

**Modularização**

É a capacidade de dividir o programa em partes distintas que desempenham tarefas específicas e que podem ser combinadas para resolver problemas mais complexos.

A modularização nos permite escrever programas mais legíveis, mais fáceis de manter e reusar e, muitas vezes, com melhor desempenho.

**Modularização em JavaScript**

Em JavaScript, a modularização é feita geralmente através da criação e do uso de funções, pois são estruturas de código que permitem que o usuário organize seus programas em partes menores e mais simples.

Como as funções desempenham o papel de módulos, quando as utilizamos, trazemos todas as vantagens dos módulos para o programador. Se elas não existissem, os programas seriam mais complexos, com bastante duplicação de código etc. De fato, isso ocorria nas primeiras linguagens de programação. Era uma tarefa difícil detectar e corrigir os erros lógicos e de sintaxe da linguagem.

Sempre que possível, se faz necessária a divisão do código, a fim de que não sejam construídos códigos extensos e de difícil entendimento e manutenção.

Essa é uma das principais regras das boas práticas de programação.

Também é necessário compreendermos que a maioria das linguagens de programação disponibiliza um conjunto de módulos prontos aos programadores, úteis por reunirem de uma maneira organizada e concisa funções testadas de uso mais comum (como funções matemáticas, por exemplo). Esses módulos são as chamadas bibliotecas.

**Funções**

No Javascript existem diversas maneiras de se declarar uma função. A maneira mais primitiva é usando declarações a partir da palavra-chave function.

Da mesma maneira que uma declaração de variável associa um valor a um nome de variável, uma declaração de função associa uma função a um nome ou identificador.

Vamos analisar a anatomia de uma declaração de função:

function olaMundo() {

console.log('Olá Mundo!');

}

Essa declaração de função consiste em:

* A palavra-chave function;
* O nome da função, ou seu identificador, seguido de parênteses;
* O corpo da função, ou o bloco de instruções necessárias para desempenhar uma tarefa, delimitado pelas chaves.

Para que o código dentro de uma função seja executado, é necessário realizar o processo de **chamar a função**. Para chamar uma função no seu código, é necessário incluir o nome da função seguida de parênteses.

Por exemplo:

function olaMundo() {

console.log('Olá Mundo!');

}

olaMundo(); // chamando a função aqui!

Uma característica interessante do Javascript é a possibilidade de realizar chamadas de função antes da efetiva declaração dessa mesma função.

Por exemplo:

olaMundo();

function olaMundo() {

console.log('Olá Mundo!');

}

Quando estamos declarando uma função, podemos especificar **parâmetros**.

Parâmetros permitem que funções aceitem entradas e manipulem essas entradas para desempenhar alguma tarefa. Usamos parâmetros para definir qual a informação deve ser passada para a função quando ela é chamada.

Veja como especificar parâmetros em uma declaração de função:

function calcularArea(comprimento, altura) {

var area = comprimento \* altura;

console.log(area);

}

Quando chamamos uma função com entradas, especificamos nos parênteses que seguem o nome da função os valores que desejamos repassar como entrada para essa função. Os valores que são passados para uma função quando ela é chamada são chamados **argumentos**. Argumentos podem ser passados como valores ou como variáveis.

Por exemplo:

function calcularArea(comprimento, altura) {

var area = comprimento \* altura;

console.log(area);

}

calcularArea(10, 6); // chamando a função com valores como argumentos

var comprimento\_retangulo = 5;

var altura\_retangulo = 6;

calcularArea(comprimento\_retangulo, altura\_retangulo); // chamando a função com variáveis como argumentos

Ao utilizar **parâmetros padrão**, nós lidamos com situações em que um argumento não é passado para uma função que está esperando por um argumento.

function ola(nome = 'Estranho') {

console.log('Olá, ' + nome + '!');

}

ola('Maria'); // retorna "Olá Maria!"

ola(); // retorna "Olá Estranho!"

Veja outro exemplo com a declaração de uma função com múltiplos parâmetros padrão:

function listaDeCompras(item1 = 'leite', item2 = 'pão', item3 = 'ovos') {

console.log('Lembrar de comprar ' + item1);

console.log('Lembrar de comprar ' + item2);

console.log('Lembrar de comprar ' + item3);

Quando uma função é chamada, o computador irá executar o corpo dessa função e avaliar o resultado daquela chamada de função. Se não for explicitamente realizada a partir de uma instrução return, por padrão, o resultado é undefinied.

function calcularArea(comprimento, altura) {

var area = comprimento \* altura;

return area; // retorna o valor atribuído a variável 'area' e encerra a execução da função.

}

console.log(calcularArea(10, 6));

Nós podemos usar o valor de retorno de uma função dentro de outra função.

Essas funções que são chamadas dentro de outras funções são comumentes chamadas de **funções auxiliares**.

Veja o exemplo de uma função para converter graus Celsius para Fahrenheit:

// Fórmula para conversão:

// fahrenheit = 9/5\*(celsius) + 32

function multiplicaPorNoveQuintos(numero) {

return numero \* (9 / 5);

}

function converteParaFahrenheit(celsius) {

return multiplicaPorNoveQuintos(celsius) + 32; // invocação da nossa função auxiliar!

}

console.log(converteParaFahrenheit(15)); // Retorna 59

Nós podemos usar funções para segmentar pequenas tarefas quando precisamos. Escrever funções auxiliares pode ajudar a quebrar tarefas grandes e complexas em tarefas menores e mais simples!

Outra forma de definir uma função é com a chamada **expressão de função**. Ela pode ser útil quando precisamos definir uma função dentro de alguma expressão.

Por exemplo, para armazenarmos uma expressão de função dentro de uma variável:

var eFimDeSemana = function(dia) {

if (dia === 'Sábado' || dia === "Domingo") {

return true;

} else {

return false;

}

}

console.log(eFimDeSemana('Segunda-Feira'))

Para invocar uma expressão de função, basta escrever o nome da variável na qual a função foi armazenada, seguido dos argumentos envoltos de parênteses.

O Javascript também permite escrever funções usando a sintaxe conhecida como **função flecha**, ou Arrow function. Esse tipo de sintaxe permite omitir a palavra-chave function, deixando a definição de uma função bem mais reduzida.

Veja um exemplo:

var calcularArea = (comprimento, altura) => {

return comprimento \* altura;

}

console.log(calcularArea(10, 6));

As Arrow Functions foram introduzidas no JavaScript com o lançamento do ECMAScript 2015, também conhecido como ES6.

Sua sintaxe concisa e a maneira como é utilizada contribuíram para o sucesso considerável entre os desenvolvedores.

Quando falamos de Arrow Functions, elas são sempre expressões, e portanto, são sempre funções anônimas.

Para demonstrar, vamos agora reescrever  a função como uma Arrow Function:

// declaração da função

function sayHelloWorld() {

return 'Hello World!'

}

// declaração da função como expressão

const sayHelloWorld2 = function() {

return 'Hello World!'

}

// declaração da arrow function

const sayHelloWorld3 = () => 'Hello World!'

// chamado da função

console.log(sayHelloWorld())

console.log(sayHelloWorld2())

console.log(sayHelloWorld3())

// saída:

// Hello World!

// Hello World!

// Hello World!

A Arrow Function é caracterizada por uma seta (=>), os parâmetros ficam do lado esquerdo da seta e o corpo da função fica do lado direito da seta.

Ao executar, teremos o mesmo resultado no console.

Ou seja, escrevemos 3 funções que fazem a mesma coisa, mas com 3 notações diferentes.

Comparando a Arrow Function com as outras funções, podemos perceber vários benefícios:

* Ela pode ser escrita em apenas em 1 linha de código.
* Sem a palavra-chave function.
* Sem a palavra-chave return.
* Sem o uso de chaves { }

Por isso ela é bem concisa e prática de usar

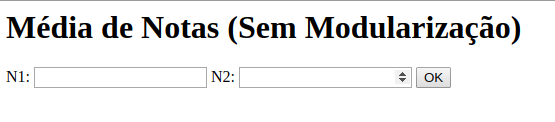
**Usabilidade**

Imaginemos um programa que calcula a média obtida por um aluno nas suas duas notas de uma disciplina. Esse é um problema pequeno e simples. Poderíamos resolvê-lo usando uma página que tem dois campos numéricos de entrada e alterar o valor de um campo texto de saída com a média dos dois campos de entrada.

Veja isso em um exemplo:

Na Figura 1, temos dois campos numéricos, N1 e N2; ao selecionar 3 e 6, respectivamente, e clicar em OK, a média gerada desses dois campos será 4.5, cujo resultado expressa o comportamento esperado.

Figura 01 - Média de Notas sem Modularização



Podemos ver no HTML (Código 1), os dois campos de entrada, o primeiro é o N1 e o segundo o N2, e a declaração do botão (onclick).

Código 1 - Média de Notas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Média de Notas (Sem Modularização)</h1>*

*N1:<input type="number" id="N1" value="">*

*N2: <input type="number" id="N2" value="">*

*<button onclick='document.getElementById("resultado").innerHTML = (Number(document.getElementById("N1").value) + Number(document.getElementById("N2").value)) / 2'>OK</button>*

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

*</body>*

*</html>*

No botão onclick, vemos um longo trecho JavaScript. Nele, o valor do innerHTML é alterado no campo "resultado", que é o campo texto e que recebe o resultado dos dois valores do N1 e do N2, ou seja, ele pega o valor do campo N1, transforma em número; usa o mesmo comportamento na outra linha, só que dessa vez, no campo N2, pega o valor, transforma em número, soma e divide por 2.

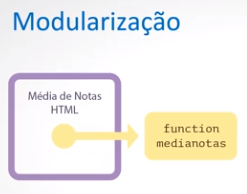
Então, esse é o primeiro exemplo de como fazer a média de dois campos de entrada.

Note como fica difícil entender a página HTML quando escrevemos o código dessa maneira. Imaginemos como seria difícil manter todo um sistema se estivéssemos usando essa forma de escrever a página. Felizmente, já vimos como melhorar o entendimento para esse caso.

Mais precisamente, pode-se escrever a alteração do campo texto em uma função JavaScript (Figura 2) que pode ser chamada, por exemplo, medianotas.

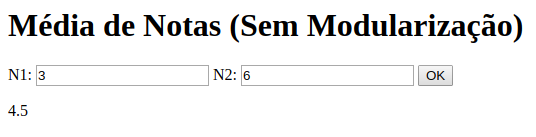
Para facilitar ainda mais a leitura do código, usa-se variáveis para armazenar separadamente o valor de cada um dos campos de entrada. Além disso, para melhorar a estrutura de nossos arquivos, escrevemos o código JavaScript em um arquivo externo.

Figura 02 - Código JavaScript em um arquivo externo



Nesse segundo exemplo, em relação à página HTML, o comportamento é praticamente o mesmo: se colocarmos uma nota (3 e 6, por exemplo) e clicar em OK, novamente a média vai aparecer, ou seja, o resultado será 4.5 (Figura 3).

Figura 03 - Média de Notas com Modularização



Porém, se olharmos o HTML (Código 2), veremos uma sessão JavaScript que vai referenciar um source, ou seja, um arquivo JavaScript externo, e o onclick invoca a função medianotas deste arquivo. Essa função estará declarada exatamente no arquivo “script0.js” (Código 3), que é o arquivo JavaScript.

Código 2 - Média de Notas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Média de Notas (Com Modularização)</h1>*

*N1:<input type="number" id="N1" value="">*

*N2: <input type="number" id="N2" value="">*

*<button onclick='medianotas()'>OK</button>*

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

*<script src="script0.js"></script>*

*</body>*

*</html>*

***script0.js***

*function medianotas() {*

*var x = Number(document.getElementById("N1").value);*

*var y = Number(document.getElementById("N2").value);*

*document.getElementById("resultado").innerHTML = (x + y) / 2;*

*}*

Nesse arquivo, foi declarada a function medianotas; pega-se o valor do elemento N1 da página, atribuindo ele à variável x. Faz-se a mesma coisa, só que para o campo N2, que é atribuído à variável y. E o valor do innerHTML do campo "resultado" é alterado para a média de x e y, ou seja, foi somado x com y e divido por 2.

O comportamento é exatamente o mesmo da outra página, só que agora foi feito algo mais modularizado (Figura 3).

**Estrutura das Declarações de Funções**

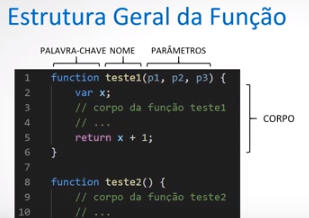
De uma maneira geral, uma função JavaScript é definida usando a palavra-chave function, seguida por um nome e parênteses. As funções podem, opcionalmente, receber valores, que chamamos de parâmetros, e podem, também opcionalmente, retornar um valor, o qual chamamos de retorno da função.

As regras para os nomes das funções são as mesmas que temos para os nomes das variáveis, ou seja:

* devem começar com uma letra, $ ou \_
* podem conter apenas letras, dígitos, sublinhados e $
* não podem ser palavras reservadas JavaScript

Na (Figura 4), temos a estrutura da definição de duas funções, teste1 e teste2.

Figura 04 - Estrutura Geral da Função

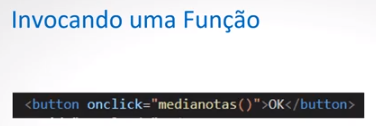


Os parâmetros da função, ou seja, os valores que a função pode receber como entrada, têm seus nomes listados dentro dos parênteses. Caso tenhamos mais de um parâmetro, como na função teste1, os nomes dos parâmetros são separados por vírgulas. Caso não haja parâmetros, como na função teste2, temos apenas os parênteses sem nada entre eles.

Por fim, temos o código a ser executado pela função, também chamado de corpo da função, colocado dentro das chaves, abrindo e fechando o bloco da função. Nesse bloco, ou seja, no corpo da função, os parâmetros funcionam como variáveis locais. Além disso, no corpo da função, podemos declarar variáveis (chamadas de variáveis locais) e comandos da linguagem. Uma variável local somente pode ter seu valor acessado dentro da função em que foi declarada. No exemplo, as variáveis locais p1, p2, p3 e x só podem ser referenciadas dentro do corpo da função teste1 e na expressão de retorno dessa função.

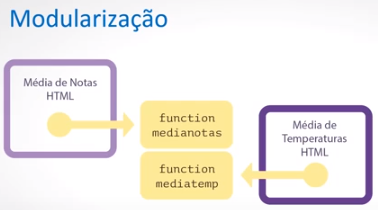
O código dentro da função será executado quando "algo" invocar, ou chamar, a função. Isso pode acontecer quando um evento ocorrer no HTML. No exemplo (Figura 5), vimos que quando um usuário clica no botão OK, a função medianotas é chamada e seu corpo executado. A chamada de uma função também pode acontecer dentro do próprio código JavaScript. De fato, uma função pode ser chamada quantas vezes forem necessárias dentro de um arquivo JavaScript.

Figura 05 - Invocando uma Função



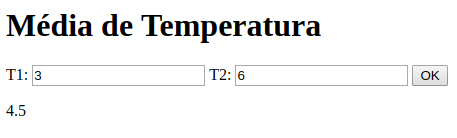
Vimos um exemplo de uma página HTML que utilizava uma função para calcular a média das notas de um aluno. Imaginemos agora que uma outra página HTML também precise calcular a média de dois campos numéricos, porém, em outro contexto onde a página está calculando a média de duas temperaturas (Figura 1). Para ilustrar melhor o problema, vamos supor que os nomes dos campos de entrada e saída são diferentes. O que poderíamos fazer seria, simplesmente, usar uma cópia da função medianotas, que apresentamos, e alterar o nome dos campos de entrada e de saída usados na função, passando a chamar essa nova função de mediatemp.

Figura 01 – Modularização



Neste exemplo (Figura 2), temos uma página bastante similar à anterior, mas agora estamos trabalhando com a média de temperaturas. Temos os campos T1 e T2, porém, veja que o comportamento, se colocarmos 3 e 6, vai resultar na mesma média, 4.5.

Figura 02 - Média de Temperaturas



E se olharmos o HTML (Código 1), veremos também que é um HTML bastante parecido com o outro, com dois campos T1 e T2, e também alterando o nome desses campos no HTML, o id deles, para "temp1" e "temp2".

Código 1 - Média de Temperaturas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Média de Temperatura</h1>*

*T1: <input type="number" id="temp1" value="">*

*T2: <input type="number" id="temp2" value="">*

*<button onclick='mediatemp()'>OK</button>*

*<p id="temperatura"></p>*

*<script src="script1.js"></script>*

*</body>*

*</html>*

***script1.js***

*function mediatemp() {*

*var x = Number(document.getElementById("temp1").value);*

*var y = Number(document.getElementById("temp2").value);*

*document.getElementById("temperatura").innerHTML = (x + y) / 2;*

}

Porém, no onclick, é chamada a mesma função mediatemp, que estará em um outro arquivo JavaScript, chamado “script1.js” que, se olharmos para esse arquivo, ele é praticamente igual ao arquivo do último exemplo, onde temos a declaração de duas variáveis, a variável x e a variável y .

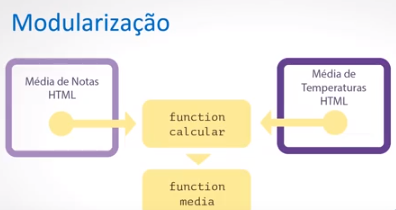
Nele, a variável x recebe o valor do campo "temp1", que é exatamente uma das diferenças com relação ao último arquivo. A variável y recebe o valor do campo "temp2", então é alterado o valor do innerHTML, do elemento de saída que tem um nome chamado "temperatura", que é outra diferença, porém a fórmula (x + y)/ 2 que nos dá a média de x e y é a mesma, ou seja, temos uma página com o comportamento bastante similar ao último exemplo.

Note que temos duas funções praticamente idênticas que alteram apenas os nomes dos campos de entrada e saída. Ambas as funções usam a mesma fórmula para calcular a média. Na prática, estamos duplicando o código, e isso é muito ruim. Isso porque será necessário testar e manter ambas as funções. Ou seja, caso você precise alterar a fórmula, vai ter que alterar e testar em vários lugares. Além do trabalho de procurar e trocar a fórmula diversas vezes, você corre o risco de esquecer de alterar uma delas e de o programa ficar inconsistente. Para evitar isso, será que não poderíamos “unificar” essas duas funções?

A resposta é sim. A idéia é termos uma função que recebe como parâmetro os nomes dos campos de entrada e de saída, e usa esses nomes para pegar os valores dos campos de entrada e escrever a média no campo de saída. Podemos até mesmo criar uma outra função que isole o cálculo dessa média. Dessa forma, criaremos a função calcular, que em seu corpo chamará a função media (Figura 3).

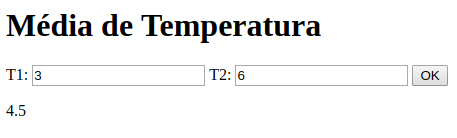
Isso é muito comum em sistemas grandes, nos quais observamos situações de funções que chamam outras funções.

Figura 03 - Média de Temperaturas



Então, do ponto de vista do comportamento da página, teremos exatamente a mesma coisa que tínhamos nos exemplos anteriores. A página de notas continua a mostrar a média de notas após o clique, e vai escrever a média dos campos que você colocar. A página das temperaturas também é a mesma coisa (Figura 4), para o valor que você escrever nos campos (T1 e T2) será gerada a média 4.5 para 3 e 6 ou a média 5 para 3 e 7.

Figura 04 - Média de Temperaturas



Porém, a estrutura interna da página é diferente, observe que o HTML de média de notas (Código 3) é muito similar ao que já tínhamos com os dois campos, N1 e N2, porém, no onclick, vou chamar uma nova função, calcular, que estará nesse arquivo JavaScript "script3.js" e vai passar os nomes dos campos N1 e N2, que são os campos de entrada, e o nome do campo de saída, que é o campo "resultado".

Código 3 - Média de Notas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Média de Notas (Mais Modularizado)</h1>*

*N1: <input type="number" id="N1" value="">*

*N2: <input type="number" id="N2" value="">*

*<button onclick="calcular('N1', 'N2', 'resultado')">OK</button>*

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

*<script src="script2.js"></script>*

*</body>*

*</html>*

***script2.js***

*function calcular(entrada1, entrada2, saida) {*

*var x = Number(document.getElementById(entrada1).value);*

*var y = Number(document.getElementById(entrada2).value);*

*var m = media(x,y);*

*document.getElementById(saida).innerHTML = m;*

*}*

*function media(a, b) {*

*return (a + b)/2;*

*}*

Ao acessarmos a página de média de temperaturas (Código 4), veremos que a página vai ser muito similar ao que já vimos nos exemplos anteriores.

Porém, o onclick do botão é idêntico ao que vimos na outra página, média de notas (Código 3). Ele também chama a função calcular, que estará no mesmo arquivo, ou seja, na prática, os dois arquivos estão importando o mesmo arquivo JavaScript e o onclick dos dois botões, nos dois arquivos, estão chamando a mesma função, apenas passando parâmetros diferentes. Assim, no Código 4, a função calcular dos campos de entrada passa de N1 e N2 , para 'temp1' e 'temp2' e o campo de saída "temperatura". Mas, na prática, os dois estão chamando a mesma função.

Código 4 - Média de Temperaturas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Média de Temperaturas (Mais Modularizado)</h1>*

*N1:<input type="number" id="temp1" value="">*

*N2: <input type="number" id="temp2" value="">*

*<button onclick="calcular('temp1', 'temp2', 'temperatura')">OK</button>*

*<p id="temperatura"></p>*

*<script src="script2.js"></script>*

*</body>*

*</html>*

Veja agora essa função no arquivo que os dois HTMLs estão importando, o “script2.js” (Código 5). Ele tem a função calcular que recebe o nome do campo de entrada1, o nome do campo de entrada2 e o nome do campo de saida .

Código 5 - Média.js

*function calcular(entrada1, entrada2, saida) {*

*var x = Number(document.getElementById(entrada1).value);*

*var y = Number(document.getElementById(entrada2).value);*

*var m = media(x,y);*

*document.getElementById(saida).innerHTML = m;*

*}*

*function media(a, b) {*

*return (a + b)/2;*

*}*

Assim, ela pega o valor do elemento que tem o identificador passado como campo de entrada1, atribui a x, pega o valor do elemento que tem identificador passado como segundo argumento chamado entrada2, e atribui a y. E vai atribuir à variável m o resultado da chamada de uma função, a função media, passando esses valores que ele pegou como entrada x e y.

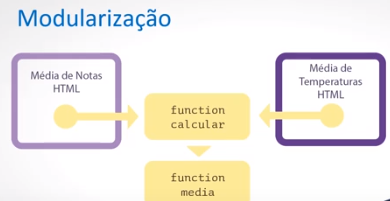
A função media, é uma outra função, que tem dois parâmetros, a e b e retorna, a média de a com b. A partir disso, ele soma a e b e divide por 2, dando o resultado como retorno.

Nesse momento, a chamada de função passando x e y usa esses argumentos e recebe a média de ambos, atribuindo isso à variável m. Por fim usaremos o valor dessa variável para alterar o innerHTML do elemento que tem o identificador igual ao que foi utilizado na chamada da função calcular, passado como argumento para o parâmetro saida e ele vai escrever o valor m.

Por exemplo, se você acessar o arquivo “Media de Notas.html” (Código 3), vai ver que foi passado o identificador ‘resultado’, como terceiro argumento. E o que essa função vai fazer é escrever nesse campo chamado de "resultado" o valor de m . Então, é exatamente isso, só que feito de maneira mais modularizada.

Neste exemplo (Figura 5), dizemos que o código de alterar os campos do HTML e o código de calcular a média estão encapsulados dentro das funções calcular e media, respectivamente, já que o HTML agora chama a função calcular para alterar o resultado na página, que, por sua vez, chama a função media para calcular a média dos valores obtidos.

Figura 05 – Modularização



Note que o código HTML não sabe como a função calcular está implementada, assim como a própria função calcular também não sabe como a função media está implementada. Isso permite que você altere os corpos dessas funções sem precisar alterar o corpo do HTML ou da função calcular.

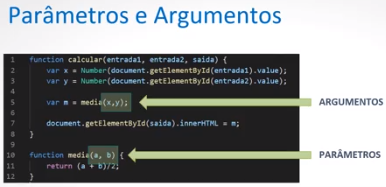
No exemplo, ambas as funções tinham parâmetros. Nesse caso, sempre que chamarmos essas funções, precisaremos indicar os valores desses parâmetros.

***Chamamos esses valores de argumentos.***

Os termos parâmetro e argumento costumam ser confundidos entre si. No entanto, é importante notar que parâmetro **é o nome da variável usada na declaração da função**, e argumento **é o valor utilizado na chamada da função**. Esse valor pode ser um valor constante, uma expressão, por exemplo, aritmética, ou o nome de uma variável, conforme foi feito no exemplo que veremos agora na (Figura 6).

Neste exemplo, usamos as variáveis x e y como argumentos para chamar a função media que tem parâmetros a e b.

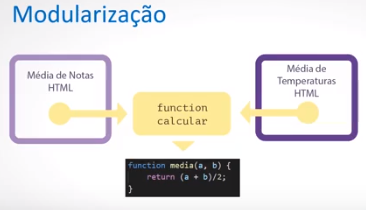
Figura 06 - Parâmetros e Argumentos



As funções podem retornar um valor. Isto, porém, não é obrigatório. Em outras linguagens de programação, esse tipo de função é chamado de procedimento. Ou seja, outras linguagens de programação diferenciam funções, que possuem retorno, de procedimentos, que não possuem retorno.

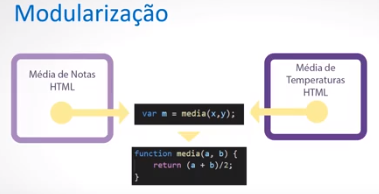
Nesse último exemplo (Figura 7), vimos que a função media tem um retorno, o qual é indicado com a palavra-chave return. Quando o JavaScript atingir uma declaração de retorno, a função irá parar de executar. Se a função foi chamada a partir de uma instrução, o JavaScript "retornará" para executar o código após a instrução de chamada. O valor de retorno é "devolvido" de volta ao "chamador".

Figura 07 - Função “media”



No exemplo (Figura 8), a variável m da função calcular recebe o valor retornado pela função media. Como vimos, esse valor é então usado para atualizar a página HTML.

Figura 08 - Função “calcular”

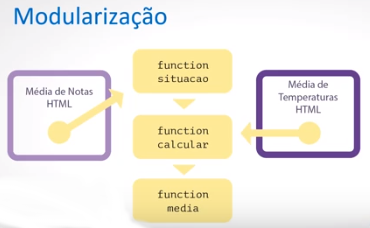


Existe ainda a possibilidade de fazer com que uma função retorne mais do que um valor, utilizando Arrays. Além disso, poderíamos também utilizar Arrays para melhorar ainda mais a modularização do nosso exemplo, fazendo com que a função média recebesse uma quantidade qualquer de campos de entrada e escrevesse no campo de saída a média dos valores de todos os campos.

Nos exemplos vistos até então, estamos trabalhando com funções bem simples. Porém, é importante ressaltar que essas funções poderiam ter, digamos, 200 linhas de código. Além disso, as funções vão permitir que você divida o problema em partes menores que depois podem ser combinadas para resolver um problema maior.

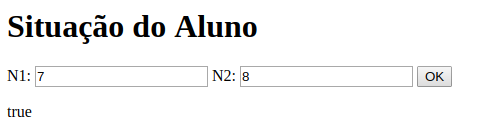
Imagine então o caso de um programa que informa se um determinado aluno, baseado em sua média, está aprovado ou não. Para esse exemplo, vamos supor que a média para aprovação seja 7. Note que esse problema pode ser quebrado em dois. O primeiro problema é calcular a média do aluno. Já o segundo, é verificar 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 01 - Compondo Funções



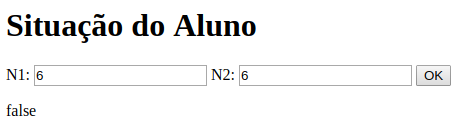
Neste exemplo, iremos alterar um pouco a página da média do aluno, que irá escrever na tela a sua situação. Como foi sugerido, a média será 7, então se o aluno tirar 7 e 8, ele será aprovado, e aparecerá true na tela (Figura 2).

Figura 02 - Situação do Aluno true



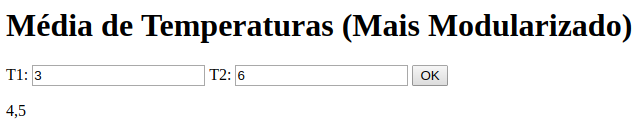
Se ele tiver uma média inferior a 7 (Figura 3), por exemplo 6 e 6, irá aparecer false na tela.

Figura 03 - 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 04 - Média de Temperaturas



Veja que a média de notas, o HTML da média de notas, não foi alterado. Temos a declaração dos campos de entrada, 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”. A única alteração nesta página HTML foi, na função situacao do arquivo JavaScript.

Código 1 - Média de Notas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<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="script3.js"></script>*

*</body>*

*</html>*

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

Código 2 - Média de Temperaturas.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções </title>*

*</head>*

*<body>*

*<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="script3.js"></script>*

*</body>*

*</html>*

Permanece no arquivo JavaScript a função media, havendo apenas uma alteração na função calcular. A função calcular recebe os campos de entrada entrada1, entrada2 e de saida. E, uma função, que isola a leitura dos campos de entrada, chamada ler\_entrada\_calcular\_media.

Código 3 - Média.js

*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;*

*}*

Essa função recebe apenas os nomes dos campos de entrada, faz a leitura atribuindo a x, em seguida, faz a leitura do segundo campo, entrada2, atribuindo a y, e retorna a média desses dois campos. Está invocando a função média que vai pegar esses dois campos (a + b)/2, 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.

Se voltarmos para a função calcular, a variável m , vai receber exatamente a média desses dois campos de entrada e vai alterar o campo de saída para o valor dessa média.

Essa função calcular (Código 3), no Código 2, não foi alterada. A página de alterar a média de temperaturas 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. Ele usa a mesma função calcular que foi usada para ler\_entrada\_calcular\_media dos dois campos de entrada (entrada1, entrada2).

Porém, o que ele vai escrever no campo de saída, não será o valor dessa média, mas o retorno da chamada da função resultado passando essa média.

No final do arquivo, temos a declaração da função resultado, que recebe o valor m e retorna a comparação de m com 7, que é a média.

Então, se em m, o valor passado for maior ou igual a 7, ela retornará true, caso contrário, ela retornará false.

E é esse retorno da função resultado, 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 05 - Média de Notas

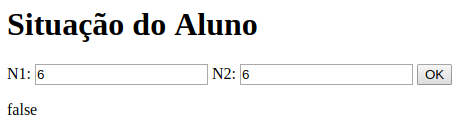
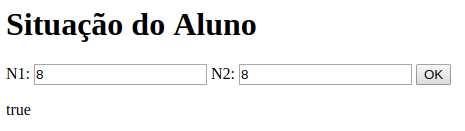


Figura 06 - Média de Notas



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 utilizamos variáveis, em todos os tipos de fórmulas, atribuições e cálculos.

Vimos no exemplo anterior (Código 3) 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 , e usava essa variável para passar à função resultado e alterar o innerHTML com a resposta da função resultado passando o valor m.

A mudança agora (Código 4) é que não será utilizada uma variável local dentro da função situacao, simplesmente será chamada a função resultado, passando o parâmetro, que é o resultado da função ler\_entrada\_calcular\_media, passando novamente os campos de entrada1 e entrada2.

Assim, o que acontece 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 - Média.js

*function situacao(entrada1, entrada2, saida) {*

*document.getElementById(saida).innerHTML =*

*"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 07 - Média de Notas

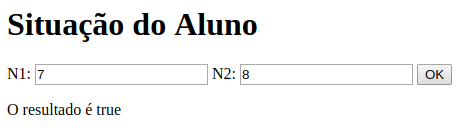
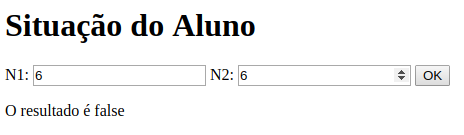


Figura 08 - Média de Notas



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.

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 - Erros de Retorno.html

*<html>*

*<head>*

*<meta charset="UTF-8" />*

*<title>Funções</title>*

*</head>*

*<body>*

*<h1>Erros Comuns no Retorno</h1>*

*N1: <input type="number" id="entrada" value="">*

*<button onclick="testeRetorno()">OK</button>*

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

*<script src="script4.js"></script>*

*</body>*

*</html>*

O campo de entrada continua sendo numérico, ele vai se chamar entrada, e, ao clicar o botão no onclick dele, chamamos a função teste\_retorno, que está no arquivo script4.js, além do parágrafo de saída chamado resultado.

No arquivo JavaScript (Código 6), temos a função teste\_retorno. Essa função pega o valor do elemento que tem o identificador entrada, transforma isso em um número e atribui a variável x. Ela declara uma variável y, e usa essa variável y  para escrever no parágrafo "O resultado é" com o valor da variável y.

Código 6 – script4.js

*function testeRetorno() {*

*var x = Number(document.getElementById('entrada').value);*

*var y;*

*y = teste1(x);*

*//y = teste2(x);*

*//y = teste3(x);*

*//y = teste4(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;*

*}*

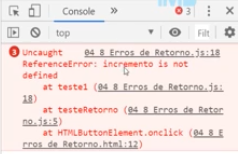
Ao longo desse exemplo, vamos usar cinco funções, teste1 até teste5  para ilustrar o comportamento dessa questão de quebrar a linha em retorno e o uso do modo rigoroso de JavaScript dentro de funções.

No primeiro teste, vemos a função teste1, note que ela está usando o modo rigoroso, ou seja, o modo rigoroso de JavaScript está funcionando dentro dessa função teste1.

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.

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 09 - Média de Notas



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, e este exige que declaremos as variáveis antes de fazermos qualquer atribuição a elas. Nesse caso, não temos essa declaração, 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  e usar a função teste2.

Na função teste2, 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



Voltando para o exemplo (Código 6), apagamos a função teste2 e atribuímos a y o valor da função teste3.

Essa função tem a declaração da variável incremento recebendo 100, porém, note que essa declaração apresenta uma quebra de linha que está dentro, digamos assim, da declaração da variável incremento.

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



Agora, vamos deletar a função teste3 e ver como é a função teste4 que está definida.

Estamos com a quebra de linha tanto nas linhas que são a declaração da variável incremento, como também no retorno. Note que foi quebrada a declaração do retorno, assim a palavra-chave return  tem a soma do valor com o incremento que eu quero retornar, e só depois disso o ponto-e-vírgula (;) foi colocado.

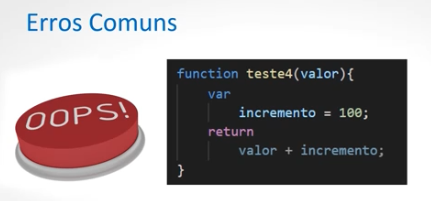
Se recarregarmos e inserirmos um valor 5, o comportamento esperado é que o resultado seja 105, porém, o que apareceu é "O resultado é undefined" (Figura 12). Isso porque o que o JavaScript fez foi colocar um ponto-e-vírgula (;) e considerou esse comando como completo, e no retorno apresentou um valor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

Figura 12 - Média de Notas



É recomendado que você nunca quebre linha em uma declaração de retorno.

Figura 13 - Erros comuns



***Saiba mais:***

<https://www.w3schools.com/js/js_functions.asp>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function>

https://www.youtube.com/watch?v=mc3TKp2XzhI&t=1490s

<https://www.youtube.com/watch?v=ItzRdMj1lzw>

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions/Default_parameters>

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions