<https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/What_are_browser_developer_tools>

**PRIMEIROS PASSOS JAVASCRIPT**

Ferramenta do Desenvolvedor = acessado no browser

var myImage = document.createElement('img');

myImage.setAttribute('src','https://blog.mozilla.org/press/wp-content/themes/OneMozilla/img/mozilla-wordmark.png');

document.querySelector('h1').appendChild(myImage);

O método appendChild() insere um novo nó na estrutura do DOM ...

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript>

Contudo, por favor, não faça isso. É uma má prática poluir seu HTML com JavaScript, e isso é ineficiente — você teria que incluir o atributo onclick="criarParagrafo()" em todo botão que você quisesse aplicar JavaScript.

Usando uma estrutura feita de puro JavaScript permite a você selecionar todos os botões usando uma instrução. O código que nós usamos acima para servir a esse propósito se parece com isso:

const botoes = document.querySelectorAll('button');

for(var i = 0; i < botoes.length ; i++) {

botoes[i].addEventListener('click', criarParagrafo);

}

Isso talvez parece ser mais do que o atributo onclick, mas isso vai funcionar para todos os botões, não importa quantos tem na página, e quantos forem adicionados ou removidos. O JavaScript não precisará ser mudado.

No exemplo interno, você pode ver essa estrutura em volta do código:

document.addEventListener("DOMContentLoaded", function() {

...

});

Explicação event listener = <https://developer.mozilla.org/pt-BR/docs/Web/API/EventTarget/addEventListener>

<https://developer.mozilla.org/en-US/docs/Web/API/Event/type>

Isso é um *event listener* (ouvidor de eventos*)*, que ouve e aguarda o disparo do evento "DOMContentLoaded" vindo do *browser*, evento este que significa que o corpo do HTML está completamente carregado e pronto. O código JavaScript que estiver dentro desse bloco não será executado até que o evento seja disparado, portanto, o erro será evitado (você irá [aprender sobre eventos](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events) mais tarde).

No exemplo externo, nós usamos um recurso moderno do JavaScript para resolver esse problema: Trata-se do atributo defer, que informa ao *browser* para continuar renderizando o conteúdo HTML uma vez que a tag <script> foi atingida.

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

Neste caso, ambos script e HTML irão carregar de forma simultânea e o código irá funcionar.

Nota: No caso externo, nós não precisamos utilizar o evento DOMContentLoaded porque o atributo defer resolve o nosso problema. Nós não utilizamos defer como solução para os exemplos internos pois defer funciona apenas com scripts externos.

Uma solução à moda antiga para esse problema era colocar o elemento script bem no final do body da página (antes da tag </body>)

#### **async e defer**

Atualmente, há dois recursos bem modernos que podermos usar para evitar o problema com o bloqueio de scripts — async e defer (que vimos acima).

* Um comentário de uma linha é escrito depois de duas barras. Por exemplo:
* // Eu sou um comentário

Um comentário de múltiplas linhas é escrito entre os caracteres /\* e \*/. Por exemplo:  
/\*

Eu também sou

um comentário

\*/

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/A_first_splash>

Eventos são ações que acontencem no navegador, como um botão sendo clicado, ou uma página carregada, ou um vídeo tocando; ações as quais podemos responder executando blocos de código. Os construtores que monitoram os acontecimentos de eventos são chamados de **event listeners**, e os blocos de código executados em resposta ao acontecimento do evento são chamados de **event handlers**.

Adicione a seguinte linha abaixo da chave de fechamento da sua função conferirPalpite():

envioPalpite.addEventListener('click', conferirPalpite);

Aqui nós estamos adicionando um event listener ao botão envioPalpite. Esse é um método que aceita a inserção de dois valores (chamados de argumentos) — o tipo de envento que estamos monitorando (neste caso o evento click) como um string (sequência de texto), e o código que queremos executar quando o evento ocorrer (neste caso a função conferirPalpite() — note que não temos que especificar os parênteses quando estivermos escrevendo dentro de [addEventListener()](https://developer.mozilla.org/pt-BR/docs/Web/API/EventTarget/addEventListener)).

botaoReinicio = document.createElement('button');

Gera um novo elemento button

botaoReinicio.textContent = 'Iniciar novo jogo';  
Define o texto do rótulo do novo elemento button

document.body.appendChild(botaoReinicio);

adiciona o elemento ao final do nosso HTML existente

var reiniciarParas = document.querySelectorAll('.resultadoParas p');

for (var i = 0 ; i < reiniciarParas.length ; i++) {

reiniciarParas[i].textContent = '';

}

Esse código cria uma variável contendo uma lista de todos os parágrafos dentro de <div class="resultadoParas"> usando o método querySelectorAll(), e então faz o loop em cada um, removendo o conteúdo de texto dos mesmos.

Em JavaScript, tudo é um objeto. Um objeto é uma coleção de funcionalidades relacionadas armazenadas em um único agrupamento. Você pode criar seus próprios objetos.

Neste caso particular, nós primeiro criamos a variável campoPalpite que armazena uma referência ao campo de inserção de texto do formulário em nosso HTML — a linha seguinte pode ser achada entre nossas declarações de variáveis próximas ao topo:

var campoPalpite = document.querySelector('.campoPalpite');

Para pegar essa referência, usamos o método [querySelector()](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/querySelector) do objeto [document](https://developer.mozilla.org/pt-BR/docs/Web/API/Document). querySelector() pega um pedaço de informação — um [seletor CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors) que seleciona o elemento ao qual você quer referenciar.

Como agora campoPalpite contém referência ao elemento [<input>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/Input), ele terá agora acesso a um número de propriedades (basicamente variáveis armazenadas dentro de objetos, sendo que alguns não podem ter seus valores alterados) e métodos (basicamente, funções armazenadas dentro de objetos). Um método disponível para elementos de inserção <input>, é o focus(), então agora podemos usar essa linha para focar o campo de inserção de texto:

campoPalpite.focus();

Variáveis que não contém referências a elementos de formulário não terão focus() disponível para elas. Por exemplo, a variável palpites contém referência de um elemento [<p>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/p), e contagemPalpites contém um número.

[**Brincando com objetos do navegador**](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/A_first_splash#brincando_com_objetos_do_navegador) **(usando o console)**

Vamos brincar um pouco com alguns objetos do navegador.

1. Primeiro abra seu programa em um navegador.
2. Em seguida, abra as [ferramentas de desenvolvimento do navegador](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_are_browser_developer_tools), e certifique-se de que a aba do console JavaScript esteja aberta.
3. Digite campoPalpite e o console irá lhe mostrar que a variável contém um elemento [<input>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/Input). Você também irá notar que o console completa automaticamente os nomes de objetos existentes dentro do ambiente de execução, incluindo suas variáveis!
   1. Agora digite o seguinte:

campoPalpite.value = 'Olá';

A propriedade value representa o valor atual inserido no campo de texto. Você verá que inserindo esse comando, nós mudamos o valor desse objeto!

1. Agora tente digitar palpites e pressione *return*. O console irá mostrar que a variável contém um elemento [<p>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/p).
2. Agora tente inserir a linha seguinte:

palpites.value

O navegador irá retornar undefined, porque value não existe em parágrafos.

1. Para mudar o texto dentro de um parágrafo, você precisa da propriedade [textContent](https://developer.mozilla.org/pt-BR/docs/Web/API/Node/textContent). Tente isso:

palpites.textContent = 'Onde está meu parágrafo?';

1. Agora algo divertido. Tente inserir as linhas abaixo, uma por uma:
2. palpites.style.backgroundColor = 'yellow';
3. palpites.style.fontSize = '200%';
4. palpites.style.padding = '10px';

palpites.style.boxShadow = '3px 3px 6px black';

Cada elemento em uma página tem uma propriedade style, que contém um objeto no qual estão inseridos em suas propriedades todos os estilos incorporados de CSS aplicados ao respectivo elemento. Isso nos permite configurar dinamicamente novos estilos CSS nos elementos usando JavaScript.

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_went_wrong>

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/Strings>

[**Aspas simples x aspas duplas**](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/Strings#aspas_simples_x_aspas_duplas)

1. Em JavaScript, você pode escolher aspas simples ou duplas para envolver suas strings. Ambas linhas abaixo funcionará bem:
2. var sgl = 'Single quotes.';
3. var dbl = "Double quotes";
4. sgl;

dbl;

Copy to Clipboard

1. Há poucas diferenças entre as duas, e qual você usar é de preferência pessoal. Você deve escolher uma e permanecer nela, entretanto; diferentes aspas no código pode ser confuso, especialmente se você usa diferentes aspas na mesma string! O seguinte retornará erro:

var badQuotes = 'What on earth?";

1. O navegador interpretará como a string não tivesse fechada, porque o outro tipo de aspas pode aparecer dentro da sua string. Por exemplo, ambos exemplos abaixo são okay:
2. var sglDbl = 'Would you eat a "fish supper"?';
3. var dblSgl = "I'm feeling blue.";
4. sglDbl;

dblSgl;

Copy to Clipboard

1. Entretanto, você não pode incluir o mesmo tipo de aspas dentro da sua string, se você usa para conter seu texto. O seguinte será um erro, como é confuso para o navegador onde a string termina:

var bigmouth = 'I've got no right to take my place...';

Isto nos leva muito bem ao nosso próximo assunto.

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/Arrays>

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Indexed_collections>

**trabalhando com ARRAYS e OBJETOS**

var arr = [42]; // Cria um array com apenas um elemento:

// o número 42.

var arr = Array(42); // Cria um array sem elementos

// e arr.length é definido como 42; isso é

// equivalente a:

var arr = [];

arr.length = 42;

Você também pode povoar o array quando o cria:

var myArray = new Array('Olá', myVar, 3.14159);

var myArray = ['Manga', 'Maçã', 'Laranja']

**Nota :** o operador do array (colchetes) também é usado para acessar as propriedades do array (arrays também são objetos em JavaScript). Por exemplo,

var arr = ['um', 'dois', 'três'];

arr[2]; // três

arr['length']; // 3

O método [forEach()](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach) disponibiliza um outro jeito de iterar sobre/em um array:

var cores = ['vermelho', 'verde', 'azul'];

cores.forEach(function(cor) {

console.log(cor);

});

// vermelho

// verde

// azul

**Alternativamente, você pode encurtar o código para o parâmetro do forEach**

com Arrow Functions ES6.

var cores = ['vermelho', 'verde', 'azul'];

cores.forEach(cor **=>** console.log(cor));

// vermelho

// verde

// azul

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/Silly_story_generator>

Mensagens de erro mais comuns

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

**2ª parte: PROGRAMANDO EM JAVASCRIPT**

[**https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building\_blocks**](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks)

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals>

**if (***condicao***) {**

*codigo para executar caso a condição seja verdadeira*

**} else {**

*senão, executar este código*

**}**

[Operadores lógicos: AND, OR e NOT](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals#operadores_l%C3%B3gicos_and_or_e_not)

Se você quiser testar várias condições sem escrever instruções aninhadas if...else, os [operadores lógicos](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators) poderão ajudá-lo. Quando usado em condições, os dois primeiros fazem o seguinte:

&& — AND; permite encadear duas ou mais expressões para que todas elas tenham que ser avaliadas individualmente como true assim toda a expressão retorna true.

|| — OR; permite encadear duas ou mais expressões para que uma ou mais delas tenham que ser avaliadas individualmente como true assim toda a expressão retorna true.

O último tipo de operador lógico, NOT, expressado pelo operador !, pode ser usado para negar uma expressão. Vamos combiná-lo com OR no exemplo acima:

### [Exemplo de operador ternário](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/conditionals#exemplo_de_operador_tern%C3%A1rio)

<https://programandosolucoes.dev.br/2021/04/13/operador-ternario-javascript/#:~:text=O%20operador%20tern%C3%A1rio%2C%20%C3%A9%20um,ser%20substitu%C3%ADdo%20por%20apenas%201.>

Você não precisa apenas definir valores de variáveis com o operador ternário; Você também pode executar funções ou linhas de código - qualquer coisa que você gosta. O exemplo ao vivo a seguir mostra um seletor de temas simples em que o estilo do site é aplicado usando um operador ternário.

<label for="theme">Select theme: </label>

<select id="theme">

<option value="white">White</option>

<option value="black">Black</option>

</select>

<h1>This is my website</h1>

Copy to Clipboard

var select = document.querySelector('select');

var html = document.querySelector('html');

document.body.style.padding = '10px';

function update(bgColor, textColor) {

html.style.backgroundColor = bgColor;

html.style.color = textColor;

}

select.onchange = function() {

( select.value === 'black' ) ? update('black','white') : update('white','black');

}

Aqui nós temos um elemento [<select>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/select) para escolher um tema (preto ou branco), além de um simples [<h1> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements) para exibir um título do site. Nós também temos uma função chamada update(), que leva duas cores como parâmetros (entradas). A cor do plano de fundo do site é definida para a primeira cor fornecida e sua cor de texto é definida para a segunda cor fornecida.

Finalmente, nós também temos um evento listener [onchange](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event) que serve para executar uma função que contém um operador ternário.

Começa com uma condição de teste — select.value === 'black'.

Se este retornar true, nós executamos a função update() com parâmetros de preto e branco, o que significa que acabamos com a cor de fundo do preto e cor do texto de branco.

Se retornar false, nós executamos a função update() com parâmetros de branco e preto, o que significa que a cor do site está invertida.

Código em LOOP

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Looping_code>

for (var i = 0; i < 100; i++) {

ctx.beginPath();

ctx.fillStyle = 'rgba(255,0,0,0.5)';

ctx.arc(random(WIDTH), random(HEIGHT), random(50), 0, 2 \* Math.PI);

ctx.fill();

}

um comando  break  irá imediatamente sair do loop e fazer o navegador passar para o código seguinte.

A instrução [continue](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue) funciona de maneira semelhante ao  break, mas ao invés de sair inteiramente do loop, ele pula para a próxima iteração do loop.

while:

initializer

while (exit-condition) {

// code to run

final-expression

}

e finalmente do...while:

initializer

do {

// code to run

final-expression

} while (exit-condition)

**Funções – blocos reutilizaveis de código**

[**https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions)

### [A expressão function (Operador function)](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions#a_express%C3%A3o_function_operador_function)

Uma expressão function é parecida com uma declaração function e tem até a mesma sintaxe (veja [Operador function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function) para detalhes):

function [nome]([param] [, param] [..., param]) {

instruções

}

nome

O nome da função. Pode ser omitido, e neste caso a função é conhecida como função anônima.

param

O nome de um argumento a ser passado para a função.

instruções

As instruções que formam o corpo da função.

### [O gerador de declaração de função (instrução function\*)](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions#o_gerador_de_declara%C3%A7%C3%A3o_de_fun%C3%A7%C3%A3o_instru%C3%A7%C3%A3o_function*)

**Nota:** Expressão Arrow function é uma tecnologia experimental, parte da proposta do ECMAScript 6, e ainda não é amplamente suportada pelos navegadores.

Uma expressão arrow function tem uma sintaxe curta e conecta seu valor lexicamente (veja [arrow functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions) para mais detalhes):

([param] [, param]) => {

instruções

}

param => expressão

param

O nome de um argumento. Quando não há argumentos deve ser indicado com o uso dos parênteses ( ). Para apenas um argumento os parênteses não são obrigatórios. (por exemplo foo => 1)

instruções ou expressão

Múltiplas instruções precisam ser envolvidas por chaves. Uma única expressão não requer chaves. A expressão também é implicitamente o valor de retorno  dessa função.

### [O construtor Function](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Functions#the_function_constructor)

**Nota:** O uso do construtor Function para criar funções não é recomendado uma vez que é requerido o corpo da função como string, o que pode impedir algumas otimizações por parte do motor JS e pode também causar outros problemas.

Como todos os outros objetos, objetos [Function](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Function) podem ser criados usando o operador new:

new Function (arg1, arg2, ... argN, corpoDaFuncao)

arg1, arg2, ... argN

Nenhum ou mais nomes para serem usados pela função como nomes formais de argumentos. Cada um deve ser uma string em conformidade com as regras para um identificador JavaScript válido ou uma lista com tais strings separadas por vírgula; por exemplo "x", "oValor", ou "a, b".

corpoDaFuncao

Uma string contento as instruções JavaScript correspondendo a definição da função.

Invocar o construtor Function como uma função (sem usar o operador new) the o mesmo efeito de invocá-lo como um construtor comum.

**= = = = = = = = = =**

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Functions>

A distinção é que métodos são funções definidas dentro de objetos. Funções embutidas de navegador (métodos) e variáveis (que são chamadas propriedades) são armazenadas dentro de objetos estruturados, para tornar o código mais estruturado e fácil de manipular.

Mas você também pode criar funções que não tem nome:

function() {

alert('hello');

}

Isto é chamado**Função anônima** — não tem nome! E também não fará nada em si mesma. Você geralmente cria funções anônimas junto com um  manipulador de eventos, o exemplo a seguir poderia rodar o código dentro da função sempre que o botão associado é clicado:

var myButton = document.querySelector('button');

myButton.onclick = function() {

alert('hello');

}

Voce também pode atribuir uma função anônima para ser o valor de uma variável, por exemplo:

var myGreeting = function() {

alert('hello');

}

Esta função agora poder ser invocada usando:

myGreeting();

Isso dá efetivamente um nome a função; você também pode atribuir uma função para ser o valor de múltiplas variáveis , por exemplo:

var anotherGreeting = function() {

alert('hello');

}

Esta função agora pode ser invocada usando qualquer das funções abaixo

myGreeting();

anotherGreeting();

Mas isso pode simplesmente ser confuso, então não faça! Quando criar funções, é melhor ficar com apenas uma forma:

function myGreeting() {

alert('hello');

}

De modo geral você irá usar funções anônimas só para rodar um código em resposta a um disparo de evento — como um botão ao ser clicado — usando um gerenciador de eventos. Novamente, Isso é algo parecido com:

myButton.onclick = function() {

alert('hello');

// Eu posso colocar código aqui

// dentro o quanto eu quiser

}

**Nota**:

As mesmas regras de escopo não se aplicam a laços (ex.: for() { ... }) e blocos condicionais (ex.: if() { ... }) — eles parecem muito semelhantes, mas eles não são a mesma coisa! Tome cuidado para não confudir-se.

Explicação com relação ao uso ou não do parenteses após o nome de uma função...

displayMessage é o nome de uma função criada em ... <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Build_your_own_function#chamando_a_fun%C3%A7%C3%A3o>

1. btn.onclick = displayMessage;

De maneira semelhante à nossa linha  closeBtn.onclick... dentro da função, aqui estamos chamando algum código em resposta a um botão sendo clicado.

Mas, neste caso, em vez de chamar uma função anônima contendo algum código, estamos chamando nosso nome de função diretamente.

Você pode estar se perguntando por que não incluímos os parênteses após o nome da função. Isso ocorre porque não queremos chamar a função imediatamente — somente depois que o botão foi clicado.

Se você tentar mudar a linha para

btn.onclick = displayMessage();

e salvar e recarregar, você verá que **a caixa de mensagem aparece sem que o botão seja clicado!**

Os parênteses neste contexto são às vezes chamados de "operador de invocação de função imediata".

Você só os usa quando deseja executar a função imediatamente no escopo atual.

No mesmo sentido, o código dentro da função anônima não é executado imediatamente, pois está dentro do escopo da função.

Exemplo 1

      btn.onclick = displayMessage;

Exemplo 2

      btn.onclick = function() {

        displayMessage('Woo, this is a different message!');

      }

Se quisermos especificar parâmetros dentro de parênteses para a função que estamos chamando (exemplo 2), então não podemos chamá-la diretamente (exemplo 1) — precisamos colocá-lo dentro de uma função anônima para que não fique no escopo imediato e, portanto, não seja chamado imediatamente. Agora ele não será chamado até que o botão seja clicado.

# Valores de retorno de função

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Return_values>

**Introdução a eventos**

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Events>

Eventos são ações ou ocorrências que acontecem no sistema que estamos desenvolvendo

Lista de eventos: <https://developer.mozilla.org/en-US/docs/Web/Events>

Cada evento disponivel possui um **manipulador de eventos**(event handler), que é um bloco de código (geralmente uma função JavaScript definida pelo usuário) que será executado quando o evento for disparado. Quando esse bloco de código é definido para rodar em resposta a um evento que foi disparado, nós dizemos que estamos **registrando um manipulador de eventos**. Note que manipuladores de eventos são, em alguns casos, chamados de **ouvinte de eventos**(event listeners) — eles são praticamente intercambiáveis para nossos propósitos, embora estritamente falando, eles trabalhem juntos. Os ouvintes escutam o evento acontecendo, e o manipulador é o codigo que roda em resposta a este acontecimento.

**Nota**: É importante notar que eventos web não são parte do core da linguagem JavaScript — elas são definidas como parte das APIs JavaScript incorporadas ao navegador.

### [Objetos de evento](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Events#objetos_de_evento)

Às vezes, dentro de uma função de manipulador de eventos, você pode ver um parâmetro especificado com um nome como event, evt, ou simplesmente e. Isso é chamado de **event object**, e é passado automaticamente para os manipuladores de eventos para fornecer recursos e informações extras.

Exemplo:

function bgChange(e) {

var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random(255) + ')';

e.target.style.backgroundColor = rndCol;

console.log(e);

}

btn.addEventListener('click', bgChange);

Aqui você pode ver que estamos incluindo um objeto de evento, **e**, na função, e na função definindo um estilo de cor de fundo em e.target — que é o próprio botão. A propriedade target do objeto de evento é sempre uma referência ao elemento em que o evento acabou de ocorrer. Portanto, neste exemplo, estamos definindo uma cor de fundo aleatória no botão, não na página.

**Note**: Você pode usar qualquer nome que desejar para o objeto de evento — você só precisa escolher um nome que possa ser usado para referenciá-lo dentro da função de manipulador de eventos. e/evt/event são mais comumente usados pelos desenvolvedores porque são curtos e fáceis de lembrar. É sempre bom manter um padrão.

e.target é incrivelmente útil quando você deseja definir o mesmo manipulador de eventos em vários elementos e fazer algo com todos eles quando ocorre um evento neles. Você pode, por exemplo, ter um conjunto de 16 blocos que desaparecem quando são clicados. É útil poder sempre apenas definir a coisa para desaparecer como e.target, ao invés de ter que selecioná-lo de alguma forma mais difícil.

 No exemplo a seguir (veja [useful-eventtarget.html](https://github.com/mdn/learning-area/blob/master/javascript/building-blocks/events/useful-eventtarget.html) para o código-fonte completo; veja também a [execução ao vivo](https://mdn.github.io/learning-area/javascript/building-blocks/events/useful-eventtarget.html) aqui), criamos 16 elementos [<div>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/div) usando JavaScript. Em seguida, selecionamos todos eles usando [document.querySelectorAll()](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/querySelectorAll)  e, em seguida, percorremos cada um deles, adicionando um manipulador onclick a cada um, de modo que uma cor aleatória seja aplicada a cada um deles quando clicados:

#### Delegação de eventos

O borbulhar também nos permite aproveitar a **delegação de eventos** — esse conceito depende do fato de que, se você quiser que algum código seja executado quando clicar em qualquer um de um grande número de elementos filho, você pode definir o ouvinte de evento em seu pai e ter os eventos que acontecem neles confluem com o pai, em vez de precisar definir o ouvinte de evento em cada filho individualmente.

Um bom exemplo é uma série de itens de lista — Se você quiser que cada um deles apareça uma mensagem quando clicado, você pode definir o ouvinte de evento click no pai <ul>,  e ele irá aparecer nos itens da lista.

Este conceito é explicado mais adiante no blog de David Walsh, com vários exemplos — veja [Como funciona a delegação de eventos em JavaScript.](https://davidwalsh.name/event-delegate)

## [Conclusão](https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Events#conclus%C3%A3o)

Agora você deve saber tudo o que precisa saber sobre os eventos da Web nesse estágio inicial. Como mencionado acima, os eventos não são realmente parte do núcleo do JavaScript — eles são definidos nas APIs da Web do navegador.

Além disso, é importante entender que os diferentes contextos nos quais o JavaScript é usado tendem a ter diferentes modelos de evento — de APIs da Web a outras áreas, como WebExtensions de navegador e Node.js (JavaScript do lado do servidor). Não esperamos que você entenda todas essas áreas agora, mas certamente ajuda a entender os fundamentos dos eventos à medida que você avança no aprendizado do desenvolvimento da web.

Galeria de Imagens

<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Image_gallery>