Programação Estruturada

Aula 04 - Funções

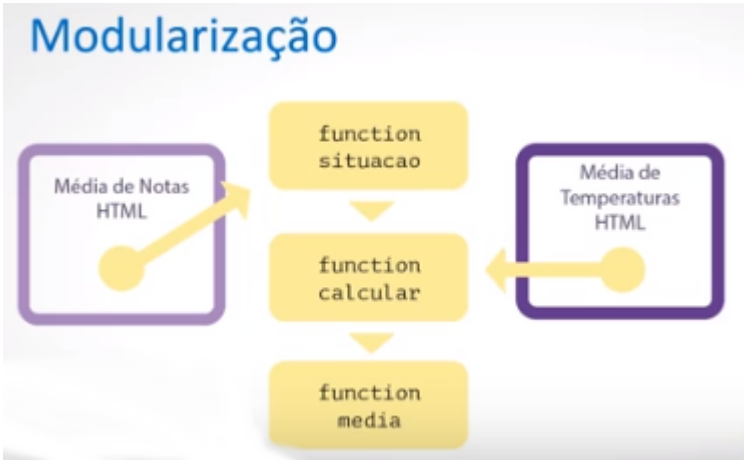
Videoaula 04 - Compondo Funções

Videoaula 04 - Compondo Funções

Nos exemplos que vimos até então, estamos trabalhando com funções bem simples. Porém, é importante ressaltar que essas funções poderiam ter, digamos, 200 linhas de código. Além disso, como dito no início da aula, as funções vão permitir que você divida o problema em partes menores que depois podem ser combinadas para resolver um problema maior.

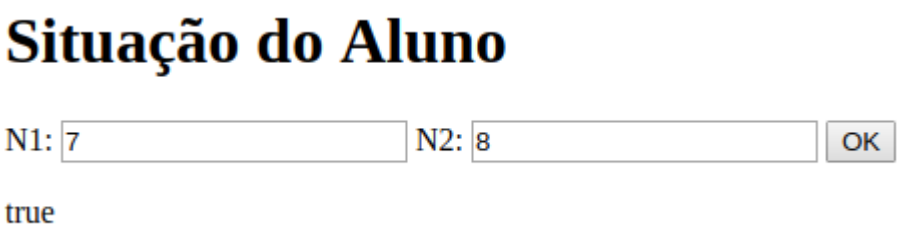
Imagine então o caso de um programa que informa se um determinado aluno, baseado em sua média, está aprovado ou não. Para esse exemplo, vamos supor que a média para aprovação seja 7. Note que esse problema pode ser quebrado em dois. O primeiro problema é calcular a média do aluno. Já o segundo, é verificar se com essa média o aluno está aprovado ou não. Vamos ver agora que, nesse caso, podemos usar uma função diferente para cada um dos problemas (Figura 1).

Figura 1 - Compondo Funções



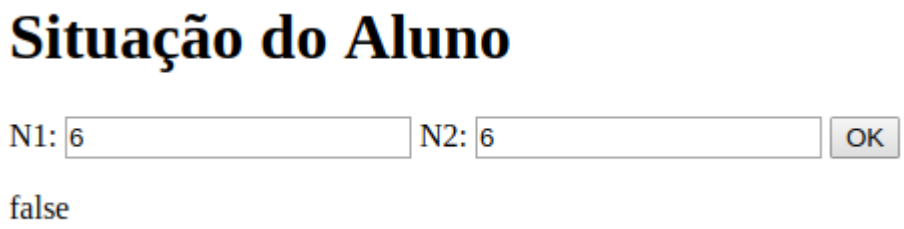
Neste exemplo, iremos alterar um pouco a página da média do aluno, que irá escrever na tela a sua situação. Como foi sugerido, a média será 7, então se o aluno tirar 7 e 8, ele será aprovado, e aparecerá `true` na tela (Figura 2).

Figura 2 - Situação do Aluno `true`



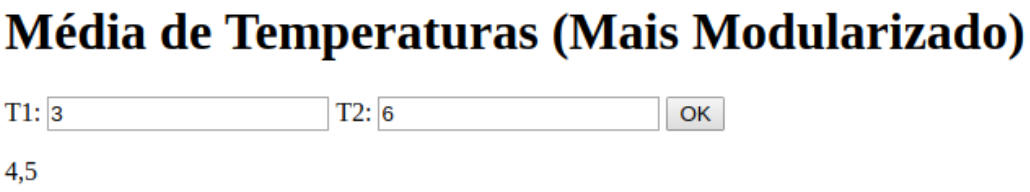
Se ele tiver uma média inferior a 7 (Figura 3), por exemplo 6 e 6, irá aparecer `false` na tela.

Figura 3 - Situação do Aluno `false`



Esse HTML vai importar um JavaScript que também será usado pela média de temperaturas (Figura 4), porém, não vamos alterar o comportamento dessa página, ele continuará sendo o mesmo; vamos apenas ver como foi a alteração no JavaScript.

Figura 4 - Média de Temperaturas



Veja que a média de notas, o HTML da média de notas, não foi alterado. Nas linhas 11 e 12, temos a declaração dos campos de entrada, na linha 13 é o botão da declaração da função, chamando agora a função `situacao`, ao invés da função para calcular a média e o campo de saída do "`resultado`" (linha 14). A única alteração nesta página HTML foi na linha 13, na função `situacao` do arquivo JavaScript.

Código 1 - 04_6 Média de Notas.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('N1', 'N2', 'resultado')">OK</button>
      <p id="resultado"></p>

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

   </body>
</html>

```

```javascript
function situacao(entrada1, entrada2, saida) {
   var m = ler_entrada_calcular_media(entrada1, entrada2);
   document.getElementById(saida).innerHTML = resultado(m);
}

function calcular(entrada1, entrada2, saida) {
   var m = ler_entrada_calcular_media(entrada1, entrada2);
   document.getElementById(saida).innerHTML = m;
}

function ler_entrada_calcular_media(entrada1, entrada2) {
   var x = Number(document.getElementById(entrada1).value);
   var y = Number(document.getElementById(entrada2).value);

   return media(x,y);
}

function media(a, b) {
   return (a + b)/2;
}

function resultado(m) {
   return m >= 7;
}

```

O arquivo de média de temperaturas (Código 2), não foi alterado, continua exatamente o mesmo que vimos no último exemplo.

Código 2 - 04_6 Média de Temperaturas.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>Média de Temperaturas (Mais Modularizado)</h1>

      T1: <input type="number" id="temp1" value="">
      T2: <input type="number" id="temp2" value="">
      <button onclick="calcular('temp1', 'temp2', 'temperatura')">OK</button>
      <p id="temperatura"></p>

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

   </body>
</html>

```

```
function situacao(entrada1, entrada2, saida) {
   var m = ler_entrada_calcular_media(entrada1, entrada2);
   document.getElementById(saida).innerHTML = resultado(m);
}

function calcular(entrada1, entrada2, saida) {
   var m = ler_entrada_calcular_media(entrada1, entrada2);
   document.getElementById(saida).innerHTML = m;
}

function ler_entrada_calcular_media(entrada1, entrada2) {
   var x = Number(document.getElementById(entrada1).value);
   var y = Number(document.getElementById(entrada2).value);

   return media(x,y);
}

function media(a, b) {
   return (a + b)/2;
}

```

```
22 function resultado(m) {
23     return m >= 7;
24 }
25
```

Nas linhas 18 a 20 (Código 3), permanece no arquivo JavaScript a função `media`, havendo apenas uma alteração na função `calcular`. Na linha 6, a função `calcular` recebe os campos de entrada `entrada1`, `entrada2` e de `saida`. E, na linha 11, criei uma função, que isola a leitura dos campos de entrada, chamada `ler_entrada_calcular_media`.

Código 3 - 04_6 Média.js

```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 04</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Situação do Aluno</h1>
10
11     N1: <input type="number" id="N1" value="">
12     N2: <input type="number" id="N2" value="">
13     <button onclick="situacao('N1', 'N2', 'resultado')">OK</button>
14     <p id="resultado"></p>
15
16     <script src="script.js"></script>
17
18   </body>
19 </html>
20
```

```
1  function situacao(entrada1, entrada2, saida) {
2      var m = ler_entrada_calcular_media(entrada1, entrada2);
3      document.getElementById(saida).innerHTML = resultado(m);
4  }
5
6  function calcular(entrada1, entrada2, saida) {
7      var m = ler_entrada_calcular_media(entrada1, entrada2);
8      document.getElementById(saida).innerHTML = m;
9  }
10
11 function ler_entrada_calcular_media(entrada1, entrada2) {
```

```
12    var x = Number(document.getElementById(entrada1).value);
13    var y = Number(document.getElementById(entrada2).value);
14
15    return media(x,y);
16 }
17
18 function media(a, b) {
19    return (a + b)/2;
20 }
21
22 function resultado(m) {
23    return m >= 7;
24 }
25
```

Essa função recebe apenas os nomes dos campos de entrada, faz a leitura na linha 12 do primeiro campo, atribuindo a `x`. Em seguida, faz a leitura na linha 13 do segundo campo, `entrada2`, atribuindo a `y`, e retorna a média desses dois campos na linha 15. Obviamente, ele está invocando a função média que vai pegar esses dois campos `(a + b)/2`, na linha 19, e vai retornar a média desses dois valores. Assim, a função `ler_entrada_calcular_media` lê a entrada e retorna a média dos valores desses dois campos de entrada (linha 11).

Se voltarmos para a função `calcular` (linha 6), a variável `m` (linha 7), vai receber exatamente a média desses dois campos de entrada e vai alterar o campo de saída para o valor dessa média (linha 8).

Essa função `calcular` (linhas 6 a 8 - Código 3), no código 2, não foi alterada. A página de alterar a média de temperaturas (linha 13) continua calculando a média dos dois campos de entrada, mas não sofre influência dessa alteração no JavaScript, isso ficou isolado. O código JavaScript (Código 3) foi um pouco mais modularizado, mas o HTML da média de temperaturas (Código 2) não foi alterado.

Porém, nesse JavaScript (Código 3), foi criada uma nova função que retorna à situação do aluno (linha 1). Ele usa a mesma função `calcular` que foi usada na linha 7, na linha 2, para `ler_entrada_calcular_media` dos dois campos de entrada (`entrada1, entrada2`).

Porém, o que ele vai escrever no campo de saída, na linha 3, não será o valor dessa média, mas o retorno da chamada da função `resultado` passando essa média. E precisamos ver o que essa função `resultado` faz.

No final do arquivo, nas linhas 22 a 24, temos a declaração da função `resultado`, que recebe o valor `m` e retorna a comparação de `m` com 7, que é a nossa média (linha 23).

Então, se em `m`, o valor passado for maior ou igual a 7, ela retorna `true`, caso contrário, ela retorna `false`. E é esse retorno da função `resultado`, na linha 3, que se usa para alterar o `innerHTML` e escrever esse valor no campo de saída, como mostra a Figura 5. Nesse caso, o aluno é reprovado, pois a média foi 6, média que está abaixo de 7, mas se for maior ou igual a 7, ele é aprovado (Figura 6).

Figura 5 - Média de Notas (04_6 Médias de Notas)

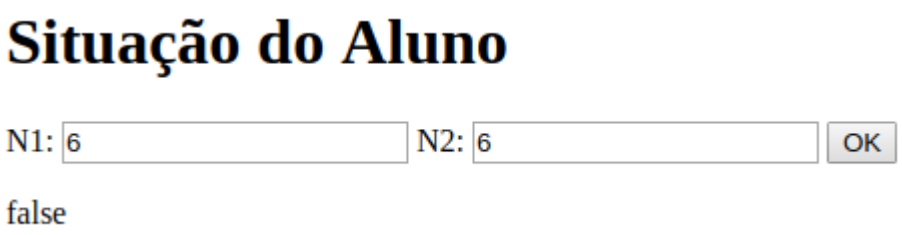
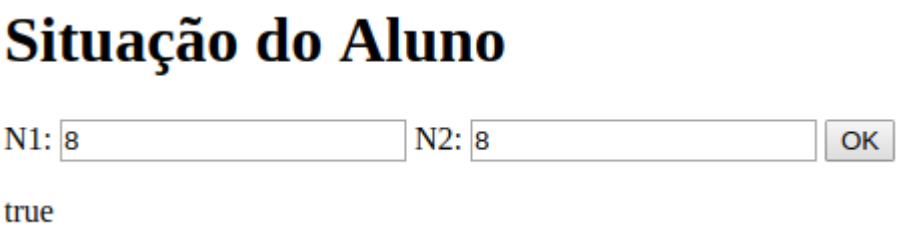


Figura 6 - Média de Notas (04_6 Médias de Notas)



O parâmetro de uma função pode ser o resultado da chamada de outra função, ou até da mesma função. Além disso, chamadas a funções podem ser usadas da mesma maneira que você utiliza variáveis, em todos os tipos de fórmulas, atribuições e cálculos. Vamos ver?

Você viu no exemplo anterior (Código 3), na linha 1, que, ao chamar a função `situacao`, era chamada a função `ler_entrada_calcular_media`, para atribuir a média dos dois campos de entrada para a variável `m` (linha 2), e usava essa variável para passar à função `resultado` e alterar o `innerHTML` com a resposta da função resultado passando o valor `m` (linha 3).

A mudança agora (Código 4) é que não utilizarei uma variável local dentro da função `situacao`, vou simplesmente chamar a função `resultado`, passando o parâmetro, que é o resultado da função `ler_entrada_calcular_media`, passando novamente aí os campos de `entrada1` e `entrada2`.

Assim, o que acontece na linha 3 é que o JavaScript vai executar primeiro essa parte `ler_entrada_calcular_media(entrada1, entrada2)`, ou seja, essa chamada de função vai ter um retorno que será a média dos dois campos de entrada, e esse retorno vai ser usado para chamar a função `resultado`.

Código 4 - 04_7 Média.js

```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 04</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Situação do Aluno</h1>
10
11     N1: <input type="number" id="N1" value="">
12     N2: <input type="number" id="N2" value="">
13     <button onclick="situacao('N1', 'N2', 'resultado')">OK</button>
14     <p id="resultado"></p>
15
16     <script src="script.js"></script>
17
18   </body>
19 </html>
20
```

```
1  function situacao(entrada1, entrada2, saida) {
2    document.getElementById(saida).innerHTML =
3      "O resultado é " + resultado(ler_entrada_calcular_media(entrada1, entrada2));
```

```
}

function calcular(entrada1, entrada2, saida) {
    var m = ler_entrada_calcular_media(entrada1, entrada2);
    document.getElementById(saida).innerHTML = m;
}

function ler_entrada_calcular_media(entrada1, entrada2) {
    var x = Number(document.getElementById(entrada1).value);
    var y = Number(document.getElementById(entrada2).value);

    return media(x,y);
}

function media(a, b) {
    return (a + b)/2;
}

function resultado(m) {
    return m >= 7;
}

```

Então, em termos de comportamento é exatamente o mesmo, estamos chamando a função `resultado`, passando a média dos campos que forem lidos, os campos de `entrada1` e `entrada2`. Mais que isso, nesse exemplo, estamos usando o retorno dessa chamada, a função `resultado`, para concatenar esse valor que for retornado pela chamada da função resultado ao texto “O resultado é”.

Dessa forma, ao inserirmos os valores cuja média seja maior ou igual a 7, ele vai escrever "`O resultado é true`" (Figura 7), e, caso contrário, se tivermos uma média menor que 7, ele vai escrever "`O resultado é false`" (Figura 8).

Figura 7 - Média de Notas (04_7 Médias de Notas.html)

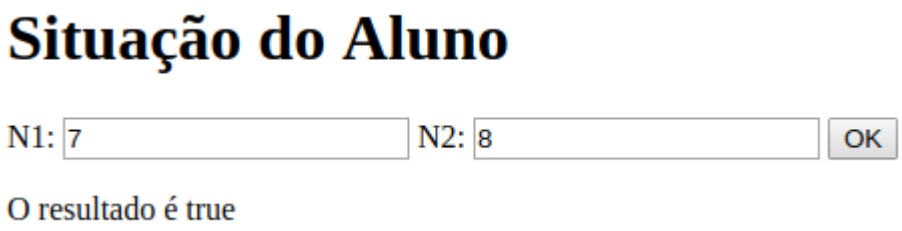
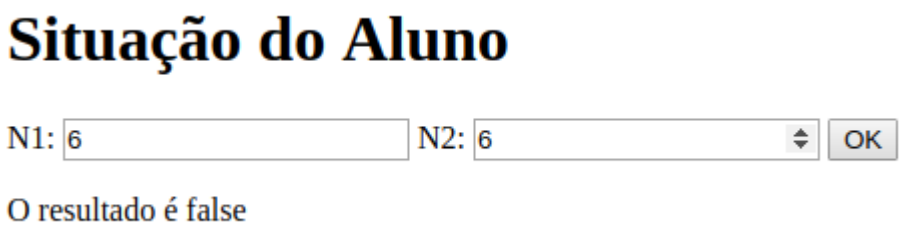


Figura 8 - Média de Notas (04_7 Médias de Notas.html)



Antes de encerrarmos, é muito importante falar sobre um erro bastante comum nos programas JavaScript que envolvem o retorno das funções e a quebra de linhas. Esse erro acontece porque concluir as instruções com ponto-e-vírgula (;) é opcional em JavaScript, o qual irá concluir a declaração de retorno no final da linha, porque isso é considerado uma declaração completa.

Vejamos, na prática, a razão disso. Nesse exemplo, também discutiremos rapidamente o uso do modo rigoroso dentro das funções.

Nesse exemplo, temos uma página HTML mais simples, com apenas um campo de entrada e um parágrafo onde será escrita a saída (Código 5).

Código 5 - 04_8 Erros de Retorno.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>Erros Comuns no Retorno</h1>

    N1: <input type="number" id="entrada" value="">
    <button onclick="testeRetorno()">OK</button>
    <p id="resultado"></p>

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

  </body>
</html>

```

```
function testeRetorno() {
    var x = Number(document.getElementById('entrada').value);

    var y;
    y = teste1(x);
    //y = teste2(x);
    //y = teste3(x);
    //y = teste4(x);
    //y = teste5(x);

    document.getElementById('resultado').innerHTML = "O resultado é "+y;
}

function teste1(valor){
    // Note que estamos usando o modo rigoroso dentro da função.
    "use strict";

    incremento = 100
    return valor + incremento
}
function teste2(valor){
    // Não estamos usando o modo rigoroso aqui.
    incremento = 100;
    return valor + incremento;
}
function teste3(valor){
    var
        incremento = 100;
    return valor + incremento;
}
function teste4(valor){
    var
        incremento = 100;
    return
        valor + incremento;
}
function teste5(valor){
    var
        incremento = 100;
    return;
    valor + incremento;
}

```

O campo de entrada continua sendo numérico, ele vai se chamar `entrada` (linha 11), e, ao clicar o botão no `onclick` dele, chamamos a função `teste_retorno` (linha 12), que está no arquivo `JavaScript 04_8 Erros de Retorno.js` (linha 15), que você verá daqui a pouco, além do parágrafo de saída chamado `resultado` (linha 13).

No arquivo JavaScript (Código 6), temos a função `teste_retorno` (linha 1). Na linha 2, essa função pega o valor do elemento que tem o identificador `entrada`, transforma isso em um número e atribui a variável `x`. Na linha 4, ela declara uma variável `y`, e a ideia é que ele use essa variável `y` na linha 11 para escrever no parágrafo "`O resultado é`" com o valor da variável `y`.

Código 6 - 04_8 Erros de Retorno.js

```
1  <html>
2    <head>
3      <meta charset="UTF-8" />
4      <title>Programação Estruturada - Aula 04</title>
5    </head>
6    <body>
7      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>
8
9      <h1>Erros Comuns no Retorno</h1>
10
11     N1: <input type="number" id="entrada" value="">
12     <button onclick="testeRetorno()">OK</button>
13     <p id="resultado"></p>
14
15     <script src="script.js"></script>
16
17   </body>
18 </html>
19
```

```
1  function testeRetorno() {
2    var x = Number(document.getElementById('entrada').value);
3
4    var y;
5    y = teste1(x);
6    //y = teste2(x);
7    //y = teste3(x);
8    //y = teste4(x);
9    //y = teste5(x);
10
11   document.getElementById('resultado').innerHTML = "O resultado é "+y;
```

```
12 }
13
14 function teste1(valor){
15    // Note que estamos usando o modo rigoroso dentro da função.
16    "use strict";
17
18    incremento = 100
19    return valor + incremento
20 }
21 function teste2(valor){
22    // Não estamos usando o modo rigoroso aqui.
23    incremento = 100;
24    return valor + incremento;
25 }
26 function teste3(valor){
27    var
28       incremento = 100;
29    return valor + incremento;
30 }
31 function teste4(valor){
32    var
33       incremento = 100;
34    return
35       valor + incremento;
36 }
37 function teste5(valor){
38    var
39       incremento = 100;
40    return;
41    valor + incremento;
42 }
43
```

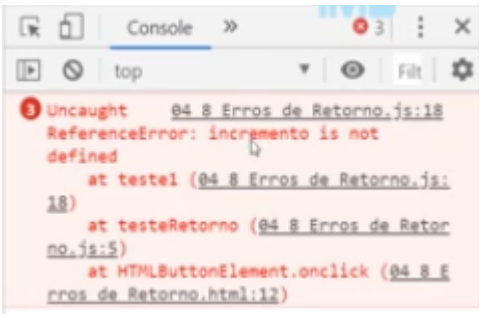
Ao longo desse exemplo, vamos usar cinco funções, `teste1` até `teste5` (linhas 5 a 9), para ilustrar o comportamento dessa questão de quebrar a linha em retorno e o uso do modo rigoroso de JavaScript dentro de funções.

No primeiro teste, nas linhas 14 a 20, vemos a função `teste1`, note que ela está usando o modo rigoroso, ou seja, o modo rigoroso de JavaScript está funcionando dentro dessa dessa função `teste1` (linha 16).

Na linha 18, temos uma variável incremento recebendo o valor 100, note que não colocamos o ponto-e-vírgula (;), estamos retornando a soma do valor que você passou com o valor do incremento (linha 19).

Então, vamos agora para a página HTML (Figura 9) e escrever o valor 10. O comportamento esperado era que, ao clicar em OK, o valor 110 fosse escrito, mas note que isso não aconteceu.

Figura 9 - Média de Notas (04_8 Erros de Retorno.html)



Clicando em F12 para depuração, vemos que a variável *incremento is not defined*, ou seja, a variável *incremento* não está definida. Isso porque, ao voltarmos para a função `teste1` (Código 6), observamos que, de fato, estamos usando o modo rigoroso (linha 16), e este exige que declaremos as variáveis antes de fazermos qualquer atribuição a elas. Nesse caso, não temos essa declaração (linha 18), então identificamos que a variável `incremento` não foi declarada e que, de fato, temos um erro.

Vamos, agora, voltar para a função `testeRetorno`, apagar a função `teste1` da linha 5 e usar a função `teste2`. Na função `teste2`, que está declarada aqui da linha 21 a 25, não está sendo utilizado o modo rigoroso, então ela é bastante similar à função `teste1`. Assim, vamos recarregar a página (Figura 10), e colocar um outro valor, o valor 5, e ao clicarmos no OK, de fato, temos a exibição do texto "`O resultado é 105`", que é 5 somando com o incremento que tem o valor 100.

Figura 10 - Média de Notas (04_8 Erros de Retorno.html)

Voltando para o exemplo (Código 6), vou apagar a função `teste2` e atribuir a `y` o valor da função `teste3`. Bom, essa função tem a declaração da variável `incremento` recebendo 100, porém, note que essa declaração apresenta uma quebra de linha entre as linhas 27 e 28, que está dentro, digamos assim, da declaração da variável `incremento`.

Qual seria o comportamento apresentado? Vamos voltar para o arquivo HTML (Figura 11), note que o resultado continua sendo exibido, "`O resultado é 104`", não houve problemas nessa quebra de linha porque o comando não foi declarado completo.

Figura 11 - Média de Notas (04_8 Erros de Retorno.html)

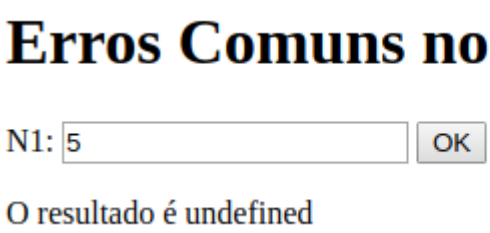
Agora, vamos deletar a função `teste3` e ver como é a função `teste4` que está definida entre as linhas 31 e 36.

Estamos com a quebra de linha tanto nas linhas 32 e 33, que é a declaração da variável `incremento`, como também no retorno. Então, note que eu, na intenção de arrumar o meu código, quebrei a declaração do retorno, assim a linha 34 tem a palavra-

chave `return` e a linha 35 tem a soma do valor com o `incremento` que eu quero retornar, e só depois disso coloquei o ponto-e-vírgula (;). Vamos ver como é o comportamento disso na página?

Se recarregar e inserir um valor 5, o comportamento esperado é que o resultado seja 105, porém, o que apareceu é "`O resultado é undefined`" (Figura 12). Isso porque o que o JavaScript fez foi colocar um ponto-e-vírgula (;) na linha 34 e considerou esse comando como completo, e no retorno apresentou um valor indefinido.

Figura 12 - Média de Notas (04_8 Erros de Retorno.html)



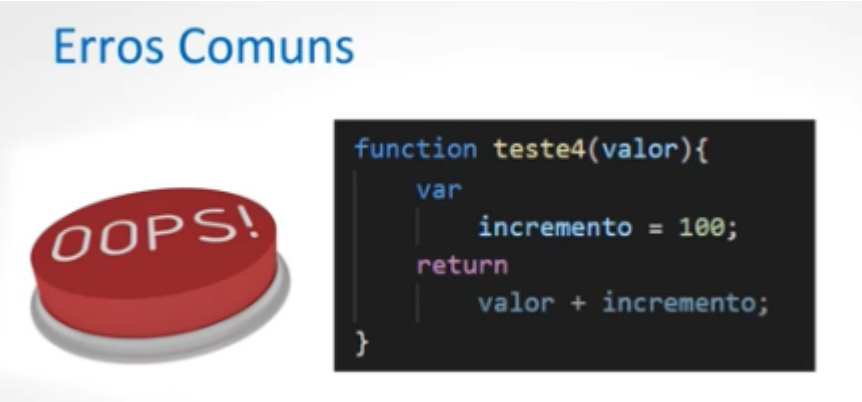
Como é que eu sei disso? Podemos fazer um outro teste. Vamos voltar para a função `testeRetorno` e apagar a função `teste4`, deixar a atribuição a `y`, a chamada da função `teste5` e, nessa função, o que eu fiz foi colocar explicitamente o ponto-vírgula (;) na linha 40, essa função que está entre as linhas 37 e 42, e o JavaScript faz internamente a ação, quando você quebra a declaração do retorno de uma função.

E veja, se eu voltar lá para a página HTML, colocar o 5, por exemplo, exatamente o que eu tenho é "`O resultado é undefined`".

Então lembre-se: ao quebrar a linha do retorno de uma função o que acontece é completar o retorno, e ele vai retornar `undefined`.

Pelo motivo que vimos neste exemplo (Figura 13), recomendo que você nunca quebre linha em uma declaração de retorno.

Figura 13 - Erros comuns



Termina aqui esta aula sobre funções. Nesta aula, você aprendeu conceitos relacionados à modularidade de programas de computador, em especial, a usar funções para atingir a modularidade. Dessa forma, você aprendeu a usar funções para dividir problemas grandes e complexos em problemas menores e mais simples de se resolver. Uma vez resolvidos, as soluções desses pequenos problemas podem ser combinadas para solucionar o problema maior e mais complexo.

Faça as atividades propostas, a fim de exercitar o que você aprendeu nesta aula. Bons estudos!