê aqui que o `m`, de fato, ele recebeu o valor 8, tá? Então nesse valor 8 já foi o resultado da `media`, e claramente podemos ver que `m` está com o valor errado.

Figura 5 - Resultado do *Step Over*



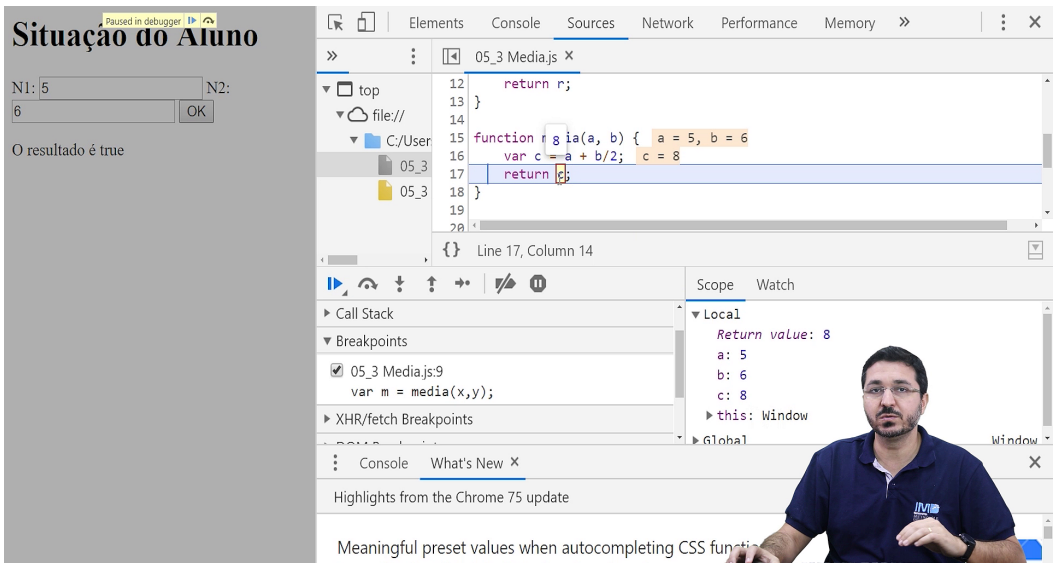
Então, o que a gente pode fazer é tentar entrar na função `media`, onde identificamos o erro, pra ver o que aconteceu lá dentro. Eu posso dar um *resume* (`paused in debugger`), aqui em cima do HTML, e ele termina a execução da minha página HTML. Aí eu posso, sem nenhum problema, reiniciar essa execução simplesmente clicando em OK.

Reiniciei, como ele identificou que o programa está dentro da função `media`, ao invés de dar um *step over*, para passar por cima da chamada dessa função, vou dar um *step into*, e, como eu falei, nesse caso aqui, essa função não é assíncrona, não iremos entrar nos detalhes do que é uma função assíncrona mas, por não ser uma função assíncrona, o comportamento *step into* vai ser exatamente igual ao comportamento do F9, do *step*, e ele vai entrar na função `media`. Certo?

Veja que ele foi direto para a função `media`, e o nosso escopo acrescentou a variável `c`, que continua indefinida, porque essa linha 16 não foi executada ainda.

E eu queria mostrar outra coisa, o *call stack*, a pilha de chamadas que tem ao clicarmos na página HTML. O `onclick` chamou a `media`, e ele diz qual foi a linha do JavaScript que aquela chamada aconteceu. A função `situacao` chamou a função `resultado`, que chamou a função `media`. O *call stack* fica aqui nessa parte da ferramenta de depuração. Então, vamos agora dar um outro *step*, isso vai nos levar à linha 17, e aí vemos que o valor de `c` é 8.

Figura 6 - Analisando o Erro do Retorno da Função



Perceba que essa linha `c` agora deveria ter o valor 5.5 e não 8, como está aparecendo aqui no escopo, dizendo que a variável c está com valor 8. Então, claramente, o erro está aqui nessa linha, como você já viu nos exemplos sobre as outras maneiras de visualizar os valores dessas variáveis.

Salvamos e vamos refazer a execução, ok? Ao executar novamente a página HTML, voltamos a ficar pausados na linha 9, vamos agora fazer um *step over*, pois acreditamos que resolvemos o problema, e vemos que o *step over* coloca em `m`, de fato, o valor 5.5. Então, conseguimos corrigir o erro, só que agora usando a ferramenta de depuração que é bem mais eficiente para você acompanhar a execução do programa, tá certo?

Termino aqui a nossa aula sobre depuração, na qual você aprendeu os principais conceitos de depuração e como executar essa atividade de várias maneiras, desde usando as saídas de seus programas JavaScript até o uso da ferramenta de depuração do Chrome.

#FicaDica

Recomendo que você leia o documento de referência da ferramenta de depuração JavaScript do Chrome através deste link:

https://developers.google.com/web/tools/chrome-devtools/javascript/reference

Com certeza, agora você terá mais facilidade de encontrar eventuais erros que os seus programas JavaScript venham a ter. É interessante que você faça alguns exercícios para treinar o uso da ferramenta de depuração do Chrome e identificar erros. Você pode até fazer os mesmos exercícios em outros navegadores para ver como aplicar os conceitos de depuração nas ferramentas de depuração deles. Nos exercícios, você encontrará alguns programas com erros inseridos neles e terá de encontrar o erro e corrigi-lo usando a ferramenta de depuração. Vamos lá?