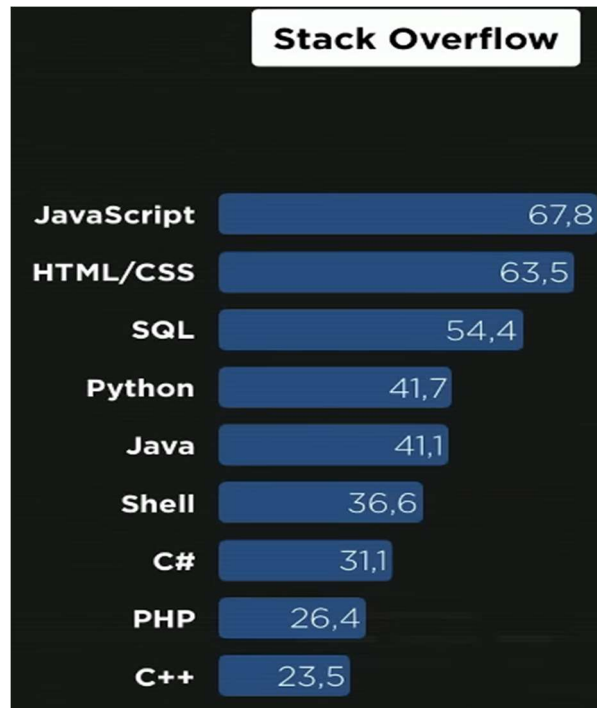


- Qual é a linguagem mais popular?
- Quais são as linguagens mais utilizadas no mundo?

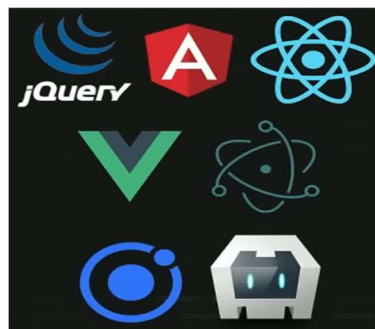
⇒ Esse é o resultado de uma pesquisa de levantamento da **Stack Overflow**, uma das maiores plataformas de reunião de programadores do mundo.



E ela mostra que, **JavaScript** é uma das linguagens com maior quantidade de material e de pessoas interessadas por ela. E o motivo disso é muito fácil de entender. Basicamente, todo site ou aplicação que você utiliza no seu celular ou no seu computador utiliza **JavaScript** na sua essência.

E quando você se tornar um programador mais experiente e quiser aprender novas tecnologias como:

- JQuery;
- Angular;
- React;
- Vue.js;
- Electron;
- Ionic;
- Cordova.



Saiba todas elas têm uma coisa em comum, todas essas tecnologias utilizam na sua essência a linguagem **JavaScript**.

Vamos aprender os fundamentos da linguagem **JavaScript**, aprender a trabalhar no modo gráfico utilizando seu navegador e construções de páginas web interativas utilizando a linguagem, vamos trabalhar um pouco com o Node.js que executa o **JavaScript** fora dos navegadores.

Serão aulas básicas que tratam dos fundamentos da linguagem, vamos trabalhar com as versões mais recentes da especificação **ECMAScript**, que é a padronização internacional da linguagem **JavaScript**.

Para iniciar precisamos entender bem um conceito muito importante, temos que entender o conceito de **CLIENTE X SERVIDOR**.

HTTP (Hypertext Transfer Protocol) é o protocolo base para as aplicações web e clientes (como aplicativos mobile, entre outros) que se conectam a web APIs. Ele permite a comunicação entre clientes e servidores através da internet.

Cliente:

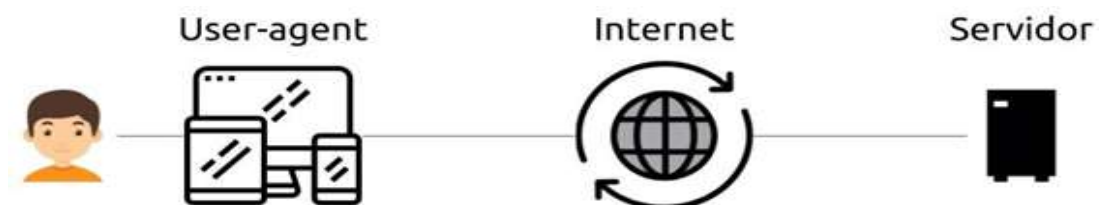
É o cliente quem sempre inicia a comunicação com o servidor.

O cliente mais conhecido é o navegador.

Quando acessamos um site pelo navegador, uma requisição é disparada pela internet para o computador no qual esse site está hospedado.

Chamamos as aplicações que agem em nome do usuário, assim como o navegador, de **user-agent**.

A Figura 1 demonstra esse fluxo de dados.



Comunicação entre o usuário e o servidor através da internet e por meio de um **user-agent**

Após enviar uma requisição, o **user-agent** aguarda até que haja uma resposta do servidor e então a comunicação se encerra.

Para obter um novo recurso do servidor o cliente deve iniciar uma nova requisição.

Servidor:

O papel do servidor web é receber uma requisição e devolver uma resposta para o cliente.

Geralmente o servidor não envia dados se o cliente não disparar primeiro uma requisição.

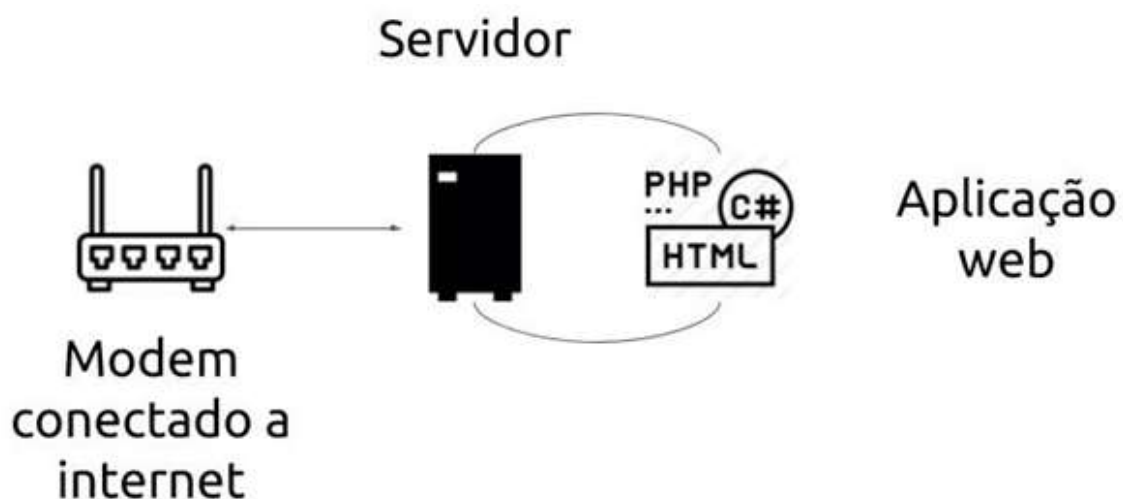
O servidor é um ou mais computadores com um único endereço como `//www.senai.com.br`. Nestes computadores podem ser hospedados documentos, imagens, áudio, vídeo, etc.

Geralmente o servidor também hospeda programas que atendem as requisições dos clientes e as processam. Chamamos esses programas de **web server**, ou servidor web.

É com um web server que nos comunicamos, sendo o computador no qual ele está instalado parte da infraestrutura que o conecta a internet.

O web server geralmente é capaz de executar aplicações escritas em linguagens como PHP, C#, Java, etc.

Na Figura 2 vemos um gráfico que ilustra isso.



Por exemplo, um servidor executando o PHP pode delegar a esse programa a tarefa de atender as requisições criando páginas HTML a partir de scripts escritos nessa linguagem.

Quando você acessa a internet, seja o que for, WhatsApp, Facebook, Instagram, o site do Google, o site do YouTube, o seu Gmail, no momento que você usa um dispositivo, ele é um **cliente** e que precisa de dados de um servidor. Às vezes de até mais de um servidor.

Agora você consegue entender que existe tecnologias para clientes, e existem tecnologias para servidores.

O JavaScript surgiu como uma tecnologia para clientes, é uma tecnologia client-side, ou seja, JavaScript funciona muito mais do lado do cliente.

Uma outra coisa que é muito importante de entender junto com JavaScript são as tecnologias relacionadas.

Sem elas o website não vai conseguir existir.

O conteúdo, o design e as interações.

E quando a gente está falando de desenvolvimento de sites, temos três tecnologias, **HTML**, quem dá todo o texto, imagens, sons e vídeos, o **CSS**, que deixa tudo bonito e o **JavaScript** que é o responsável pelas integrações e interações.

O **JavaScript** permite você fazer quase tudo, até mesmo modificar o documento na sua parte HTML e CSS. Esse é o poder da linguagem.

HTML e CSS não são linguagens de programação, são tecnologias de construção de sites, marcação e estilo, aqui você se diz desenvolvedor.

Mas JavaScript sim, é linguagem de programação, aqui você se diz programador.

Assim que vamos aprofundando os estudos vem as seguintes perguntas:

JavaScript e Java são a mesma coisa? São diferentes? Por que eles têm um nome tão parecido se eles são diferentes? Ou se eles são iguais? Porque tem um script no final de um e não tem um script no final do outro? JavaScript e ECMAScript, tem alguma relação entre eles? O que é esse ECMAScript?

A resposta é simples.

Como o HTML era muito estático, em 1995, um ex-funcionário da Silicon Graphics, que trabalhava para o Jim Clark, passou da Silicon Graphics para dentro da Netscape. Brandon Naick, era um desenvolvedor e a tarefa dele era a seguinte, criar uma linguagem que desse mais funcionalidades ao simples HTML que estava na época.

O Brandon Naick tinha um projeto e batizou a sua linguagem inicialmente de **Mocha**, que é um tipo de cafezinho. E nessa mesma época, estava surgindo uma linguagem lá da **Sun Microsystems**, chamada **Java**, em 1995, o Bryden Naick estava criando a linguagem **Mocha** e estava nascendo ao mesmo tempo a linguagem **Java**. E essa linguagem **Java** ganhou um sucesso muito grande na mídia, que estava chamando a linguagem de: **linguagem do futuro**. E aí a Netscape pensou, nós estamos criando uma linguagem, **Java** é linguagem do futuro, nossa linguagem não vai nem se chamar **Mocha**, na verdade ela tinha até um outro nome, **Mocha** foi só um nome inicial, ela se chamava **LiveScript**. E o pessoal da Netscape falou assim, sucesso, o nome **Java** faz sucesso? Não vamos chamar nossa linguagem de **LiveScript**, nós vamos chamar nossa linguagem de **JavaScript**.

O nome JavaScript não vem porque a linguagem se parece com Java, vem porque na época estava fazendo sucesso falar a linguagem Java, estava muito falada, e aí os caras da Netscape, numa grande jogada de marketing, falaram assim tá aí, vou usar esse nome Java na minha linguagem também, e acabaram lançando a linguagem JavaScript. Na verdade, se você analisar as linguagens Java e JavaScript, são bem diferentes entre si.

É claro que alguns comandos se parecem bastante. E nesse momento você pensa, eu aprendi Java e JavaScript, o if é igualzinho. Na verdade, o if não é o if do JavaScript nem o if do Java. As duas linguagens foram baseadas numa outra linguagem chamada linguagem C. Então é por isso que alguns comandos se parecem bastante, mas são linguagens completamente diferentes. JavaScript é uma coisa, Java é outra. Não confunda.

JavaScript foi uma linguagem que fez bastante sucesso e o sucesso atraiu a atenção de algumas empresas, e uma dessas foi a Microsoft, a criadora do Windows. A ideia da Microsoft foi a seguinte, JavaScript é uma linguagem legal, vamos nós mesmos criar a nossa própria linguagem. Isso porque ela tinha acabado de lançar um outro navegador que basicamente

era o código base do Mosaic, o Mosaic ficou com a NCSA, ele foi passado para uma empresa, que foi comprada pela Microsoft, que pegou o código do Mosaic e lançou um navegador chamado Internet Explorer.

O Internet Explorer foi lançado muito perto do ano de 95, junto com o Windows 95. E aí a Microsoft não deixou para trás, vamos criar nosso próprio JavaScript com os mesmos comandos, resolvendo alguns problemas que a galera já tinha identificado e vamos batizar nossa linguagem não de JavaScript, e sim de JScript, porque pode rolar um processo. A Netscape obviamente pensou, nossa, mas nós criamos a linguagem JavaScript, veio a Microsoft, meio que copiou utilizando os mesmos comandos, deu uma melhorada em outra, vai acabar virando bagunça. E a Netscape teve uma sacada muito boa, que foi padronizar a linguagem.

E isso aconteceu em 1997, quando a Netscape procurou uma empresa de padronização. Nesse caso, uma empresa europeia, a ECMA, que é a Associação Europeia de Fabricantes de Computadores. A ECMA na Europa é tipo uma ISO nos Estados Unidos, ela padroniza as coisas. E foi aí, com a Netscape cedendo o código do JavaScript para a ECMA, para uma padronização, que surgiu a linguagem ECMAScript.

Basicamente, ECMAScript é a linguagem JavaScript padronizada. Hoje, quando se estuda JavaScript, as pessoas pensam que vão estudar JavaScript, mas você vai estudar ECMA, isso significa que você vai estudar a versão padronizada do JavaScript.

E a evolução do JavaScript não foi só a evolução da linguagem, várias ferramentas surgiram com a evolução do JavaScript. E foram ferramentas que surgiram e que levam o JavaScript para outro patamar.

- **jQuery;** - O jQuery não é um framework, ele é um conjunto de bibliotecas. Foi muito famoso durante muitos anos, foi criado por desenvolvedores da Mozilla. Era uma biblioteca que facilitava muito o uso de interatividades em JavaScript.
- **Angular;** - O Angular é criado e mantido pelo Google ele facilita a criação de aplicações web, é uma linguagem menos imperativa, é uma linguagem mais declarativa.

- **React;** - O React tem mais ou menos a mesma funcionalidade do Angular, mas ele tem algumas flexibilidades maiores, Existe o React Native para criação de aplicações em celular.
- **Vue.js;** - O VUE.JS foi criado por um ex-programador do Google que segundo ele mesmo, já estava cansado de usar o Angular e criou uma versão bem melhor.
- **Electron;** - O Electron, hoje é mantido pela GitHub, é especializado para a criação de interfaces gráficas, para programas de interface gráfica. Você com certeza já utilizou ou vai utilizar um programa feito em Electron. O próprio Visual Studio Code da Microsoft, foi feito em Electron. Aqueles programas que você instala em seu computador, para acessar o WhatsApp, o Discord.
- **Ionic;** - O Ionic é um SDK para criar aplicações para dispositivos móveis, utiliza o próprio Node.js, como outros desses aqui usam o Node.js para rodar o JavaScript fora do navegador.
- **Cordova.** - O cordova é a evolução de uma tecnologia antiga chamada de PhoneGap, que hoje evoluiu, está na mão da Apache.

Existem vários outros frameworks, API, bibliotecas, principalmente também para criação de jogos.

São exemplo:

- **Phaser**
- **PixJS**
- **Impact**
- **Melon.js**
- **Crafty.js,**



existem várias bibliotecas e várias tecnologias para criação de jogos em JavaScript.

1. Framework (Estrutura):

- Definição: Um framework é uma estrutura ou ambiente de desenvolvimento que fornece uma base para criar software. Ele define a

estrutura e as diretrizes sobre como o código deve ser organizado e executado.

- Exemplo: Ruby on Rails é um framework popular para desenvolvimento web que fornece ferramentas e estruturas pré-definidas para criar aplicativos web de forma eficiente.

2. SDK (Kit de Desenvolvimento de Software):

- Definição: Um SDK é um conjunto de ferramentas, bibliotecas e documentação que facilitam o desenvolvimento de software para uma plataforma específica.

- Exemplo: O Android SDK é usado para desenvolver aplicativos para dispositivos Android, fornecendo APIs e ferramentas para compilar, depurar e testar aplicativos.

3. API (Interface de Programação de Aplicativos):

- Definição: Uma API é um conjunto de regras e protocolos que permite que diferentes softwares se comuniquem e interajam entre si.

- Exemplo: A API do Google Maps permite que desenvolvedores integrem mapas e funcionalidades de geolocalização em seus aplicativos, utilizando métodos e endpoints específicos para acessar esses recursos.

4. Biblioteca:

- Definição: Uma biblioteca é um conjunto de códigos e funcionalidades pré-definidos que podem ser reutilizados em diferentes projetos para realizar tarefas específicas.

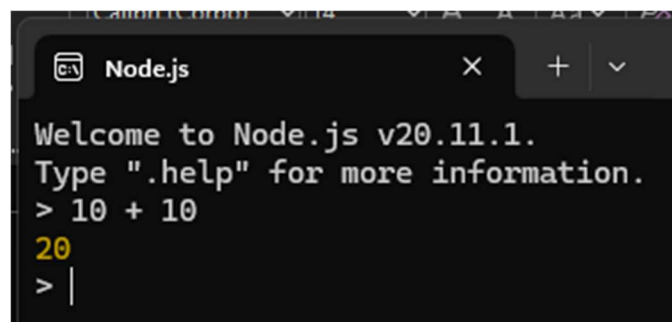
- Exemplo: A biblioteca jQuery é amplamente utilizada para simplificar o desenvolvimento de aplicativos web, oferecendo funções para manipulação de DOM, animações, requisições AJAX, entre outras funcionalidades.

Resumindo, um framework é uma estrutura que orienta o desenvolvimento de software, um SDK é um conjunto de ferramentas para desenvolver em uma plataforma específica, uma API define como os

softwares se comunicam e uma biblioteca oferece funcionalidades específicas para serem reutilizadas em projetos. Cada um desses conceitos desempenha um papel importante no processo de desenvolvimento de software, oferecendo ferramentas e recursos para os desenvolvedores.

Antes de iniciar o desenvolvimento em JavaScript, é necessário saber:

- 1- Instalar e organizar pastas com Visual Studio Code;
- 2- Instalar o NODE.JS;
 - 2.1 – <https://nodejs.org/en>
 - 2.2 – Baixar a versão Recommended
 - 2.3 – Siga a instalação até o final
 - 2.4 – Abra o Node.Js no menu iniciar
 - 2.5 – Para testar digite 10 + 10 e de Enter, ele deverá retornar o resultado 20.



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 10 + 10
20
> |
```

- 2.6 – Para sair digite .exit
- 3- Saber identificar o HTML5, CSS3 dentro de código;
- 4- Fazer comentários, para isso temos as simbologias:
 - 4.1– HTML
<!--Comentário aqui-->
 - 4.2– CSS
/*Comentário aqui*/
 - 4.3 - JavaScript
// comentários em uma única linha.
/* */ Comentar trechos de um código.

Dando inicio ao Desenvolvimento, vamos começar a desenvolver nossos primeiros scripts em JavaScript e aprender a disparar Janelas Simples.

- 1- Criar uma pasta para exercitar o JavaScript dentro de **Documentos** com nome **Javascript**;
- 2- Abrir a pasta e clicar com o direito do mouse abrir o terminal do Git Bash e chamar o **Visual Studio Code**;
- 3- Criar uma pasta chamada **aula1** e dentro dela criar um arquivo chamado **exercicio1.html**
- 4- Iniciar o corpo de um **HTML** simples, como aprendemos no inicio da unidade curricular **Codificação para Front-End**, colocar o Titulo **Exercício 1 JavaScript**, criar um título **<h1> Meu primeiro Script** e um ** Exercício 1** .

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device
6      <title>Exercício 1 JavaScript</title>
7  </head>
8  <body>
9      <h1>Meu primeiro Script</h1>
10     <span>Exercício 1</span>
11 </body>
12 </html>
```

Vamos aprender a colocar CSS dentro do HTML, para simplificar as aulas quando o código for pequeno.

- 1- Dentro do **<head>** vamos criar a **tag <style>** que será onde daremos estilo a esse nosso **exercicio1.html**;
- 2- Aplicar os estilos no **body**;

```
background-color: aqua;
color: blue;
text-align: center;
font-size: 25px;
```

- 3- Antes do fechamento do corpo do Site na **tag </body>**, para que os scripts sejam carregados depois do código base, vamos criar uma **tag <script>** (lembrando que tem como criar scripts externos, igual ao css, mais para frente iremos fazer isso, agora estamos apresentando um exemplo básico);

```
<body>
  <h1>Meu primeiro Script</h1>
  <span>Exercício 1</span>

  <script>

  </script>
</body>
</html>
```

4- Dentro da tag <script> vamos programar em JavaScript.

- ⇒ Não iremos dizer que programamos em HTML e nem em CSS, mas podemos dizer que iremos programar em JavaScript, porque ela sim é uma linguagem de programação. HTML é uma linguagem de marcação de conteúdo, e CSS é uma linguagem de estilos, são folhas de estilo, então você não diz que você programe nenhuma delas, a não ser o JavaScript.
- ⇒ Aqui dentro, podemos escrever o comando em JavaScript, sempre em letras minúsculas, na maioria das vezes, quando não for, irei explicar, porque tem diferença no caso do JavaScript.

SINTAXE BÁSICA

No JavaScript, as instruções (ou comandos) são chamadas de declaração. Uma instrução pode conter várias linhas, que serão lidas (linha por linha) e interpretadas pelo navegador, sequencialmente.

1. Vamos começar mostrando um alerta na tela:

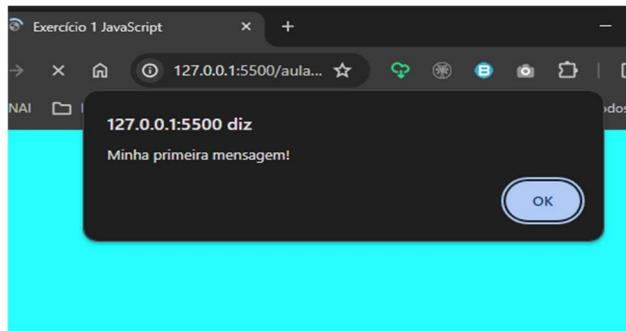
1.1 Alerta:

Exercicio1.html

⇒ **window.alert('Minha primeira mensagem!')**

- Antigamente era preciso colocar ponto e vírgula no final de cada comando, hoje em dia ele não é mais obrigatório.

⇒ Mostra uma janela de alerta!



- Aqui podemos perceber que a parte de estilo já foi exibido, só que o conteúdo ainda não apareceu, o Meu primeiro Script! E o Exercício1 não apareceu ali atrás, mas ele já disse aqui, Minha primeira mensagem. Então esse `window.alert` é o seu primeiro comando em JavaScript, que podemos até simplificar para somente `alert`, não tem problema.

Mesmo colocando o `<script>` no final, ele executou o comando, sem que o conteúdo apareça efetivamente na tela. Mesmo se atualizar o site, o conteúdo não aparece, mas a minha primeira mensagem apareceu, só quando clicar em OK, o conteúdo será exibido.

Essa é a interatividade que iremos criar.

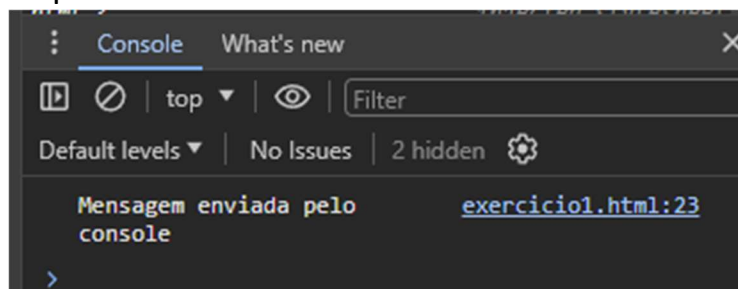
1.2 Console.

Exercicio1.html

⇒ `console.log('Mensagem enviada pelo console')`

```
21 <script>
22   window.alert('Minha primeira mensagem!')
23   console.log('Mensagem enviada pelo console')
24 </script>
```

⇒ `console.log` é uma mensagem visível apenas ao ser inspecionada.



⇒ A mensagem, no caso, é “Mensagem enviada pelo console”.

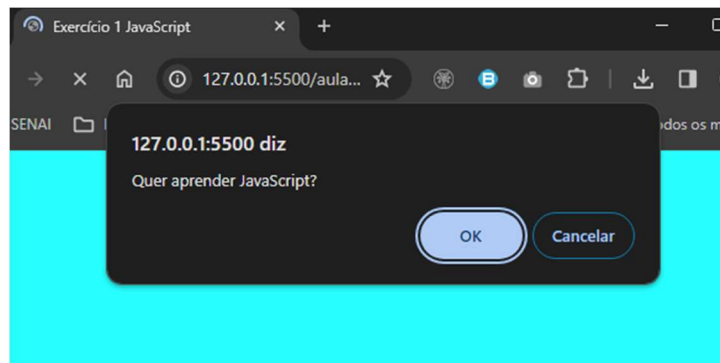
- Agora vamos pedir uma confirmação:

1.3 Confirmações:

Exercicio2.html

⇒ `window.confirm('Quer aprender JavaScript')`

⇒ Mostra uma janela de confirmação!



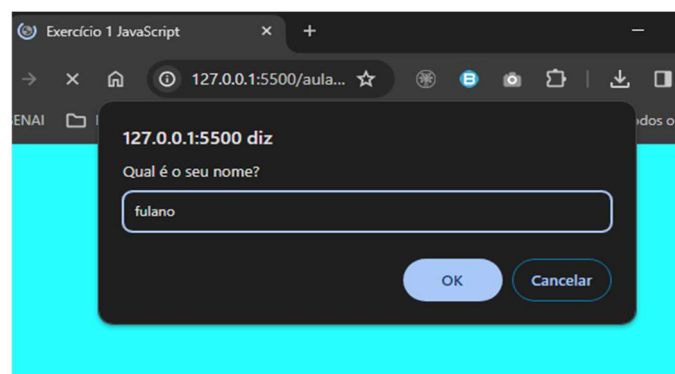
- O primeiro comando a ser executado é o meu `window.alert`. Aparece, Minha primeira mensagem. Depois de clicar no OK, será exibido o `window.confirm`, que está perguntando, Quer aprender JavaScript? com um Ok ou Cancelar.

1.4 Perguntas.

Exercicio2.html

⇒ `window.prompt('Qual seu nome?')`

⇒ Mostra uma janela com uma pergunta!



- Qual é o seu nome? Vou responder Fulano e clicar OK. Agora vem a pergunta, mas o nome não apareceu na tela. Não apareceu porque não mandamos ainda. Na verdade, não tem nem como testar no

momento em que está esse código, para isso é necessário saber qual foi o botão que a pessoa apertou, se ela apertou OK, se ela apertou Cancela. No prompt eu não sei qual foi o nome que a pessoa digitou, isso porque temos que utilizar aqui na frente dessas funcionalidades algumas variáveis.

2. Tipos de Dados e Variáveis;

VARIÁVEIS

Variáveis são elementos que armazenam dados de diferentes tipos (que serão utilizados em um algoritmo) na memória do computador.

Para criar uma variável em JavaScript, basta declará-la, digitando:

- A palavra "var" com o sinal de igual (=);
- O identificador;
- O valor que queremos atribuir a ela.

Exemplos:

⇒ Precisamos sempre nos lembrar que um único sinal de = será visto como **recebe**.

Vamos guardar 3 números:

n1	1	- Aqui falamos que n1 recebe 1
n2	8.5	- Aqui falamos que n2 recebe 8.5
n3	12	- Aqui falamos que n3 recebe 12

Assim teremos:

```
var n1 = 1
```

```
var n2 = 8.5
```

```
var n3 = 12
```

⇒ Precisamos sempre nos lembrar que um único sinal de = será visto como **recebe**.

⇒ **Quando colocamos os valores dentro das variáveis chamamos isso de atribuição, quer dizer que iremos atribuir um valor a uma variável.**

Agora vamos criar uma nova variável de tamanho maior para guardar outro tipo de dados, pois existem variáveis de tamanhos diferentes, para armazenar dados de tamanhos diferentes.

Vamos guardar 3 cadeias de caracteres - strings:

S1	sexta-feira	- Aqui falamos que n1 recebe 1
S2	Estou feliz	- Aqui falamos que n2 recebe 8.5
S3	JavaScript	- Aqui falamos que n3 recebe 12

Assim teremos:

```
var s1 = "sexta-feira"
```

```
var s2 = 'Estou feliz'
```

```
var s3 = `JavaScript`
```

Como podemos perceber, as cadeias de caracteres, nossas palavras, estão entre aspas, e podemos usar em JavaScript 3 tipos de aspas, sendo as duplas, a simples e a crase. São os três tipos de forma de delimitar um string dentro da linguagem JavaScript

Agora, podemos ver que temos 6 espaços, onde cada um tem seu nome definido, para evitar confundi-los. E o nome de cada uma dessas variáveis se chama Identificadores.

E para dar nome a essas variáveis existem algumas regras:

- Podem começar com: letra, \$ ou _
Ex: n1, s1, muito raro começar com \$ ou _, mas podemos utilizar.
- Não podem começar com números
EX: 1S, não podemos fazer.
- Podemos usar letras ou números
Ex: n1, n2, n10
- Podemos usar acentos e símbolos
Ex: média com acento, utilizar símbolo de π para representar o pi
- Não podem conter espaços
EX: Nota 1

- Não podem ser palavras reservadas
EX: function, alert, ou outras palavras que são comandos de JavaScript

Dicas na hora de criar variáveis:

- Maiúsculas e Minúsculas fazem diferença.
- Tente escolher nomes coerentes para as variáveis.

Representação dos tipos de dados:

Inteiros => 10, 2, -30

Reais => 0.2, -8.5, 8.4 – números com ponto flutuante, ou float.

No JavaScript não há a diferença entre esses tipos de dados, todos são vistos como do **tipo number**.

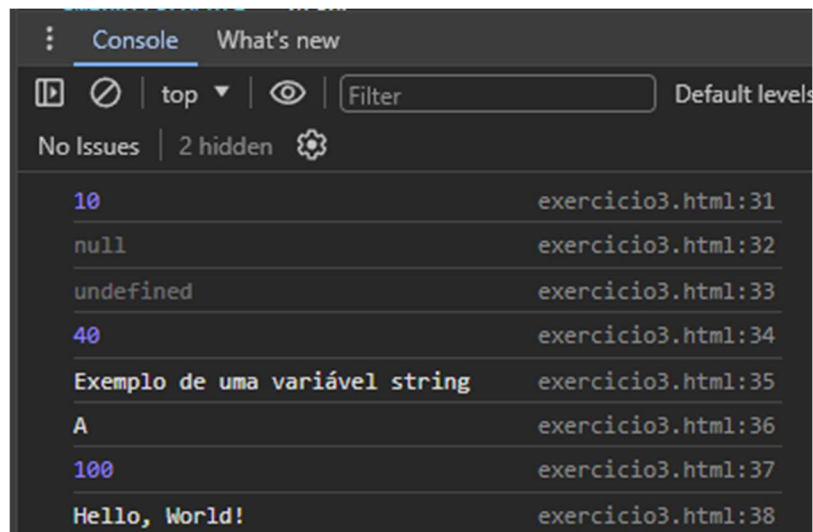
String => "Hoje", 'Java', `tem` - são cadeias de caracteres, um string também pode ser um conjunto de números, por exemplo: o número do cpf, identidade, telefone. Como eles possuem . e -, são considerados strings, por serem um conjunto de caracteres compostos por números, pontos e traços.

Boolean => true – false, verdadeiro e falso

Exercicio3.html

```
21 <script>
22   var exemploBoolean = 10
23   var exemploNull = null
24   var exemploUndefined
25   var exemploNumber = 40
26   var exemploString = "Exemplo de uma variável string"
27   var exemploCaractere = "A"
28   var $ = 100
29   var $$ = "Hello, World!"
30
31   console.log(exemploBoolean)
32   console.log(exemploNull)
33   console.log(exemploUndefined)
34   console.log(exemploNumber)
35   console.log(exemploString)
36   console.log(exemploCaractere)
37   console.log($$)
38   console.log($$)
39
40 </script>
```


- Ao inspecionar o Site, na área do console podemos ver os resultados.



- Vamos ver o resultado no NODE.JS

```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var exemploBoolean = 10
undefined
> var exemploNull = null
undefined
> var exemploUndefined
undefined
> var exemploNumber = 40
undefined
> var exemploString = "Exemplo de uma variável string"
undefined
> var exemploCaractere = "A"
undefined
> var $ = 100
undefined
> var $$ = "Hello, World!"
undefined
```

Como podemos perceber, após declarar a variável, recebemos em todas esse nome undefined, que significa não definido. Precisamos definir nosso **typeof**.

Nossos tipos de dados ou **data types** são:

Tipos Primitivos

1. String (Cadeia de Caracteres)

A primeira categoria de tipos de dados em JavaScript são as strings. Elas representam texto e são definidas usando aspas simples ou duplas.

Exemplo:

```
var nome = "João";
```

```
var mensagem = 'Olá, Mundo!';
```

2. Number (Número)

Os números são usados para representar valores numéricos. Isso pode ser um número inteiro ou um número de ponto flutuante.

2.1 - **Infinity** - é uma propriedade do *objeto global*, ou seja, é uma variável no escopo global. O valor inicial de Infinity é Number.POSITIVE_INFINITY. O valor Infinity (positivo) é maior do que qualquer outro número. Este valor se comporta matematicamente como infinito; por exemplo, qualquer número positivo multiplicado por Infinity é Infinity, e qualquer coisa dividida por Infinity é 0.

2.2 - **Nan** – Not a Number - é normalmente encontrado quando o resultado de uma operação aritmética não pode ser expresso como um número.

Exemplo:

```
var idade = 30;
```

```
var altura = 1.75;
```

3. Boolean (Booleano)

Os valores booleanos representam verdadeiro ou falso e são usados em lógica condicional.

Exemplo:

```
var aprovado = true;
```

```
var reprovado = false;
```

4. Undefined e Null

- Undefined representa uma variável que foi declarada, mas não foi inicializada.
- Null é usado para representar a ausência de valor.

Exemplos:

```
var valorNaoInicializado;
```

```
var valorNulo = null;
```

5. Symbol (Símbolo)

Símbolos são valores únicos e imutáveis, frequentemente usados como chaves de propriedades em objetos.

Exemplo:

```
var simbolo1 = Symbol('chave');
```

```
var simbolo2 = Symbol('chave');
```

Tipos de Referência

6. Object (Objeto)

Os objetos são estruturas de dados complexas que podem conter várias propriedades e métodos. Eles são fundamentais em JavaScript.

Exemplo:

```
var pessoa = {  
  nome: "Maria",  
  idade: 25,  
  cidade: "Lisboa"  
};
```

7. Array (Matriz)

Arrays são objetos especiais que permitem armazenar vários valores em uma única variável. Eles são indexados numericamente.

Exemplo:

```
var frutas = ["maçã", "banana", "laranja"];  
var numeros = [1, 2, 3, 4, 5];
```

8. Function (Função)

As funções são blocos de código reutilizáveis que podem ser chamados para executar tarefas específicas. São essenciais para a programação em JavaScript.

Exemplo:

```
function somar(a, b) {  
    return a + b;  
}
```

Tipagem Dinâmica

JavaScript é uma linguagem de tipagem dinâmica, o que significa que você não precisa declarar explicitamente o tipo de uma variável. O tipo é definido automaticamente quando você atribui um valor a ela.

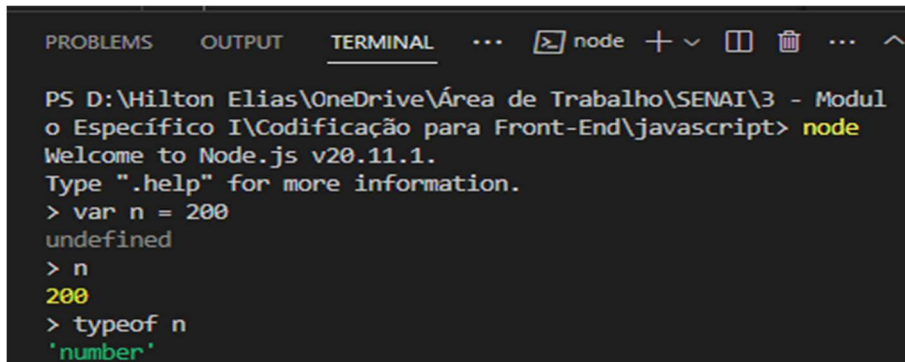
Exemplo:

```
var variavel = "Isso é uma string";  
variavel = 42; // Agora é um número
```

Para utilizar esses tipos de dados existe um comando chamado `typeof` (tipo de).

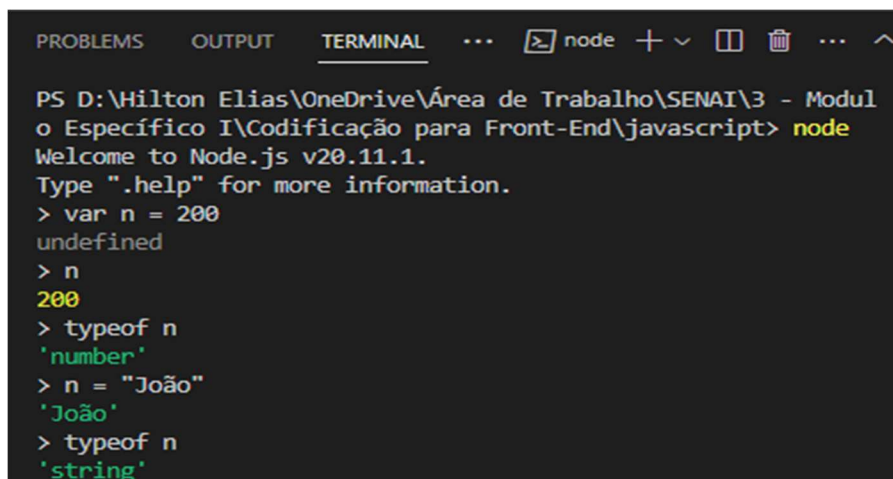
No node.js ou no terminal do Visual Studio Code, vamos testar:

- No Visual Studio Code, iniciamos digitando node, para iniciar o node.js dentro do terminal.
- Vou digitar um var n, recebe 200. Se mandar mostrar o n, ele vai dizer que é 200, mas quero saber qual é o tipo desse 200. Então vamos digitar, typeof n, assim vai me retornar, n é um number.



```
PROBLEMS OUTPUT TERMINAL ... node + - [ ] [ ] ... ^
PS D:\Hilton Elias\OneDrive\Área de Trabalho\SENAI\3 - Modul
o Específico I\Codificação para Front-End\javascript> node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var n = 200
undefined
> n
200
> typeof n
'number'
```

- No JavaScript, ele é um pouco diferente de outras linguagens, agora se eu declarar, n = "João", ele vai aceitar, e você fica em dúvida. O n é um número, como ele aceita João?



```
PROBLEMS OUTPUT TERMINAL ... node + - [ ] [ ] ... ^
PS D:\Hilton Elias\OneDrive\Área de Trabalho\SENAI\3 - Modul
o Específico I\Codificação para Front-End\javascript> node
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var n = 200
undefined
> n
200
> typeof n
'number'
> n = "João"
'João'
> typeof n
'string'
```

- Ao digitar novamente type of n, ele vai retornar string. Em JavaScript quando defino a variável n, não defino um tipo primitivo, somente falamos que ela é uma variável, um espaço de memória, pode aumentar e diminuir conforme o código for rodando.
- Podemos fazer direto assim, type of n, é um valor literal, que o JavaScript chama. Typeof 6 é literal, literalmente um número, não coloquei dentro de uma variável.

```
> typeof 6
'number'
> 
```

- Aqui você consegue fazer typeof, de qualquer os tipos. De qualquer valor ou de qualquer variável. Mas a princípio, vamos nos concentrar bastante nos tipos number, string, boolean e trabalhar bem de leve com o NULL e o UNDEFINED, diferenciando os entre si.

Agora que já conhecemos as variáveis, vamos voltar ao código onde criamos um prompt, exercício 2, onde demos um nome: qual o seu nome? Respondendo Fulano, clicamos no OK e ele simplesmente apareceu o restante do HTML com as mensagens: Meu primeiro Script e Exercício 2, o nome Digitado não serviu em nada, pelo simples fato: não fizemos nada com esse comando.

Vamos copiar esse exercício e renomear para exercício 4. Vamos pegar o resultado da execução dele e colocar dentro de uma variável nome, que recebe o resultado do prompt que irá aparecer na janela, o window.prompt.

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

Agora vamos estar com o nome guardado dentro da variável e tudo que eu digitar no prompt vai ser guardado dentro de nome.

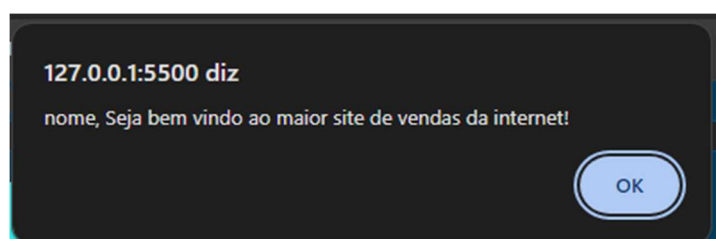
Vamos ver se funcionou utilizando outro comando que a já aprendemos que é o alert.

O window.alert, ou somente alert, e colocar ('nome, Seja bem vindo ao maior site de vendas da internet'), assim pensamos como era na lógica de programação, correto, estou dando o nome!

Nome, Seja bem vindo ao maior site de vendas da internet, certo?

Vamos executar:

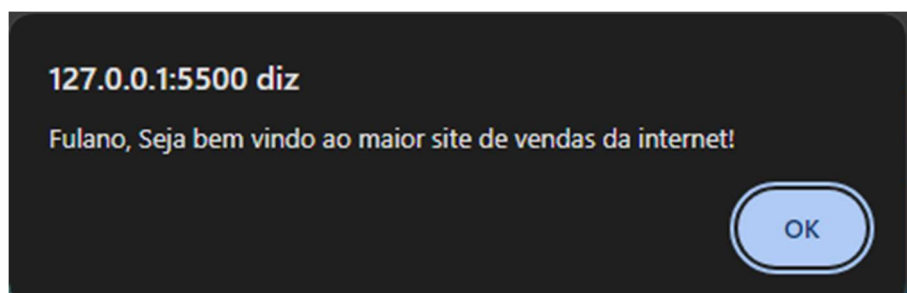
```
window.alert('nome, Seja bem vindo ao maior site de vendas da internet!')
```



Este erro é muito comum para quem está iniciando, eu não quero a palavra nome, eu quero o nome da pessoa, então vamos retirar essa variavel nome dentro das aspas, e colocar antes, para ela ficar colorida. E para unir tudo, usamos o sinal de +, que neste caso tem a função de concatenação.

```
window.alert(nome + ', Seja bem vindo ao maior site de vendas da internet!')
```

Vamos salvar e atualizar. Fulano, Seja bem vindo ao maior site de vendas da internet!



Agora, no lugar de ler o nome de uma pessoa, vamos digitar dois números, vamos copiar o exercício 5 e renomear par exercício 6 para isso vamos precisar de dois prompts, e criar duas variáveis n1 e n2, para o primeiro número:

```
var n1 = window.prompt('Digite um número:')
```

E ler outro número:

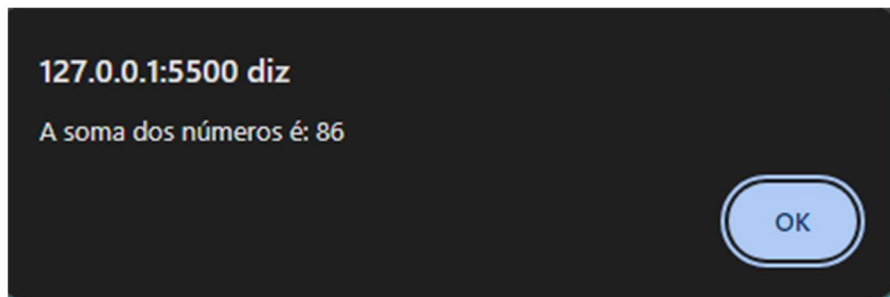
```
var n2 = window.prompt('Digite outro número:')
```

Com isso acabamos de ler dois números.

Vamos criar uma variável que recebera a soma de n1 mais n2, e fazer um window.alert

```
var s = n1 + n2  
window.alert('A soma dos números é: ' + s)
```

Vamos ver o resultado, digite um número, 8, digite outro número, 6.
Quando eu clicar em OK, vai aparecer que a soma entre os valores é 14!



Só não, vejam só: a soma dos valores é 86. Mas é simples, o que aconteceu aqui, mas não podemos ter 86, tem de dar 14.

Se olhar o exercício anterior veremos para que serve esse +, ele mais tem o efeito de concatenação.

E com isso começaremos a trabalhar os tipos de dados, isso porque o + pode servir para adição, e também pode servir para concatenação. O próprio JavaScript fica na dúvida. Se o + serve para adição e serve para concatenação também, como fazer para resolver? Como é que faz para forçar esse tipo, para eu dizer que N1 é um número e o N2 é um número?

```
var s = n1 + n2
```

Para entender o porque o JavaScript ficou confuso, temos que entender como ele funciona. Esse da soma, para ele ser adição, ele tem que ser um number de um lado e um number do outro. Se for string e string, ele vai fazer uma concatenação.

O que acontece é que o WindowPrompt retorna uma String, mesmo que eu digite um número, ele trata como um String, isso é uma característica do comando. Basicamente, N1 e N2, se eles recebem o valor do Prompt, eles recebem um valor String, o que precisamos fazer é converter de string para número.

E para isso, existem várias maneiras.

A primeira delas é utilizar o `Number.parseInt(n)` ou simplesmente `parseInt(n)`. Ele faz a conversão de um número para um número inteiro.

Se você quiser um número real, um número com vírgula, basta utilizar o `Number.parseFloat(n)` ou `parseFloat(n)`.

E ele se chama float, que é aquele ponto 5,5 ou 5.5 para o JavaScript é 5.5. Esse ponto do 5.5 chamamos de ponto flutuante ou floating point, por isso utilizar o `parseFloat`.

Parse é converter, parciar, então se eu quiser converter para inteiro ou real

E é importante saber que o N do number é maiúsculo, que o I e o F do int e do float, respectivamente, também são maiúsculos.

O JavaScript é case-sensitive, significa que caracteres em caixa alta e em caixa baixa são tratados de modo diferente. Por exemplo, as palavras sum e SUM são consideradas diferentes.

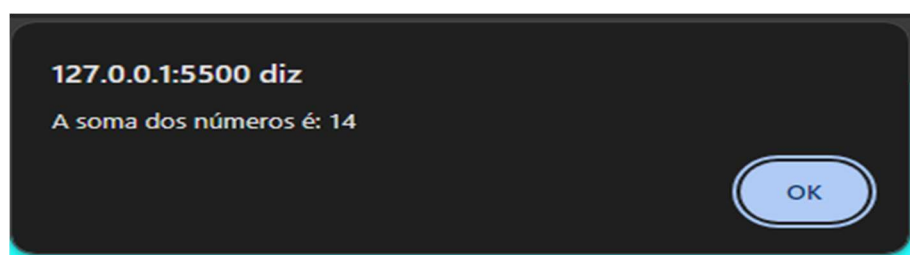
Vamos fazer com que o N1 seja inteiro e o N2 seja inteiro, vamos converter o N1 direto fazendo assim, `Number.parseInt` ou `parseInt`, temos que fechar os parênteses duas vezes, colocar tudo dentro de parênteses.

Dica: Para colocar tudo isso dentro de parênteses, seleciono tudo e abro parênteses, ele automaticamente já coloca dentro de parênteses, isso Visual Studio Code.

Basicamente o que eu estou fazendo aqui é, antes colocar dentro de N1, converter ele para inteiro. Então, a minha variável N1 vai receber o que vier do prompt da janela, que está escrito digite seu número, convertido para um número inteiro.

```
var n1 = Number.parseInt(window.prompt('Digite um número:'))  
var n2 = Number.parseInt(window.prompt('Digite outro número:'))
```

Vamos salvar e ver se está funcionando. Digitem um número, o mesmo 8 e o mesmo 6, quando clicar em OK, agora a soma dos valores é 14, agora está funcionando exatamente da maneira que queremos, porque fizemos a manipulação desses dados e fizemos a conversão deles.

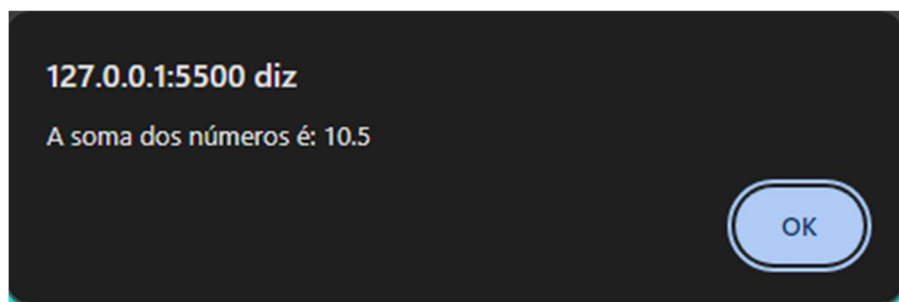


Mas com isso termos uma pequena limitação, se atualizar, digite um número, não foi dito se o número era inteiro ou real, então vou digitar 2.5 e aqui eu vou digitar 8. Se fizer 2.5 mais 8 o resultado será 10.5.

Mas quando eu der OK, a soma deu 10, porque não 10.5? Porque mandei converter para inteiro, se eu quiser real, eu vou usar float. Então eu tenho que saber mais ou menos o tipo que eu vou precisar.

```
var n1 = Number.parseInt(window.prompt('Digite um número:'))  
var n2 = Number.parseInt(window.prompt('Digite outro número:'))
```

Salvei, atualizei, vou fazer o mesmo exemplo agora, 2.5 com 8, ele deu 10.5.

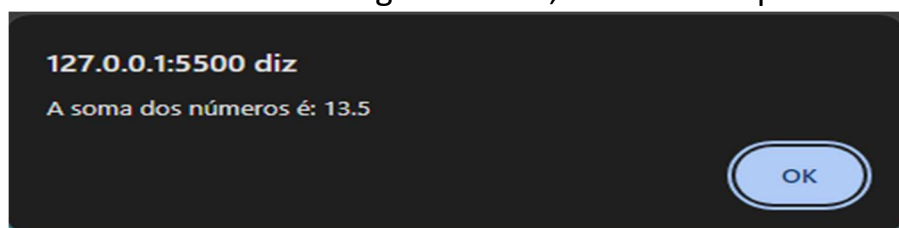


Agora sabemos que é preciso saber qual é o tipo. E isso aqui é uma grande realidade há muito tempo no JavaScript. Só que agora nas versões mais atuais, temos um modo simplificado, que é só utilizar Number(n). Coloco Number e ele se vira, ele sabe se é inteiro, ele sabe se é real e faz a conversão para o tipo definido. Mas essa síntese, só Number, não elimina as demais, porque às vezes é preciso efetivamente tratar como real, assim vou utilizar o parseFloat, se quiser efetivamente inteiro, forçar inteiro, vou utilizar o parseInt. Se eu quiser que o próprio JavaScript decida de acordo com o valor que ele recebeu, se é inteiro ou se é real, eu vou utilizar somente number.

Vamos fazer o teste, em vez de Number.parseFloat, eu vou só usar Number. Vamos ver se ele vai conseguir se virar:

```
var n1 = Number(window.prompt('Digite um número:'))  
var n2 = Number(window.prompt('Digite outro número:'))
```

Vamos ver se ele vai conseguir se virar, 5.5 + 8 tem que dar 13.5:



Desta forma, é uma maneira mais recente de fazer, está nas versões mais novas do ECMAScript.

E podemos também transformar de número para string, existem duas maneiras, a primeira é utilizar `string(n)`, assim como fizemos com o `number(n)`, posso utilizar `string(n)`, ou `string` o valor que quiser.

```
window.alert('A soma dos números é: ' + String(s))
```

A segunda é `n.toString()`, entre parênteses o que eu quero converter, vai gerar uma string, ou coloco o que eu quero converter, consigo girar ao contrário e jogar isso para uma string.

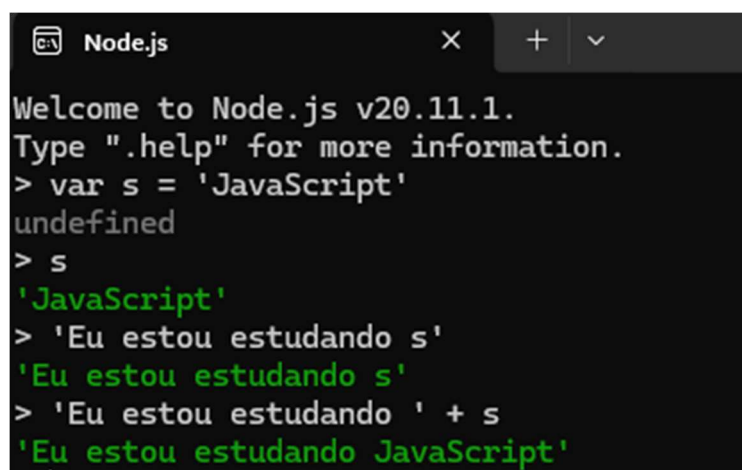
```
window.alert('A soma dos números é: ' + s.toString())
```

Com isso, agora vamos aprender como fazer formatação de novas strings, principalmente a partir do ECMAScript dos mais novos.

Vamos criar uma variável chamada `s`, e essa variável `s` está como valor JavaScript. Já aprendemos que se escrever: 'estou aprendendo `s`', não vai retornar **estou aprendendo JavaScript**, vai **mostrar estou aprendendo `s`**.

```
var s = 'JavaScript'  
'Eu estou aprendendo s'
```

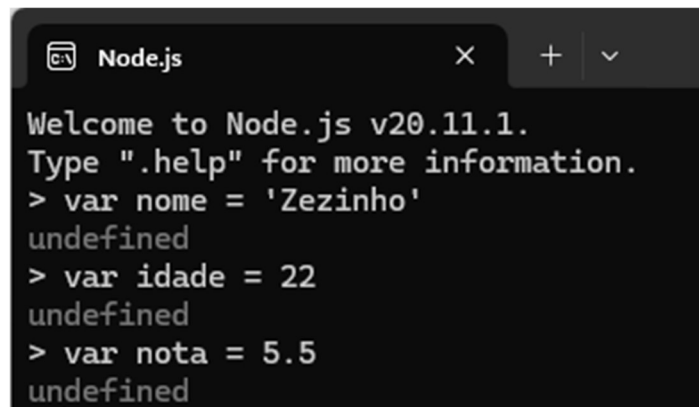
Vamos praticar no Node.



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var s = 'JavaScript'
undefined
> s
'JavaScript'
> 'Eu estou estudando s'
'Eu estou estudando s'
> 'Eu estou estudando ' + s
'Eu estou estudando JavaScript'
```

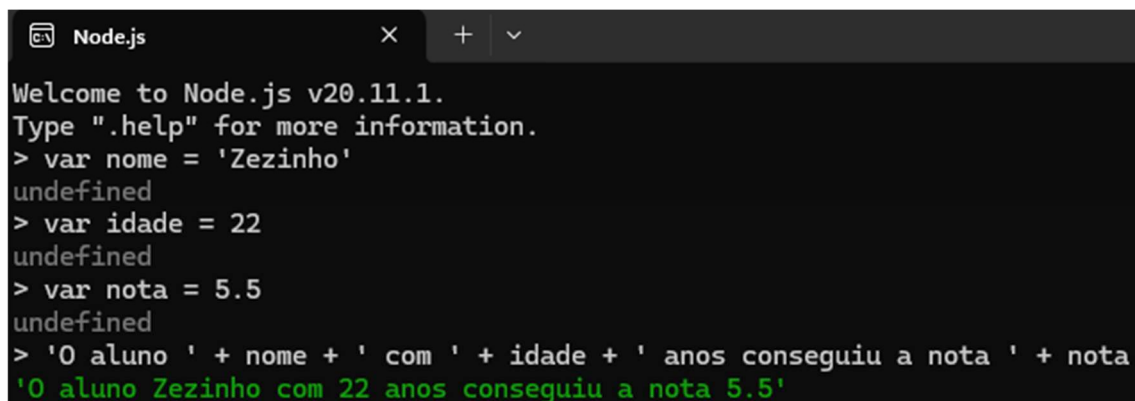
Como já sabemos o mais quando utilizando entre strings ele vai concatenar, e quando for entre números irá somar.

Vamos para outro exemplo, vamos criar três variáveis, nome, idade e nota.



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var nome = 'Zezinho'
undefined
> var idade = 22
undefined
> var nota = 5.5
undefined
```

Agora vamos fazer ele escrever assim, o aluno Zezinho de 22 anos conseguiu a nota 5.5.



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var nome = 'Zezinho'
undefined
> var idade = 22
undefined
> var nota = 5.5
undefined
> 'O aluno ' + nome + ' com ' + idade + ' anos conseguiu a nota ' + nota
'O aluno Zezinho com 22 anos conseguiu a nota 5.5'
```

Quando mandamos ele fazer:

```
var s = 'JavaScript'
```

'Eu estou aprendendo s' => não faz interpolação

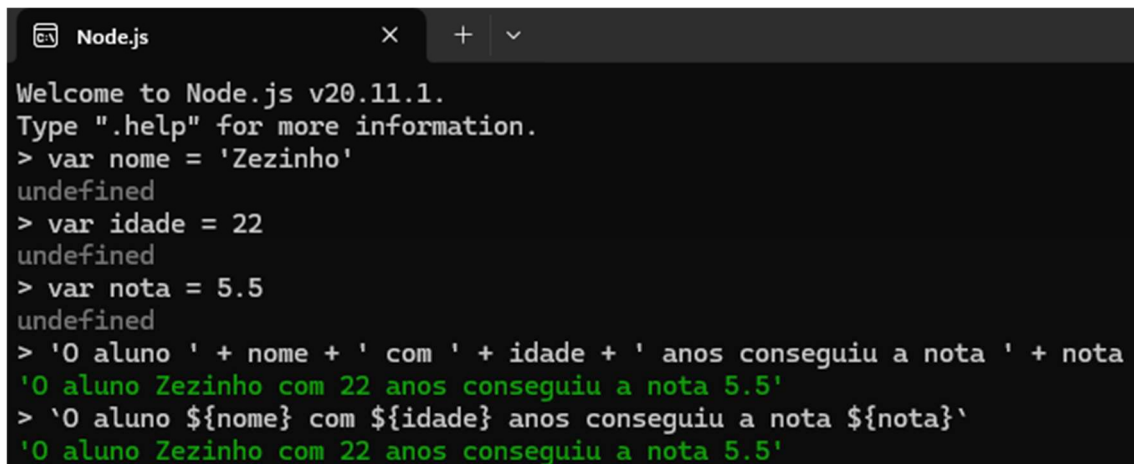
Mas quando mandamos fazer:

'Eu estou aprendendo' + s => ele faz a concatenação.

Nós acabamos de utilizar a concatenação, só que nas versões mais recentes de JavaScript tem uma forma muito fácil que se chama templatesStrings, formatador de strings.

`Eu estou aprendendo \${s}` => vamos utilizar crase, porque ela é a delimitação do que a chamamos de template string. Utilizar entre crases e utilizr o símbolo \$ seguido de {}, que no JavaScript se chama placeholder.

Vamos ver a diferença agora no node, vamos fazer um template string:



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var nome = 'Zezinho'
undefined
> var idade = 22
undefined
> var nota = 5.5
undefined
> 'O aluno ' + nome + ' com ' + idade + ' anos conseguiu a nota ' + nota
'O aluno Zezinho com 22 anos conseguiu a nota 5.5'
> `O aluno ${nome} com ${idade} anos conseguiu a nota ${nota}`
'O aluno Zezinho com 22 anos conseguiu a nota 5.5'
```

Como podem perceber, o comando, ficou bem menor, a organização fica melhor, retorna exatamente o mesmo resultado, na parte de cima utilizando concatenação, e embaixo utilizando template string, que é uma novidade das últimas versões do ECMAScript e que os navegadores mais atuais estão totalmente compatíveis com ela.

Lembrete: Nunca se esqueça, essa é a forma que vamos utilizar, aspas simples para string simples, ou aspas duplas também para string simples, sem interpolação, e nós vamos utilizar template strings, quando quiser interpolar, utilizando o placeholder.

E outras coisas podem ser feitas com string, utilizando essa mesma variável `s` que já criamos, posso utilizar `s.length`, não tem parênteses no final, porque é um atributo, mais pra frente vou exemplificar melhor, ele me diz qual é o tamanho da string, quantas letras tem essa string.

Outra coisa que eu posso fazer, é transformar para letras maiúsculas, utilizando o método, dessa vez tem parênteses, `toUpperCase()`, lembrando que o `U` e o `C`, também são em letras maiúsculas.

E por último o `toLowerCase()` vai transformar para letras minúsculas.

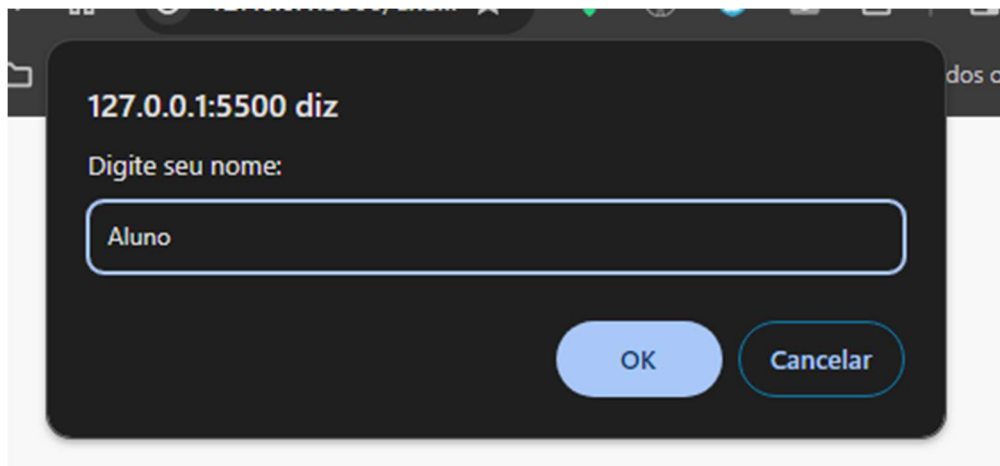
Agora vamos praticar, abra um novo documento no Visual Studio Code e nomeie de `exercico8.html`.

Abra um HTML simples, dentro do <body>

Vamos criar uma variável nome, e depois escrever a resposta na tela com o comando document.write.

```
<h1>Aula de JavaScript</h1>
<span>Exercicio 8</span>
<br>

<script>
  var nome = window.prompt('Digite seu nome:')
  document.write(`Seu nome tem ${nome.length} letras.`)
</script>
```



E posso utilizar também tags HTML. Por exemplo, posso colocar um <h2>.

```
<h1>Aula de JavaScript</h1>
<span>Exercicio 8</span>
<br>

<script>
  var nome = window.prompt('Digite seu nome:')
  document.write(`<h2>Seu nome tem ${nome.length} letras.</h2>`)
</script>
```

Aula de JavaScript

Exercicio 8

Seu nome tem 5 letras

Mas o jeito certo de fazer, é criar uma tag <style>, dentro colocar o body, com fonte normal 20pt, Arial, ponto e vírgula.

Vamos melhorar, Colocar um Olá, um placeholder para o nome e ! Seu nome tem tantas letras.

```
<script>
  var nome = window.prompt('Digite seu nome:')
  document.write(`Olá, ${nome}! Seu nome tem ${nome.length}
letras`)
</script>
```

Aula de JavaScript

Exercicio 8

Olá, Aluno! Seu nome tem 5 letras

```
<script>
  var nome = window.prompt('Digite seu nome:')
  document.write(`Olá, ${nome}! Seu nome tem ${nome.length}
letras<br>`) //br para quebrar a linha
  document.write(`Seu nome em maiúsculas é
${nome.toUpperCase()}`)
</script>
```

Aula de JavaScript

Exercicio 8

Olá, Aluno! Seu nome tem 5 letras

Seu nome em maiúsculas é ALUNO

```
<script>
  var nome = window.prompt('Digite seu nome:')
  document.write(`Olá, ${nome}! Seu nome tem ${nome.length}
letras<br>`)
  document.write(`Seu nome em minúsculas é ${nome.toLowerCase()}`)
</script>
```

Aula de JavaScript

Exercicio 8

Olá, Aluno! Seu nome tem 5 letras
Seu nome em minúsculas é aluno

E podemos também entrar com a informação toda bagunçada, que na saída em maiúscula ou minúscula ele irá corrigir, Veja o exemplo ConCeiçãO:

Aula de JavaScript

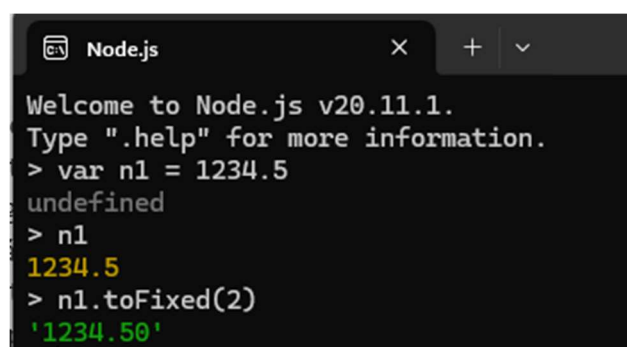
Exercicio 8

Olá, ConCeiçãO! Seu nome tem 9 letras
Seu nome em minúsculas é conceição
Seu nome em maiúsculas é CONCEIÇÃO

Essas são algumas funcionalidades pequenas para formatar strings. E para finalizar, vamos aprender a formatar números.

Vamos considera a formatação de números uma variável n1 com 1234.5. Vamos abrir o Node.

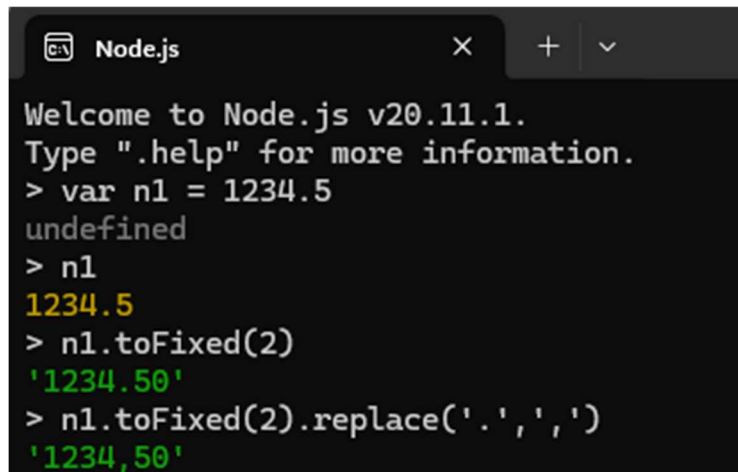
Vamos escrever esse n1 com duas casas decimais, utilizando o toFixed() para fixar duas casas.



```
Node.js x + v
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var n1 = 1234.5
undefined
> n1
1234.5
> n1.toFixed(2)
'1234.50'
```


Mas no Brasil a gente usa bastante vírgula, então podemos fazer assim, `n1.toFixed(2).replace(",")`, eu vou trocar uma coisa por outra, aqui entre aspas.

Vamos trocar o ponto por vírgula, aqui temos que fazer um esforço um pouco maior para fazer a conversão de números e com isso acabamos de aprender o `toFixed`, que é um método interno de todos os `numbers`.



```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> var n1 = 1234.5
undefined
> n1
1234.5
> n1.toFixed(2)
'1234.50'
> n1.toFixed(2).replace('.', ',')
'1234,50'
```

Agora vou passar uma dica bem valiosa, quem não é muito simples, na verdade ele é bem avançado, vamos supor que 1234,50 fosse o salário de uma pessoa, como é que faz para mostrar o real.

Vamos formatar da seguinte maneira, `n1.toLocaleString()` que é uma string localizada, que é por parte do mundo, e eu vou localizar por pt-BR, em português, vírgula e entre chave {} vou colocar as configurações, os atributos, isso aqui é um objeto em JavaScript, vamos ver isso mais para frente. Mas aqui dentro vou colocar algumas configurações, como por exemplo, o meu `style` vai ser `currency`, que tem que ser uma string, ele mostrar em valor monetário, vírgula `currency` vai ser real do Brasil, BRL.



```
> n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})
'R$ 1.234,50'
```

N1, vai ser local string, 'ptBR', que quer dizer, quero localizar para o Brasil e o estilo vai ser `currency`, que é dinheiro, e `currency` vai ser Brasil-Real.

Se eu quiser em dólar, basta colocar o USD, ele já vai ficar em dólares, se você colocar EUR, ele vai ficar em euro.

```
> n1.toLocaleString('pt-BR', {style:'currency', currency: 'USD'})
'US$ 1.234,50'
> n1.toLocaleString('pt-BR', {style:'currency', currency: 'EUR'})
'€ 1.234,50'
```

Aqui no Node ele não faz a inversão, ele coloca a vírgula antes e o ponto depois. No seu navegador, ele vai fazer direto com a vírgula no final. Você não precisa utilizar o replace, mas se quiser você pode aqui no final botar replace e fazer exatamente como a gente fez antes.

3 . OPERADORES

O JavaScript possui várias famílias de operadores, mas não vamos focar em todas as famílias. Vamos falar sobre **os operadores aritméticos, os operadores de atribuição, os operadores relacionais, os lógicos e o operador ternário.**

Existem outras famílias de operadores dentro do JavaScript, inclusive o typeof, é um tipo de operador.

Vamos iniciar com os aritméticos e os operadores de atribuição.

Esses operadores

+	-	*	/	%	**
---	---	---	---	---	----

São todos os operadores aritméticos da linguagem JavaScript. Eles são os operadores usados para fazer cálculos.

E para exemplificar cada um deles, vou colocar esses operadores e os operandos, que são aqueles que vão fazer os operadores funcionarem.

Então todos esses operadores são operadores que a gente chama de binários, são operadores que são de dois operandos.

No caso coloquei 3 e 2 em todos para poder ver o resultado da operação de cada um deles e o resultado na tela, que seria para poder mostrar um resultado.

3	+	2	=	5	Soma
3	-	2	=	1	Subtração
3	*	2	=	6	Multiplicação
3	/	2	=	1.5	Divisão
3	%	2	=	1	Resto divisão inteira
3	**	2	=	9	Potencia

Tomem bastante cuidado com o uso dos operadores, isso porque tanto na programação quanto na matemática existe uma coisa chamada

precedência de operadores, exemplo $5 + 3 / 2$, quando uma mesma expressão tem soma e divisão primeiro, a gente faz primeiro a divisão.

Vamos testar no Node e vamos fazer os exemplos.

```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 3+2
5
> 3-2
1
> 3*2
6
> 3/2
1.5
> 3%2
1
> 3**2
9
```

E agora vamos fazer o $5 + 3 / 2$. Você vai fazer $5 + 3 = 8 / 2 = 4$, mostrar 4, aí ele mostra 6,5. Por quê? Porque ele fez o $3/2 = 1,5 + 5 = 6,5$. Mas tem programa que deu erro, o JavaScript não deu erro, por que minha conta deu errado, quando você programa, a linguagem só dá erro se você tiver um erro sintático.

```
Node.js
> 5+3/2
6.5
```

Vamos tentar fazer um erro sintático $5 + / 2$. Ele me dá um erro, a minha expressão regular deu erro. $5 + / 2$ é um erro, inclusive ele marcou aqui,

```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 5+/2
5+/2
  ^
Uncaught SyntaxError: Invalid regular expression: missing /
```

No primeiro caso a expressão está correta, sintaticamente correta, aritmeticamente errada se você queria calcular a média entre 5 e 3. Mas na prática, esse comando não tem erro.

Agora se quiser fazer $5 + 3 / 2$, quero fazer $5 + 3$ primeiro e depois dividir por 2, basta colocar em parênteses. Se você utilizar os parênteses, você muda a ordem de precedência, isso é, você vai fazer primeiro $(5 + 3)$, que vai dar 8, dividido por 2, o resultado vai ser 4,

```
Node.js
> (5+3)/2
4
```

E com isso temos um novo assunto, que é a ordem de precedência dos operadores. Não só em linguagem JavaScript, mas em qualquer linguagem de programação a ordem é sempre essa.

Em qualquer expressão, em primeiro lugar, serão analisados todos os parênteses, segundo lugar as potências, terceiro lugar a multiplicação, a divisão e o resto da divisão, eles têm a mesma ordem de precedência, não necessariamente a multiplicação vem antes, qualquer um desses três vem antes, e se por acaso mais de um deles estiver na mesma expressão, você vai fazer da esquerda para a direita, quem aparecer primeiro, e por fim, as somas e as subtrações.

()		
**		
*	/	%
+	-	

Agora vamos guardar os valores em algum lugar, vamos precisar usar um operador que a já vimos, que é o operador de atribuição.

Vamos declarar algumas variáveis var a até f.

var a = 5 + 3	8	
var b = a % 5	3	
var c = 5 * b ** 2	45	Faz a potencia
var d = 10 - a / 2	6	Divide primeiro
var e = 6 * 2 / d	2	Faz da esquerda para direita
var f = b % e + 4 / e	3	Faz o resto da divisão e divisão depois soma

Vamos fazer no node.

```
Node.js
> var a = 5 + 3
undefined
> var b = a % 5
undefined
> var c = 5 * b ** 2
undefined
> var d = 10 - a / 2
undefined
> var e = 6 * 2 / d
undefined
> var f = b % e + 4 / e
undefined
```

O var f, sabemos que seu valor deu 3, ele vai dar um undefined, basta mostrar o f, que está valendo 3.

```
> a
8
> b
3
> c
45
> d
6
> e
2
> f
3
```

Outra coisa que podemos fazer são as auto atribuições, são atribuições à própria variável. Por exemplo:

```
var n = 3
n = n + 4
n = 7
```

Vamos para o node praticar.

```
Node.js
> var n = 3
undefined
> n = n + 4
7
> n = n - 5
2
> n = n * 4
8
> n = n / 2
4
> n = n ** 2
16
> n = n % 5
1
```

Tudo isso que aconteceu aqui, fez a variável n partir de 3, assumir vários valores e terminar com 1, isso são auto atribuições.

Mas podemos simplificar todas essas auto atribuições, a primeira não é auto atribuição, é atribuição simples.

Se eu pego uma variável, por exemplo, $n = n + 4$, o segundo da lista, podemos reescrever de uma maneira encurtada, que é o $n += 4$, mas só serve, se a variável, receber ela mesma, ai somo + 4.

Vamos usar bastante essa síntese do += , pois a grande maioria das linguagens de programação aceita esse tipo de autorreferência.

```
Node.js
> var n = 3
undefined
> n += 4
7
> n -= 5
2
> n *= 4
8
> n /= 2
4
> n **= 2
16
> n %= 5
1
```

Vamos aprender mais dois operadores muito usados, que são os operadores de incremento.

```
var x = 5
x = x + 1    x++
x = x - 1    x--
```

Vamos praticar no Node.js

Declarar uma variável n valendo 10, se eu mandar mostrar n, mostrou 10. Agora, se eu fizer n++, ele mostrou 10, mas se eu mandar mostrar n agora, ele está valendo 11, se fize n--, parece que ele está valendo 11, mas se eu mandar mostrar n, ele está valendo 10.

```
Node.js
> var n = 10
undefined
> n
10
> n++
10
> n
11
> n--
11
> n
10
```

Isso ocorre porque nas linguagens de programação tem como ++ virantes, que seria um pré-incremento.

Por exemplo, vou mostrar o valor de n, 10. Se no lugar de n++ eu colocar ++n, ele já soma antes, se eu colocar --n, ele já tira antes.

```
> n
10
> ++n
11
> --n
10
```

Mas não significa que n++ está errado, isso é só uma questão de ordem. Existe o pré-incremento e o pós-incremento, assim como existe o pré-decremento e o pós-decremento.

Agora vamos aos **operadores relacionais, lógicos** e o **operador ternário**. Os **operadores relacionais** do JavaScript são os **operadores relacionais** da grande maioria das linguagens de programação.

São os operadores de maior, menor, maior ou igual, menor ou igual, igual ou diferente.

>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual
==	Igual
!=	Não igual

Agora no exemplo, vamos adicionar valores, vou usar inteiros, mas podem ser valores reais. Vamos descobrir o resultado de todas essas expressões.

É importante dizer, que para toda expressão que tenha um operador relacional ligado a ela, o resultado dessa expressão vai ser sempre um valor **booleano**, será do tipo **verdadeiro** ou **falso**.

5	>	2	true	Maior que
7	<	4	false	Menos que
8	>=	8	true	Maior ou igual
9	<=	7	false	Menor out igual
5	==	5	true	Igual
4	!=	4	false	Não igual

Lembrete, o sinal de =, a igualdade no JavaScript não é um sinal de igual, e um símbolo de igual no JavaScript, é a mesma coisa que muitas outras linguagens de programação, a gente lê como **recebe**, ele é um **operador de atribuição**, ele não é um **operador de igualdade**. O **operador de igualdade** do JavaScript é o ==, e essa != quer dizer **não igual**.

Vamos para o Node.js exercitar.

```
Node.js
Welcome to Node.js v20.11.1.
Type ".help" for more information.
> 5 > 2
true
> 8 < 4
false
```

Podemos fazer com variáveis:

<pre>var a = 8 var b = 15 a > b a < b</pre>	<pre>Node.js > var a = 8 undefined > var b = 15 undefined > a > b false > a < b true</pre>
---	--

Quando temos **operadores relacionais** e **operadores aritméticos** na mesma expressão, primeiro fará os aritméticos, depois os relacionais.

<pre>a <= b - 10 8 <= 15 - 10 8 <= 5</pre>	<pre>Node.js > a <= b - 10 false</pre>
---	--

Vejam outros exemplos:

preço >= 200.50	O preço é maior ou igual a 200.50?
idade < 18	A idade é menor que 18?
curso == 'JavaScrip'	O curso é igual a JavaScript?
n1 != n2	n1 é diferente de n2?

Vejam que, neste caso do **curso == JavaScript**, podemos fazer a comparação de variáveis **string** com **string**. O **curso** é igual a **string JavaScript**.

Agora vamos falar sobre os **operadores relacionais**, conhecidos por **operadores de identidade**.

Veja no exemplo:

5 == 5	true
5 == '5'	true

Para o JavaScript, vai ser igual, isso porque o sinal de igualdade não testa o **tipo**, mesmo sabendo que temos o valor **5 inteiro** e temos o valor **5 em string**.

O que o JavaScript faz é analisar se esses 5 têm o mesmo valor que o outro, e são dois valores de tipos diferentes, mas eles têm a mesma grandeza.

E para resolver este problema, existe um outro **operador que é o de identidade**, também conhecido como **operador de igualdade restrita**, que no lugar de dois sinais de igual ==, são três sinais de igual ===.

Nesse caso específico, estou testando se os dois 5, são idênticos. E idêntico é o mesmo valor e do mesmo tipo, e neste caso, eles têm o mesmo valor, mas eles não têm o mesmo tipo.

5 === '5'	false
5 === 5	true

Vamos para o Node.js exercitar.

```
Node.js
> 5 == '5'
true
> var x = 5
undefined
> var y = '5'
undefined
```

Se testar typeof de x é number e typeof de y é string, então eles são de tipos diferentes.

```
> typeof x
'number'
> typeof y
'string'
> x == y
true
> x === y
false
```

O mesmo acontece se colocar `x != y`, e se colocar `x !== y` é **desigual restrito** que se chama no JavaScript, ele dá true.

O **desigual restrito** é que eles são de tipos diferentes, mas o valor interno é o mesmo.

```
Node.js
> x != y
false
> x !== y
true
```

Agora veremos os **operadores lógicos**, que no JavaScript são três:

1 - a exclamação;	!
2 - os dois es comerciais e	&&
3 - os dois pipes.	

Em primeiro lugar, vamos à **exclamação, !**, sempre que ela aparecer, quer dizer **negação, não**.

Sempre que aparecer em dois **es comerciais, &&**, isso é uma **conjunção ou** chama de **E**, que é nosso **E lógico**.

E os dois **pipes, ||**, são **disjunções**, ou então é o nosso **OU lógico**.

Exemplos:

1. Se eu peço uma caneta, mas ela **NÃO pode ser azul**. Basta entregar uma caneta de qualquer cor, **menos da cor azul**.
2. Se eu digo, eu quero uma **caneta azul E uma caneta vermelha**. Eu só vou ficar satisfeito se você me entregar **as duas**, porque eu quero **uma azul, E uma vermelha**.
3. Se eu digo, quero uma **caneta azul OU uma caneta vermelha**. Se entregar **as duas, está ótimo**, queria uma ou outra, ganhei as duas. Se entregar **azul, ok**. Se eu te entregar **vermelha, ok** também. **Se eu não te entregar nenhuma ou de uma cor que você não quer, aí sim eu não fico satisfeito**.

Vamos começar pelo operador de **negação**, o **não**, ele é tratado como um **operador unário**, isso é, ele só tem **um operando**, depois da exclamação, ou é **true** ou é **false**, ou uma expressão que vai resultar em **true** ou **false**.

	true	false
!	false	true

Na primeira linha, uma coisa que não é verdadeira é falso e depois uma coisa que não é falsa é verdadeira.

Agora para o operador de conjunção, que são os dois ex comerciais, **&&**. Esse operador, assim como o de disjunção também, é um operador binário, isso é, eu tenho dois valores lógicos, um de cada lado, e aí vão me dar um resultado lógico dado o resultado dessa expressão.

true	&&	true	true
true		false	false
false		true	false
false		false	false

Na primeira linha, **true** e **true**, isso é, verdadeiro e verdadeiro. Quero uma caneta azul e uma caneta vermelha, eu consegui as duas, fico satisfeito. Para qualquer outra combinação, se consegui a vermelha, mas não consegui a azul, se eu consegui a azul, mas não consegui a vermelha, ou se eu não consegui nenhuma delas, ou de uma outra cor, para todos esses outros casos, a minha resposta é **falsa**.

E por fim, a **disjunção**, que são os dois **pipes**, **||**. Também é um **operador binário**, que tem dois **valores lógicos**, um de cada lado. Dependendo da posição deles e do resultado lógico, eu tenho também um valor lógico de resultado. E na **disjunção**, **basta que um deles seja verdadeiro para o resultado ser verdadeiro**.

true		true	true
true		false	false
false		true	false
false		false	false

Por exemplo, a primeira linha, os dois são verdadeiros, então o resultado é verdadeiro também, na segunda linha primeiro é verdadeiro, mas o segundo é falso, mas basta que um seja verdadeiro para me dar o resultado verdadeiro. Na terceira linha, o segundo é verdadeiro. mas o primeiro é falso, mesmo assim verdadeiro como resultado, e no último não tem nenhum verdadeiro, aí sim o meu resultado é falso.

Vamos fazer alguns exemplos práticos no Node.js:

```
var a = 5
var b = 8
a > b && b % 2 == 0
```

Node.js

```
> var a = 5
undefined
> var b = 8
undefined
> a > b && b % 2 == 0
false
```

Antes de iniciar é preciso entender que, quando temos **operadores aritméticos, relacionais e lógicos na mesma expressão**, Primeiro, vou fazer todos os **operadores aritméticos** (**%**, **>**), depois eu vou fazer os **operadores relacionais** (**==**), e depois eu vou fazer os **operadores lógicos** (**&&**).

Vamos primeiro ao operador aritmético **b % 2**, o **b** valendo 8 que dividido por 2 dá 4, resta 0. Esse lado deu verdade (**0 == 0**). Agora tenho um **E** aqui, mas eu tenho um **operador relacional**. O (**5 > 8**), então esse lado dá falso. Se esse lado dá falso e esse lado da verdadeiro, é a mesma coisa que testar, falso e verdadeiro, que dá falso.

```
> a > b && b % 2 == 0
false
```

Outro exemplo:

```
a <= b || b / 2 == 2
```

Node.js

```
> a <= b || b / 2 == 2
true
```

Primeiro faço os **operadores aritméticos**, então vou fazer primeiro (**b / 2**), que é 4, é igual a 2? Não, então isso deu **falso**. Agora eu tenho o **operador relacional**, o **a** é menor ou igual a **b**? (**5 <= 8**) então esse lado dá **verdadeiro**. Neste caso, tendo um **verdadeiro** e um sendo **falso**, tendo um **OU** como comparação no meio, basta que um seja **verdadeiro** para o resultado ser **verdadeiro**.

Lembrete, se por acaso em uma mesma expressão tiver um **E**, um **OU** e um **NÃO**, a ordem de execução será sempre essa: primeiro o **NÃO**, depois o **E**, depois o **OU**.

Vejam outros exemplos:

idade >= && idade <= 17

idade maior ou igual a 15 e idade menor ou igual a 17

estado == 'RJ' estado == 'SP'	o estado é Rio de Janeiro ou o estado é São Paulo
salario > 1500 && sexo != 'M'	salário é maior do que 1.500 e sexo é diferente de masculino

Agora que vimos uma quantidade maior de operadores, vamos ver como fica a ordem de precedência, sempre de cima para baixo, dentro de uma expressão.

Em primeiro lugar, vamos fazer os **operadores aritméticos** (parênteses, o asterisco, a exponenciação, a divisão, a multiplicação, o módulo).

Depois que fizer todos os **operadores aritméticos**, você vai para os **operadores relacionais**, que não têm **ordem de precedência**, quem aparecer primeiro vai ser feito primeiro, a leitura é da esquerda para a direita, e por fim, vamos fazer os **operadores lógicos**, primeiro, o **NÃO**, depois o **E**, e por último o **OU**.

Operadores aritméticos	() , ** , * , / , % , + , -
Operadores relacionais	> , < , >= , <= , == , !=
Operadores lógicos	! , && ,

E para finalizar, vamos falar sobre o **operador ternário**, ele é composto por dois símbolos que aparecem na mesma expressão, a interrogação, **?**, e os dois pontos, **:**.

? **:**

Ele se chama **ternário** porque ele tem **três partes**.

teste **?** true **:** false

Esses três blocos são os blocos de **teste**, o bloco **verdadeiro** e o bloco **falso**. Por isso se chama operador ternário, porque ele une **três operandos**.

O primeiro, é um teste lógico, é um resultado que dá **verdadeiro** ou **falso**.

O que está no meio é o que vai acontecer quando esse teste lógico for **verdadeiro** e no final, o que vai acontecer quando esse teste lógico for **falso**.

Vamos a um exemplo prático:

<code>media >= 7</code>	<code>?</code>	<code>'Aprovado'</code>	<code>:</code>	<code>'Reprovado'</code>
----------------------------	----------------	-------------------------	----------------	--------------------------

Vamos fazer um teste no Node.js:

<code>var media = 5.5</code>	<pre>> var media = 5.5 undefined > media >7?'Aprovado': 'Reprovado' 'Reprovado'</pre>
------------------------------	--

O resultado do teste lógico, que é esse aqui, é falso. Então ele vai cair aqui para o reprovado.

Vamos modificar a média:

<code>media += 3</code>	<pre>> media += 3 8.5 > media >7?'Aprovado': 'Reprovado' 'Aprovado'</pre>
-------------------------	--

Agora a minha média, é 8.5, se aplicar o mesmo comando no Node.js, agora a média é maior do que 7, então, ele me mostra aprovado.

Outro exemplo:

<code>var x = 8</code> <code>var res = x % 2 == 0 ? 5 : 9</code>	<pre>> var x = 8 undefined > var res = x % 2 == 0 ? 5 : 9 undefined > res 5</pre>	
---	--	--

Primeiro faço os **operadores aritméticos**, o **operador ternário** é sempre o último a ser feito, vai fazer $(x \% 2 == 0)$. Se o $x \% 2$ é igual a zero, o $x \% 2$ é $8 / 2$ que dá 4 e resta 0.

Zero é igual a zero? Sim, **verdade**.

Se é verdade, vou executar a primeira parte, assim o 5 que vai ser atribuído para res.

Mais um exemplo:

```
var idade = 19
var r = idade >= 18 ? 'Maior' : 'Menor'
```

```
> var idade = 19
undefined
> var r = idade >= 18 ? 'Maior' : 'Menor'
undefined
> r
'Maior'
```

A minha variável idade vai receber 19 anos, vou fazer a minha variável r receber a idade é maior ou igual a 18? Interrogação, maior, dois pontos, menor.

O que acontece aqui?

A minha variável r vai receber ou **Maior** ou **Menor**, vai depender se a minha idade é **maior ou igual a 18**.

A minha idade é **19, é maior ou igual a 18?**

Verdade, então ele vai pegar a palavra maior, e colocar dentro da variável r.