



Mundo Tech

PROGRAMAÇÃO BACK-END FUNDAMENTOS BÁSICOS DE BACK-END

SUMÁRIO

Mas, antes de começar.....	3
Estilo CSS.....	3
Index	6
Funções em JavaScript.....	8
Estruturas de controle e repetição.....	9
Estrutura sequencial.....	9
Estrutura de decisão	10
Estrutura de repetição	13
Funções.....	15
Obtendo valores de uma entrada com JavaScript	23
Referências.....	26



MAS, ANTES DE COMEÇAR...

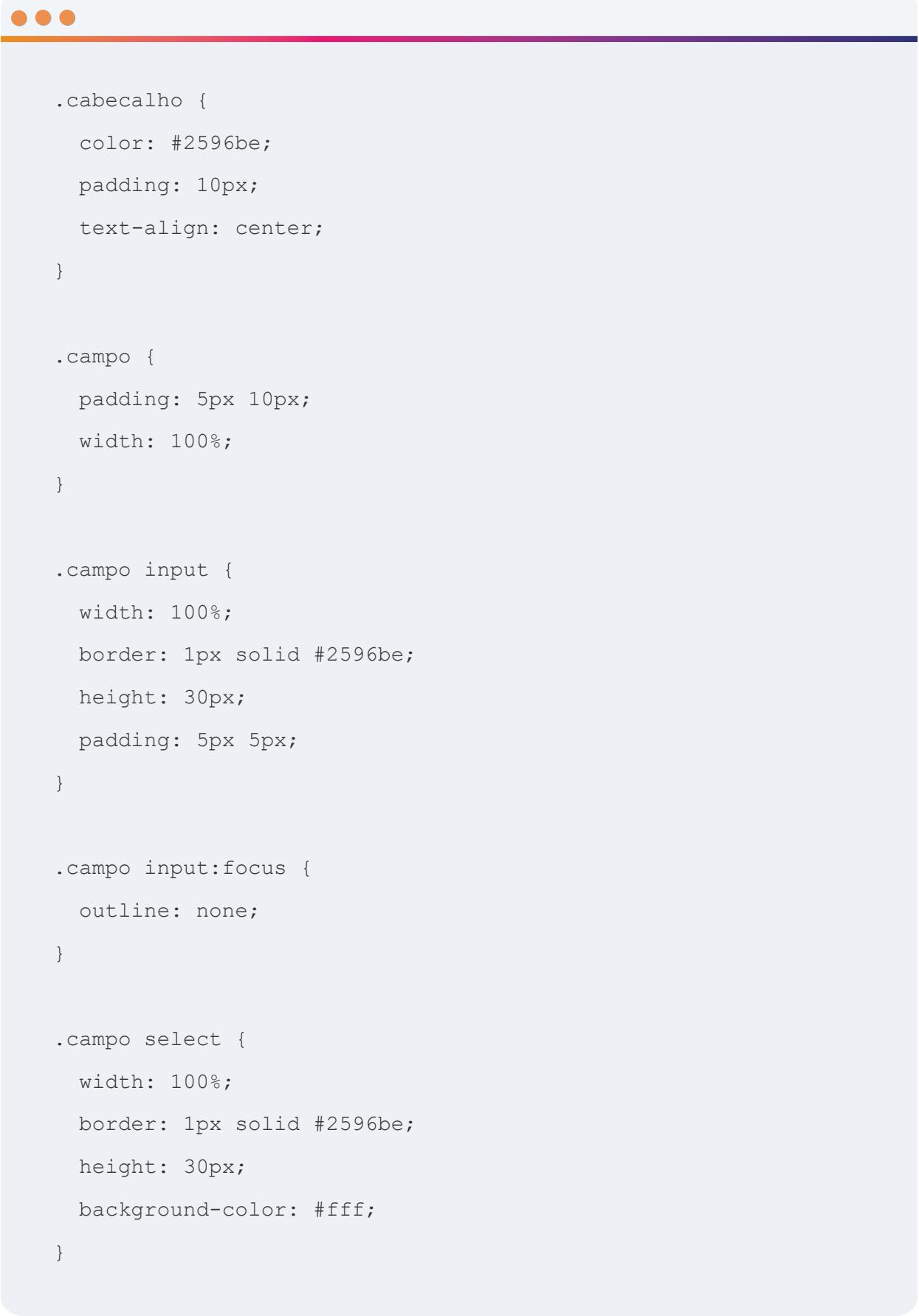
Você recebeu da equipe os arquivos HTML (index.html) e CSS (estilo.css), o código-fonte correspondente, portanto confira a seguir:

Estilo CSS

```
* {  
    margin: 0px;  
    padding: 0px;  
    box-sizing: border-box;  
}  
  
body {  
    font-family: "Roboto", sans-serif;  
}  
  
label {  
    color: #2596be;  
}  
  
button {  
    padding: 5px 30px;  
    margin-top: 10px;  
    background-color: #fff;  
    color: #2596be;  
    border: 1px solid #2596be;  
}
```



```
button:hover {  
    background-color: #2596be;  
    color: #fff;  
}  
  
.conteudo {  
    width: 100%;  
    max-width: 500px;  
    padding-top: 50px;  
    margin: auto;  
    height: 100vh;  
    max-height: 900px;  
    background-color: #fff;  
}  
  
.formulario {  
    height: auto;  
    margin: 0px 0px;  
    padding: 5px 10px 50px 10px;  
    background-color: rgb(240, 240, 240);  
    border-radius: 6px;  
}  
  
form {  
    width: auto;  
    margin: 0px 40px;  
}
```



```
.cabecalho {  
    color: #2596be;  
    padding: 10px;  
    text-align: center;  
}  
  
.campo {  
    padding: 5px 10px;  
    width: 100%;  
}  
  
.campo input {  
    width: 100%;  
    border: 1px solid #2596be;  
    height: 30px;  
    padding: 5px 5px;  
}  
  
.campo input:focus {  
    outline: none;  
}  
  
.campo select {  
    width: 100%;  
    border: 1px solid #2596be;  
    height: 30px;  
    background-color: #fff;  
}
```

```
.campo select:focus {  
    outline: none;  
}  
  
.botao {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}  
  
.resultado {  
    padding: 5px;  
    margin-top: 10px;  
    background-color: #fff;  
    border: 1px solid #2596be;  
    display: block;  
}
```

Index

```
<!DOCTYPE html>  
<html lang="pt-br">  
<head>  
    <meta charset="UTF-8">  
    <title>Calculadora</title>  
    <link href="http://fonts.googleapis.com/css?family=Roboto"  
        rel="stylesheet" type="text/css">  
    <link rel="stylesheet" href="estilo.css" />  
    <script src="script.js"></script>  
</head>
```

```
<body>
<div class="conteudo">
<div class="formulario">
<form method="" action="">
<div>
<h2 class="cabecalho">Calculadora</h2>
</div>
<div class="campo">
<label>Valor 1:</label>
<input type="text" id="txtValor1">
</div>
<div class="campo">
<label>Valor 2:</label>
<input type="text" id="txtValor2">
</div>
<div class="campo">
<label>Operação:</label>
<select name="cbxOperacao" id="cbxOperacao">
<option value="+>">Adição</option>
<option value="->">Subtração</option>
<option value="/>">Divisão</option>
<option value="*>">Multiplicação</option>
</select>
</div>
<div class="botao">
<button type="button" onclick="calcularValores()">Calcular</button>
```

```
</div>

<div class="campo">
    <label>Resultado:</label>
    <span class="resultado" id="txtResultado">-</span>
</div>
</form>
</div>
</body>
</html>
```

A partir daí você deve analisar o arquivo HTML e identificar quais são os campos que devem ser referenciados no arquivo JavaScript e dar prosseguimento no processo de implementação utilizando os recursos das funções.

Boa sorte com seu primeiro desafio!

• •

FUNÇÕES EM JAVASCRIPT



Como você tem acompanhado, as aplicações web dinâmicas surgiram após o nascimento do desenvolvimento web. Com a crescente popularidade dos aplicativos da web, o JavaScript tornou-se uma das linguagens mais importantes do mundo. Desta forma, como recursos complementares, você compreenderá a utilização de estruturas de controle e repetição, bem como de funções. A principal diferença entre JavaScript e a maioria das outras linguagens é que JavaScript nos permite criar uma função como uma entidade autônoma e passá-la como argumento para uma outra função, que pode aceitá-la como parâmetro, assim como qualquer outro objeto. Na sequência, você verá, também, como obter o valor de uma entrada de um formulário utilizando funções em JavaScript. Está preparado? Então, dê continuidade a sua leitura!



Estruturas de controle e repetição

As estruturas de controle são subdivididas em três grandes categoriais: estruturas sequenciais, estruturas de decisão e estruturas de repetição. É importante que você saiba diferenciar as estruturas sequenciais das estruturas de decisão e de repetição. Existe a possibilidade de inserir qualquer uma delas (inclusive uma dentro da outra) nas estruturas sequenciais, e, geralmente, é isso mesmo o que ocorre.



Figura 1 - Exemplo de funcionamento das estruturas de controle

Fonte: elaborada pelo Autor (2022)

Estrutura sequencial

As estruturas sequenciais representam ações primitivas, executadas em uma sequência linear, de cima para baixo, da esquerda para a direita. Uma estrutura computacional realizará as ações de maneira linear, comando a comando, podendo ou não ocorrer desvios de fluxos em seu contexto. Confira um exemplo de estrutura sequencial:

```
const numero1 = 5;
const numero2 = 3;

// Adiciona os dois números
const soma = numero1 + numero2;

// Exibe a soma
console.log('A soma do ' + numero1 + ' e ' + numero2 + ' é: ' +
soma);
```

Estrutura de decisão

A estrutura de decisão indica uma decisão, tomada após o teste de uma condição. No caso, os algoritmos que aprendemos são sequenciais, por isso, quando uma estrutura de decisão é utilizada, o código sofrerá um desvio dentro do algoritmo, fazendo com que ele haja de outra forma, de acordo com a condição colocada. A estrutura condicional básica do JavaScript considera a instrução “if”. Caso o resultado da avaliação seja verdadeiro, a ação será executada. Confira um exemplo:



```
//Condição é verdadeira: imprime na tela “Este  
número é maior do que 0.”  
let numero = 2;  
if (numero > 0) {  
    console.log("Este número é maior do que 0.");  
}  
// Código depois do if
```



```
//Condição é falsa: não faz nada e passa para a  
próxima instrução fora do bloco do if  
let numero = 2;  
if (numero > 0) {  
    console.log("Este número é maior do que 0.");  
}  
// Código depois do if
```

Figura 2 - Exemplo de estrutura de decisão

Fonte: elaborada pelo Autor (2022)

Uma declaração é usada para fazer o programa tomar uma rota, ou outra, dependendo do resultado de uma avaliação da expressão. O condicional verifica a expressão que você passa a ele por um valor verdadeiro ou falso. O bloco “if...else” é um tipo de instrução condicional que executará um bloco de código quando a condição na instrução “if” for verdadeira. Se a condição for falsa, o bloco “else” será executado. Confira um exemplo:

```
if (navigator.cookieEnabled) {  
    alert("O navegador suporta cookies.");  
} else {  
    alert("O navegador não suporta cookies.");  
}
```

A instrução “if” avalia a condição entre parênteses “()” que, no exemplo apresentado, se o navegador tiver a funcionalidade de armazenamento de cookies habilidade será exibida a mensagem: “O navegador suporta cookies.”, caso contrário, exibirá “O navegador não suporta cookies.”.

Haverá momentos em que você desejará testar várias condições. É aí que entra o bloco “else...if”. Quando a instrução “if” for falsa, o computador passará para a instrução “else...if”. Se isso também for falso, então ele se moverá para o bloco “else”. Confira um exemplo:

```
// Utilizando a função console para exibir informações no navegador.  
let semaforo = "vermelho";  
if (semaforo === "verde") {  
    console.log("Siga!");  
} else if (semaforo === "vermelho") {  
    console.log("Pare!");  
} else {  
    console.log("Atenção!");  
}
```

Há momentos em JavaScript em que você pode considerar usar uma instrução “switch” em vez de uma instrução “if...else”. Instruções “switch” podem ter uma sintaxe mais limpa em relação a instruções complicadas “if...else”. Dê uma olhada no exemplo a seguir. Em vez de usar uma longa declaração “if...else”, você pode optar por uma declaração “switch”, que é mais fácil de compreender.

```
const animal = "cachorro";  
switch (animal) {  
    case "cachorro":  
        console.log("Eu possuo um cachorro.");  
        break;  
    case "gato":  
        console.log("Eu possuo um gato.");  
        break;  
    case "cobra":  
        console.log("Eu possuo uma cobra.");  
        break;  
    case "papagaio":  
        console.log("Eu possuo um papagaio.");  
        break;  
    default:  
        console.log("Eu não possuo um animal de estimação.");  
        break;  
}
```

Na programação, uma instrução “switch” é uma instrução de fluxo de controle que testa o valor de uma expressão em vários casos. O algoritmo passará pela instrução “switch” e verificará a igualdade estrita (==) entre o caso e a expressão. Se um dos casos corresponder a expressão, o código dentro dessa cláusula do caso será executado. No exemplo acima temos, como expressão, a variável “animal”. Já como casos, temos as cláusulas: cachorro, gato, cobra e papagaio. Se nenhum dos casos corresponder à expressão, a cláusula “default” será executada.

As instruções “break” fazem com que o algoritmo interrompa a execução do “switch” quando o caso for correspondido. Se as instruções “break” não estiverem presentes, o computador continuará com a execução da instrução “switch” mesmo se uma correspondência for encontrada.

Estrutura de repetição

Os laços são uma das principais estruturas de controle do JavaScript. Com um loop, é possível automatizar e repetir indefinidamente um bloco de código, por quantas vezes você quiser que ele funcione. O JavaScript fornece muitas maneiras de iterar através de loops.

a) While

O “while” é a estrutura de looping mais simples que o JavaScript fornece. Adicionamos uma condição após a palavra-chave “while” e fornecemos um bloco que é executado até que a condição seja avaliada como verdadeira. O exemplo a seguir usa a instrução “while” para gerar os números ímpares entre 1 e 10 no console:

```
let contador = 1;  
while (contador < 10) {  
    console.log(contador);  
    contador = contador + 2;  
}
```

Primeiro, você deve declarar e inicializar a variável “contador” como 1. Segundo, executar a instrução dentro do loop, se a variável “contador” for menor que 10. Em cada iteração, enviar a contagem para o console e aumentar a contagem da variável “contador” de 2 em 2. Terceiro, após cinco iterações, o contador terá obtido o valor 11. Portanto, a condição “contador” < 10” se tornará falsa, logo o loop é encerrado.

b) For

A segunda estrutura de looping muito importante em JavaScript é a de “for”. Usamos a palavra-chave “for” e passamos um conjunto de três instruções: a inicialização, a condição e a parte de incremento. Exemplo:



```

// Exibe os números de 1 a 5
// Inicializa as variáveis
const numero = 5;
// Looping de i = 1 até 5
// Em cada interação, a variável i é incrementada de 1 em 1
for (let i = 1; i <= numero; i++) {
    console.log(i); // Exibe o valor de i
}

```

No exemplo apresentado é possível identificar que o algoritmo exibe os números de 1 a 5. Logo, a cada iteração, a variável “i” é incrementada em 1, exibindo os valores a cada loop até a condição ser satisfeita. Acompanhe como é o funcionamento do exemplo apresentado:

Interação	Variável	Condição: i <= numero	Ação
Primeira	i = 1 numero = 5	true	1 é impresso. i é incrementado em 2.
Segunda	i = 2 numero = 5	true	2 é impresso. i é incrementado em 3.
Terceira	i = 3 numero = 5	true	3 é impresso. i é incrementado em 4.
Quarta	i = 4 numero = 5	true	4 é impresso. i é incrementado em 5.
Quinta	i = 5 numero = 5	true	5 é impresso. i é incrementado em 6.
Sexta	i = 6 numero = 5	false	O loop é finalizado.

O mesmo exemplo apresentado anteriormente pode ser reescrito utilizando a estrutura de repetição “while”. Confira um exemplo:

```
// Exibe os números de 1 a 5
// Inicializa as variáveis
let i = 1, numero = 5;
// Looping de i = 1 até 5
while (i <= numero) {
    console.log(i); // Exibe o valor de i
    i = i + 1; // Incrementa o valor de i de 1 em 1
}
```

Funções

Em qualquer programa JavaScript moderadamente complexo, tudo acontece dentro das funções. As funções são uma parte essencial do JavaScript. Mas o que é uma função? Uma função é um bloco de código, autocontido. Uma função é declarada usando a palavra-chave “function” (MORRISON, 2008; FLANAGAN, 2013; HAVERBEKE, 2018; CAELUM, 2022).

As regras básicas para nomear uma função são semelhantes às para nomear uma variável. É melhor escrever um nome descritivo para sua função. Por exemplo, se uma função for usada para adicionar dois números, você poderá nomear a função como “adicionarNumeros”. O corpo da função é escrito dentro de “{}”. Por exemplo:

```
// Declaração da função
function saudacao() {
    console.log("Olá mundo!");
}
```

No programa citado acima, foi declarada uma função chamada “saudacao()”. Embora para usar essa função você precise chamá-la. Confira como você pode chamar esta função.

```
// Chamada da função saudacao  
saudacao();
```

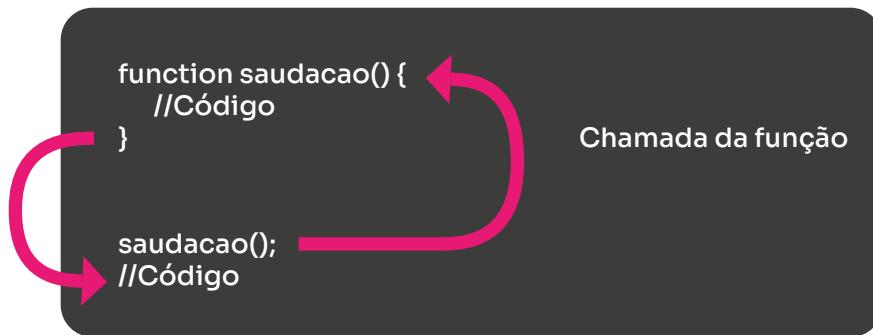


Figura 3 - Exemplo de uma chamada de uma função

Fonte: elaborada pelo Autor (2022)

Uma função também pode ser declarada com parâmetros. Um parâmetro é um valor que é passado ao declarar uma função. Confira esta definição:

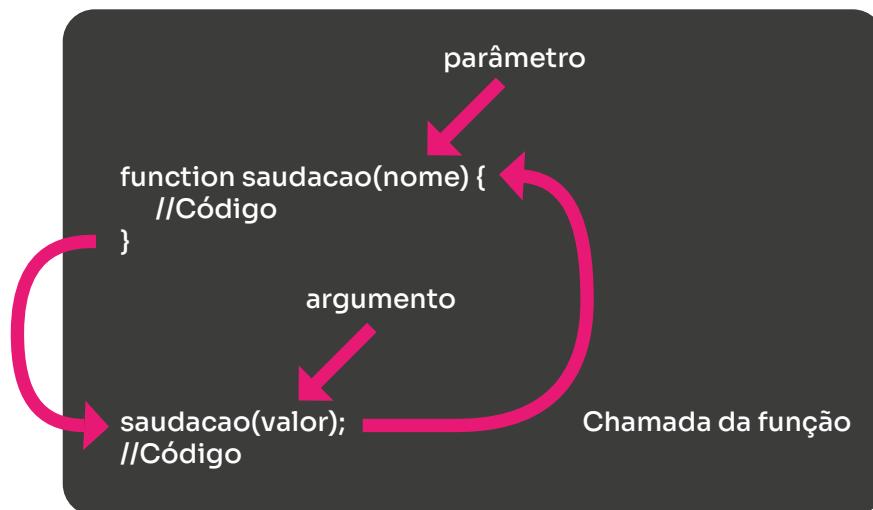


Figura 4 - Exemplo de parâmetros e argumentos de uma função

Fonte: elaborada pelo Autor (2022)

Agora você verá o funcionamento da função JavaScript com parâmetro. Confira um exemplo:

```
// Declaração da função  
function saudacao(nome) {  
    console.log("Olá " + nome + "!");  
}  
  
// Leitura do valor  
let valor = prompt("Entre com o seu nome: ");  
  
// Chamada da função  
saudacao(valor);
```

No programa acima, a função “saudacao()” é declarada com um parâmetro “nome”. O usuário é solicitado a inserir um nome, que é armazenado na variável “valor”. Então, quando a função é chamada, um argumento é passado para a função.



Quando um valor é passado ao declarar uma função, ele é chamado de parâmetro. E quando a função é chamada, o valor passado é chamado de argumento.



```
// Programa para realizar a adição de dois números usando função  
// Declaração da função  
function adicao(a, b) {  
    console.log(a + b);  
}  
  
// Chamada da função  
adicao(3, 4); // Resultado: 7  
adicao(2, 9); // Resultado: 11
```

No programa anterior, a função “adicao()” é usada para encontrar a soma de dois números. A função é declarada com dois parâmetros “a” e “b”. A função é chamada usando seu nome e passando dois argumentos, os valores “3” e “4” em uma chamada e “2” e “9” em outra chamada. Observe que você pode chamar uma função quantas vezes quiser. Você pode escrever uma função e chamá-la várias vezes com argumentos diferentes.

A instrução “return” de uma função pode ser usada para retornar o valor para uma chamada de função. A instrução “return” denota que a função terminou. Qualquer código após “return” não será executado. Se nada for retornado, a função retornará um valor “undefined”.



```
// Função para somar dois números
// Declaração da função
function adicao(a, b) {
    return a + b;
}

// Coleta duas entradas do usuário
let numero1 = parseFloat(prompt("Entre com o primeiro número: "));
let numero2 = parseFloat(prompt("Entre com o segundo número: "));

// Chamada da função
let resultado = adicao(numero1, numero2);

// Exibe o resultado
console.log("A soma é " + resultado);
```

No programa acima, a soma dos números é retornada pela função usando a instrução “return”. E esse valor é armazenado na variável “resultado”.

Acompanhe alguns benefícios de usar uma função (HAVERBEKE, 2018):

- › A função torna o código reutilizável, pois você pode declará-la uma vez e usá-la várias vezes.
- › A função torna o programa mais fácil, pois pequenas tarefas são divididas em uma função.
- › A função aumenta a legibilidade do código-fonte.

Em JavaScript, as funções também podem ser definidas como expressões. Por exemplo:

```
// Programa para encontrar o dobro de um número  
// Função é declarada dentro de uma variável  
  
let valor = function (numero) {  
    return numero * numero;  
};  
  
// Impressão do resultado  
console.log(valor(4));  
  
// Pode ser usado como valor de variável para outras variáveis  
let calculo = valor(3);  
console.log(calculo);
```

No programa acima, a variável “valor” é usada para armazenar o resultado de uma função. Neste exemplo, a função é tratada como uma expressão. Ainda no exemplo, a função é chamada usando o nome da variável que, neste caso, é denominada de “valor”. A função apresentada acima é chamada de função anônima. Perceba que uma função anônima é uma função sem nome. Confira outro exemplo de uso de uma função anônima:

```
● ● ●  
let exibir = function() {  
    console.log("Eu sou uma função anônima!");  
};  
  
// Chamada da função  
exibir();
```

Neste exemplo, a função anônima não tem nome entre a palavra-chave “function” e os parênteses “()”. Como é preciso chamar a função anônima posteriormente, você deve atribuir a função anônima à variável “exibir”.

As funções de setas ou arrow functions são uma introdução recente ao JavaScript. Elas são muito utilizadas em vez das funções “regulares”, descritas na seção anterior. Você encontrará as duas formas utilizadas em todos os lugares. Visualmente, elas permitem que você escreva funções com uma sintaxe mais curta (HAVERBEKE, 2018).

O exemplo a seguir define uma expressão de função que retorna a soma de dois números:

```
● ● ●  
let adicionar = function (valorA, valorB) {  
    return valorA + valorB;  
};  
  
// Chama a função e exibe o resultado  
console.log(adicionar(10, 20)); // Resultado: 30
```

O exemplo a seguir é equivalente à expressão de função acima chamada de “adicionar()”, mas usa-se uma função de seta:

```
● ● ●  
let adicionar = (valorA, valorB) => valorA + valorB;  
  
// Chama a função e exibe o resultado  
console.log(adicionar(10, 20)); // Resultado: 30
```

Mas note que não há um nome aqui para esta função. As funções do tipo *arrow function* são anônimas. Você deve atribuí-las a uma variável. Neste exemplo, a função de seta tem uma expressão “valorA + valorB” utilizada para retornar o resultado da expressão. No entanto, se você usar a sintaxe de bloco, ou seja, com as chaves “{}”, você precisará especificar a palavra-chave “return”:

```
● ● ●  
let adicionar = (valorA, valorB) => {  
    return valorA + valorB;  
};  
  
// Chama a função e exibe o resultado  
console.log(adicionar(10, 20)); // Resultado: 30
```

Se você tiver apenas um argumento, os parênteses em torno dos parâmetros podem ser omitidos, tornando a expressão ainda mais curta. Confira:

```
● ● ●  
let dobro = numero => numero * 2;  
console.log(dobro(3)); // Resultado: 6
```

Este resultado pode ser obtido pelo modo tradicional, utilizando a sintaxe de bloco:

```
● ● ●  
let dobro = function (numero) {  
    return numero * 2;  
};  
console.log(dobro(3)); // Resultado: 6
```

Se não houver argumentos, logo os parênteses devem estar vazios e sempre devem estar presentes. Confira um exemplo:

```
● ● ●  
let mensagem = () => console.log("Olá mundo!");  
mensagem();
```

As funções do tipo *arrow function* podem parecer estranhas e não muito legíveis no início, mas isso muda rapidamente à medida que você se acostuma com a estrutura. Elas são muito convenientes para ações simples de uma linha. Entretanto, às vezes você precisa de uma função mais complexa, com várias expressões e declarações. Nesse caso, é possível colocá-las entre chaves. A principal diferença é que as chaves exigem uma instrução “return” dentro delas para retornar um valor (assim como uma função regular faz) (HAVERBEKE, 2018). Confira um exemplo:

```
● ● ●  
let soma = (a, b) => { // A chave abre uma função multilinha  
    let resultado = a + b;  
    return resultado; // Se usarmos chaves, precisamos de uma  
    // instrução "return" explícita  
};  
  
console.log(soma(1, 2)); // Resultado: 3
```

Obtendo valores de uma entrada com JavaScript

Os campos de entrada nos permitem receber dados dos usuários. Existem muitos tipos de campos de entrada e, embora obter seu valor seja feito de maneira semelhante em cada caso, é necessário aplicar algumas técnicas para obter esses dados. Aqui está um exemplo básico. Neste exemplo, é criado um campo de entrada de texto e, por meio de uma função, seu valor é coletado e impresso no console.



```
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="UTF-8" />
        <title>Exemplo</title>
    </head>
    <body>
        <input
            type="text"
            id="txtCampo"
            placeholder="Digite o texto"
            onblur="capturarValor()"
        />
        <script>
            function capturarValor() {
                const valor = document.getElementById("txtCampo").value;
                console.log(valor);
            }
        </script>
    </body>
</html>
```

O valor da entrada, armazenado na variável “valor”, será impresso no console quando a função “capturarValor()” for chamada. Cada entrada pode receber um atributo que estabeleça um evento que acione a execução de uma função. No exemplo anterior, usamos o evento DOM (Document Object Model) “onblur” para fazer chamadas para nossa função de manipulador de eventos. Além disso, usamos a função “getElementById” para acessar diretamente o elemento HTML chamado de “txtCampo” para, posteriormente, com a função “value”, obter o seu valor. O evento “onblur” aciona a função “capturarValor” quando o campo perde o foco (ou seja, o usuário navega para fora do campo).

Cada tipo de entrada exigirá eventos diferentes para acionar a função do manipulador. Essa é a parte de recuperar a entrada do usuário que requer alguma reflexão. Além disso, vários eventos podem ser usados com a mesma entrada. Entender quando esses eventos são acionados e como eles interagem com seu campo de entrada é a chave para que seus manipuladores de eventos funcionem corretamente. O exemplo a seguir exibirá o texto inserido no campo de entrada ao clicar no botão “Capturar” usando JavaScript.

```
<!DOCTYPE html>

<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <title>Exemplo</title>
  </head>
  <body>
    <input type="text" placeholder="Digite o texto" id="txtCampo" />
    <button type="button" onclick="capturarValor();">Capturar</button>
    <h1 id="txtResultado"></h1>
    <script>
      function capturarValor() {
        // Selecionando o elemento de entrada e obtendo seu valor
        var valor = document.getElementById("txtCampo").value;
        // Exibindo o valor coletado usando o console
        console.log(valor);
      }
    </script>
  </body>
</html>
```

```
// Exibindo o valor coletado na própria página HTML  
document.getElementById("txtResultado").innerHTML = valor;  
}  
</script>  
</body>  
</html>
```

Observe que a propriedade “innerHTML” é usada para definir as tags HTML e também o conteúdo que é inserido entre elas. Essa propriedade é um dos componentes do DOM que permite aos desenvolvedores manipular as aparências das páginas da web. Além disso, você tem a oportunidade de modificar ou substituir elementos HTML. Assim, no exemplo apresentado, o elemento <h1>, representado pelo identificador “txtResultado”, recebe o valor armazenado da variável “valor” e exibe este conteúdo na própria página HTML quando o botão “Capturar” for acionado.

Dica



Quer saber mais sobre os eventos disponíveis no JavaScript? Acesse esta documentação:
<https://developer.mozilla.org/pt-BR/docs/Web/Events> e confira as diversas possibilidades de uso.

Durante seus estudos, você compreendeu como utilizar as estruturas de controle e de repetição com o objetivo de automatizar diversas necessidades de negócio em seus projetos. Além disso, você viu os principais recursos oferecidos com a utilização de funções. Com o uso de funções, é possível criar um código compacto que traz um fácil entendimento. Esse recurso pode eliminar a poluição do seu código. Assim, as funções são os blocos de construção fundamentais para bibliotecas de código reutilizáveis. Como complemento, você viu como obter o valor de uma entrada de um formulário utilizando JavaScript. Isso é muito útil, pois, com a utilização de funções e dos campos de entrada, conseguimos receber qualquer dado dos usuários. Avance para novos tópicos, pois ainda há conteúdos interessantes para você conhecer! Continue seus estudos!



REFERÊNCIAS

DESENVOLVIMENTO web com HTML, CSS e JavaScript: curso WD-43. **Caelum**, [s.d.]. Disponível em: <https://www.caelum.com.br/download/caelum-html-css-javascript.pdf>. Acesso em: 15 jul. 2022.

FLANAGAN, D. **JavaScript**: o guia definitivo. 6. ed. Porto Alegre: Bookman, 2013.

HAVERBEKE, M. **Eloquent JavaScript**. 3. ed. San Francisco. 2018.

MORRISON, M. **Use a Cabeça! JavaScript**. Rio de Janeiro: Alta Books, 2008.



