# BootCamp JavaScript

## História, Evolução e Aplicações

JS – Linguagem interpretada, significa que ela é rodada em tempo real e o código está disponível instantaneamente, não precisa esperar um processo de compilação.

Baseada em protótipos: Uma base para as estruturas de dados do Javascript, conjunto de funções e possibilidades em comum em todas as estruturas de dados.

Multiparadigma: é possível escolher se vc deseja usar funções orientadas a objetos, programação funcional ou estruturada.

Utilizada em aplicações web cliente-side = parte que terá interação com o usuário.

Segue o padrão ECMAScript = Padrão para várias linguagens, conjunto de normas que definem quais funcionalidades estão sendo lançadas para a linguagem.

EcmaScript 3: try/catch

EcmaScript 5 -> JSON support

EcmaScript 6 = orientação a objetos e muitas outras funções.

Aplicações:

Programação Web

Aplicativo de celulares

SmartWatches

Games

Internet of things

APIs

## Recursos Básicos

Comentário de código igual a linguagem c/c++.

Variável: funciona igual uma variável em matemática, podendo ser alterado ao longo do código.

Constante (sempre em letra maiúscula): assume um valor que não pode ser manipulado mais depois da definição.

Funções: usa a terminologia function com o nome da função logo em seguida.

## Console pg Web e Terminal

Web: Utilização do console web na área de inspecionar da página do google chrome.

Terminal: utilização do node.js.

## Estrutura de Projeto

Para importar um arquivo JS em uma página HTML é necessário utilizar o <script src=”endereço do arquivo js”></script>

Para importar arquivos CSS para o HTML, é necessário utilizar o

<link rel=”stylesheet” href=”endereço do arquivo”/>

## Interação com o DOM

Document object model – são os elementos que existem na página.

## Mercado de Trabalho

Frameworks e Bibliotecas:

Principais bibliotecas utilizadas pelas empresas ou pelo mercado.

VueJS

Angular

React

jQuery

Sites populares para programadores:

W3C, MDN web docs, DIO, Stackoverflow, youtube, GitHub e LinkedIn.

# Syntaxes JavaScript

Programação de alto nível que integra páginas web e app mobile.

Realiza scripts dinâmicos que realizam interações em páginas web e aplicativos.

## Tipagem:

Funciona como uma regra de uso de dados, quanto mais forte a tipagem, mais obrigatório é a declaração do tipo de dado.

A tipagem em JavaScript é fraca, a declaração dos dados acontece de modo dinâmico.

## Tipos Primitivos

Boolean (valores verdadeiro ou false), null, undefined, Number, String, Array, Object, Function.

Typeof() = retorna o tipo primitivo da variável escolhida entre ().

## Variáveis:

São dados que podem sofrer mudanças.

Var – escopo global e local, podendo ser alterada e caso não seja set um valor, ele será null. Pode receber valor comparados, exemplo var x = 5<2;

Let – escopo local, podendo ser alterado e também recebendo null caso não tenha valor.

Const – Escopo local, somente leitura, não podendo ser alterado, e deve possuir um valor set nele, não assumindo o null automático.

Escopo global: quando a variável é declarada fora de qualquer bloco, sua visibilidade fica disponível em todo o código.

Escopo Local: quando a variável é declarada dentro de um bloco, sua visibilidade pode ficar disponível ou não. Em outras palavras, ela precisa ser declarada dentro de uma função ou fora dela, porém caso a declaração seja feita dentro da função, ela não pode ser chamada por um código fora da função, é necessário chamar a própria função para ter visibilidade da variável ou constante.

Não inicia var com número, não utilizar espaço, não usar palavras reservadas e declare a variável no topo do código.

O “=” significa atribuição, o sinal de comparação é com “==” e o “===” é o sinal de idêntico.

## Operadores Aritméticos

Único diferente é a potência, que é escrita com “\*\*”.

## Vetores e Objetos

Arrays são um tipo de lista, ou matriz de variáveis, onde cada variável possui um índice. Os valores podem ser de vários tipos.

Let array = [‘string’, 1, true, false, [‘array1]…]

Um array pode guardar um array dentro e pode guardar valores de tipos diferentes no mesmo array.

### Manipulação do Array

Array.forEach()

forEach() – itera um array;

push() – adiciona um item no final do array;

pop() – remove um item no final do array;

shift() – remove um item do início do array;

unshift() – adiciona um item no início do array;

indexOf() – retorna o índice do valor;

splice() – remove ou substitui um item pelo índice;

slice() – retorna uma parte de um array existente;

### Objetos

Dados que possuem propriedades e valores que definem suas características. Deve ser declarado entre “{}”

As propriedade de objetos podem ser atribuídas à variáveis, facilitando a manipulação do objeto, isso se chama desestruturação.

Var xicara{

Cor: ‘azul’, == var cor = xicara.cor;

Tamanho: ‘p’,

Funcao: tomarCafe()

}

## Estruturas Condicionais

São instruções para realizar determinadas tarefas a partir de uma condição, seja de decisão ou repetição

### Decisão:

Podemos usar as palavras reservadas “if” e “else”.

“else if” para estabelecer mais condições.

Ninho de if’s:

Quando se usa um “if” dentro de outro “if”.

If ternário:

Verificação de uma única linha:

X != -1 && y!=3 ?

Switch/case: funciona como uma estrutura de condição. Server para substituir o “else if”. Em JS é necessário colocar um break após o término da condição, para não entrar em loop eterno.

### Laços e repetições:

Estruturas condicionais que repetem uma instrução até atingir determinada condição.

- for; não esquecer de declarar uma nova variável com let para o funcionamento do condicionamento.

- for/in; repetição a partir de uma propriedade

- For/of; repetição a partir de uma valor. Não funciona com objetos pois as propriedades variam, diferentes do índice em um array que sempre serão números inteiros.

- While; verificação da condição é feita antes da execução

- Do/while; a verificação da condição é feita depois da execução

## Funções e suas particularidades:

São blocos de comando e instruções para a execução de determinadas tarefas.

Utilização do function(declaração de parametro){

Instrução

}

As funções podem receber em sua declaração, parâmetros, que servem como variáveis, onde sua atribuição pode ser feita durante a chamada da função.

### Tipos de funções:

Declarativas: declaradas pelo function, são as mais comuns. É necessário que a função possua um nome.

Expressões de funções: são funções atribuídas à expressões, o nome é opcional:

Var função = function(){

}

Arrow function: são funções de expressão de sintaxe curta. Sempre serão anônimas, sendo assim não podem ser nomeadas. Deve ser declarada com parênteses ”()”, seguido de “=>” e depois chaves “{}”

Var função = () => {

}

### Funções aritméticas:

Number(): para converter valores em números;

Prompt() – para registrar entradas de usuário;

Alert() – para mostrar mensagem ao usuário;

TemplateStrings – para usar strings junto com expressões;

## Operadores:



### Atribuição:

=;

+=; x+=y; x = x+y;

\*=; x = x\*y;

/=;

%=;

### Aritmética:

+; -; \*... esse tipo de símbolo que pode ser usado para cálculos e determinações.

### Comparação:

==; igual a (pode comparar o valor 2 com uma string que tenha o valor 2 nela, retornando igual, mas os tipos são diferentes);

===; mesmo valor e mesmo tipo;

!=; diferente

!==; valor e tipos diferentes;

### Lógica:

&& = “e” lógico;

|| = “ou” lógico;

! = “não” lógico;

### Condicional (ternário):

(condition) ? expression1 : expression2

Equivale a um if else.

Se a condição for verdadeira roda a expressão 1, se for falso roda a expressão 2.

# Variáveis e Tipos

Atribuição de valores:

Utilização do Camel Case e Upper Snake case: formas de escrita do nome das variáveis. Não se usa espaço, se usa ou o underline ou uma letra maiúscula.

Ex: Camel case: variavelLegal.

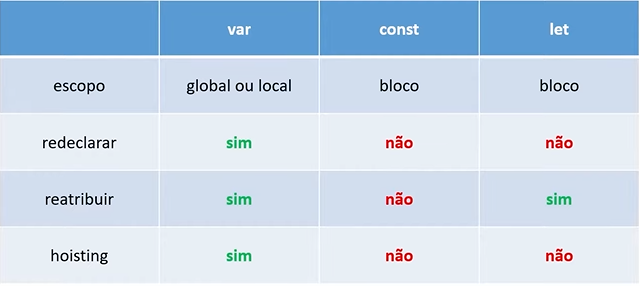
Upper Snake Case: VARIÁVEL\_LEGAL.

Var e Let: podem ter seus valores alterados:

Let só funciona dentro de um bloco de código ou só funciona fora do bloco de código, no entanto é necessário declará-la antes de fazer uma menção a variável **LET**.

Hoisting: é possível atribuir um valor a um variável antes de declará-lo. Só funciona com o **VAR.**

**Const**: Utilizar o Snake\_Upper\_Case, seu valor não pode ser alterado nem redeclarar uma constante. Não possui Hoisting. Além disso, a atribuição de valor deve ser feito na declaração, pois caso isso não seja feito, a linguagem vai entender a tentativa de passar um valor como uma reatribuição.



## Estruturas de Dados

Coisas que são semelhantes em todas as linguagens de programação.

Javascript é uma linguagem de tipagem dinâmica. Ou seja, não é preciso especificar o tipo do valor que vc está declarando.

Divididos em dois grupos:

Primitivos: numbers, string, Boolean, null, undefined – A utilização desses métodos é em letra minúscula.

Compostos: objects e arrays. – A utilização desses métodos é feito por letra maiúscula, mesmo que o nome seja o mesmo dos tipos primitivos.

Ex. let nome = “João”; === string;

Let String = “João” === object.

### Strings

Comumente utilizada para textos;

Valores declarados entre aspas ou crase.

Para concatenação de textos é necessário utilizar crase.

Não esquecer que a contagem de índice começa no zero, mas a medição de tamanho começa no 1.

Existem diversos métodos para alterar strings, como o replace.

Método muito usado é o length.

### Numbers

Números inteiros ou decimais.

Para converter um dado em outro: Ex nome da variável.toString().

### Boolean

Valores verdadeiros ou falsos. Usado com valores de comparação com o == e o ===.

### Arrays

Lista iterável de elementos: propriedade do lenght e do index.

forEach() – itera um array;

push() – adiciona um item no final do array;

pop() – remove um item no final do array;

shift() – remove um item do início do array;

unshift() – adiciona um item no início do array;

indexOf() – retorna o índice do valor;

splice() – remove ou substitui um item pelo índice;

slice() – retorna uma parte de um array existente;

Outro importante são o mapFilter e Reduce.

### Objeto

Let person = {asdada};

Assim que se declara um objeto. Pode ser um dicionário de dados ou um hash map, também é muito comparado com uma agenda ou informações de cadastro de uma pessoa, pois é capaz de guardar diversos tipos de dados diferentes no mesmo objeto.

Para passar uma variável como parâmetro para outro objeto é feito entre [].

Ex: let person = {}

Let mom = {nameOfMom}

Person[mom] = Maria;

Chamada de person;

Person

{nameOfMom = maria,}

### Empty, null e undefined

Empty: o valor é declarado e colocado como 0;

Null: o valor é declarado, mas ele não existe;

Undefined: não é nem declarado o valor;

## Função

Estrutura: function nome (parâmetro){

return: retorna oq vc deseja e para o funcionamento;

}

Função anônima: funções que representam expressões.

Uma variável pode armazenar uma função

Const soma = function (a, b){

Return a+b;

}

Função autoinvocável:

IIFE (immediately invoked function expression)

Uma função anônima entre parênteses, seguida por outro par de parênteses, que representam sua chamada.

Ainda é possível passer parâmetro para essa função.

(

Function(){  
let name = “Dio”;

Return name;

}

)();

Callbacks:

Uma função passada como argumento para outra função; você passa a ter mais controle das chamadas de funções

Const calc = function(operação = outra função, num1, num2){

Return operação(num1, num2);

}

## Parâmetros

Valores padrão:

Hoje é possível colocar um número padrão para uma variável dentro da própria declaração do parâmetro.

Objeto “argument”:

Um array com todos os parâmetro passados quando a função foi invocada;

Function findMax(os parâmetros podem ser indefinidos, então é preciso chamar o arguments para saber){

Let max = - Infinity;

For(let i=0; i<argument.length; i++){

If(arguments[i]>max){

Max = arguments[i];

}  
}

Return max;

}

## Arrays e Objetos

Arrays: spread uma forma de lidar separadamente com elementos

O que era parte de um array se torna um elemento independente.

Function sum (x, y, z){

Return x + y + z;

}

Const numbers = [1, 2, 3];

Console.log(sum(…numbers));

Isso é importante pois ao separar os elementos, vc consegue passar todos os valores do array como parâmetro, pois antes o array como um todo seria um único parâmetro.

Rest: combina os argumento em um array;

O que era um elemento independente se torna parte de um array.

Function confereTamanho(...args){

Console.log(args.length)

}

Objetos:

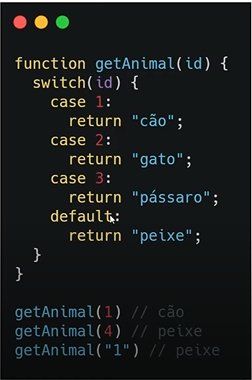
Object destructuring: entre chaves{}, podemos filtrar apenas os dados que nos interessam em um objeto;



## If/else e Switch

Condicionais, que verificam se um argumento é verdade e decide o que deve ser chamado.

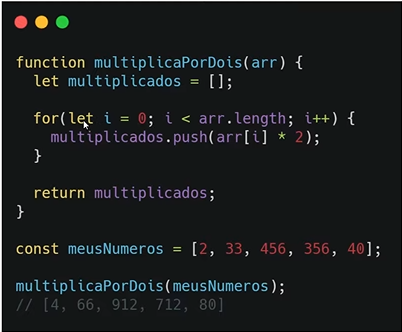
Switch case: serve para muitas verificações de condições, aqui vc pode ter comparação de tipo e valor, ou seja, consegue usar ===;



Nesse exemplo: o case 1 significa que é o caso em que o id = 1.

## For e While

For loop dentro de elementos iteráveis (arrays, strings)



For ... in

Em caso de objetos, caso vc queira percorrer todos as propriedades de um objeto, vc precisa fazer:

For (prop in obj){

Console.log(obj);

Isso vai passar o nome das propriedades.

}

For(prop in obj){

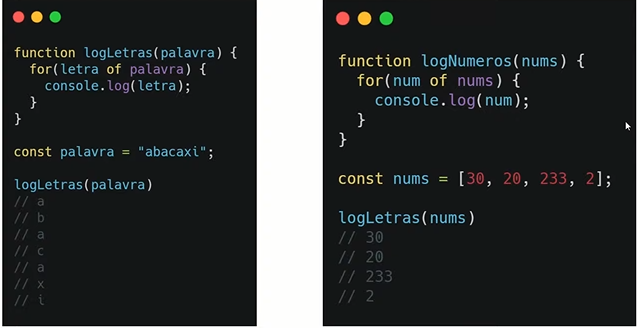
Console.log(obj[prop]);

Pega o valor de cada propriedade;

}

For ... of

Loop entre estruturas iteráveis (arrays, strings)



While: Executa instruções até que a condição se torne falsa.

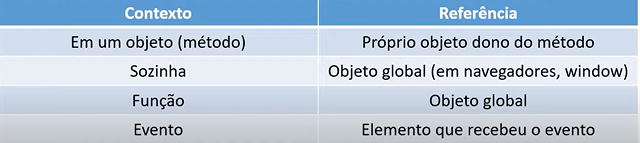
Do ... while: sempre executa a instrução uma vez e depois verifica a condição e ai vai parar só quando a condição seja falsa.

## This

A palavra reservada this é uma referência de contexto.



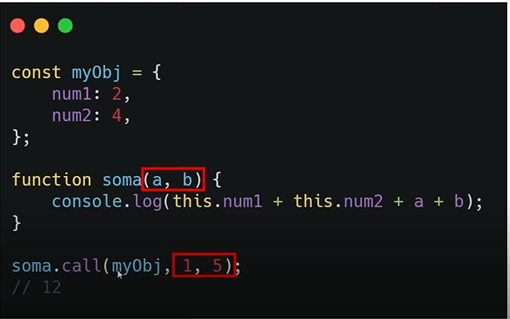
O this esta se referindo ao objeto pessoa e pegando a propriedade firstName e lastName...

O valor do this pode mudar de acordo com o lugar no código onde foi chamada.

Manipulação de valor do this:

Call: coloca o nome do objeto que vc quer que o this se refira. Ela também serve para passar os argumento para uma função que tenha parâmetros.



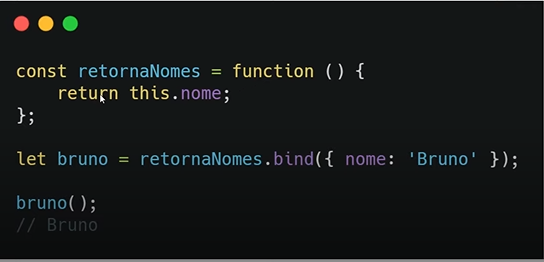


Apply: Implementação muito parecida com a call, com uma diferença, é possível passar parâmetros para essa função dentro de uma array.

Ou seja, é preciso passar as informações com []. Na imagem acima, o 1 e o 5 ficariam = [1, 5].

Bind:

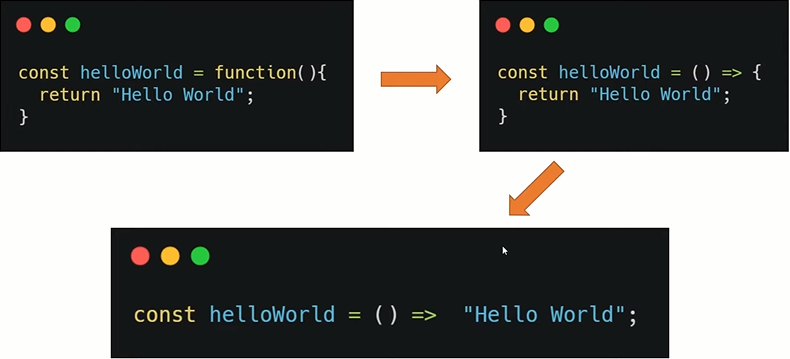
Clona a estrutura da função onde é chamada e aplica o valor do objeto passado como parâmetro.



## Arrow functions

Sintaxe

Representada por =>, equivale a uma constate que se refere a uma função, para tornar a vida mais fácil é só substituir o = pela =>.



Caso tenha uma tinha, não precisa das chaves e do return.

A arrow function não faz hoisting.

O valor this não podem ser usados na arrow function. Não existe o objeto arguments e o construtor não pode ser utilizado.

# Debugging e Error Handling

Tipos de Erros:

Objeto error do ECMA:

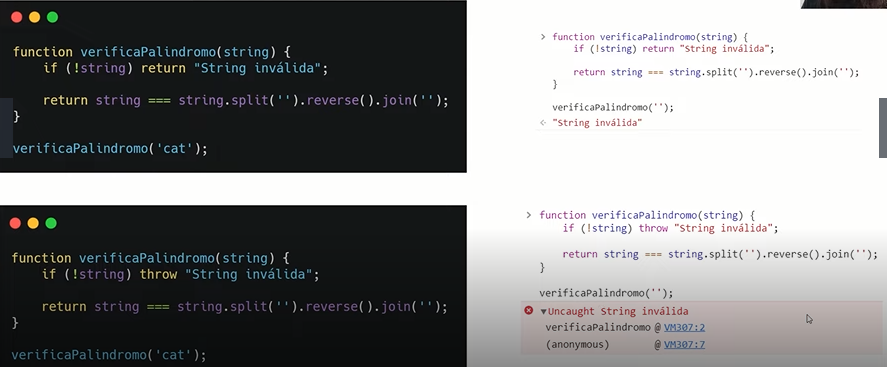
Acontece em tempo de execução: esqueceu alguma coisa, como uma declaração de variável ou de alguma virgula.

DOMexception:

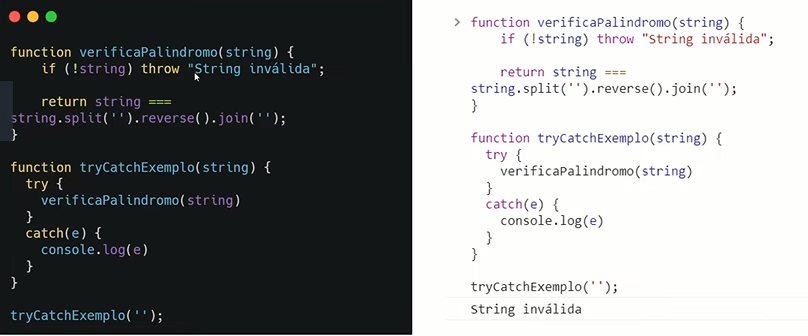
Erros relacionados a rodagem do código em uma página da Web. Caracteres inválidos ou elementos inválidos em partes do código etc.

## Tratamento de Erros:

Throw: Trata o que foi declarado como um erro propriamente dito. Funciona igual ao return, mas com a mensagem de erro e suas explicações.



Try/Catch: o try vai rodar o bloco de código e caso apareça um erro, o catch irá pegar o erro.



É possível utilizar o throw junto com o try/catch.

Finally: seria um complemento do try/catch. Ele funciona como um código que sempre irá rodar, não importa se um erro foi achado ou não.

## Criar Error Personalizados

O objeto Error: normalmente é usado somente a mensagem de erro, pois os outros parâmetros talvez não sejam suportados.



O erro também pode ter um nome específico:



# Assincronicidade

Não ocorre ou não se efetiva ao mesmo tempo;

O JS roda de maneira síncrona.

### Promises:

Objeto de processamento assíncrono. Inicialmente, seu valor é desconhecido. Ela pode, então, ser resolvido ou rejeitado. O resultado só virá depois, nesse meio tempo muita coisa pode acontecer. Pode ficar resolvida, pendente ou rejeitada.



### Async/Await

Funções assíncronas precisam dessas duas palavras-chave.

Para fazer uma função assíncrona é necessário usar ‘async’ antes do function.

O await serve para mudar o estado da linguagem JS, pois como ela é uma linguagem síncrona, ela nunca vai parar para ver o valor da promise, sendo assim, é necessário parar o código de forma manual com o ‘await’.

Mesmo na chamada de uma função assíncrona, é necessário usar o await antes de chama-lá, se não vai dar tudo errado.



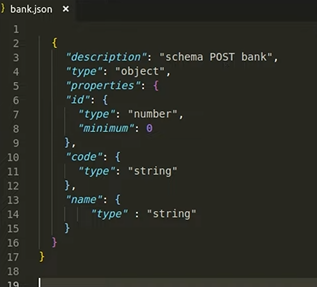
## Consumir API’s

### API’s

Application Programming Interface.

Uma API é uma forma de intermediar os resultados do back-end com o que é apresentado no front-end. Sendo assim, ela possibilita a comunicação entre plataformas.

Você consegue acessá-la por meio de URL’s.



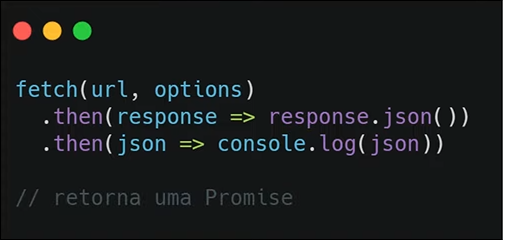
JSON: JavaScript Object Notation.

É muito comum que API’s retornem seus dados no formato .json, portanto precisamos tratar esses dados quando os recebermos. É um formato de troca de informações, muito utilizado na comunicação de front-ends com APIs.

O formato do JSON é a de um objeto do JS.

### Fetch

Consumidor API



É necessário usar o await quando for utilizar o fetch, pois ele retorna uma promise.

Você pode fazer operações no Banco de Dados (POST, GET, PUT, DELETE, etc).



# Orientação a Objetos

## Paradigmas

Imperativo e Declarativo:

Imperativo: como vc vai resolver os problemas.

Declarativo: No que vc vai fazer para resolver o problema

A orientação a objetos é do tipo Imperativo. É necessário falar como as coisas vão ser chamadas e realizadas.

Na OO os programas são objetos que possuem propriedades (nome, idade...)

Possuem 4 pilares:

Herança: O objeto filho herda propriedades e métodos do objeto pai, sendo os objetos filhos capazes de ter suas próprias características.

Polimorfismo: Objetos podem herdar a mesma classe pai, mas se comportarem de forma diferente quando invocamos seus métodos.

Abstração: Tornar um problema complexo em algo cada vez mais simples, simplificando sua avaliação, classificação ou para permitir a resolução de algo.

Encapsulamento: Cada classe tem propriedades e métodos independentes do restante do código.

## Protótipos e Classes

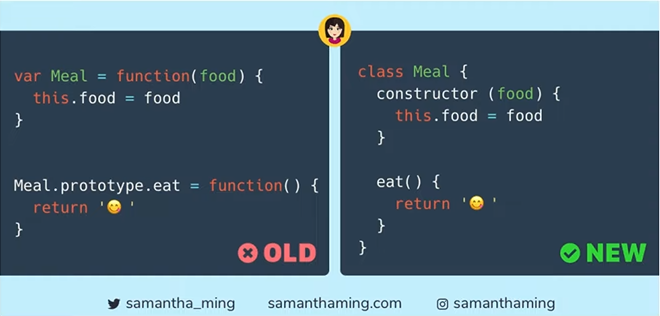
### OOJS

Protótipos: são esqueletos de todos os objetos em JS e o Object.prototype estará no topo da cadeia.

Todo objeto possui como classe pai o Object.prototype. Sendo assim, os métodos permitido de ser utilizados em objetos, estará definido na classe Object.prototype.

#### Classe:

Uma sintaxe feita para facilitar a escrita.



O JS não possui classes nativamente. Todas as classes são objetos e a herança se dá por prototypes.



Super() – Utilizar métodos construtores da classe pai. Passa os argumento enviados para classe ascendente.

# D.O.M

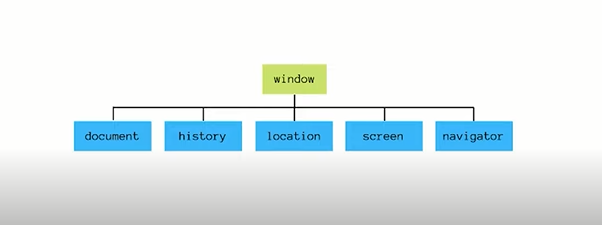
## Document Object Model

O DOM html é um padrão de como acessar e modificar os elementos html de uma página.

DOM vs BOM

Browser object model

Tudo o que está dentro do objeto window.



O BOM seria o window e o DOM é o document, ou seja, o DOM faz parte do BOM.

## Manipulação do DOM

Métodos:

Estrutura HTML:



Selecionando os elementos de uma página:





Estas chamadas retornam Arrays.

Outros métodos.

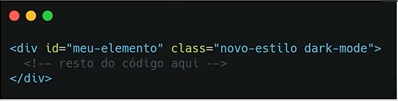


## Manipulação de estilos Html

Como acessar classes no CSS com o JS = Element.classList.

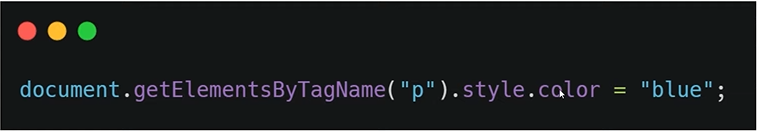


Resultado:



Este código simplesmente manipulou os elementos da classe que já estava criada no CSS.

Como acessar o CSS diretamente pelo JS, ou seja, não existe um arquivo css.



Utilização do .style

## Eventos

Qualquer tipo de ação que o usuário faça em uma página web.

### Tipos:

Eventos de mouse:

Mouseover, mouseout.

Evento de Click:

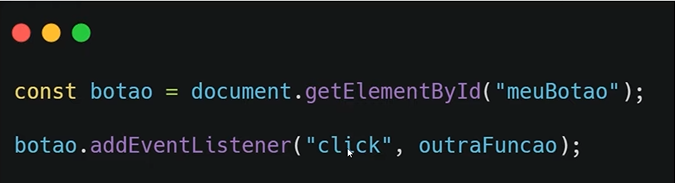
Click, dbclick

Eventos de atualização:

Change, load

### Acionar eventos

Event listener: diretamente no JS, cria um evento que vai ser acionado quando o usuário realizar determinada ação.



Atributo HTML: Especifica a função a ser chamada diretamente no elemento html.

