**JavaScript**

JS é uma linguagem interpretada

**Interpretada** - Tá sendo rodada em tempo real, o código tá sendo disponibilizado automaticamente, instantaneamente

Compilada - Antes de ser mandado pro console ou pro browser, voce precisa passar pelo compilador pro computador conseguir entender as instruções

Baseada em **protótipos** -

**Protótipo** - Uma base para uma grande quantidade de dados, que é um conjunto de funções em comum em todas essas estruturas

**Multiparadigma** - Pode escolher se quer POO, Funcional, etc

Comumente utilizada em aplicações Web Client-Side - Linguagem da WEB, linguagem da Internet

Segue o padrão **ECMAScript**

**Recursos Básicos**

* **Functions** - Métodos que fazem uma lógica
* **Console** - pra fazer teste
* **DOM** - Document Object Model - Elementos que existem na página

**hoisting** - Quando não pode, precisa declarar antes de usar.

**Let** - bloco - não posso redeclarar - posso reatribuir - nao posso hoisting

**Var** - global ou local - posso redeclarar - posso reatribuir - posso hoisting

**Const** - bloco - não posso redeclarar - não posso reatribuir - nao posso hoisting

Javascript é de **tipagem dinâmica**, ou seja, não precisa passar o tipo necessariamente

**Tipos**

**Primitivos** - Não tem método dentro deles - Letra minúscula

* - Numbers
* - Strings
* - boolean
* - null
* - undefined

**Compostos, não primitivos**

* - Objetos
* - Arrays

**String** - Textos ou frases

Entre aspas ou crases

$`{let}` - **Interpolação**

**split** = separa pelo que você passar dentro do split

**includes** = tem ? e ele retorna true ou false

**startsWith** = começa com isso que eu passei?

**endsWith** = termina com isso que eu passei?

**Principais métodos da String**

* Concatenação
* Propriedade Length
* Iterabilidade
* Formatação
* Index de letras

**Numbers** - Números inteiros ou aritméticos

**Floor** - arredonda pra baixo

**Ceil** - arredonda pra cima

**Booleanos** - Verdadeiro ou False

**Arrays - Vetores**

* Lista iterável de elementos
* Tem o length
* let array = []
* array.push - adiciona no final do array
* array.pop - remove o último do array
* array.shift - remove o primeiro do array
* array.unshift - adiciona no começo do array
* array.every - todos os itens
* array.some - algum deles é
* array.reverse - troca a ordem do array

**Objetos - Estrutura do tipo chave e valor**

* .keys - mostra as keys do obj passado
* .values - mostra os valores do obj passado

**Empty - Null - Undefined**

**Empty** = 0 - Declarou mas nao tem nada, só inicializou mas não tem valor dentro

**Null** = Null - Tem o valor null, existe mas é Nulo de fato

**Undefined** = Literalmente não foi declarado, nem inicializado, nem nada, ele é indefinido

**Functions**

Variáveis criadas dentro de uma função apenas podem ser utilizadas dentro dela

Quando invocamos o return, a função para de ser executada

Uma variável pode armazenar uma função

**Tipos de funções**

* **Função anônima** - Sem nome
* **Função auto invocável** - IIFE (Immediately invoked function expression), função anônima entre parênteses, seguida por outro par de parênteses, que representa sua chamada - NÃO PODEM RECEBER PARÂMETROS
* **Callbacks** - Função passada como argumento para outra

**Parâmetros**

* **Valores padrão** - Posso passar um valor padrão para a variável ex: function exponencial(num = 1){} - Se não for passado, é 1
* **Arguments** - Array com todos os parâmetros que foram passados quando a função foi invocada

**Arrays**

* **Spread** - Uma forma de lidar separadamente com elementos do array ex: ...numbers - numbers = array, assim ele pega os valores do array.
* **Rest** - Combina os argumentos em um array, contrário do spread

**Objetos**

Entre chaves, podemos filtrar apenas os dados que nos interessam do objeto

**Loops**

* **If/Else**
* **Switch** - Comparação de tipo e valor, precisa de um default no final, ideal quando se precisa comparar muitos valores
* **For**
* **For...In** - Loop entre propriedade enumeraveis, utilizado para objetos
* **For...Of** - Igual o de cima, só que para string ou array
* **While** - Executa até que seja falsa
* **Do While** - Igual a de cima, mas pelo menos uma vez ele executa

**This**

* É uma referência de contexto
* **Call** - eu passo o objeto que eu quero que ele se refere - É possivel passar parâmetros para essa função separado por virgulas
* **Apply** - diferença pro call, é que tu passa dentro de um array [1,2], no call é 1,2
* **Bind** - Clona a estrutura da função e aplica o valor do objeto passado como parâmetro

**Arrow Functions - =>**

Caso exista apenas uma linha, pode tirar chave e return

se existir apenas um parâmetro, pode tirar o parênteses

Arrow não faz hoisting, funções normais fazem

**Outras restrições da Arrow**

This sempre vai ser do objeto global

Não existe o objeto arguments

o construtor (new Objeto()) também não pode ser utilizado

**Collections**

**Map**

Uma coleção de arrays no formato [chave, valor], pode ser iterado por um loop for...of

**Métodos**

* .set - sempre chave e valor
* .get - passa a chave e ele retorna o valor
* .delete - passa a chave e ele deleta ambos. Chave e valor
* Se eu quiser retornar depois de deletado, vai voltar undefined

**Map x Objeto**

* Maps pode ter chave de qualquer tipo, objeto é sempre no formato de string
* Map possui length, objeto só se iterar
* Mais fáceis de iterar (Map)
* Utilizado quando o valor das chaves é desconhecido...
* Os valores tem o mesmo tipo

**Set**

Sets são estruturas que armazenam apenas valores unicos (não pode se repetir)

**Métodos**

* .add
* .has - ver se existe um valor dentro (retornar true ou false)
* .delete

**Set x Array**

* Em vez de length, no set se usa o size
* Set não possui os métodos map, filter, reduce e etc

**Array - Map, Filter e Reducer**

**Map**

Cria um novo array, ela não modifica o array original

Realiza as operações em ordem

**Sintaxe:**

array.map(callback, thisArg)

Callback - Função que vai ser executada em cada elemento

this Arg(Opcional) - valor de this dentro da função de callback

**Map x forEach**

* Valor de retorno
* Considera se o array auxiliar será necessário

**Filter**

Literalmente um filtro para seu array, apenas os itens que serão da condição passada serão retornados

**Sintaxe:**

array.filter(callback, thisArg)

Callback - Função que vai ser executada em cada elemento

this Arg(Opcional) - valor de this dentro da função de callback

**Reduce**

Executa uma função em todos os elementos do array e retorna um valor unico

Sintaxe:

array.reduce(callback, initialValue)

Callback - Função a ser executada a partir do acumulados

InitialValue - Valor sobre o retorno ira atuar

**Acumulador** - acumulador de todas as chamadas do CallBack

**Current Value** - elemento atual sendo acessado pela função

**POO**

**Paradigmas**

* **Imperativo** - Como vai resolver - POO é
* **Declarativo** - No que voce vai fazer - BD em SQL ou Programação Funcional

**Pilares**

* Herança
* Polimorfismo
* Encapsulamento
* Abstração

Caso um parâmetro tenha getter e setter, ele deve ser nomeado com \_ no começo

* **Herança** - O objeto filho herda propriedades e métodos do objeto pai
* **Polimorfismo** - Objetos podem herdar a mesma classe pai, mas se comportarem de forma diferente quando invocamos o método
* **Encapsulamento** - Cada classe tem propriedades e métodos independentes do restante do código - Só influencia naquela classe
* **Abstração** - Isolar algo, a fim de simplificar a sua avaliação. Tornar cad vez mais simples - Generalizar

**OOJS**

Protótipos - Todos os objetos Javascript herdam propriedades e métodos de um prototype.

**Classes**

Syntactic sugar - Uma sintaxe feita pra facilitar a escrita

JS não possui classes nativamente, todas as classes são objetos e a herança se dá por protótipos

**Assincronicidade - JS assíncrono**

**Assíncrono** - Que não ocorre ou não se efetiva ao mesmo tempo

JS roda de maneira síncrona - faz uma coisa, termina e dps começa outra

no assincrono, você pode fazer varias coisas ao mesmo tempo

**Promises** - Objeto de processamento assíncrono, que pode ser resolvido ou rejeitado - No momento que você recebe você não sabe o valor, depois ele é:

- resolved = .then()

- rejected = .catch()

**Estados**

* **Pending** - Pendente
* **Fulfilled** - Completado
* **Rejected** - Deu errado

**Async/Await**

* Funções assíncronas precisam destas duas palavras chaves
* Funções assíncronas também retornam Promises

**APIs - Application Programming Interface**

* É uma forma de intermediar os resultados do back-end com o que é apresentado no front-end.
* APIS retornar muito JSON
* **JSON - JavaScript Object Notation**

**Fetch - Usado para consumir a API**

Fetch retorna Promise

**Módulos**

Módulos tem a capacidade de exportar e importar informações de outros arquivos do mesmo tipo.

**Vantagens**

* + Organização do Código
  + Compartilhamento de variáveis em escopos diferentes
  + Explicita as dependências dos arquivos

**Exportar –**

* + **Named exports** – Quando você quer exportar mais de uma coisa e exportar uma de cada vez
  + **Default exports** – Só pode ter 1 por arquivo e vai ser o retorno padrão do arquivo.

**Importar –**

* + **Named exports** – Quando você quer importar mais de uma coisa e importar uma de cada vez
  + **Default exports** – Só pode ter 1 por arquivo e vai ser o retorno padrão do arquivo.

Podemos dar apelido! Ex: import {arquivo as Apelido} from

Importar tudo = \*

**Vincular ao HTML**

Na tag SCRIPT, coloque type=”module” (é preciso estar rodando um servidor, um live server da vida)

**Curiosidades**

Módulos sempre estão em strict mode

Pode ser utilizado em JS e MJS (módulos Java script)

Ao importar, precisa da extensão

Sempre usar ./ como ponto de partida

**Tratamento de Erros**

**Tipos de Erros**

* ECMAScript Error – Erros que ocorrem em tempo de execução

**Composto por**:

* + - 1. Mensagem
      2. Nome
      3. Linha
      4. Call Stack
* DOM Exception – Erros relacionados ao Document Object Model (DOM) relacionado a página WEB

**Tratando Erros**

Throw – Ele retorna o que eu passar, como um erro mesmo

EX:

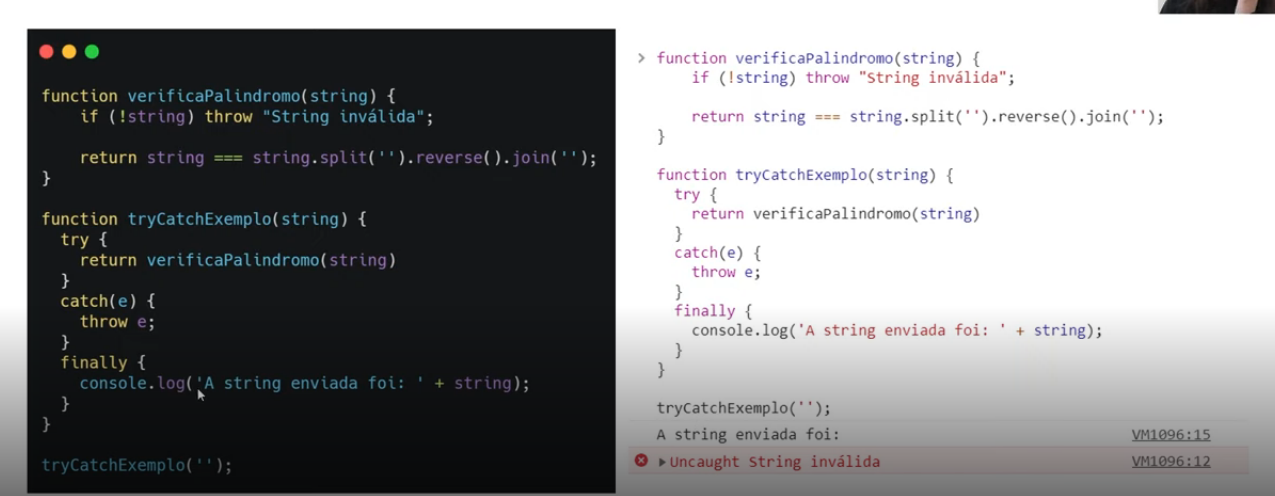


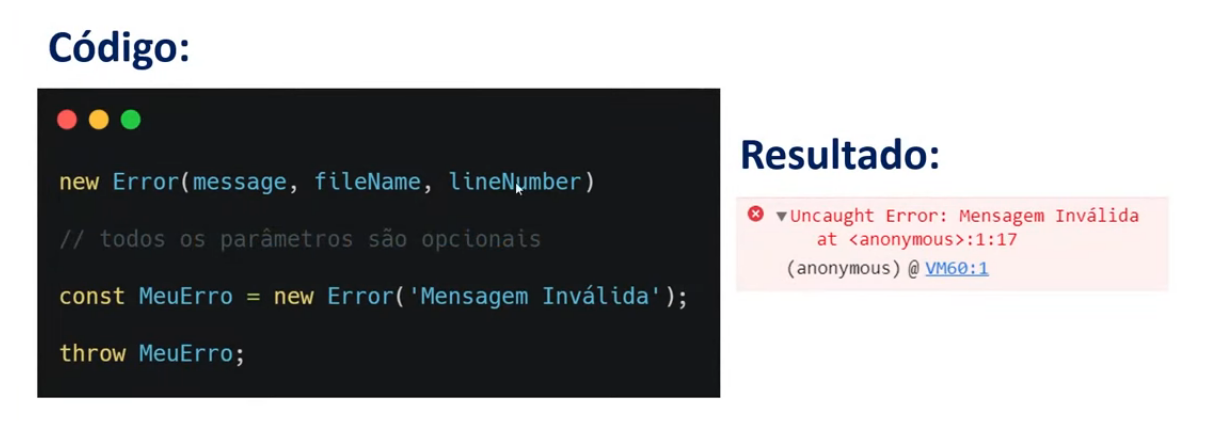
Return – Ele retorna a string, ou o number..Depende do que você colocou



**Try...Catch –** Dentro do try você verifica, se tiver algum tipo de erro, ele vai cair pro catch – Permite que a gente manipule o erro de uma maneira mais personalizada

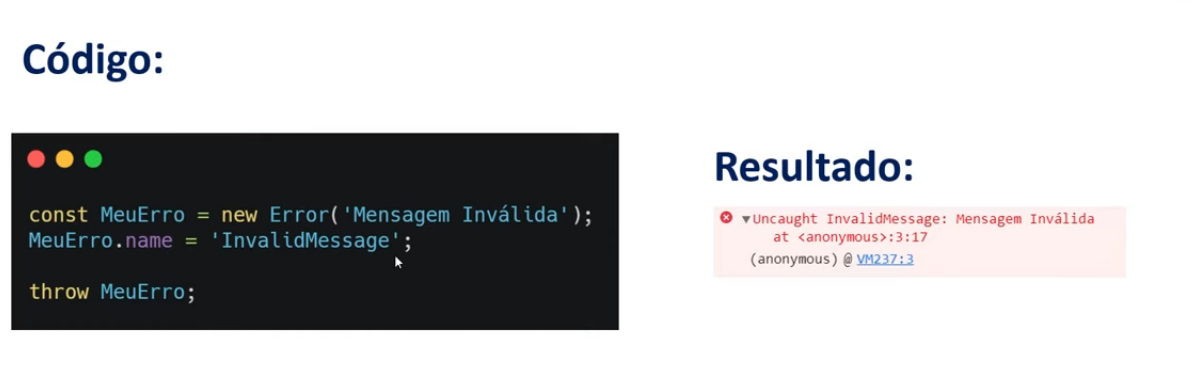
**Finally** – Uma instrução que vai ser chamada independente se caiu no try ou no catch...

**  
O Objeto Error**



Geralmente utilizada só a mensagem...

**Colocando nome no erro:**

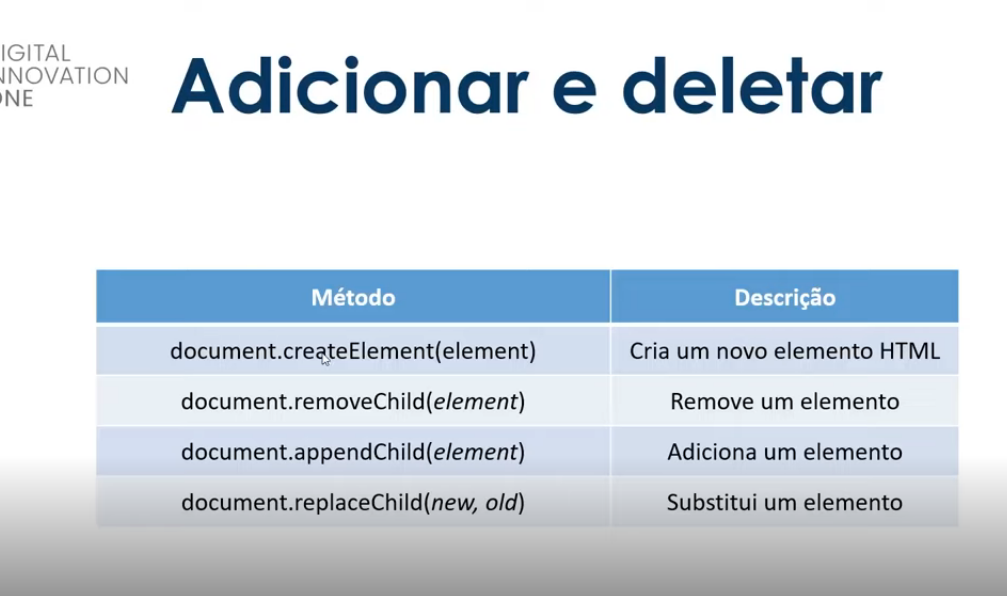


**DOM, o que é**

**Document Object Model –** Padrão de como acessar e modificar os elementos HTML de uma página – Cada elemento é chamado de Nó

**DOM X BOM**

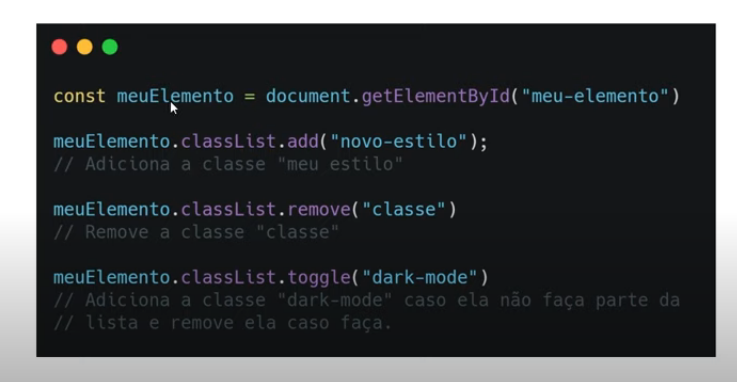
BOM – Browser Object Model – Tudo que está dentro do objeto window



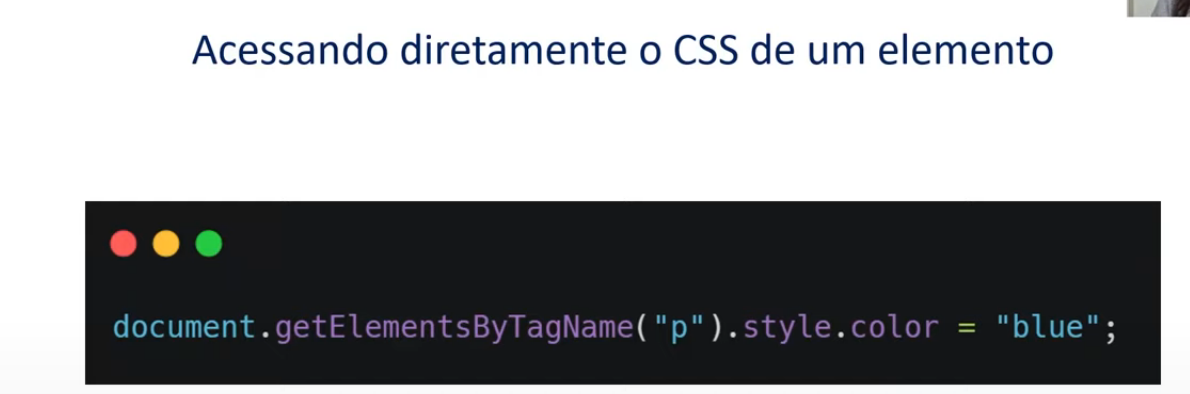
QuerySelectorAll – pode fazer a busca por mais que uma única classe

Retorna array

**Adicionado classe**



**Adicionando CSS pelo JS**

****

**Eventos –** Qualquer tipo de ação que o usuário faz numa página WEB

* Eventos do Mouse

1. Mouse Over
2. Mouse out

* Eventos de Clique

1. Click
2. dbClick – Dois cliques

* Eventos de Atualização

1. Change
2. Load



**Alterando dentro do HTML**

