

LTPW1

Capítulo 01

História do JavaScript

JavaScript é uma linguagem moderna, certo? Nem tanto! Mas ela vem evoluindo durante os últimos anos. Ao completar essa aula, você será capaz de entender de onde veio a linguagem JS, vai saber diferenciar as linguagens Java e JavaScript e vai entender o que é o tão falado ECMAScript. Chegou a hora de começar seus estudos em JS, vamos lá?



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicada - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidade de obter ganho financeiro com ele.



Antes de começar...

Eu sei que você está com muita vontade de começar a aprender JavaScript, mas eu preciso muito que você leia dois documentos antes. Vá até o meu repositório (<https://github.com/gustavoguanabara/>), na área de PDFs do **Curso de HTML e CSS3**, e leia os documentos:

- 01 - [História da Internet](#)
- 02 - [Como funciona a Internet](#)

Lá eu vou mostrar o funcionamento básico da Internet e vou considerar que você já possui essa base antes de começar.

Também recomendo treinar a [lista básica de exercícios de HTML e CSS](#) para que você entenda as tags básicas e como configurar os seletores de estilo antes de se aventurar no aprendizado de JavaScript. Muitos comandos vão começar a aparecer por aqui e talvez você não esteja totalmente familiarizado(a) com eles.

Então vai lá, investe um tempinho para ler tudo com muita paciência e entender o funcionamento básico da Internet antes de voltar aqui e dar prosseguimento ao Curso de JavaScript. Combinado? 

Desde o Mocha, passando pelo LiveScript... E o marketing!

Começamos tudo em 1995. A World Wide Web tinha acabado de ser criada, junto com a linguagem HTML e seus primeiros navegadores arcaicos. Eu não sei se você sabe, mas hoje em dia não existiriam sites modernos sem o JavaScript, mas naquela época não tinha nada disso. As interatividades eram praticamente inexistentes e nós tínhamos uma Internet muito mais estática e sem graça.

Basicamente, os navegadores que “brigavam” pelo mercado eram o recém lançado **Internet Explorer** da Microsoft e o **Netscape** da empresa de mesmo nome.



A Microsoft tinha um trunfo na manga: o **Windows**. O que ela fez foi embutir o **IE** no sistema operacional, forçando todo mundo a usar o navegador dela, sem sequer poder desinstalá-lo. O usuário padrão sequer cogitaria instalar um segundo navegador como o **Netscape**, mesmo que o segundo fosse bem melhor (e era).

Pensando em oferecer uma experiência digna de quebrar a barreira e fazer todo mundo ficar com vontade de instalar o Netscape como um navegador extra, a

empresa pensou em adicionar funcionalidades exclusivas que o IE não tinha, uma delas nos interessa bastante: INTERATIVIDADE.



E foi em 1995 que um programador chamado **Brendan Eich** (o cara aí ao lado) saiu da **Silicon Graphics** e foi contratado e começou a trabalhar na Netscape. Ele tinha criado uma linguagem chamada **Mocha**, que daria poderes especiais ao **Netscape 2.0** que sairia no fim do ano. Ao entrar na empresa, Eich e a equipe mudaram o nome da linguagem para **LiveScript**, mas essa nova alcunha demoraria pouco.

Muito se falava no mercado sobre uma poderosa linguagem chamada **Java**. Os jornais a chamavam de “Linguagem do Futuro”, aquela que mudaria o mundo dali pra frente.

Pensando em pegar uma “carona” na fama do Java, a equipe resolveu rebatizar o **LiveScript** para **JavaScript** em uma simples demonstração de decisão de puro marketing. E pra gerar ainda mais confusão, mais tarde ainda surgiu uma parceria entre a **Sun Microsystems** (criadora do Java) e a **Netscape** (criadora do JavaScript).



APRENDA MAIS: Veja um vídeo que vai esclarecer as dúvidas e te mostrar de maneira simples que **JavaScript não é Java!**

Faiçal e Conrado: <https://youtu.be/hvcEPAnKB50>

O JavaScript deu certo!

Mesmo com toda confusão gerada entre os leigos, o JavaScript agradou bastante e aqueceu ainda mais a briga entre os navegadores.

E deu tão certo que aconteceu o que sempre acontece com tudo aquilo que dá certo: o JavaScript foi CO-PI-A-DO! E a “cópia” partiu de nada mais, nada menos que a **Microsoft**! O Internet Explorer 3 sairia compatível com a linguagem **JScript**, criada pela empresa do Bill Gates.

Para combater essa sem-vergonhice da Microsoft, a Netscape resolveu normalizar a linguagem através da organização Européia especializada na criação de padrões para a computação: a **ECMA**.

A primeira versão do ECMA-Script (versão padronizada) só sairia em Junho de 1997, mas até hoje é muito importante, pois faz a linguagem evoluir a cada ano de forma contínua e descentralizada (independente de qualquer empresa particular).

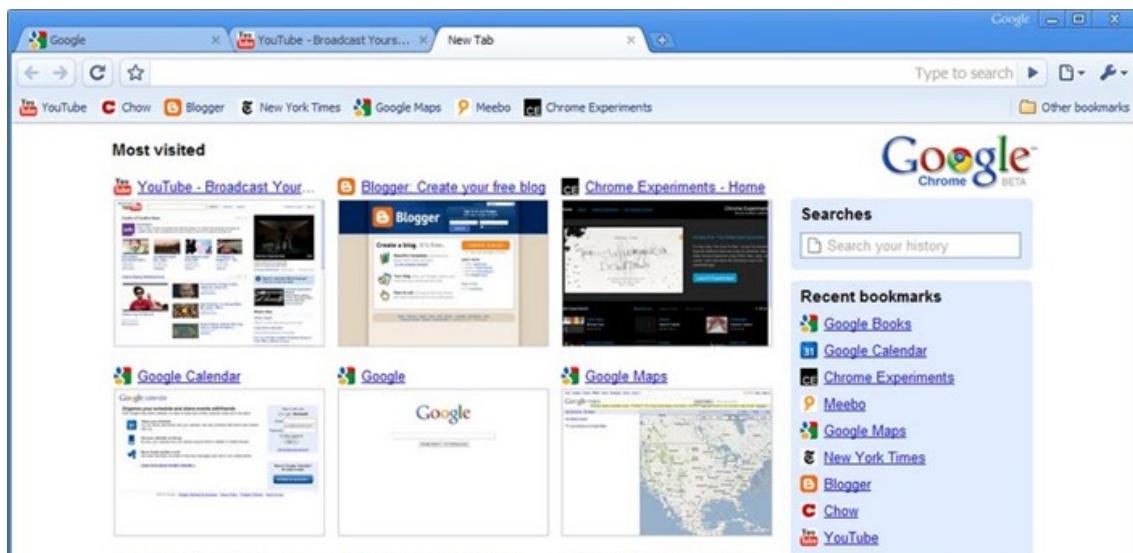
E o objetivo de padronizar a linguagem acabou sendo uma ótima escolha, uma vez que em 2008, depois de muito lutar contra a concorrência e abusos da Microsoft, a Netscape faliu e deixou o navegador pra trás.

Mas nem tudo foi em vão. Um grupo de desenvolvedores da Netscape se reuniu e fundaram a **Fundação Mozilla**, que mais tarde lançou o conhecido **Firefox**, o sucessor espiritual do Netscape e que está aí até hoje.

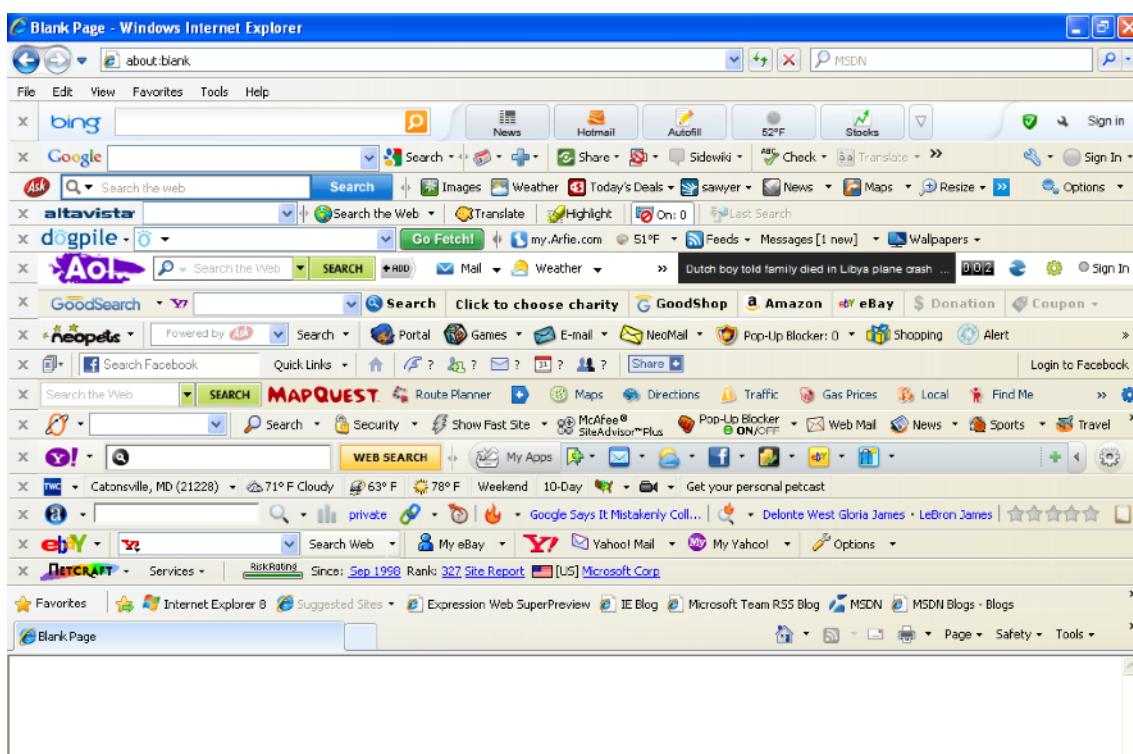


Surgimento do líder

Enquanto a Microsoft comemorava a vitória contra a Netscape, a gigante **Google** lançava em 2008 a primeira versão do **Google Chrome**. Mas a caminhada foi longa, pois foi só em 2016 que o número de pessoas usando Chrome nos computadores ultrapassou o número de pessoas usando o Internet Explorer.



A ideia era ter um navegador mais simples visualmente e sem aquelas barras gigantes com fileiras de botões (Toolbars) que existiam nos outros navegadores. Dá só uma comparada nessas imagens (o de baixo parece muito o navegador do meu pai 😊)



Outra arma muito boa que o Chrome tinha era a sua máquina para rodar JavaScript: o poderoso **V8**. Ele era bem mais eficiente do que a concorrência, entregando ótimos resultados para os usuários (mesmo consumindo mais memória que todos os demais).



CURIOSIDADE: O motor V8 do Google Chrome foi usado como base para a criação do **Note.JS**, um projeto que possibilitou levar o JavaScript para os servidores. A cada dia o projeto cresce mais, fica de olho vivo aí e dá uma estudada em Node mais tarde 😊

E o **Google Chrome** acabou com o reinado do Internet Explorer nos computadores. O navegador do Google evoluiu muito mais e ganhou o primeiro lugar nos computadores a partir de 2016. A Microsoft até tentou reverter a situação, lançando o navegador **Edge**, mais simples e leve, mas já era tarde demais.



O mundo dá voltas...

É quase inacreditável o que aconteceu em 2020 (momento em que estou escrevendo esse material).



A Microsoft, depois de tentar sucesso com o Edge nos PCs com Windows, resolveu entregar os pontos e reconhecer que a base do software criada para o Chrome era superior. E foi em 2020 que surgiu o novo Edge, baseado no **Projeto Chromium** (a parte open-source do projeto do Google).

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal *Curso em Vídeo* no YouTube. O link que vou compartilhar contigo tem a história que você leu aqui, contada de forma mais ilustrada. Reserve um tempo dos seus estudos para assistir esse vídeo todo.



Curso em Vídeo: <https://youtu.be/rUTKmc2gG8>

LTPW1

Capítulo 02

Como o JS funciona?

Agora que você já sabe de onde veio o JavaScript, chegou a hora de entender como a tecnologia funciona. É importante entender os mecanismos de uma linguagem antes de começar a estudá-la. Nesse material, falaremos de JS puro, com os objetivos primordiais definidos para a tecnologia. Vamos entender o passo-a-passo que um código JS leva para ser executado em nosso navegador.



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicado - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidades de obter ganho financeiro com ele.



Antes de começar...

... é importante deixar claro que esse capítulo do material vai explicar o funcionamento do JavaScript da maneira tradicional da sua aplicação.

JS é uma linguagem que hoje funciona em servidores? Claro, mas não foi feita pra isso! Para rodar em servidores, precisamos de um software criado por terceiros (que é o Node.js).

JS pode ser usada para criar aplicativos para Windows? Claro, mas não foi feita pra isso! Usando uma biblioteca chamada Electron, dá pra criar uns programas bem legais, tipo o VS Code da Microsoft, que é feito em JavaScript.

JS pode ser usada para criar programas para celular? Claro, mas não foi feita pra isso! Se você aprender um pouco sobre frameworks como PhoneGap, Xamarin, Ionic ou React, vai conseguir criar aplicações portáteis programando em JavaScript.



Sendo assim, não falaremos sobre os “super poderes” que a linguagem JavaScript vem ganhando com o tempo. Estamos abordando aqui a maneira primordial para qual a linguagem foi criada: interações na web, usando seu navegador.

Combinado?

Lembra de como funciona a Internet?

Lá no material PDF do **Curso de HTML5 e CSS3**, que está nesse mesmo repositório, eu expliquei como funcionava a Internet. Se por acaso você ainda não leu, está ferindo uma regra básica do nosso acordo aqui: não pular nenhum passo. No primeiro capítulo desse material eu dei os pré-requisitos desse curso aqui, e ler o material introdutório de HTML era uma exigência!



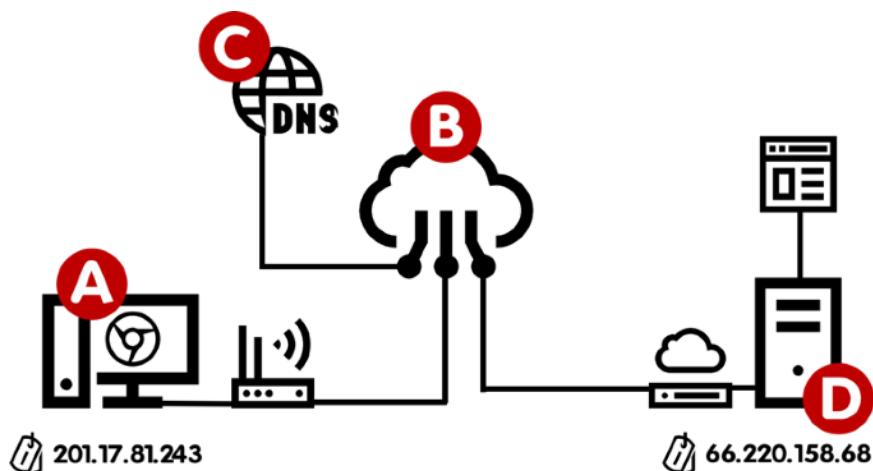
APRENDA MAIS: Para acessar todo o material do Curso de HTML5 e CSS3, basta entrar no link a seguir e clicar no link para acessar o material em PDF do curso.

<https://gustavoguanabara.github.io>

Agora que você já levou um leve puxão de orelha, vamos continuar.

A Internet funciona baseada no paradigma **cliente-servidor**. Nesse exato momento você é o **cliente**, que está com seu navegador aberto e acessando alguns sites (inclusive o repositório do GitHub, no link acima). Essa página que está aparecendo no seu computador/celular está vindo de um **servidor**, que é um outro computador que está preparado para prover os arquivos.

Vamos voltar àquela imagem que está no material de HTML:



Quando está conectado à Internet (B), um visitante abre um navegador (A) e digita uma URL (*endereço do site*), o servidor web (D) deve ser acessado e precisa consultar um servidor DNS (C) para resolver o nome do domínio, retornando o IP atual do servidor.

Uma vez que a conexão entre o cliente (A) e o servidor (D) está estabelecida, o arquivo solicitado vai ser transferido para a máquina de destino.



IMPORTANTE: Quando não solicitamos um arquivo específico ao servidor, ele vai entregar o **arquivo de índice** do site. Para sites em HTML5, esse arquivo de índice se chama `index.html`.

Quando o arquivo HTML inicial chega ao cliente, o navegador vai analisar todas as tags e vai gerar o resultado visual da página. Quando ele encontra tags de mídia (``, `<video>`, `<audio>`, etc), novas solicitações são enviadas para o servidor (D), pedindo mais arquivos (jpg, png, mp3, mp4, etc).

Quando os comandos que chegam estão em **JavaScript**, a *Engine* do navegador será a responsável por executá-los (no caso do **Google Chrome**, o motor JS se chama **V8**). Ele vai conseguir criar interatividade do visitante com os componentes do site (*DOM*) e executar rotinas definidas pelos programadores do site.



HTML e CSS não é programação. JavaScript é?

Quando estamos criando código **HTML** e **CSS**, podemos dizer que estamos “*desenvolvendo em HTML*”, mas é errado dizer que estamos “*programando em HTML*”. HTML e CSS são linguagens, mas não são **linguagens de programação**. Essas linguagens são mais classificadas como **linguagens de marcações**.



Já a **JavaScript** é uma linguagem de programação, considerada uma *linguagem de scripts não compilados*. Todos os códigos JS são enviados para o navegador do cliente e são interpretados linha-a-linha, gerando o resultado.

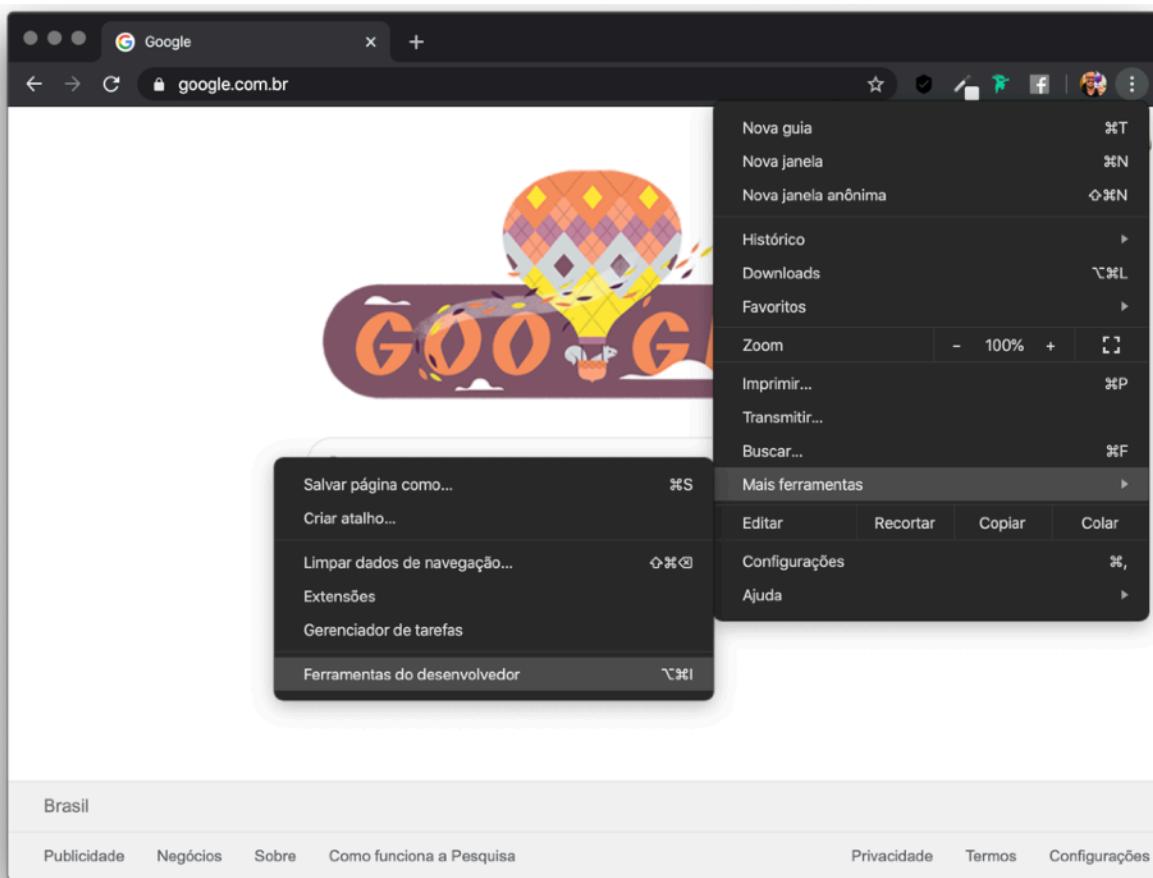
Sendo assim, podemos dizer “estou programando em JS”, mas não podemos dizer “estou programando em HTML”.

Sacou? 😊

Me mostra um exemplo desse tipo de interatividade com JS?

Mas é claro! Vamos fazer um exemplo que se tornou clássico nas minhas aulas.

Abra aí o seu **Google Chrome** (tem que ser ele, pois tem ferramentas que ajudam programadores como nós) e acesse o site www.google.com.br



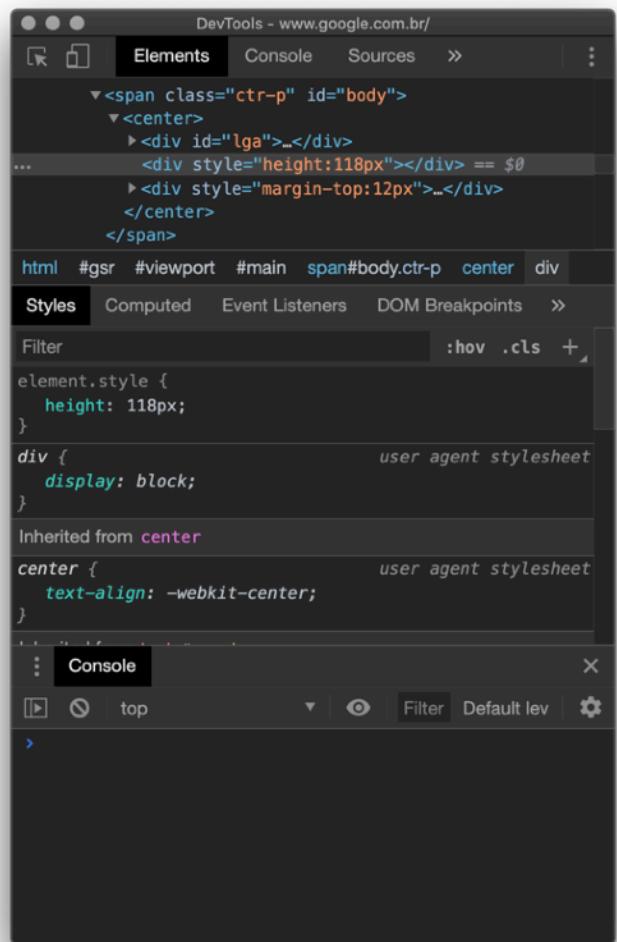
Depois clique naqueles três pontinhos que existem lá em cima na direita (veja a imagem), escolha o menu Mais ferramentas e depois Ferramentas do desenvolvedor.

A tela que vai abrir se chama **DevTools** e está representada aí ao lado. Ela pode aparecer colada ao seu navegador ou flutuando sobre a tela. Você muda isso clicando sobre os três pontinhos lá em cima na direita e mudando o Dock side. Mude essa opção para Undock e deixe ela sempre flutuando, vai ajudar bastante na visualização das coisas na maioria dos casos.

Essa ferramenta é muito útil para quem desenvolve sites. Como você pode perceber, ele mostra os elementos HTML, CSS e permite identificar detalhes do código.

É com o DevTools que você também vai ser capaz de detectar erros nos seus códigos que você vai começar a criar agora. Ele com certeza será um dos seus melhores amigos na caminhada por aprender a programar em JavaScript.

Mas o que mais importa no momento é a parte de baixo da tela, chamada **Console**. Ele permite que possamos interagir com componentes do site aberto no momento.



Deixe o site do Google aberto, vá até a janela do DevTools, procure o Console e digite o código a seguir:

```
window.alert('Olá, Mundo!')
```

Depois aperte Enter e veja a magia acontecendo. Você conseguiu fazer o site do Google falar contigo! 😱

Achei meio bobo 😞. Tem como melhorar?

Você nunca está satisfeito(a) mesmo, não é? Pois agora você vai ser capaz de mudar o site do Google como nunca imaginou antes. Clique no botão Ok do alerta que apareceu para fechá-lo e volte para a sua janela de console. Dessa vez, o comando vai ser um pouco maior, mas seu efeito vai ser mais legal! Confia em mim!

```
document.body.style.backgroundColor = 'black'
```

Eu não preciso te ensinar JS agora, mas só de olhar o comando acima e você vai perceber que praticamente vai conseguir prever o que vai acontecer. Basta ter um conhecimento básico de Inglês. E se você não conseguiu entender, não se desespere! Isso vai melhorar com o tempo.



IMPORTANTE: Em JavaScript, as letras maiúsculas e minúsculas fazem toda a diferença! Por exemplo, na linha acima, perceba que existe uma única letra C em maiúsculas. Se você esquecer de colocar assim, vai dar erro! Tecnicamente, dizemos que JS é uma linguagem **case sensitive**. Já vai aprendendo os termos aí.

Você viu isso? O site do Google ficou preto! 😠

Mas não sai por aí dizendo que você hackeou o site do Google! Na verdade, você só alterou a configuração do código que foi enviado para o seu computador. Qualquer outra pessoa que esteja acessando o Google da sua casa, vai ver o site normal, com o fundo branco.

Por hoje é só, pessoal!

Paramos por aqui por enquanto, mas você não pode parar de aprender, não pode parar de praticar. Tire um tempo pra dar uma aprofundada nos seus conhecimentos antes de prosseguir, e pra isso eu já tenho uma indicação pra te fazer.

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para um vídeo que mostra o conteúdo que você acabou de ver aqui, mas de uma maneira mais aprofundada e com outro ponto de vista. Acesse agora mesmo!



Curso em Vídeo: <https://youtu.be/Ptbk2af68e8>

LTPW1

Capítulo 03

Primeiros passos com JavaScript

Pronto! Já vai começar a codificação que você está esperando tanto! Chegou a hora de colocar a mão na massa e criar os primeiros scripts em JS. E faremos do jeito tradicional: devagar e sempre. Ao utilizar esse método que eu prefiro aplicar às minhas aulas, veremos exercícios bem simples, com a aplicação de cada tag nova isoladamente. Tem gente que acha melhor começar com um projeto grande, mas em sala eu opto sempre por esse método. Vamos lá?



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicada - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidades de obter ganho financeiro com ele.



Você já tem o seu editor de código instalado?

Se por acaso você não possui um editor instalado, vamos recomendar o uso do **Visual Studio Code** da Microsoft no seu computador para acompanhar esse curso. Na verdade, nós já discutimos sobre editores no **Curso de HTML5+CSS3**, durante o material da **Aula 04: Primeiros Passos com HTML5**. Recomendo a leitura desse material antes de começar por aqui.



Como já falei anteriormente, você pode usar qualquer editor de código se a sua preferência for o Atom, Sublime, Brackets, Notepad++ ou qualquer outro, mas eu optei pelo uso do **VS code** para preparar esse curso.



APRENDA MAIS: Para acessar todo o material do Curso de HTML5 e CSS3, basta entrar no link a seguir e clicar no link para acessar o material em PDF do curso.

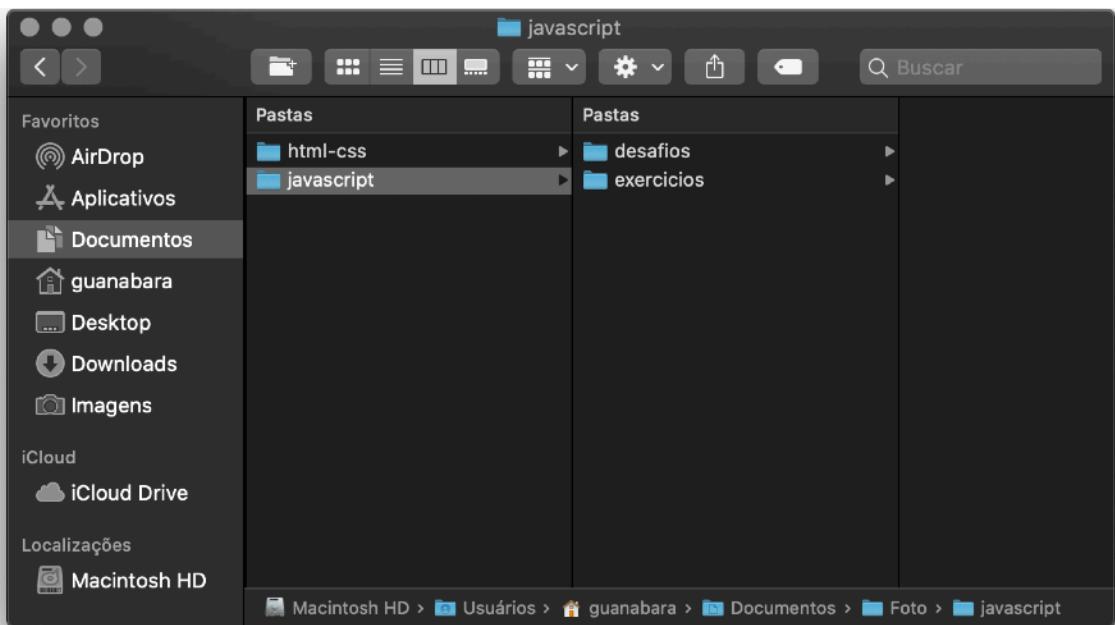
<https://gustavoguanabara.github.io>

Além de instalar o **VS code**, no PDF da **Aula 04 do Curso de HTML** eu te ensinei também a habilitar algumas configurações importantes do editor, como deixar a interface em Português, melhorar o suporte a JavaScript, ligar o Salvamento Automático, quebra automática de linhas e até mudar a interface do modo escuro para o modo claro.

Vamos organizar a nossa pasta de projeto

Durante o **Curso de HTML**, criamos uma pasta dentro dos **Meus Documentos** chamada **html-css**. Para acompanhar esse curso, vamos criar também uma pasta em **Meus Documentos** chamada **javascript** (em minúsculas, sem espaços).

Dentro da pasta **javascript** que você acabou de criar, crie também as pastas **desafios** e **exercícios**, da mesma maneira que você fez com o curso de HTML. A organização das pastas vai ficar exatamente igual ao que estamos apresentando na próxima imagem.



Agora já temos a nossa pasta de projeto **javascript**, vamos abri-la usando o **VS code** da mesma maneira que foi ensinado no outro curso (leia o material em PDF da aula 04 do curso de HTML).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
</body>
</html>
```

Veja na imagem acima que estou com a pasta **javascript** aberta, e dentro da pasta **exercicios** criei uma pasta chamada **ex001** e dentro dessa pasta, criei o arquivo **index.html**. Dentro desse documento, digitei uma exclamação ! e pressionei **Enter** para criar o código base em HTML.



IMPORTANTE: Se por acaso você estiver com a pasta **html-css** aberta e quiser fechá-la, basta acessar o menu do **VS code** e acessar a opção **Arquivo > Fechar Pasta**.

Nosso primeiro exercício JavaScript

Agora que você já criou seu primeiro exercício JavaScript, vamos ao código.

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport"
6          content="width=device-width, initial-scale=1.0">
7      <title>Primeiro JS</title>
8  </head>
9  <body>
10     <h1>Primeiro JavaScript</h1>
11     <script>
12         window.alert('Olá, Mundo!')
13     </script>
14 </body>
15 </html>
```

Note que a maneira mais simples de usar JS é incluindo o script em um documento HTML. Sendo assim, se você está fazendo o nosso **Curso de HTML** com a gente (recomendo muito), você já consegue identificar e classificar quase todas as linhas do documento acima. Apenas as **linhas 10, 11 e 12** são uma novidade!

Na recomendação atual, os scripts JS devem ser incluídos no final do corpo do documento HTML, posicionando a tag `<script></script>` logo acima do fechamento da tag `</body>`.

Tudo aquilo que estiver dentro da tag de script será considerado automaticamente JavaScript pelo seu navegador.

Nosso primeiro comando é

```
window.alert('Olá, Mundo!')
```

A palavra `window` (optional) é um objeto que indica uma referência à janela atual do navegador. A palavra `alert()` é um método capaz de disparar uma mini-janela de alerta para avisar algo ao usuário.

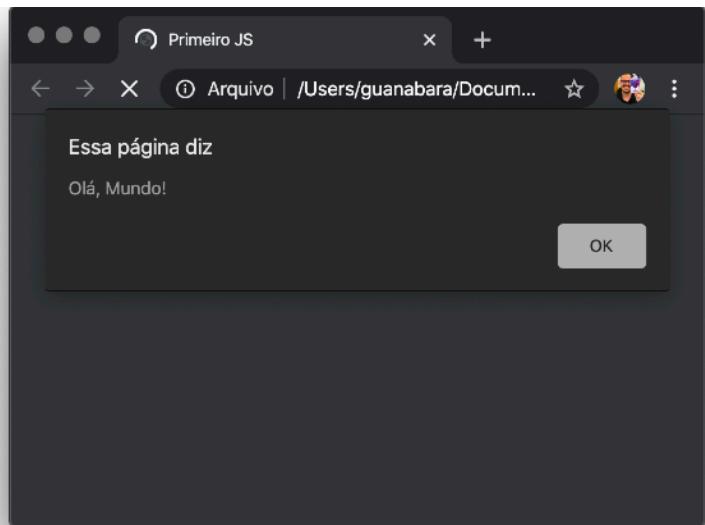
Dentro dos parênteses do método `alert()`, temos uma **string** entre aspas (que podem ser *'aspas simples'*, *"aspas duplas"* ou *'crases'* para delimitar as palavras que efetivamente aparecerão na janela de alerta).

Para executar nosso script, basta seguir o mesmo procedimento usado para abrir um arquivo HTML: vá até a pasta onde o documento está salvo (provavelmente Documentos/javascript/exercicios/ex001/, se você seguiu os passos), clique com o botão direito do mouse sobre o arquivo index.html e escolha a opção **Abrir com Google Chrome**.

Note que assim que o arquivo abre, a mensagem de alerta já aparecerá automaticamente, antes mesmo de exibir o título `<h1>` que foi colocado até mesmo antes da tag `<script>`.

Quando você apertar o botão OK do alerta, a página vai aparecer automaticamente.

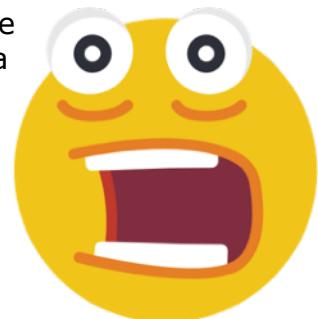
Eu sei que esse é um exercício simples, mas você precisa criar todos os passos na sua casa. É apenas com a prática que você vai poder dizer que aprendeu efetivamente. Sem a prática, nada feito!



Não funcionou? Não se desespere!

A dica que vou te dar agora vale ouro (mas você nem precisa me pagar por isso, relaxa!). Quando estamos começando a desenvolver coisas em JavaScript, é muito comum cometer pequenos erros de digitação que vão fazer nosso código simplesmente parar de funcionar.

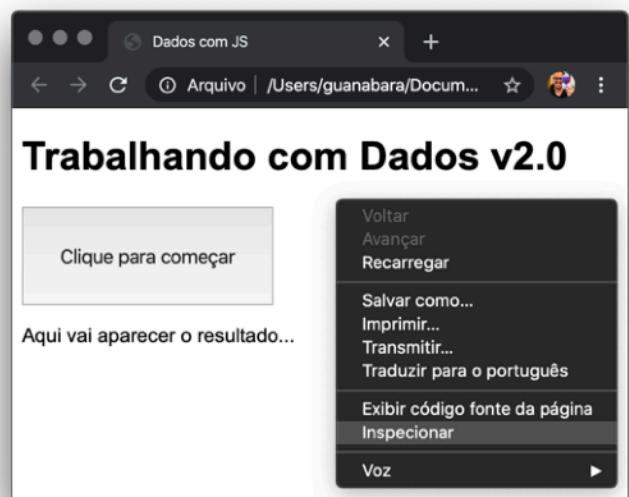
Dependendo do deslize, pode ser que uma única linha pare de funcionar ou até mesmo o script inteiro! Mas tem um jeito de usar o **Google Chrome** para ver o que aconteceu de errado.



Olha bem a **linha 11** do código que eu coloquei na página anterior. As palavras `window.alert` estão todas juntas, todas em minúsculas e não possuem nenhum espaço entre elas. Pois é EXATAMENTE assim que você tem que digitá-las! Qualquer pequeno deslize ou falha vai acarretar em um **ERRO** e seu script simplesmente não vai funcionar e a janela de alerta não vai aparecer!

Se algo do tipo acontecer com você, vá até o Chrome, clique com o botão direito do mouse sobre uma área em branco do fundo do seu site e escolha a opção **Inspecionar** (veja a tela ao lado). Outra forma é pressionando Ctrl + Alt + I no teclado ou clicar nos três pontos no canto superior direito da janela e escolher **Mais Ferramentas > Ferramentas do Desenvolvedor**.

Na janela que vai abrir, foque sua atenção no canto superior direito e veja o ícone vermelho que vai aparecer lá em cima.



```

<html lang="pt-br">
  <head>...</head>
  <body>
    <h1>Trabalhando com Dados v2.0</h1>
    <button onclick="inicio()">Clique para começar</button>
    <section id="resultado">
      <p>Aqui vai aparecer o resultado...</p>
    </section>
  </body>
</html>

```

Este ícone aqui ao lado só vai aparecer se você clicar no botão e tentar executar o código com erro no navegador.

Agora vamos para a parte de baixo da janela, na guia **Console**. Note que existe um erro em Inglês indicando qual foi o erro que você cometeu!

O erro também indica a linha aproximada onde ele foi encontrado que, mas pode se acalmar.

Ao notar que você cometeu esse erro, corrija-o, salve as alterações (mantenha sempre o Auto Save ligado) e volte a atualizar o navegador (geralmente apertando F5 ou Shift+F5).

Toda vez que você esperar uma ação do seu script e ela não acontecer, dê sempre uma olhada nessa janela de **DevTools**. Ela é uma das suas melhores amigas a partir de hoje!

IMPORTANTE! Essa janela **DevTools** não vai te mostrar falhas no seu código

HTML, apenas no seu código JavaScript, ok? Falhas ou deslizes em HTML simplesmente são ignorados pelo seu navegador. Sendo assim, se algum componente nas suas tags não estiver se comportando como você espera, simplesmente vá lá no seu código e verifique se sua tag está digitada corretamente. Não conte com o **DevTools** pra isso, infelizmente.

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!



Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyHQl1o1

LTPW1

Capítulo 04

Interações básicas com o usuário e com o navegador

A sua caminhada para aprender JavaScript já começou no capítulo anterior. Agora chegou a hora de criar uma interatividade com o usuário e também com os componentes exibidos no navegador. Vamos começar fazendo um botão reagir às nossas ações com ele e também vamos aprender a pedir dados para o usuário. Vamos nessa, que o trabalho não pode parar!

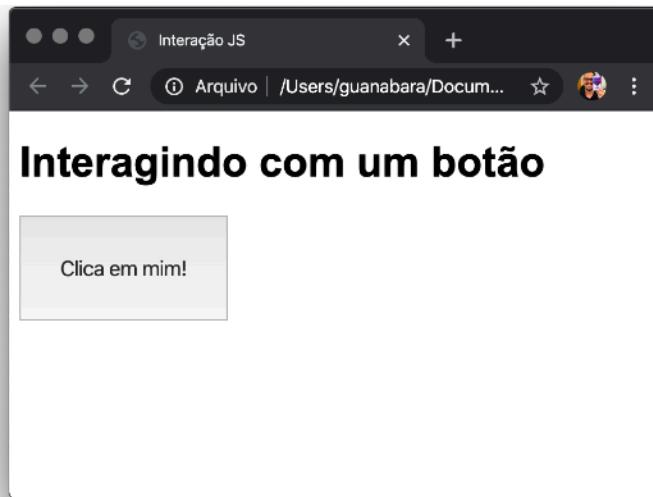


Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicado - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidade de obter ganho financeiro com ele.



Interagindo com um botão

A maneira mais simples que eu posso imaginar para que possamos interagir com uma página web é com um botão. Nosso objetivo aqui é criar uma página simples, com um botão sensível ao clique e que vai reagir de acordo com essa ação. Dá só uma olhada na imagem a seguir.



Vamos ver como isso foi criado e como ele reage ao clique. Crie uma pasta **ex002** dentro dos seus **exercícios**, adicione um arquivo **index.html** a essa pasta e digite o código a seguir. Em seguida, vamos analisar as linhas mais importantes.

```
1, <!DOCTYPE html>
2, <html lang="pt-br">
3, <head>
4,   <meta charset="UTF-8">
5,   <meta name="viewport"
6,     content="width=device-width, initial-scale=1.0">
7,   <title>Interação JS</title>
8,   <style>
9,     body { font: 12pt Arial; }
10,    button { font-size: 12pt; padding: 30px; }
11,   </style>
12, </head>
13, <body>
14,   <h1>Interagindo com um botão</h1>
15,   <button onclick="clicou()">Clica em mim!</button>
16,   <script>
17,     function clicou() {
18,       window.alert('Você clicou no botão!')
19,     }
20,   </script>
21, </body>
22, </html>
```

Vamos olhar com atenção a **linha 14**, onde criamos o botão. Existe um parâmetro na tag <button> que é o onclick. Note que dentro das aspas eu coloquei o nome de um **evento** que será tratado pelo JavaScript. A ligação entre o HTML e o JS será pelo nome desse evento (não esqueça dos parênteses no final, mais tarde você entenderá pra que eles servem).

Agora analise a **linha 16**. Ela é exatamente a definição da função que tratará do evento. Ao contrário do que fizemos no capítulo anterior (se esqueceu, retorna lá no PDF e confere), o window.alert() não está sozinho dentro do <script>. Ele está justamente dentro da função (identificada pela palavra function do JS).

Toda função em JS é relacionada a um **bloco**, que nada mais é do que um conjunto de comandos que estão entre chaves **{ }** . Tudo o que estiver entre chaves em JS, chamaremos de **bloco**.

Sendo assim, o comando que está na **linha 17** não vai executar assim que a página for carregada. No lugar disso, ela será executada só quando a função será disparada, e o modo de disparo nesse caso será feito na **linha 14**, pelo evento onclick do botão.



CÓDIGO NA MÃO: Todos os códigos dos exercícios já estão disponíveis no meu repositório público. Mas isso não significa que você deve copiar os códigos para “aprender mais rápido”. O objetivo aqui é ter o código dos exercícios sempre à mão em casos de emergência ou consulta rápida. Dá uma olhada também na pasta de **desafios**, pois lá você vai provar para si mesmo(a) que realmente aprendeu. Para acessar tudo isso, é só ir direto para o endereço a seguir:

<https://gustavoguanabara.github.io>

Interagindo com o usuário

Outra maneira de usar o JavaScript básico para criar interações é através do método prompt(). Com ele, podemos pedir para o usuário digitar dados e usar isso para causar uma resposta personalizada para ele.

Vamos começar criando uma pasta chamada **ex003** dentro da sua pasta de **exercícios** e criar um arquivo **index.html** dentro dela.

O código que virá a seguir é muito parecido com o anterior, na parte de HTML e CSS. Inclusive o botão, que na **linha 14** vai disparar o evento de clique, chamando a função inicio().

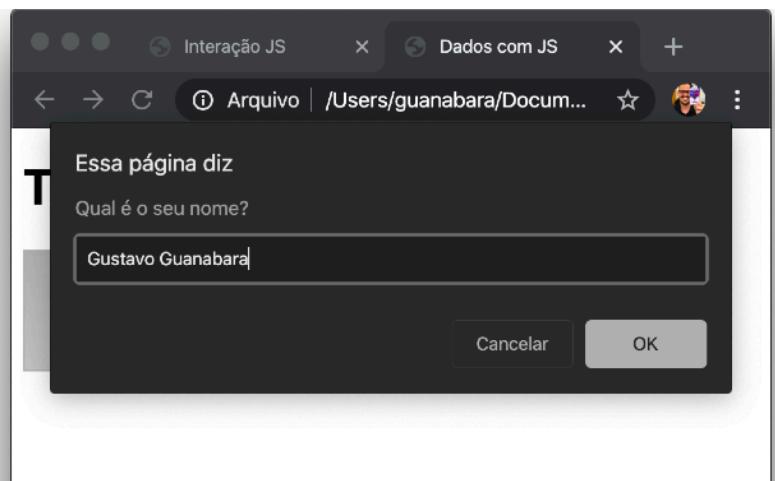
A grande novidade desse exercício está nas **linhas 17 e 18**, e é com elas que vamos nos preocupar especialmente.

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width,
6         initial-scale=1.0">
7     <title>Dados com JS</title>
8     <style>
9         body { font: 12pt Arial; }
10        button { font-size: 12pt; padding: 30px; }
11    </style>
12 </head>
13 <body>
14     <h1>Trabalhando com dados v1.0</h1>
15     <button onclick="inicio()">Clique para começar</button>
16     <script>
17         function inicio() {
18             let nome = window.prompt('Qual é o seu nome? ')
19             window.alert(`Olá, ${nome}! É um prazer te
20             conhecer!`)
21         }
22     </script>
23 </body>
24 </html>

```

A **linha 17** tem o método `window.prompt()`, que gera uma solicitação para que o usuário digite o seu nome. Essa janela se parece bastante com um alerta, mas tem a diferença de incluir uma caixa disponível para aceitar a digitação. Veja o resultado de um `prompt()` na imagem abaixo.



Mas como vamos guardar o nome do visitante? Áí entra o início da **linha 17**. A instrução `let nome` serve para declarar uma **variável** chamada `nome`, que vai guardar o nome que a pessoa vai digitar.

Em JavaScript, o símbolo de `=` não se lê como “igual”. Na verdade, sempre que você encontrar um `=`, leia como “recebe”.

Lendo então a **linha 17** depois de aprender tudo isso, ficamos com:

```
let nome = window.prompt('Qual é seu nome?')
```

“A variável `nome` vai receber o resultado de um `prompt` que vai aparecer na janela perguntando qual é o nome do usuário.”



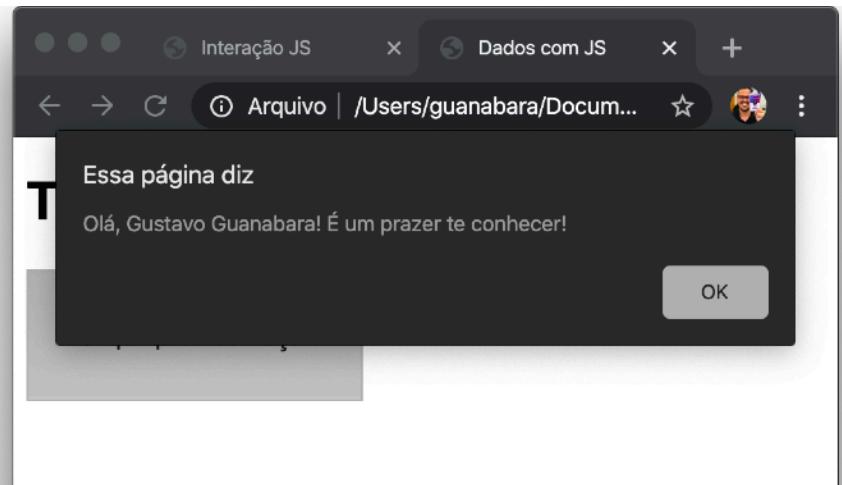
MAIS SOBRE VARIÁVEIS: Em JavaScript declaramos variáveis com as palavras `let` ou `var`. Basicamente, usando `var` nós podemos ter problemas com o escopo da variável (falaremos mais sobre isso na aula 07, mas por enquanto vamos só com `let` mesmo).

Outra coisa que podemos criar são variáveis imutáveis ou constantes. São variáveis que ficam na memória até o fim da execução do script, mas que não podem ter seu valor alterado de forma alguma. A declaração de uma constante é feita colocando a palavra `const` no lugar da palavra `let`.

Já na **linha 18**, temos também uma novidade. Note que dentro do `alert()`, usamos crases para delimitar a **string** dessa vez.

```
window.alert(`Olá, ${nome}! É um prazer te conhecer!`)
```

Uma **string** que está entre crases tem um nome especial: se chama **template string**. Esse tipo de string é uma das novidades do ECMAScript moderno e quebram um galho gigante!



Dentro de uma **template string** podemos usar **placeholders** no seu interior. Um placeholder é representado pelos símbolos `{}$` e podem ser usados para facilitar a exibição de conteúdos de variáveis ou expressões.

Olhando a **linha 18**, perceba que `${nome}` vai ser substituído pelo conteúdo da variável nome, criada na **linha 17** e que está guardando o nome da pessoa que está rodando o script.

Experimente fazer o **ex003** no seu computador e veja o resultado! Se precisar olhar o código original, não se esqueça de visitar o repositório público. Está tudo lá!

E começam os desafios!

Lá no repositório, além do material em PDF e dos códigos dos exercícios 100% disponíveis, também disponibilizamos alguns **desafios** que devem ser resolvidos. Esses desafios não incluem o código original e você deve tentar chegar à resposta sem copiar nenhum código.

Com todo o conteúdo que vimos até essa aula, você já pode resolver o **desafio d001 e d002**. Acesse o repositório público, abra a área do curso de JavaScript e clique no link de acesso aos desafios. Manda ver! Só não fica pedindo a resposta! Você consegue resolver isso sozinho(a)!



Repositório em: <https://gustavoguanabara.github.io>

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!



Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyHQl1o1

LTPW1

Capítulo 05

Interações básicas com componentes da página

No capítulo anterior, aprendemos uma maneira bem simples de interagir com o usuário usando os métodos `prompt()` e `alert()`. Agora chegou a hora de aprender com partes da sua página, utilizando alguns componentes do seu documento HTML5. Essa técnica que eu vou te ensinar agora se utiliza de um conceito maior chamado DOM, mas por enquanto não vamos entrar nesse assunto. Só confia no tio e vem!



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicada - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidade de obter ganho financeiro com ele.



Já vamos direto para um código de exemplo

Vou pular a parte de papo e conceitos e vou direto para a mão na massa nesse caso. Lá na sua pasta de projetos JavaScript, dentro de **exercícios**, crie uma pasta **ex004** e dentro dela um arquivo `index.html` com aquele código base que já fizemos algumas vezes. Modifique-o e adapte-o para que fique como apresentado a seguir:

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width,
6          initial-scale=1.0">
6  <title>Dados com JS</title>
7 </head>
8 <body>
9     <h1>Trabalhando com Dados v2.0</h1>
10    <button onclick="inicio()">Clique para começar</button>
11    <section id="resultado">
12        <p>Aqui vai aparecer o resultado... </p>
13    </section>
14 </body>
15 </html>
```

O nosso foco inicial será relembrar o código que escrevi lá na **linha 10**, onde criei um botão com uma chamada ao método `inicio()` quando for clicado pelo visitante. Isso já foi feito no capítulo anterior.

A novidade aqui está nas **linhas de 11 a 13**, onde criei uma seção identificada como `resultado`, com um parágrafo interno. Ele será o nosso elemento mais importante desse exercício.

Identificar os elementos que desejamos acessar via script é essencial para evitar dores de cabeça futuros. Quando identificamos o `<section>` com o nome `resultado`, deixamos as “portas abertas” para o JavaScript entrar.

Agora, para resolver esse exercício, vamos criar o método de resposta ao evento `onclick` do botão. Monte uma tag `<script></script>` na **linha 14**, logo depois do `</section>` e escreva o código a seguir. Não se preocupe se a confusão tomar conta da sua cabeça, logo logo eu vou explicar o script linha a linha.

```

14     <script>
15         function inicio() {
16             let nome = window.prompt('Qual é o seu nome?')
17             let res = window.document.getElementById('resultado')
18             res.innerHTML = `<p>Olá, <strong>${nome}</strong>! É
19             um grande prazer te conhecer! &#x1F596;` 
20     }
21 </script>

```

Dentro do script, entre as **linhas 15 e 19**, criamos a função `inicio()` e um bloco com {}. Dentro declaramos duas variáveis: `nome` e `res`. A variável `nome` vai receber o resultado de um `prompt()`. Até aí, nenhuma novidade.



IMPORTANTE: A qualquer momento, se você sentir alguma dificuldade nas linhas em que não explico com tantos detalhes, como a **linha 16** do código acima, talvez seja sinal de que você tenha pulado algum material anterior. A explicação mais aprofundada do uso do `prompt()` está no capítulo anterior.

Sendo assim, talvez seja melhor dar uns passos pra trás de vez em quando. Isso é super normal! Tentar aprender as coisas com pressa, geralmente causam resultados assim. E não se esqueça de praticar sempre! Só ler esse material, não adianta muita coisa. Foco total, mas sem perder a paciência! Você consegue!

A grande novidade aqui foi a **linha 17**. É nela onde vamos focar nossos esforços agora. Iniciando a linha, usamos a instrução `let` que vai criar uma variável chamada `res` (na verdade, o que é criado é um **objeto**). Mas por enquanto pense em um objeto como uma *variável especial*).

O objeto `res` será o nosso **elo** entre o JavaScript e o HTML. Lembra da **linha 11** do código HTML? Lá eu dei um `id` para o elemento `section` e agora vou tirar proveito disso. Só segue a linha 17 e tenta ler em Português:

"Na janela do navegador (window), dentro do documento atual (document), pegue o elemento que tenha o id (getElementById) com o nome 'resultado'."

É exatamente isso que você leu! A partir de agora, tudo aquilo que eu fizer com o objeto `res` em JavaScript, vai se refletir o objeto que tenha o `id` citado no código. Nesse código, se mexermos com `res`, vamos afetar diretamente a `<section>` do nosso documento HTML. Sacou?

Agora vamos olhar com atenção para a **linha 18** do código acima. Ela vai modificar o "HTML que está dentro" (`innerHTML`) da `section`. Dá só uma olhada na **linha 12** do nosso código HTML. Esse é o parágrafo que vai aparecer no navegador quando abrirmos o `index.html`. Porém, a **linha 18** vai mudar esse parágrafo quando o botão da **linha 10** for clicado.

Para facilitar a programação, usei uma *template string* (já estudamos ela nos capítulos anteriores) e incluí um *placeholder* \${nome} para personalizar a mensagem. No fim da linha, ainda coloquei um código do emoji  para a mensagem ficar mais simpática.



EMOJI? Se você não entendeu como o código 🖖 virou uma mãozinha, recomendo que abra nosso material de HTML e vá para o PDF da **Aula 05**. Lá eu explico como descobrir o código de qualquer emoji existente.

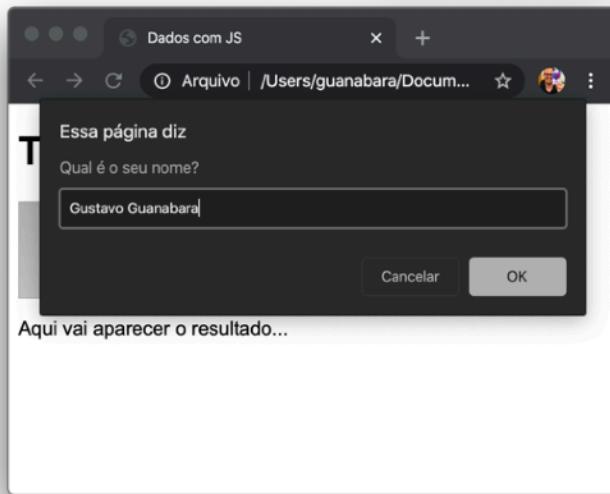
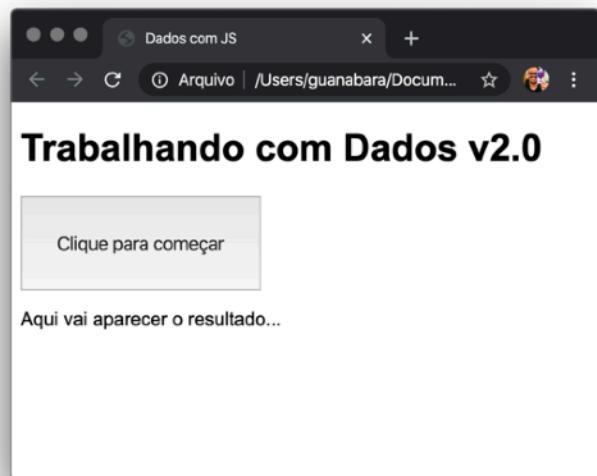
<https://gustavoguanabara.github.io>

Agora vamos fazer tudo funcionar

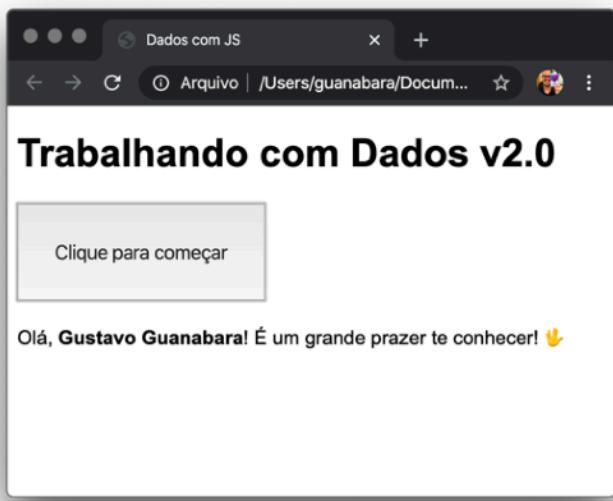
Vamos abrir o nosso arquivo index.html da pasta **ex004** e ver o funcionamento do script.

Logo no início, o parágrafo da seção que está logo abaixo do botão vai apresentar o texto padrão que configuramos no documento HTML. Talvez na sua casa, o botão fique um pouco menor, pois fiz umas configurações adicionais em CSS.

Ao clicar no botão, um prompt() será disparado, pedindo que o visitante digite o seu nome.



Por fim, depois de apertar o botão OK que fica na janela do prompt(), o parágrafo vai ser substituído pela mensagem de boas-vindas.



SUGESTÃO: Se o seu programa funcionou corretamente, meus parabéns! Só sugiro que faça uma pequena alteração na **linha 18** e veja a diferença. Substitua o símbolo de `=` por `+=` (sem espaços) e perceba a diferença! Conseguiu entender? Mais pra frente, falamos melhor sobre isso.

Não funcionou? Não se desespere!

Eu sei que já falamos sobre isso no capítulo 03, mas como dessa vez tivemos que escrever um código bem maior, acho importante repetir o conselho. Quando estamos começando a desenvolver coisas em JavaScript, é muito comum cometer pequenos erros de digitação que vão fazer nosso código simplesmente parar de funcionar.

Dependendo do deslize, pode ser que uma única linha pare de funcionar ou até mesmo o script inteiro! Mas tem um jeito de usar o **Google Chrome** para ver o que aconteceu de errado.



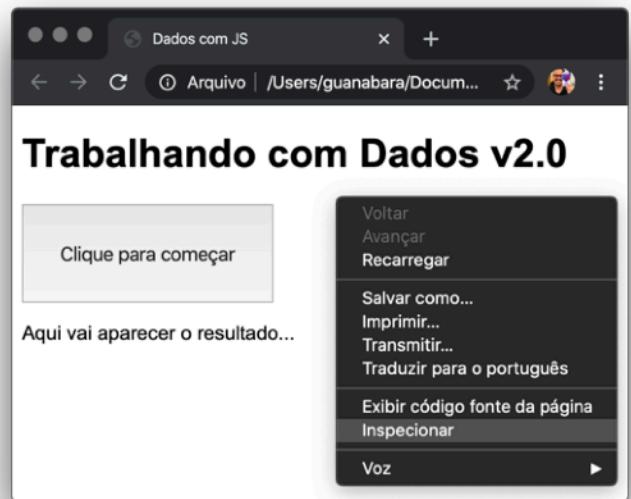
Vamos ver um erro muito comum que os iniciantes acabam cometendo. Analise a linha a seguir.

```
17 | let res = window.document.getElementById('resultado')
```

Mudei a **linha 17** do código, escrevendo `getElementById()` completamente em letras minúsculas (o que é um erro). Ao clicar no botão, o script pede o nome do visitante, mas não é capaz de mostrar o resultado na tela.

Se algo do tipo acontecer com você, vá até o Chrome, clique com o botão direito do mouse sobre uma área em branco do fundo do seu site e escolha a opção **Inspecionar** (veja a tela ao lado). Outra forma é pressionando Ctrl + Alt + I no teclado ou clicar nos três pontos no canto superior direito da janela e escolher **Mais Ferramentas > Ferramentas do Desenvolvedor**.

Na janela que vai abrir, foque sua atenção no canto superior direito e veja o ícone vermelho que vai aparecer lá em cima.



Este ícone aqui ao lado só vai aparecer se você clicar no botão e tentar executar o código com erro no navegador.

Agora vamos para a parte de baixo da janela, na guia **Console**. Note que existe um erro dizendo (em Inglês) que "window.document.getelementbyid não é uma função em inicio()". Inclusive ele me indicou que o erro apareceu na linha 17 do arquivo index.html.

O erro também indica que algo aconteceu na linha 10, mas pode se acalmar. Ele apenas está dizendo que tudo se originou quando você clicou no botão (que está na linha 10).

Ao notar que você cometeu esse erro, corrija-o, salve as alterações (mantenha sempre o Auto Save ligado) e volte a atualizar o navegador (geralmente apertando F5 ou Shift+F5).

Não se esqueça: a janela **DevTools** do Google Chrome é uma das suas melhores amigas! #BFF

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!



Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyHQl1o1

LTPW1

Capítulo 06

Fazendo umas contas com JS

Você é daquele time de pessoas que odeia Matemática, ou é dos meus e adora essa Ciência maravilhosa? Independente do lado que você esteja, com uma coisa temos que concordar: fazer cálculos simples faz parte do nosso dia-a-dia! Coisas primordiais como calcular uma média, fazer a conta do troco da padaria ou quantos anos ainda faltam pra você se aposentar (não faça essa com tanta frequência, senão vai acabar ficando triste). Nesse capítulo, vamos ver como podemos fazer conta em nossos scripts.



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicada - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidade de obter ganho financeiro com ele.



A Matemática é como o cachorro do meu vizinho

Calma! Não estou ofendendo o meu vizinho chamando de cachorro! O que estou querendo explicar é que meu vizinho tem um cachorro daqueles bem pequenos e que acha que é um cão gigante. Já viu um desses?

Quando eu vou sair de casa para ir dar aula e o cachorro me vê, começa a latir raivosamente e mostrar os dentes pra mim. É como se ele estivesse querendo me dizer: "*chega mais perto, e eu arranco o seu braço na dentada*".



Acontece que eu sei que ele não pode me fazer mal. Aí falo o nome dele, me aproximo, abixo e coloco a mão na sua cabeça. Ele imediatamente começa a abanar o rabo e às vezes até deita no chão de tanta alegria.

Matemática também é assim. Ela assusta às vezes. Se você se intimidar ela fica assustadora. Mas se você enfrentar o desafio, ela vai ser dócil contigo! Pode acreditar em mim!

Operador de Atribuição (parte 1)

A coisa mais simples que você pode fazer em JS é a atribuição. Atribuir significa jogar um valor ou o resultado de uma expressão para dentro de uma variável. Vamos a um exemplo bem simples:

```
1 let n1 = 4  
2 let n2 = 8  
3 let n3 = n1 + n2
```

As três linhas ao lado possuem o mesmo operador de atribuição, que é o símbolo de `=`. Vamos fazer um acordo aqui: sempre que você encontrar um símbolo desses no código, vai ler como se fosse a palavra **"recebe"**.

Sendo assim, vamos ler as linhas assim:

- 1 - "a variável `n1` **recebe** o valor `4`"
- 2 - "a variável `n2` **recebe** o valor `8`"
- 3 - "a variável `n3` **recebe** o resultado da expressão `n1+n2` (o valor `12`)"



E O SINAL DE IGUAL? Você pode estar se perguntando: se apenas um sinal `=` significa "recebe", como vamos representar o operador de "igual"? A resposta é simples: Em JavaScript, o "igual" é representado como `==` (dois sinais na sequência, sem espaço entre eles) ou como `===` (três sinais consecutivos, ou igualdade estrita).

Ainda existem outros operadores de atribuição, mas veremos isso mais tarde na parte 2 desse assunto, combinado?

Operadores Aritméticos

Os operadores desse tipo servem para fazer as contas básicas que estamos acostumados no nosso dia-a-dia:

Operador	Função	Exemplo	Resultado
()	Agrupamento	$(5+3)/2$	4
+	Positivo unário	+5	5
-	Negativo inário	-4	-4
**	Exponenciação	$5 ** 2$	25
*	Multiplicação	$3 * 2$	6
/	Divisão	$10 / 2$	5
%	Resto da divisão	$5 \% 2$	1
+	Adição Binária	$5 + 3$	8
-	Subtração Binária	$8 - 5$	3

Ordem de precedência

O primeiro operador (Agrupamento) não é exatamente um “operador aritmético”, mas adicionei aqui na lista para efeitos práticos. Usamos muito eles para forçar a execução de uma operação antes de outra. No exemplo que demos, se resolvermos $5+3/2$ o resultado será 6.5, já que em uma expressão aritmética com soma binária e divisão, resolvemos primeiro a divisão. Chamamos isso de **ordem de precedência**.

Como você já deve ter percebido, essa tal de “ordem de precedência” indica quais operadores serão executados primeiro. Nem tudo é resolvido na ordem em que aparece na expressão. Os que estão acima são mais poderosos. Nos lugares onde aparecerem o símbolo [os operadores possuem a mesma ordem e são poderosos da mesma maneira, sendo assim devem ser resolvidos da esquerda para a direita. Sempre que você se deparar com uma expressão aritmética, consulte a tabela acima. Se preferir, pode imprimir essa página, recortar a tabela e colar em um lugar de destaque.

Vejamos um exemplo, comparando a expressão $5+3/2$ com a expressão $(5+3)/2$:

$$5 + \boxed{3 / 2} \quad \boxed{(5 + 3)} / 2$$

No primeiro caso, quando não coloquei parênteses, a ordem de operação será para aquele operador onde a precedência vem antes, no caso a divisão, que resolveremos primeiro. Dessa maneira, o resultado da expressão à esquerda seria 6.5

Já na expressão da direita existem os parênteses - que possuem a precedência mais alta - e começaremos por eles, realizando a soma antes e depois a divisão. O resultado da segunda expressão, seria então 4.



DICA: Analise os operadores em uma expressão, como se fosse uma fila de banco. Não importa muito a ordem visual da fila, os idosos serão atendidos primeiro. Eles têm preferência! Na tabela, quem tem mais preferência são os operadores que estão mais pra cima. Os Operadores que estão ligados pelo sinal de [possuem a mesma preferência, assim quem será atendido primeiro é quem estiver à esquerda.

Não entendi os operadores % e **

Normalmente meus alunos não entendem de cara o significado do operador %. "Mas isso não é pra calcular porcentagem?", eles perguntam.

O símbolo % não tem essa finalidade, assim como o x não calcula a multiplicação. Programação tem suas particularidades, já vai se acostumando!

Vamos começar com um cálculo simples: 134 % 5. Armei a conta aqui embaixo e vamos resolvê-la até encontrar o valor inteiro da divisão (sem usar a vírgula).

$$\begin{array}{r} 134 \\ 34 \longdiv{134} \\ \hline 26 \\ 4 \end{array}$$

Quando chegamos ao resto 4, só conseguimos continuar a divisão colocando um zero a mais ao lado do resto e uma vírgula no final do 26, não é? Não queremos isso, pois estamos buscando pela **divisão inteira**.

Sendo assim, $134 \% 5 = 4$. Sacou?

Agora vamos nos focar no operador **, que mais parece que estou multiplicando duas vezes. Mas não é bem assim. Esse operador na verdade calcula o valor de uma exponenciação ou **potência**.

Por exemplo, se quisermos calcular 12^2 , basta escrever `12 ** 2` e o motor JavaScript vai entender.

Agora que você já entendeu o funcionamento de cada operador aritmético do JavaScript, vamos colocar um pouco a mão na massa para treinar a sua aplicação em expressões mais complexas.

Vamos praticar um pouquinho?

Vamos ver se você entendeu o que é ordem de precedência. Fique sempre de olho na tabela da página anterior e marque quem vem primeiro em cada expressão antes de resolvê-la de fato. Eu facilitei e já fiz o primeiro pra você. As próximas são contigo!

$$8 * 3 + 2 / 2 = \boxed{25}$$

$$1 + 3 ** 2 / 9 = \boxed{}$$

$$1 + 3 ** 2 / 9 = \boxed{}$$

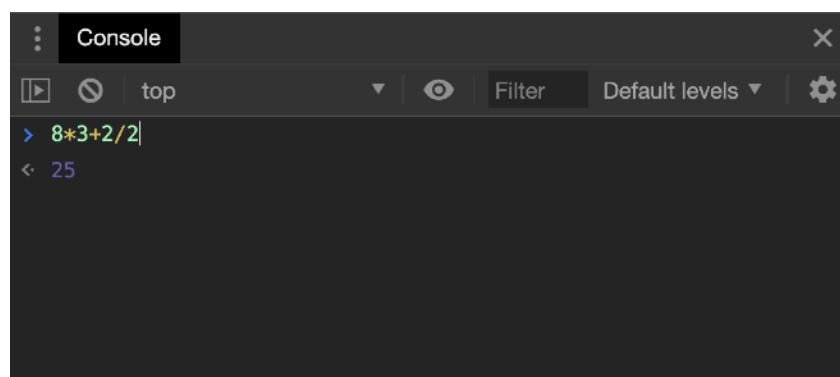
$$-3 + 4 * (10 \% 4) = \boxed{}$$

$$(8 - 2 * 3) \% 1 * 2 = \boxed{}$$

Quer saber se suas contas estão certas?

Você se lembra quantas vezes eu te disse que o **DevTools** do Google Chrome é seu melhor amigo? Pois ele vai te ajudar de novo!

Abra no seu Google Chrome as **Ferramentas do Desenvolvedor** do jeitinho que eu te ensinei no **Capítulo 03**, na seção "Não funcionou? Não se desespere!", ou simplesmente aperte Ctrl+Alt+I. Vá até a parte do **Console** e digite qualquer fórmula diretamente.



Viu? Confia no **DevTools**! Ele está sempre ao seu lado! No momento em que você achar que as coisas pararam de funcionar ou sentir que algo deu errado, é só abrir essa janelinha mágica e ela quase sempre te dará uma resposta, ou pelo menos uma dica que te levará ao resultado.

Operadores de Atribuição (parte 2)

Agora que você já conhece os operadores aritméticos básicos, podemos combiná-los aos operadores de atribuição para realizar tarefas que são muito comuns. Por exemplo, uma expressão que faremos com muita frequência é a operação de acumulador. Algo como `soma = soma + num`. Podemos simplificá-la usando operadores de simplificação aritmética:

Operador	Exemplo	Equivale a...
<code>+=</code>	<code>a += b</code>	<code>a = a + b</code>
<code>-=</code>	<code>a -= b</code>	<code>a = a - b</code>
<code>*=</code>	<code>a *= b</code>	<code>a = a * b</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>**=</code>	<code>a **= b</code>	<code>a = a ** b</code>
<code>%=</code>	<code>a %= b</code>	<code>a = a % b</code>

Pode ser que você não tenha entendido muito bem esses operadores agora. Não se preocupe! Mais pra frente voltamos neles e tudo ficará mais claro.

Operadores de incremento/ decremento

Ainda nos mantendo no mesmo assunto de simplificar expressões muito comuns usadas no dia-a-dia do programador. Outro tipo de expressão bastante usada é o incremento (somar uma unidade a uma variável) ou decremento (diminuir uma unidade a uma variável).

Operador	Exemplo	Equivale a...	Poderia usar
<code>++</code>	<code>x ++</code>	<code>x = x + 1</code>	<code>x += 1</code>
<code>--</code>	<code>x --</code>	<code>x = x - 1</code>	<code>x -= 1</code>

E se eu precisar fazer umas contas mais complexas?

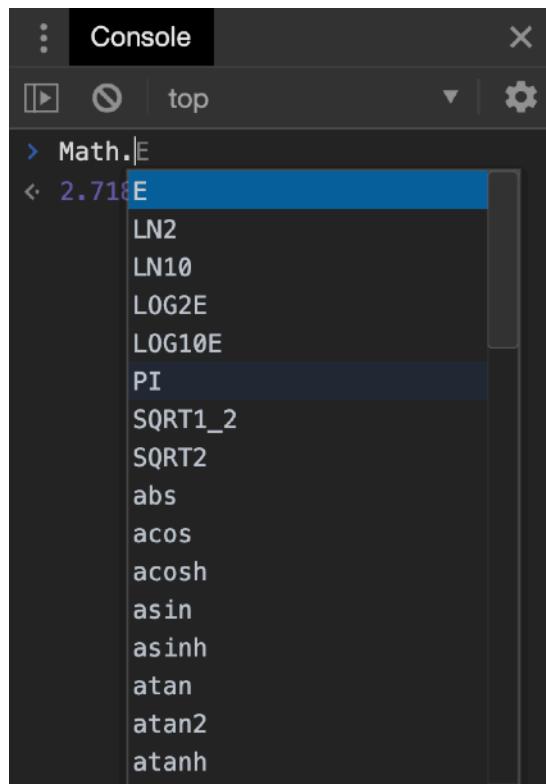
Com o uso dos operadores aritméticos e um bom conhecimento de Matemática já é possível resolver quase tudo. Porém, nem sempre queremos ficar estudando teoremas e correndo atrás de resolver tudo manualmente.

Pensando nisso, o JavaScript tem um objeto global especial chamado **Math** que possui vários métodos e atributos internos para operações Matemáticas (como o próprio nome já sugere).

Aproveita que o seu **DevTools** já deve estar aberto, vai até a janela de **Console** e digita Math. (não se esqueça de colocar a letra M maiúscula e o ponto logo depois, sem espaço nenhum).

Olha só ao lado a quantidade de métodos e atributos dentro do objeto **Math** e que podem ser utilizados em seus códigos a partir de agora.

Vou colocar aqui algumas coisas que podem ser feitas usando **Math**, usando alguns exemplos de comandos que você pode digitar no Console também, se quiser ver o resultado (só não se esqueça que o JS diferencia maiúsculas e minúsculas):



The screenshot shows the DevTools Console with the search bar containing 'Math.'. Below it, a list of properties and methods is displayed, starting with 'PI' which is highlighted in blue. Other items include LN2, LN10, LOG2E, LOG10E, SQRT1_2, SQRT2, abs, acos, acosh, asin, asinh, atan, atan2, and atanh.

Digite isso...	... o resultado será
Math.PI	O valor de PI com precisão de 15 dígitos
Math.abs(-200)	O módulo ou valor absoluto do número
Math.sqrt(81)	A raiz quadrada do número informado
Math.cbrt(27)	A raiz cúbica do número informado
Math.ceil(12.2)	Arredonda pra cima o número informado
Math.floor(15.7)	Arredonda pra baixo o número
Math.round(18.5)	Faz o arredondamento aritmético tradicional
Math.trunc(22.9234)	Trunca o valor, eliminando toda a parte fracionária
Math.min(5, 9, 2, 6, 1, 4)	Indica qual foi o menor valor informado
Math.max(5, 9, 2, 6, 1, 4)	Indica qual foi o maior valor informado
Math.random()	Gera um número aleatório entre 0 e 1



REFERÊNCIA COMPLETA: Para uma lista completa, consulte o manual oficial da **Fundação Mozilla** no endereço abaixo:

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

E o cálculo de porcentagem?

Lá na capa desse capítulo eu perguntei se você é do tipo de pessoa que odeia ou adora Matemática. Se você foi do segundo grupo, não precisa se preocupar com essa sessão. Agora, se você é daqueles que não curte a ciência dos números, provavelmente fica travado(a) quando o assunto é porcentagem.



RESUMINDO E SIMPLIFICANDO: De forma fácil, uma porcentagem é uma **razão com denominador 100** (por isso se chama "por-100-to"). Ele calcula uma medida proporcional de um valor em relação ao "todo" ou 100%.

A maneira mais simples é resolver usando uma **regra de três** simples. Vamos ver dois exemplos:

EXEMPLO 01: Uma loja de tecnologia vende um computador por **R\$1850,00**. Na época da Black Friday, o dono da loja autorizou um desconto de 20% no preço do PC para aumentar as vendas e liberar estoque. Qual será o valor do PC na Black Friday?

EXEMPLO 02: José é um ótimo funcionário. Chega sempre na hora certa e quase não falta ao trabalho. Seu salário é de **R\$3480,00** há 2 anos. No próximo mês, o chefe do José resolveu dar um aumento a ele e disse que seu salário vai subir 13%. Qual será o valor do novo salário do José?

Nos dois casos, são citados valores percentuais (20% de desconto no primeiro caso e 13% de aumento no segundo). Vamos usar como base o primeiro exemplo:

O pensamento é o seguinte: o PC custa R\$1850,00, e isso corresponde a 100% do preço. Quanto será que vai custar (x) 20% desse valor?

$$\begin{array}{rcl} \text{R\$ } 1850 & \xrightarrow{\quad\quad\quad} & 100\% \\ \text{R\$ } x & \xrightarrow{\quad\quad\quad} & 20\% \end{array}$$

Vamos resolver essa regra de três multiplicando apenas o valor, de forma cruzada:

$$x \cdot 100 = 1850 \cdot 20$$

Agora vamos deixar apenas x em evidência do lado esquerdo da expressão, passando o 100 para o lado direito, que deixa de multiplicar e passa a dividir:

$$x = \frac{1850 \cdot 20}{100}$$

Então a fórmula acima calcula quanto é 20% de R\$1850,00. Aí está a nossa **razão com denominador 100** que estávamos procurando.

Agora podemos calcular o novo preço do produto, baseado nos 20% calculados:

novo = 1850 - x

De forma similar, podemos calcular os 13% do salário do José:

$$x = \frac{3480 \cdot 13}{100}$$

Viu como é simples? Agora podemos calcular o novo salário:

novo = 3480 + x

Note que dependendo do tipo de reajuste, usamos o operador mais coerente para cada caso. No exemplo 1, tínhamos um DESCONTO, logo subtraímos o valor de x. Já no exemplo 2, tínhamos um AUMENTO, e por isso somamos o valor de x.

Se quisermos transformar isso tudo em um código simples em JavaScript, podemos escrever:

EXEMPLO 01 em JavaScript

```
1 let preço = 1850
2 let porc = 20
3 let valor = (preço * porc) / 100
4 let novo = preço - valor
```

EXEMPLO 02 em JavaScript

```
1 let sal = 3480
2 let porc = 13
3 let valor = (sal * porc) / 100
4 let novo = sal + valor
```



NÃO ESQUEÇA: Durante a programação, o símbolo de % usado no código serve para calcular o resto da divisão inteira, não tem nada a ver com a porcentagem.

Muitos exercícios

Agora chegou a hora de praticar MUITO. Acesse agora mesmo o endereço <https://gustavoguanabara.github.io/javascript/exercicios/> e execute os **exercícios 005, 006, 007, 008 e 009**.

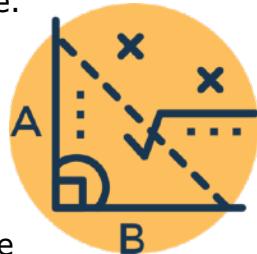
Depois crie as pastas **ex005, ex006, ex007, ex008 e ex009** no seu computador e tente fazer esses mesmos exercícios sem copiar o código que eu criei. Nesse momento, a prática é algo que você mais precisa. Se por acaso ficar difícil, pode acessar o repositório público de HTML e CSS e dar uma olhada nos comandos, mas **EVITE COPIAR**.



Por enquanto é isso

Espero que esse capítulo tenha deixado tudo bem claro pra você. Guarde essa parte do seu curso com carinho e se preferir, pode até imprimir o capítulo inteiro para quando for precisar fazer uma consulta rápida.

Matemática **não** é um bicho de sete cabeças. Treine bastante no **DevTools** do seu Google Chrome e experimente criar expressões que você está acostumado(a) no seu dia-a-dia. Não deixe também de fazer os exercícios do número 005 até o 009.



Tenho uns desafios pra você!

Lá no repositório, além do material em PDF e dos códigos dos exercícios 100% disponíveis, também disponibilizamos alguns **desafios** que devem ser resolvidos. Esses desafios não incluem o código original e você deve tentar chegar à resposta sem copiar nenhum código.

Com todo o conteúdo que vimos até essa aula, você já pode resolver os seguintes desafios:



- **d003**: Antecessor e sucessor
- **d004**: Calculando o troco
- **d005**: Conversor de medidas
- **d006**: Conversor de temperatura
- **d007**: Conversor de moedas
- **d008**: Calculando desconto de 10%
- **d009**: Aumentando o salário
- **d010**: Resolvendo Bhaskara

Acesse o repositório público, abra a área do curso de JavaScript e clique no link de acesso aos desafios. Manda ver! Só não fica pedindo a resposta! Você consegue resolver isso sozinho(a)!

Repositório em: <https://gustavoguanabara.github.io>

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!



Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyH011o1

LTPW1

Capítulo 07

Múltiplas ações, variáveis e conteúdo dinâmico

Três assuntos em um mesmo capítulo? Pois é. Estamos avançando no JavaScript e precisamos ampliar nossos horizontes um pouquinho. Preciso te mostrar alguns conceitos importantes para prosseguirmos e para que nossos scripts fiquem mais interessantes. Vá com calma, se por acaso estiver cansado(a), faça uma pausa, descansa um pouco antes de continuar. Afinal, não adianta tentar correr para aprender programação. Paciência, dedicação e prática: essa é a receita para o sucesso!



Você tem todo o direito de usar esse material para seu próprio aprendizado. Professores também podem ter acesso a todo o conteúdo e usá-lo com seus alunos. Porém todos que usarem esse material - seja para qual for a finalidade - deverão manter a referência ao material original, criado pelo **Prof. Gustavo Guanabara** e disponível no endereço do seu repositório público <https://github.com/gustavoguanabara/>. Este material não poderá ser utilizado em nenhuma hipótese para ser replicada - integral ou parcialmente - por autores/editoras para criar livros ou apostilas, com finalidade de obter ganho financeiro com ele.



Vamos avançar um pouco com as variáveis

No capítulo 4, abordamos o conceito inicial de variáveis. Agora vamos nos aprofundar um pouco mais. Como você já deve saber, uma variável é um espaço na memória do computador, celular, tablet, TV, vídeo-game ou qualquer dispositivo que você use para acessar um site ou rodar um script JS.

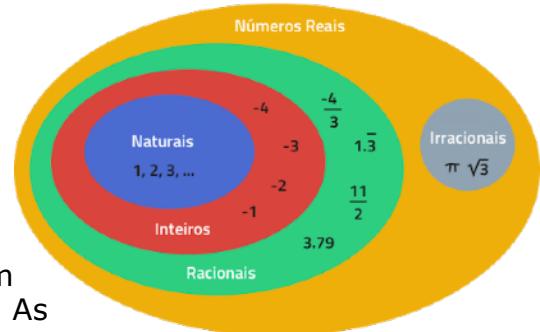


IMPORTANTE: Provavelmente você está estudando hardware, arquitetura de computadores ou qualquer matéria similar. Pois saiba que todas as variáveis são armazenadas na **memória principal** do dispositivo (geralmente memória do tipo RAM).

Quando usamos comandos do tipo:

```
var a = 10  
let b = 'JavaScript'  
const c = 3.1415
```

A primeira linha cria uma variável chamada **a**, que guarda o **número inteiro** 10. A segunda linha também cria uma variável chamada **b**, que guarda uma **cadeia de caracteres** (também conhecida como **String**) com o valor JavaScript. As aspas (simples, duplas ou crases) não fazem parte da String, apenas a delimitam.



Já a terceira linha declara uma **variável imutável** (também chamada de constante) com um **número real** com o valor 3.1415. Veja na imagem ao lado a relação existente entre números inteiros (\mathbb{Z}) e reais (\mathbb{R}).



RELEMBRANDO: Você deve se lembrar da época das suas aulas iniciais de Matemática, que **números inteiros** são aqueles que não possuem a vírgula (ponto flutuante). Já os **números reais** são aqueles que possuem parte fracionária separada por vírgula.

No JavaScript moderno, a distinção entre números inteiros e real está bem simplificada com o uso da função interna `Number()`. Antigamente usávamos bastante as funções `parseInt()` e `parseFloat()` para realizar a conversão para cada tipo específico.

Descobrindo o tipo de uma variável ou valor

A qualquer momento, podemos descobrir o tipo de uma variável ou valor, podemos usar a instrução `typeof`. Considere as linhas de exemplo acima e vamos analisar o tipo de cada uma delas através de comandos simples:

```

typeof a          // vai retornar "number"
typeof b          // vai retornar "string"
typeof c          // vai retornar "number"

```

Você pode usar o Console do seu DevTools para fazer esses testes. Inclusive, pode experimentar testes diretamente com valores ou expressões:

```

typeof 250        // vai retornar "number"
typeof '800'       // vai retornar "string"
typeof true         // vai retornar "boolean"
typeof 'false'      // vai retornar "string"
typeof 200 * 5      // vai retornar "NaN"
typeof (500 / 3)    // vai retornar "number"

```

Na lista acima, temos algumas curiosidades:

- O valor 250 é considerado **number**, mas o valor '800' é uma **string**! Isso acontece porque o 800 está entre aspas e mesmo parecendo ser um número para nós, humanos, um computador sempre vai considerar qualquer coisa delimitada por aspas (simples, duplas ou crases) como uma cadeia de caracteres.
- O valor true é um valor lógico (só aceita verdadeiro ou falso), conhecidos pelo JS como o tipo **boolean**. A mesma linha de raciocínio feita anteriormente serve para o valor 'false' testado logo em seguida. Mesmo parecendo um valor booleano, ele está entre aspas e vai ser considerado uma string.
- A penúltima linha retorna um termo que é novidade pra nós. Para o JavaScript, NaN significa NÃO É UM NÚMERO (*Not a Number*). O que a linguagem quer te dizer é que provavelmente você está tentando fazer uma conta ou mostrar um número, mas ele está gerando um resultado que não é numérico e ele não está conseguindo resolver.
- Já na última linha, a expressão pode ser resolvida e o resultado é um número. Sendo assim, se você quiser testar uma expressão usando typeof, coloque-a entre parênteses, ok?

Múltiplas ações em JavaScript

Até o momento, o que criamos aqui foram ações simples executadas em resposta a apenas um evento por vez. Somente um botão estava sendo monitorado. Agora vamos ver como podemos monitorar várias ações em um só código.

Abra seu diretório de exercícios JavaScript e crie uma pasta **ex010**. Nele, crie um arquivo chamado index.html e adicione as linhas básicas de um documento HTML5.

```

9   <body>
10  <h1>JavaScript Externo</h1>
11  <button onclick="acao1()">Ação 1</button>
12  <button onclick="acao2()">Ação 2</button>
13  <button onclick="acao3()">Ação 3</button>
14  <button onclick="acao4()">Ação 4</button>
15  <section id="saida">
16    <p>Aqui vou registrar suas ações com os botões acima.</p>
17  </section>
18  <script src="acoes.js"></script>
19 </body>

```

Note que nas **linhas de 11 até 14**, criamos quatro botões diferentes e que cada um dispara uma ação diferente e não uma única funcionalidade.

Outra coisa que estamos fazendo pela primeira vez é a **linha 18**. Com ela, estamos evitando colocar um código JavaScript dentro de um arquivo HTML. Dessa maneira, nosso código fica muito mais enxuto e organizado, já que estamos separando o que é HTML do que é JS. Ao digitar a linha 18, segure a tecla Ctrl e clique sobre o nome do arquivo acoes.js (que até o momento não existe). Ao fazer isso, disparamos um pedido ao VS Code para que ele crie o arquivo automaticamente!



ANOTE AÍ: Sempre que você tiver uma ligação com outro arquivo externo em nossos código atual, basta usar o atalho Ctrl+clique em cima do nome do arquivo. Se o arquivo existir, o VS Code vai tentar abri-lo. Se ele não existir, vai tentar criá-lo. Essa é uma **MEGA DICA!**

No documento acoes.js que criamos, escreva apenas o código das funções.

```
1 function acao1() {  
2     let resp = window.document.getElementById('saida')  
3     resp.innerHTML += '<p>Clicou no primeiro botão</p>'  
4 }  
5  
6 function acao2() {  
7     let resp = window.document.getElementById('saida')  
8     resp.innerHTML += '<p>Clicou no segundo botão</p>'  
9 }  
10  
11 function acao3() {  
12     let resp = window.document.getElementById('saida')  
13     resp.innerHTML += '<p>Clicou no terceiro botão</p>'  
14 }  
15  
16 function acao4() {  
17     let resp = window.document.getElementById('saida')  
18     resp.innerHTML += '<p>Clicou no quarto botão</p>'  
19 }
```

escopo de resp

Se você analisar o código acima com cuidado, a **linha 2** está se repetindo várias vezes nas linhas 7, 12 e 17. Até o momento, isso é necessário porque toda função tem um bloco (delimitado por {}) e toda variável declarada dentro de um bloco só vai funcionar dentro desse bloco. **ANOTE MAIS ESSA DICA AÍ!**

O nome que damos a essa característica é **ESCOPO**. De forma resumida, o escopo de uma variável é a área onde ela existe e funciona (o que chamamos de **escopo local**). Na imagem acima, marquei o escopo da variável resp que está dentro da function acao1(). A minha variável resp do acao1() não vai funcionar fora daquela linha pontilhada. Por isso tivemos que criar várias versões da mesma variável, cada uma dentro de cada ação.

Porém, essa não é a melhor maneira de resolver esse problema. Vamos ver uma nova versão, bem melhor.

```

1 let resp = window.document.getElementById('saída')
2
3 function acao1() {
4     resp.innerHTML += '<p>Clicou no primeiro botão</p>'
5 }
6
7 function acao2() {
8     resp.innerHTML += '<p>Clicou no segundo botão</p>'
9 }
10
11 function acao3() {
12     resp.innerHTML += '<p>Clicou no terceiro botão</p>'
13 }
14
15 function acao4() {
16     resp.innerHTML += '<p>Clicou no quarto botão</p>'
17 }

```

escopo global de resp

Olha só o que aconteceu! Transferi a linha que declara a variável resp para fora das funções, lá no início, na linha imediatamente acima da primeira (não pode ser no final). Agora estamos dizendo ao JS que a variável criada tem **escopo global**, o que significa que ela vai funcionar dentro de todas as funções, é só olhar a nova linha pontilhada do segundo caso.

Conteúdo dinâmico

Outra característica do código acima é o seu comportamento dinâmico. Conforme você usa o site e vai clicando nos botões, ele vai adicionando conteúdo ao final da página sem apagar o conteúdo anterior, que era o que estava acontecendo antes (veja na imagem ao lado).

Isso tudo está acontecendo, simplesmente porque estamos usando o operador `+=` no lugar de usar o a atribuição simples `=` nas linhas 4, 8, 12 e 16.



Se você não entendeu o funcionamento do `+=`, volte para o capítulo anterior, onde explicamos os diversos tipos de operadores de atribuição. Faça aí uma experiência, modificando uma das ações e substituindo o `+=` pelo `=` e veja qual foi o resultado prático.



ATENÇÃO: Nesse caso, o operador `+` não funciona como uma soma aritmética. Na verdade, ele também pode funcionar como **concatenação**, juntando uma string em outra. Sendo assim, quando usamos `+=` com strings, estamos pedindo para o JavaScript não apagar o conteúdo anterior e adicionar o novo conteúdo no final.

Hora de exercitar

Agora chegou a hora de praticar. Acesse agora mesmo o endereço <https://gustavoguanabara.github.io/javascript/exercicios/> e execute o **exercício 009** no seu computador e tente atingir esse mesmo resultado em casa, sem copiar o código que eu criei. Nesse momento, a prática é algo que você mais precisa. Se por acaso ficar difícil, pode acessar o repositório público de HTML e CSS e dar uma olhada nos comandos, mas **EVITE COPIAR**.



Tenho uns desafios pra você!

Lá no repositório, além do material em PDF e dos códigos dos exercícios 100% disponíveis, também disponibilizamos alguns **desafios** que devem ser resolvidos. Esses desafios não incluem o código original e você deve tentar chegar à resposta sem copiar nenhum código.

Com todo o conteúdo que vimos até essa aula, você já pode resolver todos os seguintes desafios, do **d001** até o **d010**.



Acesse o repositório público, abra a área do curso de JavaScript e clique no link de acesso aos desafios. Manda ver! Só não fica pedindo a resposta! Você consegue resolver isso sozinho(a)!

Repositório em: <https://gustavoguanabara.github.io>

Eu já falei sobre isso no YouTube?

Eu sei que às vezes as pessoas gostam mais de assistir vídeos do que ler livros, e é por isso que eu lanço há anos materiais no canal Curso em Vídeo no YouTube. O link que vou compartilhar contigo vai diretamente para a playlist completa do curso que já está totalmente disponível e mostra todos os procedimentos passo-a-passo. Acesse agora mesmo!



Curso em Vídeo: https://www.youtube.com/playlist?list=PLHz_AreHm4dlsK3Nr9GVvXCbpQyHQ11o1