

Introdução ao JavaScript & versionamento de código



DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA

- SSH
- Branches
- Pull Request e Merge
- Git Flow
- Introdução ao JavaScript

O **protocolo SSH** é um protocolo de segurança de comunicação e transferência de dados entre cliente e servidor com **alta encriptação** de dados.

Usamos uma **chave SSH** e o protocolo SSH para garantir a segurança de acesso aos nossos repositórios do Github (e de outros repositórios de código do mercado).

As **chaves SSH** são **arquivos** que ficam salvos em uma pasta **/.ssh** dentro da sua pasta de usuário (para usuários Windows) e contém um **código de segurança** que o servidor usa para confirmar suas credenciais.

Vamos seguir o passo-a-passo do Github para criar nossa chave.

[Link para o site do GITHUB - SSH](#)

Anteriormente no curso, vimos como criamos branches (ramificações) do nosso código fonte.

Quando criamos uma branch, deixamos na branch de origem o código em seu estado atual e passamos a alterar o código na branch nova.

Criamos, desta forma, uma versão do código fonte.

Após modificarmos o que queremos nesta versão, podemos optar por “juntar” novamente as duas versões de código, fazendo com que ambas as branches tenham a mesma versão do código fonte.

Podemos fazer isso de duas maneiras: através de um merge (junção, mescla) direto das ramificações ou através de um pedido - chamado de pull request.

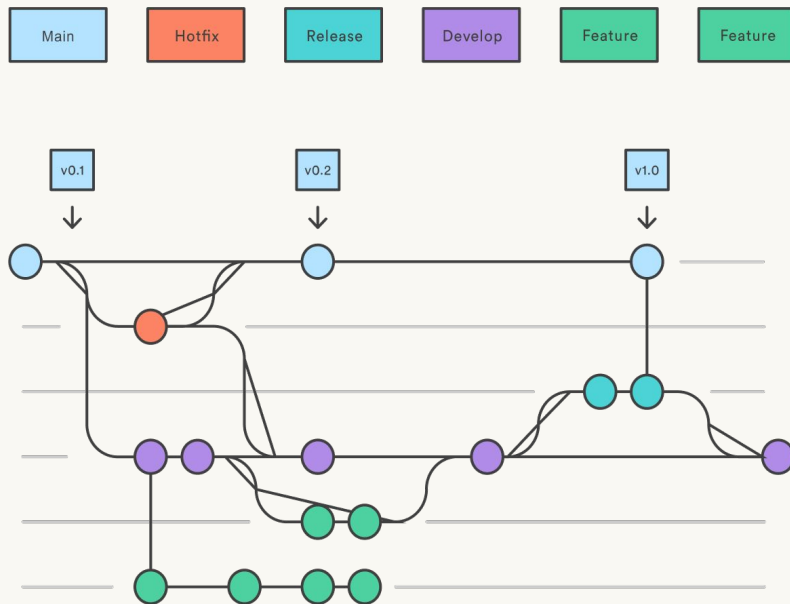
PULL REQUEST E MERGE

Para realizar o merge diretamente na linha de comando, vamos até a branch para a qual queremos trazer o código novo e executamos o comando `git merge <nome_branch_origem>`.

Todas as alterações (commits) realizados na branch de origem vão ser mescladas com a branch destino.

GIT FLOW

Um modelo bastante utilizado nas empresas atualmente para criação de branches é o modelo Git Flow:



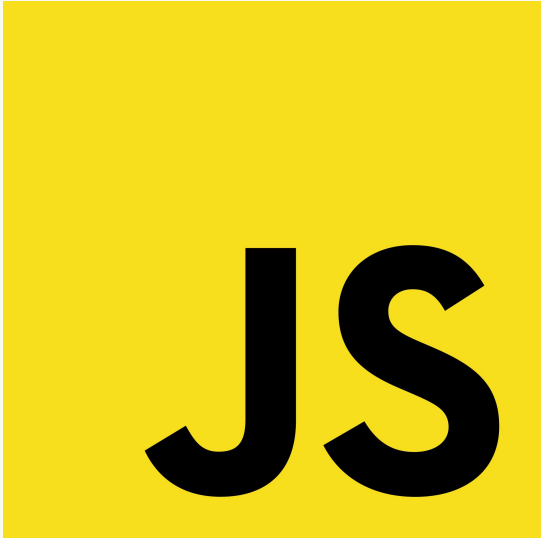
Branches no git flow:

- **main:** branch principal, onde roda o código “em produção”
- **release:** branch de homologação, onde são feitos testes
- **develop:** branch principal de desenvolvimento, de onde se originam as branches de features (funcionalidades)
- **feature:** branch onde é desenvolvida a funcionalidade
- **hotfix:** branch que se origina da main, para correções rápidas do código que está em produção

Chega de conversa! Vamos pra prática!



INTRODUÇÃO AO JAVASCRIPT

The image shows the JavaScript logo, which consists of the letters 'JS' in a bold, black, sans-serif font, centered within a bright yellow square.

BREVE HISTÓRIA DO JAVASCRIPT

A **Microsoft** lançou seu navegador **Internet Explorer** e resolveu fazer, através de engenharia reversa, sua própria versão do JavaScript - o **JScript**.

A **Netscape**, então, buscou padronizar a linguagem através de uma organização internacional - a **ECMA** (*European Computer Manufacturer Association*).

Por razões de registro de marca, a linguagem padronizada nasceu com o nome de **EcmaScript** (nome atual real da linguagem).

BREVE HISTÓRIA DO JAVASCRIPT

A primeira atualização da linguagem veio em 1999 - o **EcmaScript 3**.

Foram desenvolvidas, durante os anos seguintes, as versões **3.1** e **4**, mas nunca chegaram a ser implementadas.

A próxima atualização concreta veio só em **2009** - **ES 5**.

BREVE HISTÓRIA DO JAVASCRIPT

O **ES5** permaneceu vigente até 2016, quando foi lançada a versão **ES6** - que trouxe diversas novas funcionalidades muito utilizadas, como a sintaxe **arrow**, **promises**, novas formas de declarar variáveis e outras ferramentas que serão estudadas no decorrer do curso.

Desde então, o grupo de trabalho **TC39** - responsável pela padronização da linguagem **JavaScript** - tem tentado lançar uma nova versão anualmente. A versão atual é a **ES12** (março de 2022).

Chega de conversa! Vamos pra prática!

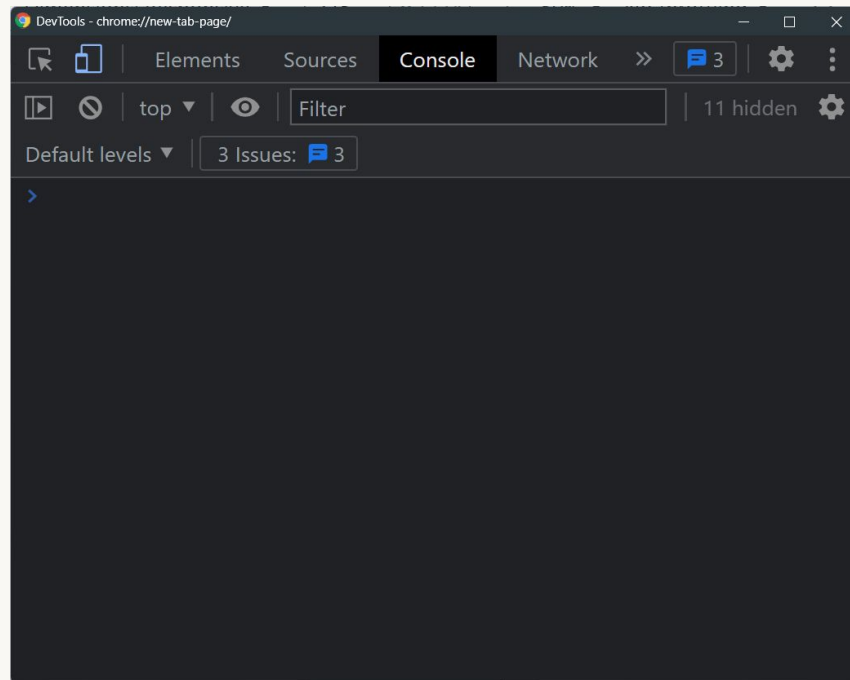


JAVASCRIPT NO NAVEGADOR

Developer Tools

- F12 do teclado
- Botão direito + inspecionar
- CTRL + SHIFT + i

Verifique se a aba selecionada é a "Console"

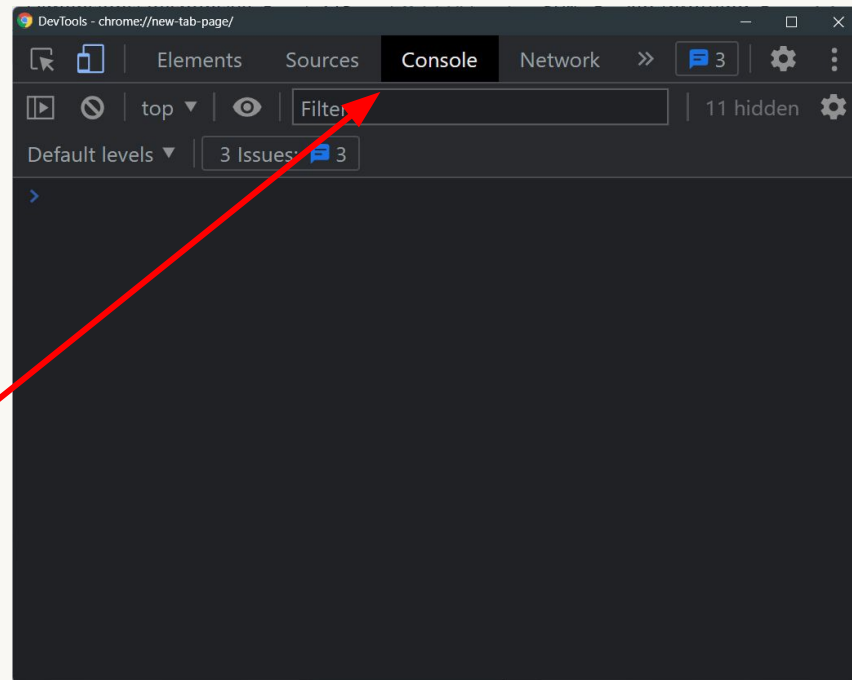


JAVASCRIPT NO NAVEGADOR

Developer Tools

- F12 do teclado
- Botão direito + inspecionar
- CTRL + SHIFT + i

Verifique se a aba selecionada é a "Console"



Console.log é um **método** JavaScript utilizado para imprimir no console do navegador alguma informação.

A sua sintaxe é:

```
console.log ("Olá mundo!")
```

JAVASCRIPT NO NAVEGADOR

Console.log é um **método** JavaScript utilizado para imprimir no console do navegador alguma informação.

A sua sintaxe é:

MÉTODO

vamos estudar sobre
métodos e funções mais
adiante no curso!

console.log ("Olá mundo!")

ARGUMENTO

Para podermos trabalhar de maneira simplificada, vamos criar nossos script no VSCode e executar o código com a extensão Code Runner.

.run

<https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner>

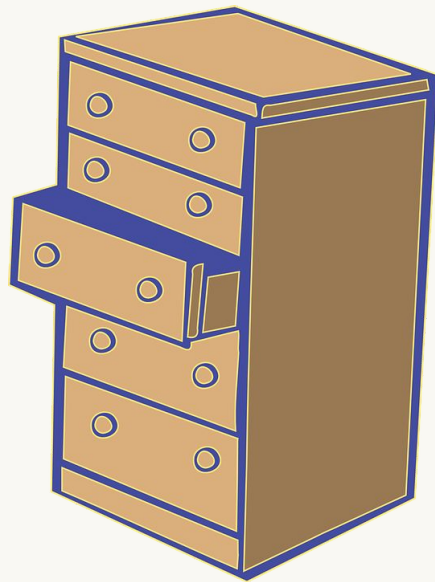
Usamos **variáveis** em nosso código sempre que queremos **armazenar um valor** para ser reutilizado posteriormente.

Uma variável funciona como uma “**etiqueta**” que indica o endereço do valor atribuído à ela na memória do nosso computador.

VARIÁVEL VAR

Podemos imaginar a memória do computador como um enorme “armário” cheio de “gavetas”.

Guardamos nosso valor “Olá mundo!” em uma gaveta de nome “**primeiraVariavel**”.

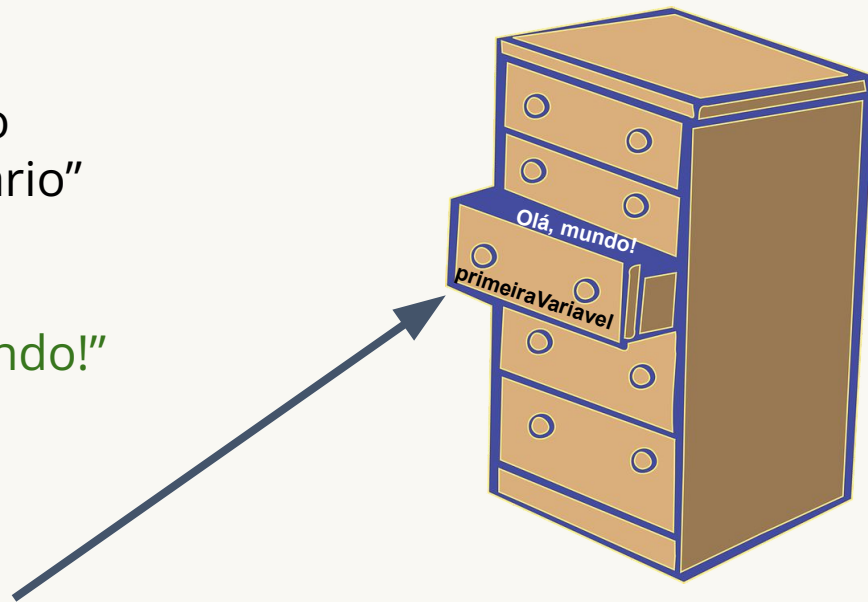


```
var primeiraVariavel = “Olá mundo!”
```

VARIÁVEL VAR

Podemos imaginar a memória do computador como um enorme “armário” cheio de “gavetas”.

Guardamos nosso valor “Olá mundo!” em uma gaveta de nome “**primeiraVariavel**”.



```
var primeiraVariavel = "Olá mundo!"
```

VARIÁVEL VAR

A sintaxe de definição de variável:

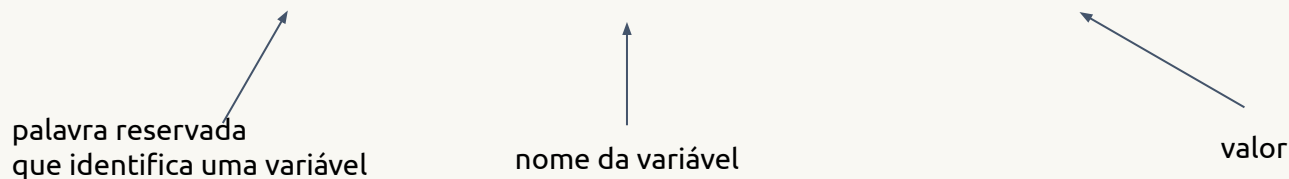
```
var primeiraVariavel = "Olá mundo!"
```


VARIÁVEL VAR

A sintaxe de definição de variável:

var primeiraVariavel = "Olá mundo!"

palavra reservada
que identifica uma variável



nome da variável

valor

A sintaxe de definição de variável:

```
var primeiraVariavel = "Olá mundo!"
```

↑
operador de atribuição
(veremos mais adiante)

Toda vez que quisermos recuperar o valor dessa variável, basta passarmos o nome dela para a operação que estamos fazendo:

```
console.log(primeiraVariavel)
```

Existem algumas regras para determinarmos o nome de uma variável:

- Só pode conter **letras, dígitos** ou os símbolos \$ e _;
- O primeiro caractere **não** pode ser um **dígito**;
- Não podemos utilizar palavras reservadas da linguagem;

Podemos utilizar duas convenções para criar nomes de variáveis compostos: o padrão **snake_case** ou o **camelCase**.

Exercício!

Vamos abrir uma empresa de convites de casamento, e temos nosso texto padrão. Porém, escrever cada um dos convites manualmente é impraticável.

Dado o texto padrão, identifique quais **variáveis** podemos utilizar no lugar do texto, para que possamos **alterar o nome** dos convidados e dos noivos sem precisar escrever todo o texto novamente:

Caro Fulano e Fulana!

Vocês estão convidados para o casamento de Beltrano e Beltrana, a ser realizado no dia 14/03/2022, às 16 horas.

Contamos com a sua presença!

*Atenciosamente,
os noivos*



Caro **Fulano** e **Fulana**!

Vocês estão convidados para o casamento de **Beltrano** e **Beltrana**, a ser realizado no dia 14/03/2022, às 16 horas.

Contamos com a sua presença!

Atenciosamente,
os noivos



Caro **Fulano** e **Fulana**!

Vocês estão convidados para o casamento de **Beltrano** e **Beltrana**, a ser realizado no dia **14/03/2022**, às **16** horas.

Contamos com a sua presença!

Atenciosamente,
os noivos



Mãos à obra!



TIPOS DE DADOS PRIMITIVOS

Nossas variáveis podem armazenar diversos tipos de dados que são definidos pelo JavaScript.

Uma variável no JavaScript pode ser sobrescrita com tipos diferentes de dados - característica que costumamos chamar de **linguagem de tipagem fraca**.

Entender os tipos de dados é importante para a boa implementação do código e para evitar erros.

TIPOS DE DADOS PRIMITIVOS

Vamos entender alguns tipos de dados **primitivos**:

- **String**
- **Number**
- **BigInt**
- **Boolean**
- **Null**
- **Undefined**
- **Symbol**

TIPOS DE DADOS PRIMITIVOS

Vamos entender alguns tipos de dados **primitivos**:

- **String**
- **Number**
- **BigInt** (*não vamos ver agora*)
- **Boolean**
- **Null**
- **Undefined**
- **Symbol** (*não vamos ver agora*)

TIPOS DE DADOS PRIMITIVOS

- **String**

String é um tipo de dado que pode ser definido como uma cadeia de caracteres - de fato, nós já estamos utilizando o tipo string em nossos exemplos.

Para criar uma string, usamos aspas **simples** ou **duplas**.

```
var nome = "Michael Nascimento"
```

TIPOS DE DADOS PRIMITIVOS

- **Number**

No JavaScript, o tipo number representa um número inteiro ou decimal - não existe diferenciação entre ambos, como em outras linguagens.

Para criar um number, não usamos aspas. Basta atribuir o valor à variável:

```
var idade = 31
```

```
var altura = 1.76
```

TIPOS DE DADOS PRIMITIVOS

- **Number**

No JavaScript, o tipo number representa um número inteiro ou decimal - não existe diferenciação entre ambos, como em outras linguagens.

Para criar um number, não usamos aspas. Basta atribuir o valor à variável:

```
var idade = 31  
var altura = 1.76
```

números decimais usam
ponto ao invés de **vírgula**!

TIPOS DE DADOS PRIMITIVOS

- **Boolean**

Um valor Boolean é um tipo de dado lógico que pode assumir dois valores: **true** ou **false**.

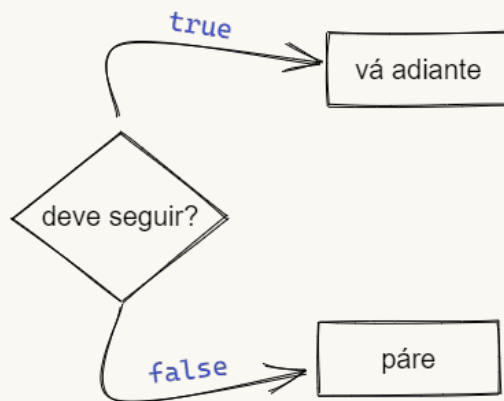
Para criar um boolean, não usamos aspas e passamos um dos dois valores à variável:

```
var temMuitoDinheiro = false
```


TIPOS DE DADOS PRIMITIVOS

- **Boolean**

Usamos os valores booleanos constantemente para determinar o **fluxo de execução** do código, principalmente dentro de **loops** e **condicionais** - assuntos que serão abordados mais adiante no curso.



- **Null e Undefined**

Os tipos de dado **null** e **undefined** tem um comportamento bastante similar, mas são diferentes.

O tipo null é um indicador de “**valor vazio**” ou “**valor desconhecido**” e deve ser **atribuído manualmente** pelo programador (nunca é atribuído pelo sistema).

O tipo undefined significa “**não definido**” e é **atribuído de forma padrão** pelo JavaScript à variáveis que não foram inicializadas com nenhum valor.

TIPOS DE DADOS PRIMITIVOS

- **Null e Undefined**

Exemplo:

```
var documento; // undefined
```

```
var telefone = null; // null
```

TIPOS DE DADOS PRIMITIVOS

Para sabermos qual é o tipo de determinado valor ou variável, podemos utilizar o operador **typeof**.

Este operador é muito útil quando quisermos garantir, por exemplo, que os argumentos passados para uma função sejam de um determinado tipo específico (veremos sobre funções mais adiante).

TIPOS DE DADOS PRIMITIVOS

Exemplos:

```
var nome = "Mika Nascimento"
```

```
var idade = 31
```

```
var temDinheiro = false
```

```
typeof nome; // "string"
```

```
typeof idade; // "number"
```

```
typeof temDinheiro; // "boolean"
```

- Git em 15 minutos - <https://www.youtube.com/watch?v=USjZcfj8yxE>
- Git Flow - [Atlassian Git Flow](#)
- Tipos de dados JavaScript - [JavaScript Data Types](#)
- História do JavaScript - [\(EN\) The Weird History of JavaScript](#)
- 25 Anos de JavaScript - [JetBrains - 25 Years of JavaScript Timeline](#)



DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!



<LAB365>