JAVASCRIPT

O que é JavaScript?

Linguagem de programação de alto nível;

Recebeu o nome por causa da linguagem JAVA, que estava no topo do mercado na época;

Entenda que: JavaScript = JS = Vanilla JavaScript;

Sua principal função é deixar a página viva;

adicionando comportamentos (alterações de HTML e CSS) através de eventos;

JavaScript é case sensitive;

Onde JavaScript é utilizado?

Interação com a página, HTML e CSS, através do DOM;

Cálculo, manipulação, e validação de dados;

Também é empregada no server-side, com Node.js;

As principais bibliotecas de Front-end são baseadas em JS (React, Vue, Angular , Svelte…).

Formas de executar JavaScript

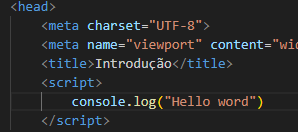
Há diversas formas de executar JavaScript;

Padrão arquivo importado no Html;

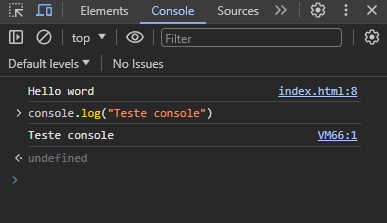
Diretamente no navegador, através do console;

Por meio de aplicações, como o JS Fiddle;

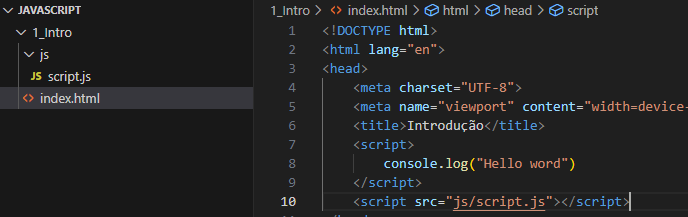
Modo 1: Dentro da tag html “<script>”:



Modo 2: Pelo console dentro do navegador:



Modo 3: Pelo arquivo file importando para o Html:



Modo 4: usando aplicativos de teste online como exemplo:

<https://jsfiddle.net/>

# Tipos de dados e Operadores:

## O que são tipos de dados?

É a forma de classificar um dado;

Temos como dado: “Matheus”, 15, true, [];

Os tipos de dados mais comuns são:

* Number;
* String;
* Boolean;
* Empty values (null, undefined);
* Object;

## $Number

Number é o tipo de dado para valores numéricos;

Em JS todos os números são considerados Number;

Sejam eles: Inteiros, ponto flutuantes ou negativos;

Alguns exemplos: 10, 52.5 - 12;

Note que nas linguagens de programação as casas decimais são após o caractere ponto (15.8);

Em JavaScript o operador typeof exibe o tipo do dado;



### Aritmética com Numbers

Podemos realizar operações aritméticas na programação;

Operadores como: +, -, \*, / podem ser utilizados;

Como um exemplo: console.log (2 + 5);

A ordem matemática também é respeitada na programação, exemplo: console.log(5 + (4\*12)): explicando o exemplo: 1º Parêntesis; 2º Expoentes; 3º Multiplicações e Divisões; e depois soma e subtração (da esquerda para a direita);

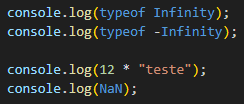
### Special Numbers

Special Numbers são dados considerados como números, mas não funcionam como eles;

Eles são:

* Infinity;
* -Infinity;
* NaN (Not a Number);

Algumas operações podem resultar nestes valores;



## $String

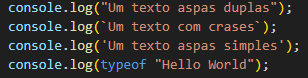
Strings são textos;

Em JavaScript temos três formas de criar dados de texto;

Aspas simples, duplas e crases;

Desta maneira: console.log(“Hello World”);

O “O efeito final” é o mesmo, mas cada um destes recursos tem suas particularidades;



### Mais sobre string

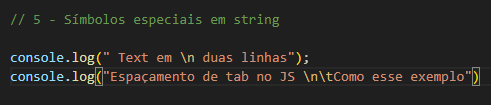
Uma string deve sempre começar e terminar com o mesmo caractere (“” | ’’ | ``);

Há algumas combinações de caracteres que tem efeitos interessante nas strings;

Por exemplo o \n, ele pula uma linha no texto;

Veja um exemplo: console.log (“Text em \n Duas linhas”);

Ou o espaçamento de tab (“Text em \t Duas linhas”);



### Concatenação

Concatenação é o recurso que une dois ou mais textos;

O operador da concatenação é o +;

Exemplo: “Meu” + “texto” + “combinado”;

Agora o recurso pode não fazer tanto sentido, mas com variáveis teremos um melhor uso para ele;



## $Interpolação (Template String)

**Ou podemos chamar de Máscara**

A interpolação é um recurso semelhante a concatenação;

Mas nos possibilita a escrever tudo na mesma string;

Esta deve ser escrita `entre crases`;

Podemos executar código JavaScript com ${algum código};



## $Booleans

Os booleans possuem apenas dois valores: true ou false;

Qualquer comparação, utilizando os sinais >, <, ==, resulta em um valor booleano;

Podemos relembrar que este tipo é importante para estruturas de condição e repetição;



## $Comparações

As comparações que podemos utilizar são:

* Maior e menor: > e <;
* Maior ou igual e menor ou igual: >= e < =;
* Igual: ==;
* Diferente: !==;
* Idêntico: ===;

Lembrando que “>” seria o símbolo de maior como exemplo:

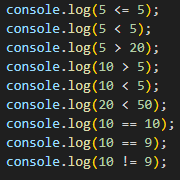
30 > 10 == true;

30 > 40 == false;

Lembrando que “<” seria o símbolo de menor como exemplo:

30 > 10 ==false;

30 > 40 == true;



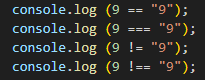
### Comparação de idêntico

Os operadores === e !== funcionam como == e !==;

Porém também levam em consideração o tipo do dado;

Estes operadores necessitam que o tipo e o dado sejam iguais/diferentes;

Devemos tentar ao máximo utilizar estes operadores;



## $Operadores lógicos

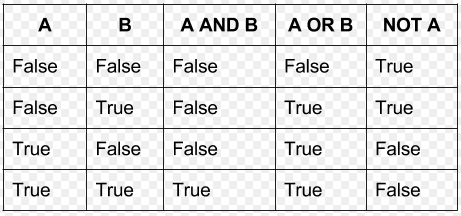
Os operadores lógicos servem para unir duas ou mais comparações;

O resultado final também é um boolean;

1. && - AND - para ser true apenas se os dois lados forem verdadeiros;
2. || - OR - para ser true, um lado como true é suficiente;
3. ! - NOT - este operador inverte a comparação;

### Tabela verdade

A tabela verdade vale para qualquer linguagem, e contém todos os resultados dos operadores lógicos;

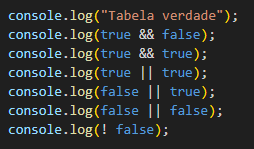


### Operadores lógicos na prática

Agora vamos aplicar AND, OR e NOT na prática;

Entender como os resultados são gerados em JavaScript;

Lembrando que Operadores lógicos são muito importantes no desenvolvimento;

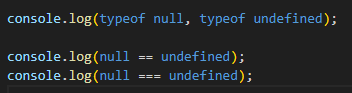


## $Empty Values

Temos duas palavras reservadas que pertencem a este grupo de dados: undefined e null;

Undefined geralmente é visto quando utilizamos um código que ainda não foi definido;

Já null, costuma ser imposto pelos programadores, para determinar que não há ainda um valor;

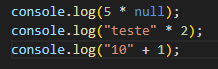


## $Conversão de tipo automática

Em JavaScript algumas operações mudam o tipo de dado, e isso acontece ‘silenciosamente’;

Exemplos:

* 5 \* null => 0
* “5” - 3> 2
* “5” + 1 => 51
* “a” \* “b” => NaN



Lembrar sempre de usar os dados corretos para as operações !

# Estrutura de programação

## 

## $Salvando valores na memória

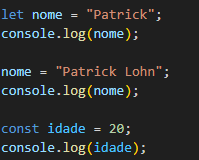
## variáveis

Até então colocamos valores nas expressões de console;

Porém isso não é tão comum pois nós utilizamos as variáveis;

Que são como containers, que salvam informações para quando precisamos utilizar;

Temos como declarar variáveis como let e const;



“let” é uma variável que pode mudar seu valor;

“const” é uma variável que não muda de valor;

### Mais sobre variáveis

Podemos criar várias variáveis em sequência, desta maneira;

* let a= 5, b = 4 , c=10;

Não podemos começar variáveis com números;

Também não podemos utilizar alguns caracteres especiais, como: “@” os únicos permitidos são ($ e \_ );

As variáveis são case sensitive (as variáveis são sensíveis a mudanças de caracteres ex: nome / Nome: mesmo sendo palavras iguais são variáveis diferentes devido ao carácter maiúsculo);



### Nomes reservados

Algumas palavras tem o nome reservado, não podemos criar variáveis com elas, elas são;

* break case catch class const continue debugger default delete do else enum export extends false finally for function if implements import interface in instanceof let new package private protected public return static super switch this throw true try typeof var void while with yield;

Resumidamente todas as palavras que são utilizadas para efetuar processos/recursos no JS;

É possível unir elas a mais outra palavra, para criar uma variável se necessário, ex: continue\_passa = 1;

### Convenção de nome de variáveis

Nos casos abaixo temos a pior forma até a melhor, para declarar nome de variáveis:

* let programadadorfullstack; (ruim)
* let programador\_full\_stack;
* let ProgramadorFullStack;
* let programadorFullStack; (Mais utilizada)

## $O ambiente JavaScript

Quando um programa é iniciado, um ambiente é criado;

Neste ambiente temos diversas funções e objetos da linguagem JavaScript;

Exemplo:console e alert;

Todo programa terá acesso a elas;

O ambiente no caso é o navegador;

## $A estrutura de uma função

Uma função é um bloco de códigos que pode ser reaproveitado ao longo do nosso programa;

Invocamos/chamamos ela pelo seu nome, e também o uso de parênteses: funcao();

Também podemos inserir parâmetros, que deixam a execução da função única, ex: soma(a,b);

Utilizamos algumas funções até então, como log de console;

### Funções do JS: prompt

A função prompt recebe um dado do usuário;

Podemos salvar este valor em uma variável;

Exemplo:

* Const x = prompt(“Digite um número :”)

Uma função pouco utilizada, mas nos permite fazer ações interessantes;



### Funções do JS: alert

A função alert emite uma mensagem na tela por um pop up;

Também não é muito utilizado, mas é um clássico do JavaScript;

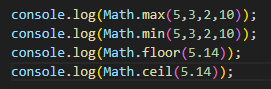


### Funções do JS: Math.x

Math é um objeto, que possui diversas funções para fins matemáticos;

Por exemplo:

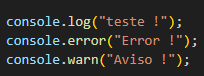
* max: encontra o maior número;
* min: encontra o menor número;
* floor: arredonda para baixo o número;
* ceil: arredonda para cima o número;



### Funções do JS: console.x

O console também é um objeto, assim como Math, tem várias funções;

A sua função principal é exibir uma mensagem de alguma categoria na aba Console;



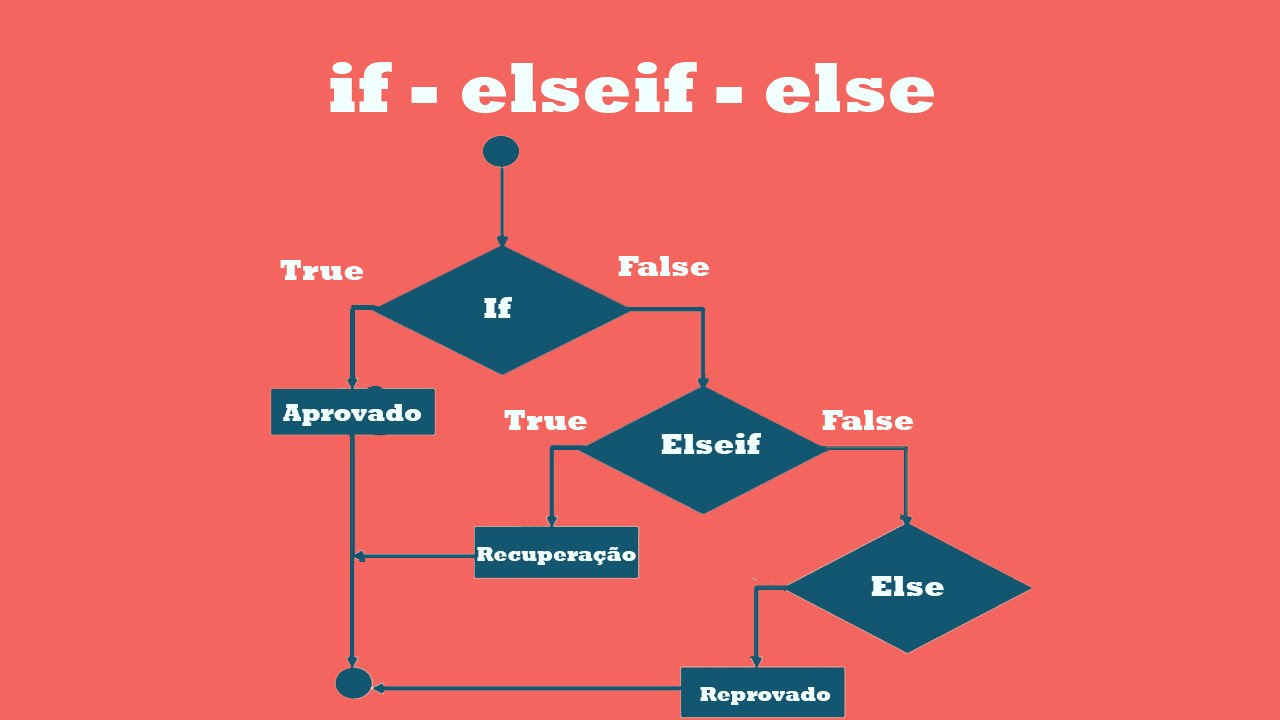
## $O que são estruturas de controle ?

Os programas são executados de cima para baixo;

Com estas estruturas podemos alterar o fluxo de execução;

O caminho dependerá das condições e comparações;

As principais são if, else e elseif;



Exemplo de fluxo com estruturas de controle.

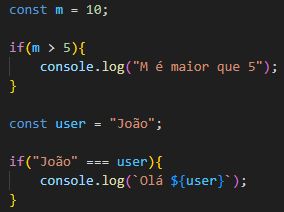
### Estrutura condicional:

### if

O if é muito utilizado na programação em geral;

Temos um bloco de código sendo executado, se uma condição for verdadeira;

A condição é válida por um boolean gerado após a execução do trecho de código do if;



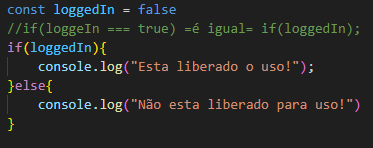
### Estrutura condicional:

### else

O else executa quando o if não atende sua condição;

Ou seja, não temos um bloco de validação, apenas do que será executado;

A ideia é: Executar algo SE (x > 5), SE NÃO execute isto;



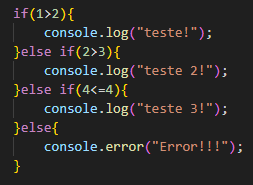
### Estrutura condicional:

### else if

O else if é uma estrutura intermediária de if e do else;

É possível adicionar novas condições, como no if;

Assim temos a possibilidade de criar várias validações, para resolver nosso problema;



## Estrutura condicional:

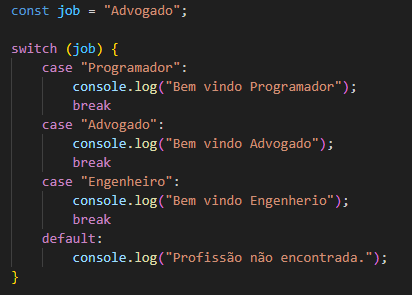
### switch

O switch pode ser utilizado para organização de um excesso de if/else;

Cada if seria um case;

para cada case, temos que adicionar um break;

E temos o default, que é como o else;



## $O que são estruturas de repetição ?

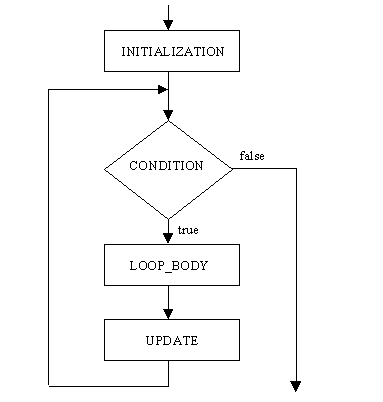
Um bloco de código que se repete até uma condição ser satisfeita;

Isso evita a repetição desnecessária do nosso código;

Alguns das estruturas são for e while;

A suas sintaxes são diferentes, mas as duas chegam no mesmo resultado;

Temos que nos atentar ao loop infinito;



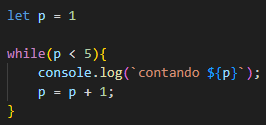
### Estrutura de repetição:

### while

O while faz uma ação até que uma condição seja atingida;

No bloco definimos o fim do loop, que é a condição;

Temos que definir também um incrementador, que é quem faz a condição ser atingida;



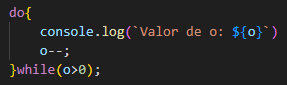
**Estrutura de repetição:**

### do while

O do while também é uma estrutura que permite repetição;

A sintaxe é semelhante ao while;

Este recurso não é tão utilizado;



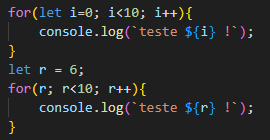
**Estrutura de repetição:**

### for

O for é a estrutura de repetição mais utilizada;

Ela condensa toda lógica em uma linha, ao primeiro olhar parece mais complexa, mas simplifica as coisas;

Na própria declaração, colocamos: incrementador, condição final e progressão;

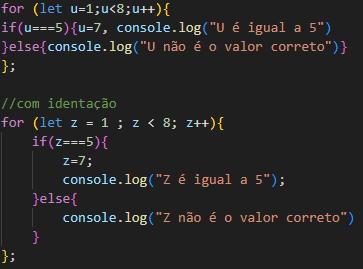


**A importância da indentação**

A indentação é um recurso utilizado para organizar múltiplos blocos de código;

Utilizamos o tab para criar um nível de indentação;

O código funciona sem, porém é interessante a adição deste recurso;



Podemos fazer o processo automaticamente pressionando;

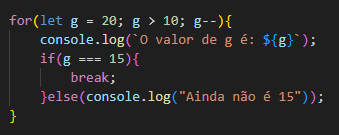
Shift + alt + f

**Forçando a saída de um loop**

Com a instrução de break podemos ejetar um loop, fazendo que com as repetições cessem;

Isso pode poupar memória, pois o código será executado menos vezes;

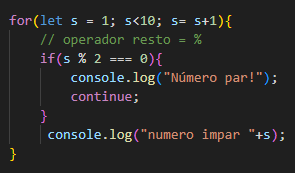
Não é tão comum, mas é um recurso válido da linguagem;



**Pulando a execução de um loop**

A palavra reservada continue, pode pular uma ou mais execuções do loop;

É um recurso utilizado de forma semelhante ao break;



# Funções

## O que são funções ?

Estruturas de código menores, podemos dividir nosso código em várias funções;

O ideal é que cada uma tenha apenas um único objetivo;

Isso nos faz poupar código, pois podemos reaproveitá las;

A linguagem tem várias funções já criadas, e nós podemos criar as nossas;

### $Definindo uma função

A estrutura da função é um pouco mais complexa;

Primeiramente utilizamos a palavra function, isso inicia uma função;

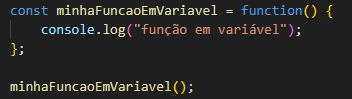
Precisamos nomeá-la após;

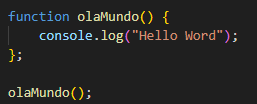
Os parâmetros, ficam entre () depois do nome;

O corpo da função fica entre {};

Geralmente uma função retorna um valor;

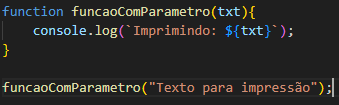
Segue maneiras de criar uma função:

Maneira 1 Maneira 2



A parte de baixo seria para chamar a função para execução;

Maneira 3: Estamos especificando parâmetro dentro da function;

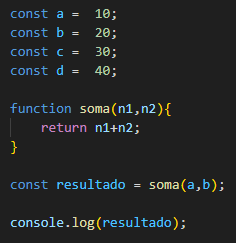
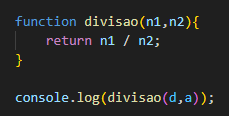


### $Retorno das funções

O retorno serve para processarmos um valor dentro da função e retornar para o programa;

A palavra reservada para este recurso é return;

Se não retornamos nada a função tem utilidade, mas não externaliza o que acontece nela;

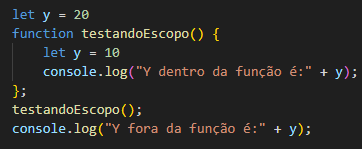


### $Escopo das funções

As funções têm um escopo separado do escopo do programa, que é o global;

Este escopo faz com que variáveis de fora não funcionem dentro;

Podemos então declarar novas variáveis, sem interferir nas já declaradas;



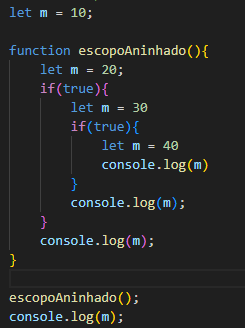
**Escopo aninhado**

### $(Nested Scopes)

As formas de criar variáveis, let e const, nos dão a possibilidade do escopo aninhado;

Que consiste em ter em qualquer bloco a declaração de variáveis separadas dos outros escopos;

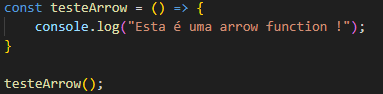
Um bloco é caracterizado por um código entre {};



## Arrow function

Arrow function é uma outra forma que temos de criar funções;

É uma sintaxe resumida, que tem algumas diferenças das funções normais;

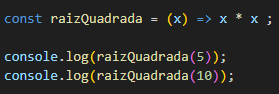


### $Mais sobre Arrow function

A arrow function pode ter uma sintaxe mais resumida;

Muito útil para funções pequenas;

Onde omitimos as {} e também a instrução de return;

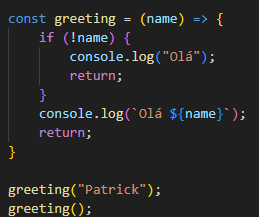
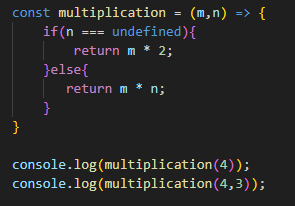


### $Argumentos opcionais

Os argumentos/parâmetros nas funções são obrigatórios, precisamos passar todos;

Porém há casos de funções que podem funcionar sem algum dos argumentos;

Para resolver isso podemos fazer uma checagem do parâmetro com um if;

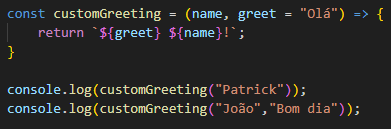


### $Argumentos com valor default

Valor default é quando o argumento tem um valor prévio;

Se for passado um novo valor, o default é substituído;

Se não, o default é utilizado na função;

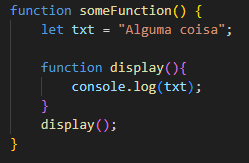


## Closure

Closure é um conjunto de funções, onde temos um reaproveitamento do escopo interno de uma função;

Pois este escopo não pode ser acessado fora da função, já que é um bloco;

Então há funções internas que aproveitam o escopo, e são chamadas de closure;

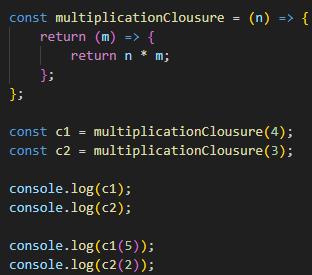


### $Mais sobre Closure

As closures também podem servir para salvar os resultados já executados;

Criando uma espécie de incrementação;

Assim temos uma variável que executa uma função e modifica seu valor;

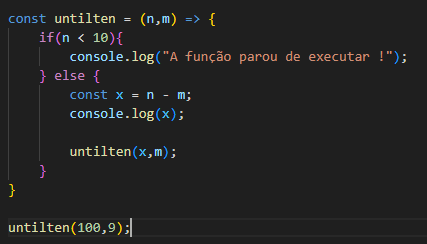


## Recursão

Um recurso que permite a função se auto invocar continuamente;

Criamos uma espécie de loop;

É interessante definir uma condição final, para parar a execução;



# Arrays e objetos

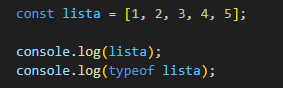
## Arrays

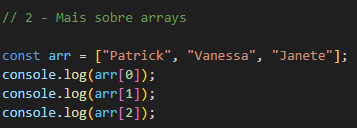
Arrays são listas;

Podemos inserir valores de qualquer tipo de dado;

Os valores são inseridos entre[];

Cada valor é separado o outro por uma vírgula;





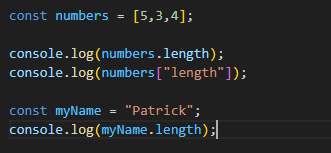
## Propriedades

Propriedades são como informações de um objeto;

Os arrays tems propriedades, assim como outros tipos de dados;

As propriedades podem ser acessadas por notação de ponto ou colchetes:

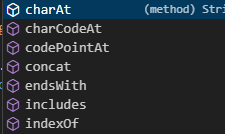
* dado.prop ou dado[´prop´]

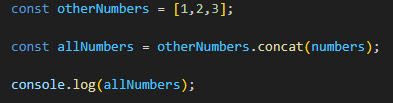


## Métodos

Métodos são como funções, acessamos com notação de ponto e utilizamos () para invocar;

Um importante conceito da OOP: Objetos são compostos por métodos e propriedades;

Como muitos dados são objetos em JS, temos métodos e propriedades neles; 



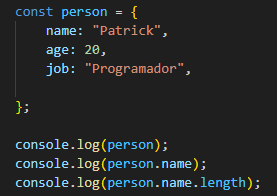
## Objetos (Object Literals)

Em JS temos um tipo de dado que é o objeto, mas seu nome técnico é object literals;

Isso porque o objeto vem da Orientação a Objetos, com outros recursos: instância, herança…

Já o literals possui apenas propriedades e métodos, nós mesmos os criamos;

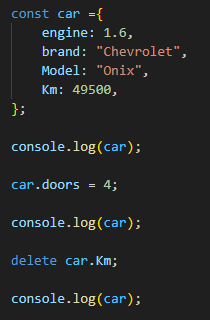
O objeto fica em um bloco de {};



### $Removendo e criando novas propriedades

Para adicionar uma nova propriedade a um objeto, utilizamos a notação de ponto e atribuímos um valor;

Já para excluir, vamos utilizar o operador delete na propriedade alvo;



### $Diferença entre arrays e objetos

Os arrays são utilizados como listas de itens, geralmente todos possuem o mesmo tipo;

Já os objetos são utilizados para descrever um item, contém as informações do mesmo, e as propriedades possuem diferentes tipos de dados;

Podemos ter também um array de objetos, isso é muito utilizado;

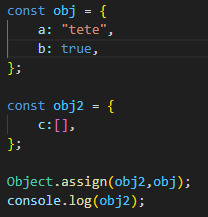
estes dois dados são muito importantes na programação;

### $Mais sobre Objetos

Podemos copiar todos as propriedades de um objeto para outro com o método assign;

O object literal é uma instância de um objeto, chamado Object;

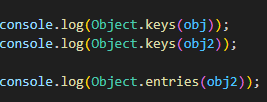
Um objeto ou array criado com const pode ter seus elementos e propriedades modificados!



### $Conhecendo melhor o objeto

Podemos verificar as propriedades de um objeto pelo método keys de Object;

Com o método entries, recebemos arrays dos nome das propriedades com seus valores;

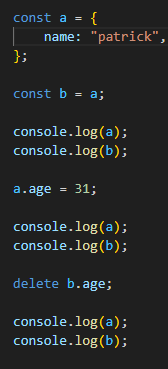


### $Mutação (Mutability)

Outra característica interessante é a mutação, isso ocorre quando criamos um objeto a partir de outro;

Este novo objeto, não é novo e sim uma referência do primeiro;

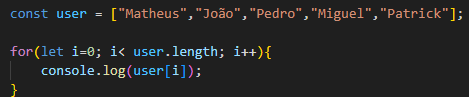
As mudanças dele, podem afetar a cópia e vice-versa;



### $Loops em arrays

Algo muito comum é percorrer os arrays através de estruturas de repetição, como for e while;

Isso serve para utilizar o resultado de cada um dos elementos de forma simples, sem repetição de código;



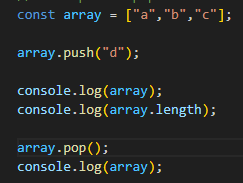
## Métodos de array:

### $push e pop

Os métodos de array são muito úteis para manipular os arrays, ou seja, alterar os seus valores de alguma forma;

Com o push adicionamos um item ao fim do array;

Com o pop temos a remoção de um elemento no fim do array;



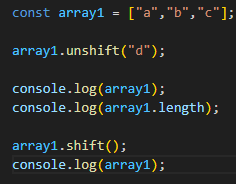
**Métodos de array:**

### $shift e unshift

Ao contrário de pop e push, temos shift e unshift;

O método shift remove o primeiro elemento do array;

Já o método unshift adiciona itens ao início do array;

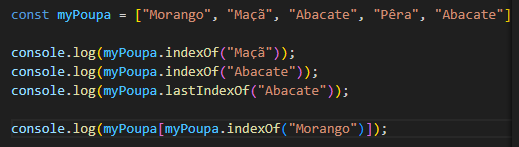


**Métodos de array:**

### $indexOf e lastINdexof

O método indexOf nos permite encontrar o índice de um elemento, que passamos como argumento para o método;

Já o lastIndexOf é utilizado quando há repetição de elementos e precisamos do índice da última ocorrência;



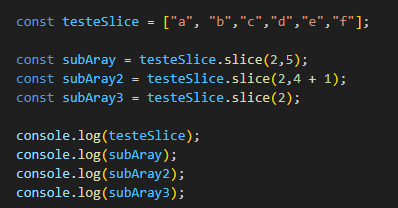
**Métodos de array:**

### $slice

O método slice é utilizado para extrair um array menor de um array maior;

O intervalo de elementos é determinado pelos parâmetros, que são: o índice de início e o índice de fim;

O último elemento é ignorado, se quisermos ele devemos somar 1 ao índice final;

****

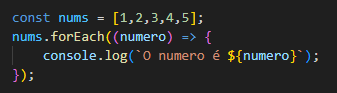
**Métodos de array:**

### $forEach

O forEach é como uma estrutura for ou while, porém é um método;

Ele percorre cada um dos elementos do array;

Para alguns sua sintaxe pode ser mais simples;

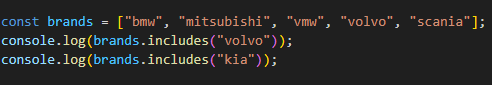


**Métodos de array:**

### $includes

O método includes verifica se o array tem um elemento;

Utilizamos no array e como argumento colocamos o elemento que buscamos;

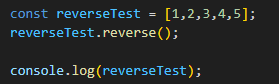


**Métodos de array:**

### $reverse

O método reverse inverte os elementos de um array;

Este método modifica o array original, então tome cuidado;



## Sobre os métodos de string

As strings também são objetos, ou seja tem métodos e propriedades;

Alguns são muito semelhantes aos de array;

Note que você pode utilizar length em uma string ou em um array;

E também acessar cada caractere pelo seu índice;

**Métodos de string:**

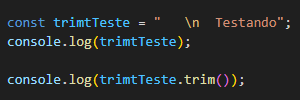
### $trim

O trim remove tudo que não é texto em uma string;

Como: caracteres especiais e espaços em branco;

Um método interessante para utilizar em sanitização de dados;

O método não modifica o texto original;

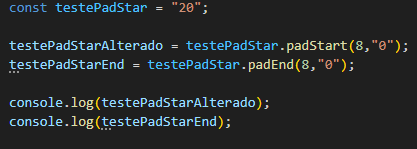


**Método de string:**

### $padStart

O método padStart insere um texto no começo da string;

O texto pode ser repetido, de acordo com o segundo argumento ao método, ele determina o máximo de caracteres do texto alvo;



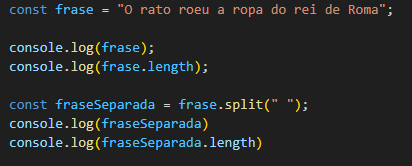
**Métodos de string:**

### $split

O split divide uma string em um array;

Cada elemento será determinado por um separador em comum;

Os mais utilizados, são: ponto e vírgula, vírgula, espaço;



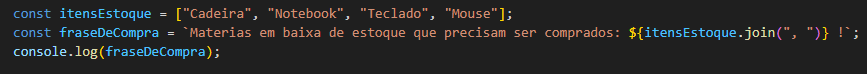
**Métodos de string:**

### $join

Já o join une array em uma string;

Podemos colocar um separador também, para formatar a string;





**Métodos de string:**

### $repeat

O método repeat repete um texto n vezes;

Onde n é o número que colocamos como seu argumento;

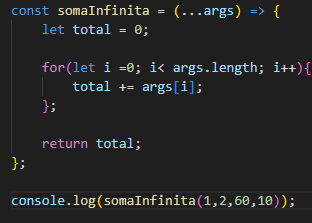


## Rest Operador / Rest Parameters

Rest Operador é caracterizado pelo símbolo “…”

Podemos utilizá-lo para receber indefinidos argumentos em uma função;

Assim não precisamos declarar exatamente o que vamos receber, deixando a função mais ampla;

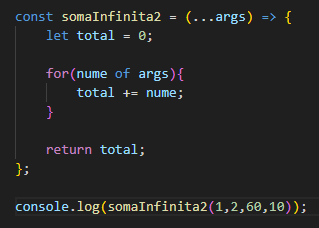


## Estrutura de repetição for…of

O for…of é uma estrutura de repetição semelhante ao for, porém mais simples;

O número de repetição é baseado no array utilizado;

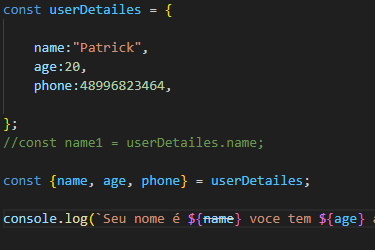
E podemos nos referir aos elementos sem precisar acessar o índice deles;



## Destructuring em objetos

O destructuring é uma funcionalidade que nos permite desestruturar algum dado;

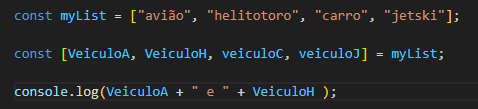
No caso dos objetos, é possível criar variáveis a partir das suas propriedades, com uma simples sintaxe;



## Destructuring em array

O destructuring também pode ser utilizado para desestruturar um array em variáveis;

A sintaxe é um pouco diferente, agora utilizaremos colchetes, e não temos nome das chaves;



## JSON

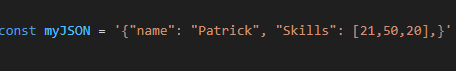
O JSON, JavaScript Object Notation, é um dado em formato de texto;

Utilizamos para comunicação entre API e front-end;

Sua formatação é rigorosa, se for mal feita o dado é invalidado e não conseguimos comunicação;

Seu formato lembra os object literals;

Regras: apenas aspas duplas e não aceita comentários;

****

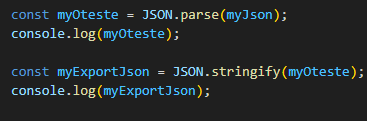
### 

### $JSON para objeto e objeto para JSON

Na maioria das vezes vamos precisar converter objetos para JSON;

Ou um JSON para um objeto JavaScript válido;

Utilizamos o objeto JSON e os métodos stringgify e parse;

****

# Orientação a Objetos

## O que é orientação a objetos ?

Um paradigma de programação, uma outra forma de programar;

Utilizando objetos como seu principal princípio;

A maioria dos softwares é desenvolvido neste paradigma;

Frameworks e bibliotecas de front-end também são desenvolvidos com POO;

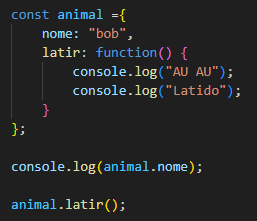
Estávamos desenvolvendo no modo procedural;

## Métodos

Métodos podem ser adicionados aos objetos;

Eles são como propriedades, mas contém uma função;

Invocamos os métodos do mesmo modo que funções;

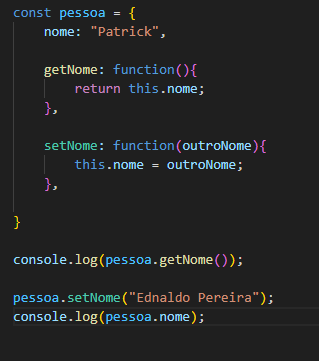


### $Aprofundando em Métodos

Os métodos são utilizados para interagir também com as propriedades do seu objeto;

Podemos exibir elas ou modificá-las;

Podemos nos referenciar com o próprio objeto com a palavra reservada this;



## Sobre o Prototype

Prototype é um recurso que faz parte da arquitetura de JavaScript;

É uma espécie de herança, onde objetos pais herdam propriedades e métodos aos filhos;

Por isso muitos dados são considerados objetos e termos objetos, como: String, Number, e outros;

Ou seja, cada dado tem um objeto pai herdou características pelo prototype;

### $Prototype na prática

O recurso fundamental do prototype que temos que entender é o fallback;

Quando uma propriedade não existe em um dado/objeto, ela é procurada no seu ancestral;

Ou seja, é por isso que temos acesso a length em strings, por exemplo;

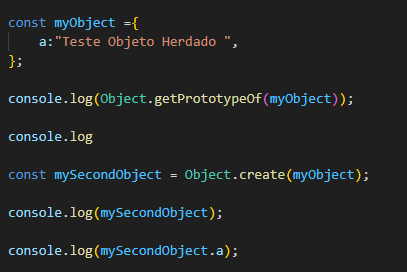
### $Mais sobre Prototype

### 

Quando criamos um objeto a partir de outro, este outro será o prototype do objeto criado;

Porém também herdará as características do objeto pai, se for um objeto, herda de Object;

Esta é a cadeira do prototype;

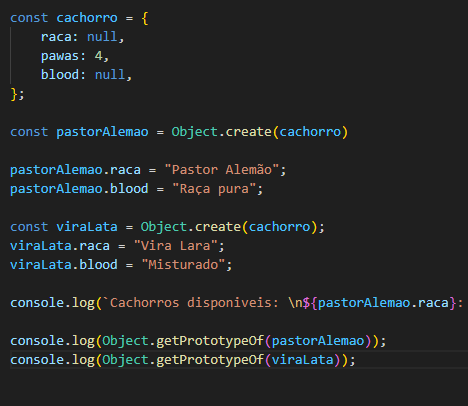


## Classes básicas

Os prototypes são originados de uma Classe;   
Que é o molde dos objetos, nela definimos os métodos e propriedades;

JavaScript já possui suas classes, porém podemos criar as nossas;

Isso é essencial para a Orientação a Objetos;

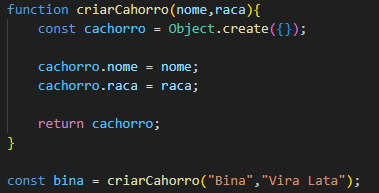


## Classes baseadas em funções construtoras

Utilizando funções como classes, conseguimos iniciar as propriedades com a criação do objeto;

Chamamos de função construtora, este recurso;

O construtor tem como objeto instanciar um objeto, ou seja, criar um novo objeto;

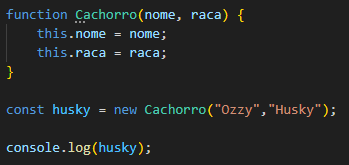


## Classes baseadas em funções

Este recurso é semelhante ao anterior, mas com uma nova palavra chave: a new;

Em vários linguagens o new é utilizado para instanciar novos objetos, em JS isso também acontece;

E eles podem partir funções;

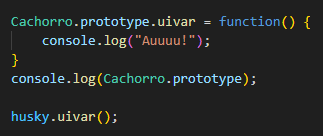


## Classes de função com métodos

Para adicionar métodos da criação do objeto, podemos acessar o prototype e colocá-los lá;

Esta é basicamente a essência de JavaScript;

Porém com a evolução da linguagem, outros recursos foram criados.

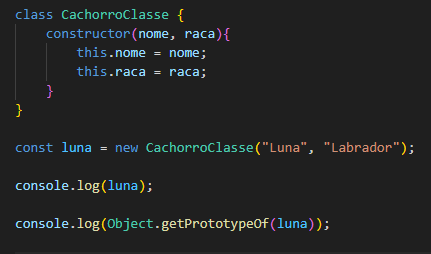


## Classes ES6

Nas versões mais atuais de JS abandonamos as functions e utilizamos as classes;

Aqui temos recursos comuns em outras linguagens, como o construtor;

Além da instância por new;



**MAIS USADO**

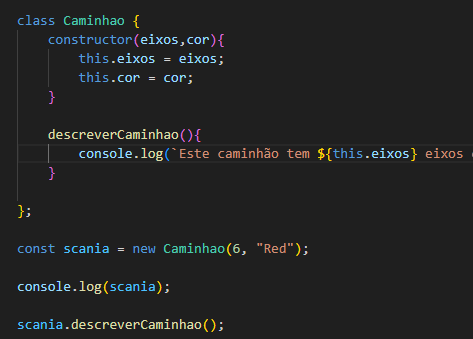
### $Mais sobre classes

Não podemos adicionar propriedade diretamente as classes;

Isso precisa ser feito ao iniciá-la ou via prototype;

Métodos da classe também podem utilizar this para se referir ao objeto instanciado;

Os métodos não precisam da palavra function;

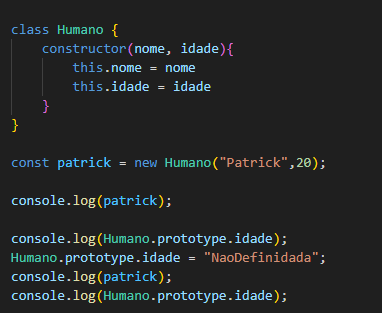


## Override nas propriedades via Prototype

As instâncias dos objetos são criadas baseadas nas classes;

Ou seja, as propriedades têm os valores definidos no construtor ou por métodos;

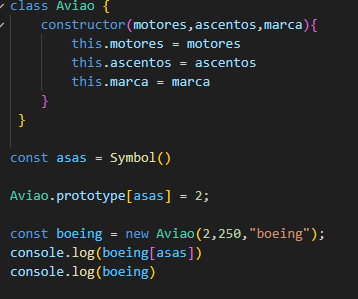
Para alterá-los podemos utilizar o prototype;



## Symbols em classes

Quando utilizamos o recurso de Symbol com classe, é possível criar uma propriedade única e imutável;

Isso é útil quando há algum dado que se repetirá em todos os objetos criados a partir da classe;



## Getters e Setters

Os getters e setters são bem famosos na Orientação a Objetos;

O get é um método utilizado para exibir o valor de algum propriedade;

E o set é utilizado para alterar o valor;

Através de métodos, temos um bloco de código para transformação de dados;

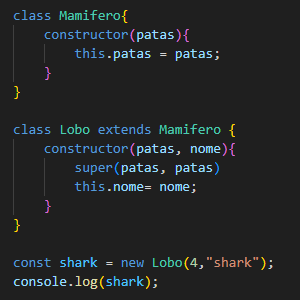


## Herança

Uma classe pode herdar propriedades de outra por meio de herança;

Utilizamos a palavra chave extends, para adicionar a classe que vai trazer as propriedades;

E super para enviar os valores para a classe pai;

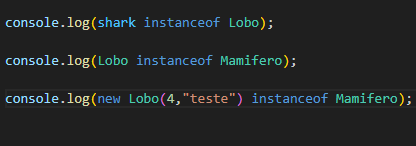


## Operador instanceof

Assim como typeof que verifica o tipo, temos o operador instanceof;

Que verifica se um objeto é pai de outro, para ter certeza da ancestralidade;

Isso é verificado com objeto => classe, e não através das classes;



# Debug e tratamento de erros

## O que é bug e debug?

Bug: um problema que ocorreu no código, muitas vezes por erro do programador, impede o funcionamento do software;

Debug: Método de encontrar e resolver o bug, em JavaScript temos diversas estratégia para isso;

Validação: Técnicas utilizadas para ter o mínimo possível de bugs no software;

## Strict mode

O strict é um modo de desenvolvimento que deixar o JS mais rigoroso na hora de programar;

Deve ser declarado no topo do arquivo ou de funções;

O strict não limita os recursos de JS, ele baliza a forma que você programa;

Bibliotecas famosas são todas feitas em strict;

## Método de debug:

### $console.log

O método log de console é muito utilizado para debug;

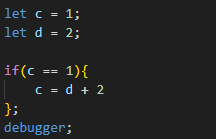
Utilizamos diversas vezes nos nossos exemplos;

**Método de debug:**

### $debugger

O debugger é uma instrução que nos permite o debug no console do navegador;

Podemos evidenciar os valores das variáveis em tempo real e com o programa executando, o que ajuda bastante;

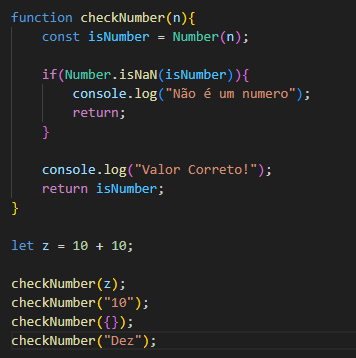


## Tratamento de dado por função

Nunca podemos confiar no dado que é passado pelo usuário;

Sempre devemos criar validações e tratamento para o mesmos dados;

Ao longo do curso aprenderemos diversas técnicas;



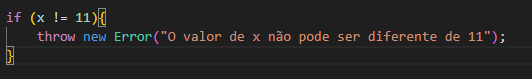
Exemplo: tratamento de dados de N Number;

## Exceptions

As exceptions são erros que nós geramos no programa;

Este recurso faz o programa ser abortado, ou seja, ele não continua sua execução;

Utilizamos a expressão throw new Error, com a mensagem de erro como argumento;

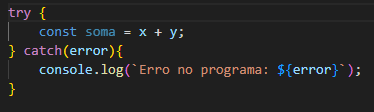


## Try Catch

Try catch é um recurso famoso nas linguagens de programação;

Onde tentamos executar algo em try, e se um erro ocorrer ele cai no bloco do catch;

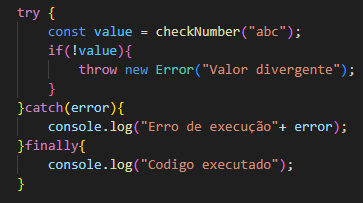
Útil tanto para debug, como também no desenvolvimento de uma aplicação sólida;



## Finally

O finally é uma instrução que vai depois do bloco try catch;

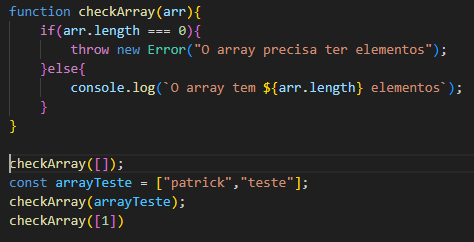
Ela é executada independente de haver algum erro ou não em try;



## Assertions

Assertions são quando os tratamentos de valores passados pelo usuário, geram um erro;

Porém este recurso tem como objetivo nos ajudar no desenvolvimento do programa, ou seja, seria algo para os devs e não para os usuários;



# Programação assíncrona

## O que é programação assíncrona ?

A programação assíncrona precisa ser utilizada quando as respostas não são obtidas de forma imediata no programa;

Chamadas a uma API são assíncronas, não sabemos quanto tempo a resposta pode demorar;

Até agora utilizamos só instruções síncronas;

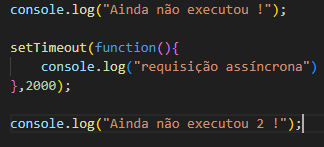
na programação assíncrona as execuções não ocorrem em formato de fila, e sim no tempo;

## Função setTimeout

A função setTimeout faz parte da programação assíncrona;

Pois estabelecemos uma ação para ser executada após um certo tempo;

Ou seja, o código continua rodando e depois temos a execução da função;

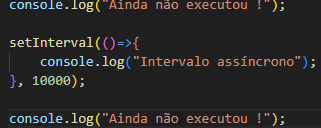


## Função setInterval

A função setInterval é semelhante a setTimeout, ela é executada após um tempo;

Porém ela não para de ser executada, temos a sua chamada definida pelo tempo de espera na execução;

É como um loop infinito com execução de tempo controlada;

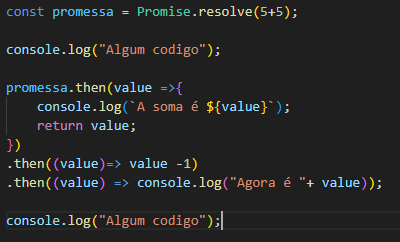


## Promises

As promises (promessas) são execuções assíncronas;

É literalmente uma promessa de um valor que pode chegar em um ponto futuro;

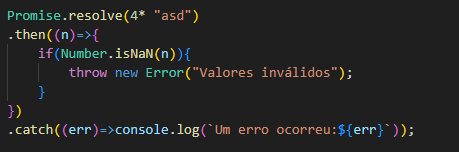
Utilizamos o objeto Promise e alguns métodos para nos auxiliar;



### $Falha nas Promises

Uma promise pode conter um erro, ou dependendo de como o código é executado podemos receber um erro;

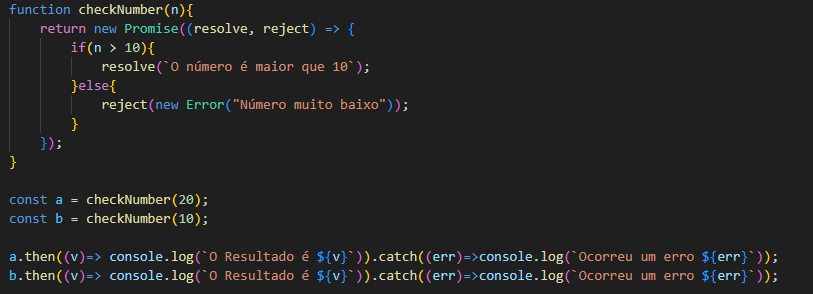
Utilizando a função catch para isso, podemos pegar o erro e exibir;



### $Rejeitando Promises

A rejeição, diferente do erro, ocorre quando nós decidimos ajetar uma promise;

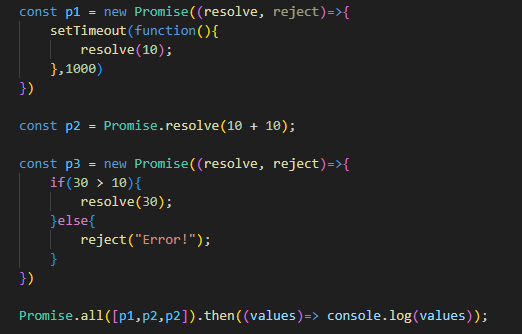
Podemos fazer isso com o método reject;



### $Resolvendo vários promises

Com o método all podemos executar várias promises;

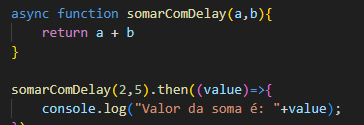
JavaScript se encarrega de verificar e retornar os seus valores finais;



## Async Function

As async functions são funções que retornam Promises;

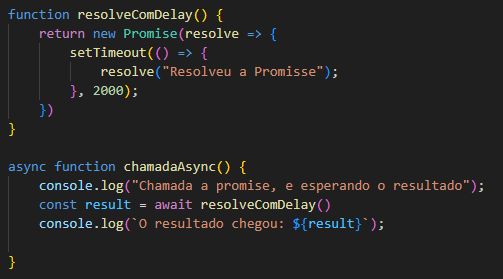
Consequentemente há a possibilidade de receber o resultado delas depois, além da utilização dos métodos de Promise;



## Assertions

A instrução await serve para aguardar o resultado de uma async function;

Tornando mais simples lidar com este tipo de função, desta maneira não precisamos trabalhar diretamente com Promises;

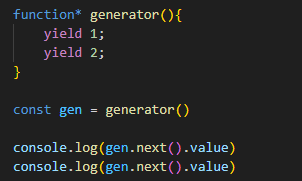


## Generators

Generators funcionam de forma semelhante às promises;

Ações podem ser pausadas e continuadas depois;

Temos novos operadores, como function\* e yield;



# JavaScript no navegador

## Protocolos da Web

Um protocolo é uma forma de comunicação entre computadores através da rede;

O HTTP serve para solicitar arquivos e imagens do servidor (Hyper Text Transfer Protocol);

É possível navegar em sites através do HTTP;

SMTP: protocolo para envio de email;

TCP: protocolo para transferência de dados;

## Conhecendo melhor as URLs

Cada arquivo que é carregado no navegador tem uma URL;

A URL (Uniform Resouce Locator) pode ser dividida em três partes;

Por exemplo: https//pkldeveloper.com.br/index.html

https é o protocolo, pkldeveloper.com.br é o domínio, que referencia um servidor (DNS > IP)

E index.html p arquivo.página que estamos acessando;

## Conhecendo o HTML

HTML (HyperText Markup Languege) é uma linguagem de marcação;

Onde estruturamos as páginas web, criando elementos;

Os elementos são chamados de tags, que podem ser: títulos, imagens, formulários, listas, post, tabelas ….

As tags são caracterizadas por:<p>Texto </p>

Podemos adicionar estilos ao HTML com CSS;

## A estrutura do HTML

Toda página HTML tem duas partes importantes: head e body;

No head inserimos as configurações da página, e importações de outros arquivos (CSS, JS);

Já no body temos os elementos que ficam visíveis para o usuário;

As tags possuem atributos que configuram os elementos;

## HTML e JavaScript

Podemos adicionar JavaScript ao HTML por meio da tag script, em arquivo externo ou script na página;

Algumas tags tem atributos que podem executar JS, mas isso não é muito utilizado;

Sempre que houver um link entre um arquivo e outro, uma chamada HTTP é executada;

JavaScript pode ser utilizada para manipular elementos do HTML e alterar estilos;

## HTML e o DOM

O DOM é uma representação fiel do HTML da página;

Ele é utilizado para acessar o HTML através de JS, acessamos os elementos/tags;

Assim podemos modificá-lo através dos métodos e propriedades dos objetos que alteram o DOM;

DOM vem de Document Object Model;

Através dele também podemos atrelar eventos ao HTML, como click ou pressionar teclas do mouse;

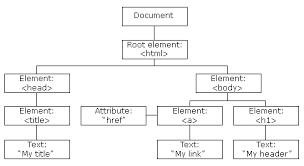
### $DOM

O DOM pode modificar completamente uma página;

É possível alterar: elementos, atributos, estilização;

Adicionamos e removemos elementos;

O DOM cria uma árvore do HTML, os elementos são chamados de nós;

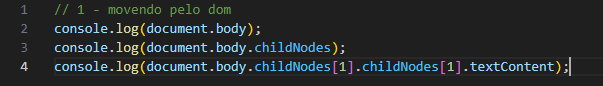


### $Movendo-se pelo DOM

Todos os elementos podem ser acessados através de document.body;

A partir deste elemento pai, vamos encontrando os childNodes (nós);

E podemos acessar suas propriedades, e consequentemente modificá-los;



## Selecionando elementos

Temos várias formas de selecionar especificamente um elemento, ou um conjunto deles;

A diferença entre eles é a forma de seleção, que pode ser por: classe, id, seletor de CSS;

Alguns exemplos são: getElementsByTagName, getElementByld, querySelector.

### $Encontrando elemento por tag

Com o método getElementsByTagName selecionamos um conjunto de elementos por uma tag em comum;

O argumento é uma string que leva tag a ser selecionada;



### $Encontrando elementos por id

Com o método getElementByID selecionamos um único elemento, já que o id é o único na página;

O argumento é uma string que leva o id a ser selecionado ;



### $Encontrando elementos por classe

Com o método getElementsByClassName selecionamos um conjunto de elementos por uma classe em comum;

O argumento é uma string que leva a classe a ser selecionada;

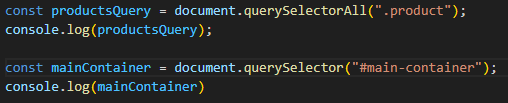
Veja como os atributos do HTML começam a fazer mais sentido em conjunto com JS;



### $Encontrando elementos por CSS

Com o método querySelectorAll selecionamos um conjunto de elementos por meio de um seletor de CSS;

E com o querySelector apenas um elemento, com base também um seletor de CSS;



## Alterando o HTML

Podemos mudar praticamente toda a página com DOM;

Adicionar, remover e até clonar elementos;

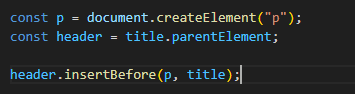
Alguns métodos muito utilizados são: insertBefore, appendChild, replaceChild;

### $Alterando o HTML com insertBefore

O insertBefore cria um elemento antes de um outro elemento;

É necessário criar um elemento com JS, isso pode ser feito com createElement;

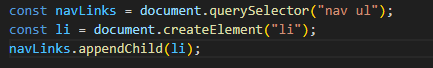
O elemento de referência pode ser selecionado com alguns dos métodos que vimos antes;



### $Alterando o HTML com appendChild

Com o appendChild é possível adicionar um elemento dentro de outro;

Este elemento adicionado será o último elemento do elemento pai;

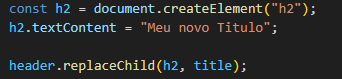


### $Alterando o HTML com replaceChild

Já o método replaceChild é utilizado para trocar um elemento;

Novamente precisamos do elemento pai;

E também o elemento para ser substituído e o que vai substituir;

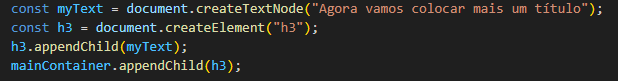


## Criando nós de texto

Os textos podem ser manipulados com método também;

Temos o createTextNode, que cria um nó de texto;

E este nó pode ser inserido em um elemento;



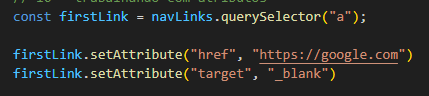
## 

## Trabalhando com atributos

Podemos ler e alterar os valores dos atributos;

Para ler vamos utilizar o método getAttribute;

E para alterar utilizamos setAttribute, este leva o nome do atributo e o valor para alterar;



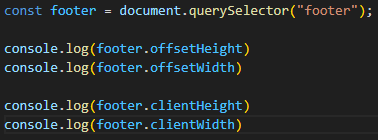
## Altura e largura dos elementos

## 

É possível também pegar valores com altura e largura de elementos

Vamos utilizar as propriedades: offsetWidth e offsetHeight;

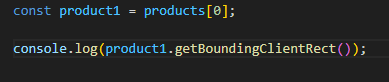
Se queremos desconsiderar as bordas, temos:  
clientWidth e clientHeight;



## Posição do elemento

Com o método getClientBoundingRect podemos pegar várias informações do elemento;

Como: posição no eixo x, y, altura, largura e outros;



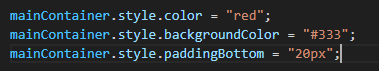
## Estilos com JS

Todo elemento possui uma propriedade chamada style;

A partir dela conseguimos alterar as regras de CSS;

Note que regras separadas por traço viram camelCase, exemplo:

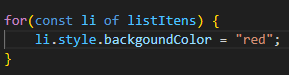
background-color => backgorundColor;



### $Alterando estilos de HTMLCollection

HTMLCollection aparece quando selecionamos vários elementos de uma vez;

Podemos passar por cada um dos elementos com um for of, e estilizar individualmente cada item;



# Eventos no JavaScript

## O que são eventos ?

Ações atreladas a algum comportamento do usuário;

Por exemplo: click, alguma tecla, movimento da tela e do mouse;

Podemos inserir lógica quando estes eventos ocorrem;

E podemos disparar eventos em certo elementos;

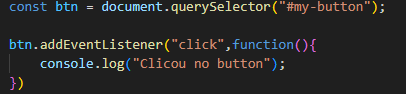
Esta técnica é conhecida como event handler;

## Como acionar um evento

Primeiramente precisamos selecionar o elemento que vai disparar o evento;

Depois vamos ativar um método chamado addEventListener;

Nele declaramos qual o tipo do evento, e por meio de callback definimos o que acontece;

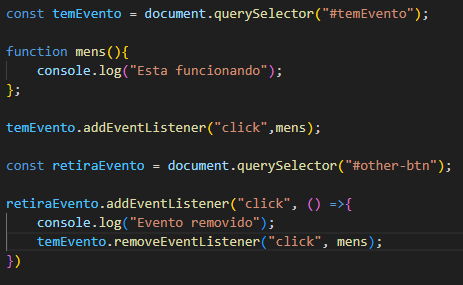


## Removendo eventos

Há situações que vamos querer remover os eventos dos elementos;

O método para isso é removeEventListener;

Passamos o evento que queremos remover como argumento;

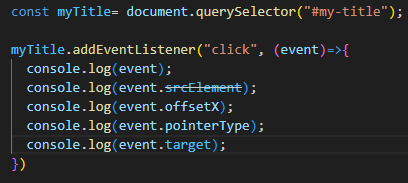
****

## 

## Objeto do evento

Todo evento possui um argumento especial, que contém informações do mesmo;

Geralmente chamado de event ou e;

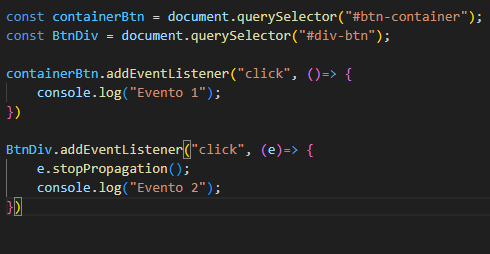


## Propagação

Quando um elemento de um evento não é claramente definido pode haver propagação;

Ou seja, um outro elemento ativar o evento;

Para resolver este problema temos o método stopPropagation;

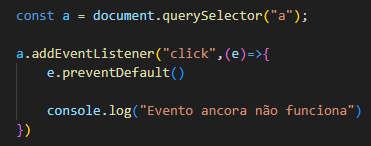


## Ações default

Muitos elementos tem ações padrão no HTML;

Como os links que nos levam a outras páginas;

Podemos remover isso com o método preventDefault;



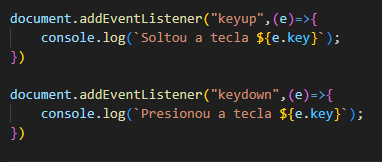
## Eventos de tecla

Os eventos de tecla mapeiam as ações no teclado;

Temos a disposição keyup e keydown;

O keyup ativa quando a tecla é solta;

E keydown quando é pressionada;



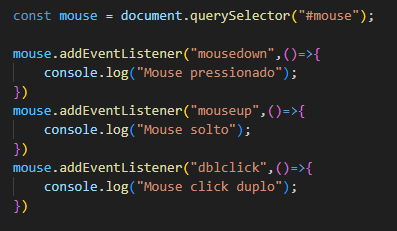
## Outros eventos de mouse

O mouse pode ativar outros eventos;

mousedown: pressionou botão do mouse;

mouseup: soltou botão do mouse;

dblclick: clique duplo no mouse;

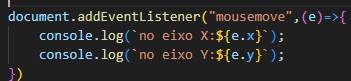
****

## Movimento do mouse

É possível ativar um evento a partir da movimentação do mouse;

O evento é o mousemove;

Com o objeto de evento podemos detectar a posição do ponteiro do mouse;

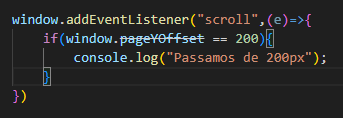


## Eventos por scroll

Podemos também adicionar um evento ao scroll do mouse/página;

Isso é feito pelo evento scroll;

Podemos determinar que algo aconteça após chegar numa posição escolhida da tela;

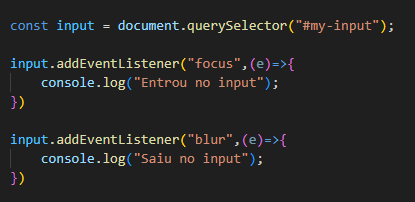


## Eventos por foco

O evento focus é disparado quando focamos em um elemento;

Já o blur é quando perde o foco do elemento;

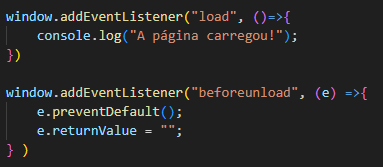
Estes são comuns em inputs;



## Eventos de carregamento de página

Podemos adicionar um evento ao carregar a página, que é o load;

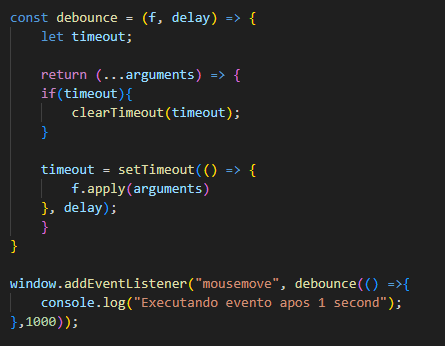
E quando o usuário sai da página, que é o beforeunload;



## Técnica de debounce

O debounce é uma técnica utilizada para fazer um evento disparar menos vezes;

Isso poupa memória do usuário, pois talvez nem sempre o evento seja necessário;



# ES6 +

## O que é JS ES6+

São as novas versões de JavaScript;

Cada uma delas trouxe recursos que ajudam muito no desenvolvimento;

Estes recursos são essenciais para trabalhar com frameworks/libs como react, Vue e Angular;

Agilizam muito o desenvolvimento com JS;

## Variáveis com let e const

Temos duas novas formas de declarar variáveis a partir do ES6, que são **let** e **const**;

O **let** é uma forma de atribuir valor, e poder modificar depois;

Já **const** declara uma constante, podemos atribuir um valor e não alterar;

O grande diferencial são os escopos em bloco;

## Arrow functions

As **arrow functions** são um recurso para criar funções de forma mais simples;

Alguns aspectos a diferenciam das funções comuns;

Por exemplo o this, que é relacionado ao elemento pai de quem está executando;

## Filter

O **filter** é um método de array para filtrar dados;

O filtro é determinado por nós, **por meio de uma função;**

Resultado em um array com **apenas os elementos que precisamos;**

Nessa versões mais novas de JS temos vários métodos de array importantes como este;

## Map

O **map** também é um método de array percorre todos os elementos do mesmo;

O map é utilizado para **modificar o array de origem;**

Filter remove elementos desnecessários, map altera os que precisamos;

## Template literals

O recurso de **template literals** permite a impressão de variáveis em um texto;

Escrevemos entre crases, desta maneira: `texto`

E as variáveis são inseridas com: ${variável};

## Destructuring

O **destructuring** desestrutura dados complexos em várias variáveis;

Podemos utilizar em **arrays e objetos;**

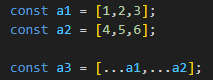
Muitas variáveis podem ser criadas em um única linha;

## Spread operator

O **spread** pode ser utilizado em **arrays e objetos**;

Utilizamos para inserir novos valores em um array ou objeto;

É um recurso que pode unir dois arrays, por exemplo;



## Classes

As **classes** são recursos fundamentais para programar orientado a objetos;

Temos acesso a recursos importantes, como:

constructor, propriedades, métodos;

Antes as classes em JS eram criadas com **constructor functions;**

## Herança

**Herança** é o recurso que nos dá a possibilidade de uma classe herdar métodos e propriedades de outro;

A palavra **extends** determina qual classe será herdada;

Para enviar propriedades para a classe pai utilizamos **super,** isso é necessário;

# 