**AULA 01 - TIPOS DE ERROS**

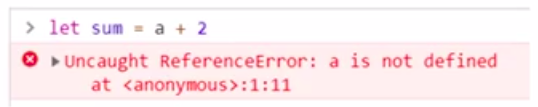
1. ECMAScript Error:

É um erro que acontece em tempo de execução, ou seja, quando o código está rodando ele percebe que algo tá errado, uma variável não foi declarada, faltou um ponto-vírgula, etc.

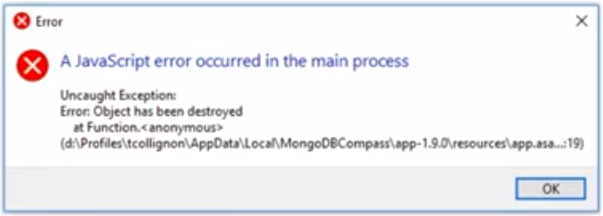
Ele é composto por: uma mensagem, nome do erro, linha em que o erro aconteceu e call stack que é uma pilha de chamadas que aconteceram no código até chegar nesse erro específico.

Exemplos:

Erro do console do navegador:



Erro do sistema operacional:



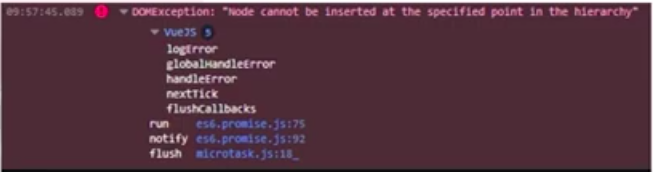
1. DOMException

São erros relacionados ao DOM. Eles ocorrem quando fazemos a relação entre o nosso código javascript à uma página da Web (ao código que está sendo rodado na página da Web), ou seja, uma página que está consumindo esses dados.

Por exemplo. Erro de string com caractere inválido:



Erro: node (nó) não pode ser inserido onde estou tentando inserir, ex: tentando inserir um nó dentro de um elemento da árvore DOM que é inválido.

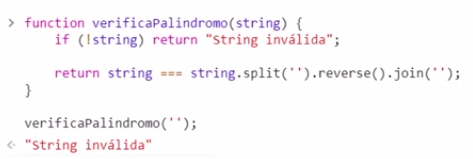


**AULA 02 - TRATANDO ERROS**

1. Throw:

Chamando o throw (figura abaixo) ao invés de return, ele vai tratar a string “String inválida” como um erro. E aí quando faço a chamada da função **verificaPalindromo(‘ ’)**, teremos um erro (Uncaught String inválida) com algumas propriedades (que veremos logo mais a frente):

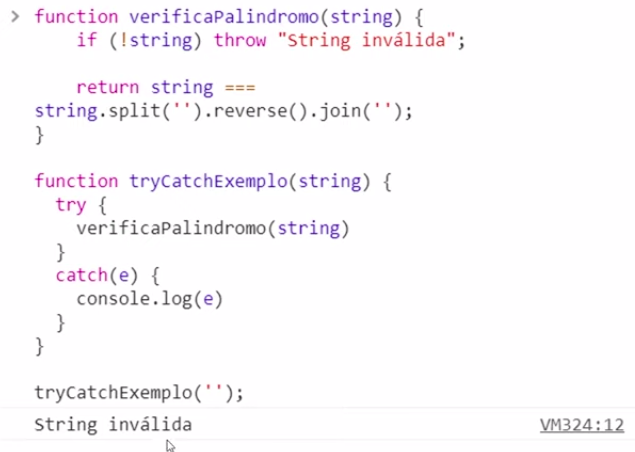
****

Repare que string inválida não está entre aspas, ou seja, não é uma string que definimos como return, conforme o exemplo abaixo: 

1. Try… catch:

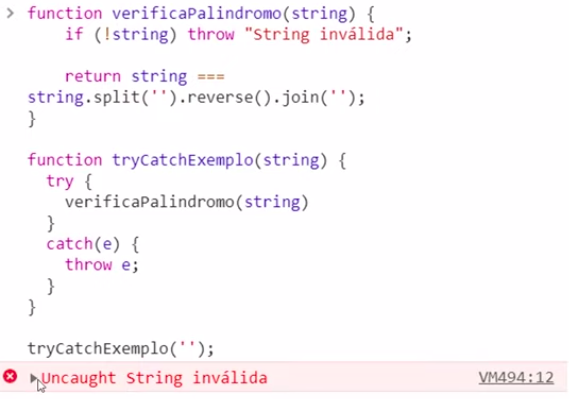
Dentro do bloco do try vou verificar um pedaço de código que eu determinar (no exemplo abaixo isso foi feito com a função **verificaPalindromo**) e se esse pedaço de código tiver algum tipo de erro ele vai ser capturado com a função catch.

Dentro do catch posso manipular o erro e todas as suas propriedades da forma que eu quiser. No ex abaixo estou pedindo para ele logar o erro através de **console.log(e)**:



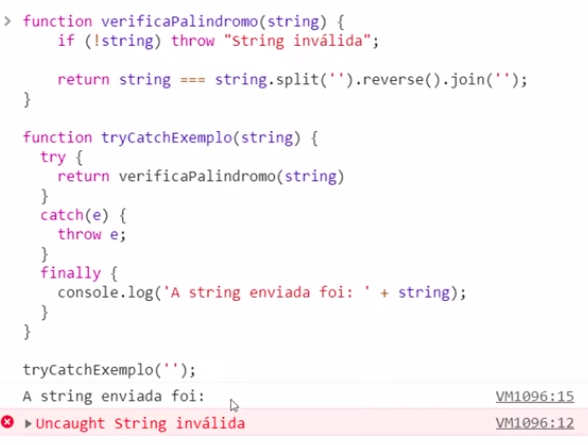
Veja que acima temos um console log de String inválida, não temos o erro *uncaught* que vimos com *throw* e também não temos aspas, ou seja, não é uma string igual temos em return.

Agora se colocarmos o **throw** como nosso **catch** teremos um erro vermelhinho parecido com o que vimos em throw:



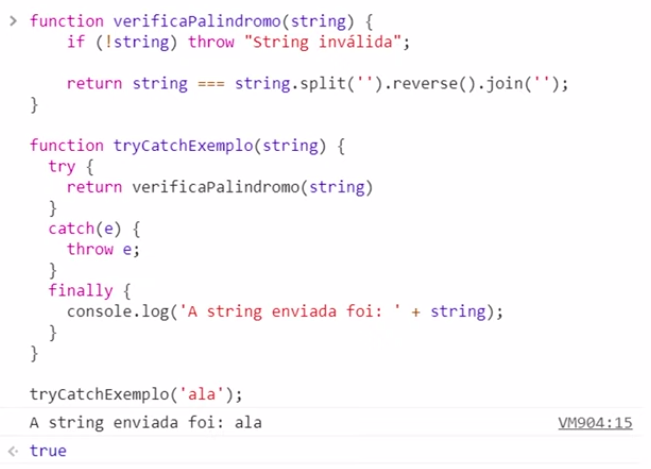
1. Finally:

O bloco **try catch** pode ser também complementado com o **finally**. Ele é uma instrução que será executada de qualquer forma, independente de ter erro ou não no código testado com o **try**.



Acima podemos ver que tivemos o retorno do **finally** mesmo tendo nosso erro no código.

Agora vamos ver o resultado de **finally** sem erro:



O finally agora retorna algo como valor junto à frase “A string enviada foi:” pois dessa vez demos um valor a ela. E o retorno lançado com **throw** agora é true que é a validação da nossa string.

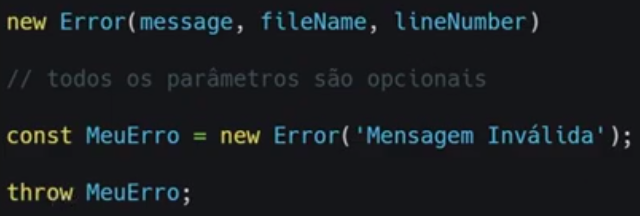
**AULA 03 - CRIANDO ERROS PERSONALIZADOS**

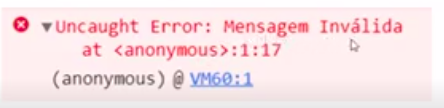
É possível manipular o **Error** que é um *objeto* que já vem do Javascript e que podemos colocar uma série de propriedades e executar uma série de métodos dele.

Ele aceita os seguintes parâmetros: *message, fileName* e *lineNumber*. E todos estes parâmetros são opcionais, porém *fileName* e *lineNumber* não são padrão então pode ser que algum browser não seja capaz de entendê-lo (consultar documentação para entender a compatibilidade e como esses parâmetros são aceitos, etc).

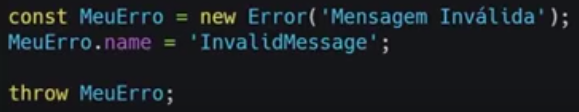
Por isso, a maioria das pessoas utilizam somente o parâmetro *message*.

Vamos manipular o erro criando uma constante e dando um nome a ela (**MeuErro**). Depois crio uma nova instância para o erro usando **new Error** e nele utilizo o parâmetro que quiser, neste caso foi apenas *message* (‘Mensagem Inválida’). Por último vou dar um **throw** no MeuErro e teremos como resultado o Uncaught Error abaixo mostrando também a linha onde está o erro (1:17):

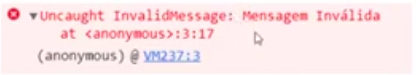




O **Error** pode também ter um nome, então, se quiser que o nome do erro seja ‘InvalidMessage’, posso depois de declarar meu erro (na primeira linha do código) acessar um método do Error através do ponto (linha dois: **MeuErro.name**) e digo que o nome do erro agora tem o nome ‘InvalidMessage’ como propriedade:



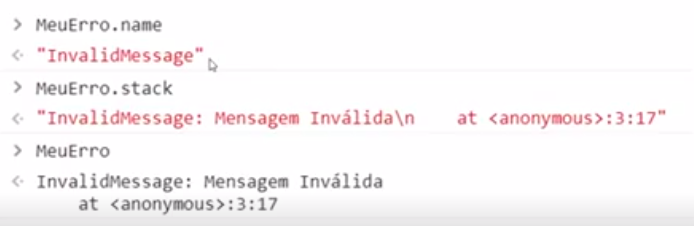
O resultado é:



Agora além de mostrar Mensagem Inválida, meu erro vai retornar também sua propriedade nome ‘InvalidMessage’.

Sempre virá -> nome: mensagem. Abaixo temos a stack com o número da linha etc, e o que foi executado até chegar no erro (anonymous) @ VM237:3.

Se formos no console do navegador, podemos testar chamando cada uma das propriedades do objeto **Error:**



Acima ele está mostrando que o name é “InvalidMessage”.

Depois mostra a stack (a pilha de informações sobre o erro) o que foi executado até então “InvalidMessage: Mensagem Inválida\n” e onde o erro foi encontrado “at <anonymous>:3:17” (linha 3 coluna 17).

E se der um console.log em MeuErro ele apresenta o nome, a mensagem e a stack (tudo).

**ATIVIDADE PRÁTICA**

Pasta **debug-error** do GitHub da Stephany: <https://github.com/stebsnusch/basecamp-javascript/tree/main/debug-error>

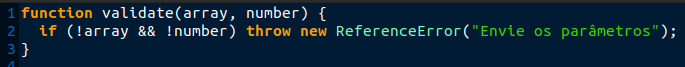
## **Atividade: validação de erros por tipo**

O objetivo é que a função receba um array e retorne ele caso o seu tamanho corresponda ao número enviado como parâmetro na função. Caso contrário, um erro será lançado.

* Crie uma função que recebe um array e um número



* Realize as seguintes validações
  + Se os parâmetros não forem enviados, lance um erro do tipo ReferenceError



* + Se o array não for do tipo 'object', lance um erro do tipo TypeError



* + Se o número não for do tipo 'number', lance um erro do tipo TypeError

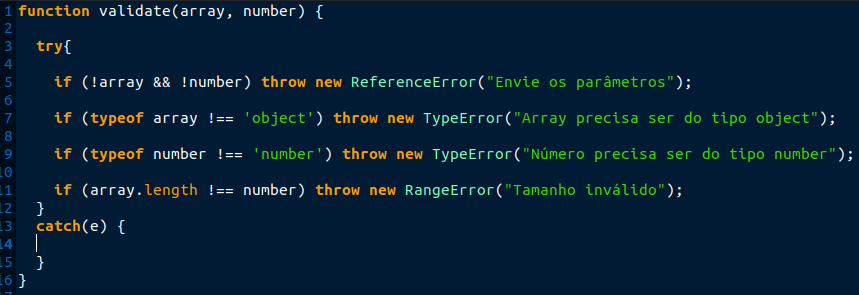


* + Se o tamanho do array for diferente do número enviado como parâmetro, lance um erro do tipo RangeError

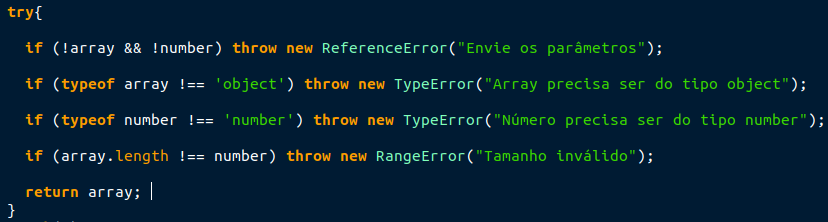


* Utilize a declaração try...catch

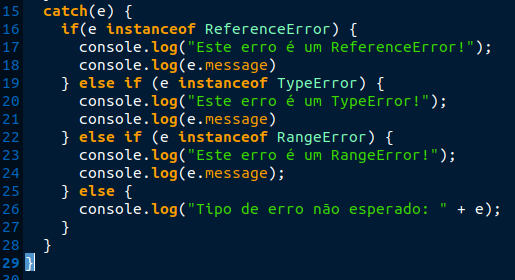
Agora é só colocar tudo isso que fizemos dentro do **try catch:**



E é claro, caso nosso array seja válida, temos que retorná-la (return array):



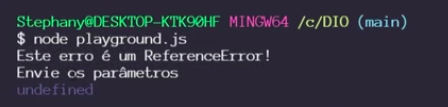
* Filtre as chamadas de catch por cada tipo de erro utilizando o operador instanceof

****

Agora chamo minha função com uma string vazia. Assim esperamos o erro “ReferenceError”. Ele retornará também o *undefined* pois não temos o retorno da array pois está vazia.



Retorno:

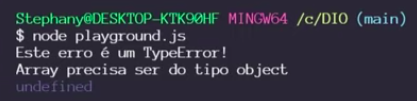


Agora verificando se a array não for do tipo *object*.

Mandei um numero ao inves de array e quero 5 itens nela, vai dar erro de tipo pois mandei numero e não array:



Retorno

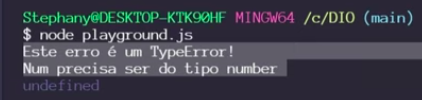


Agora vamos verificar o number.

Quero uma array com “a” itens, a não é numero então vai dar TypeError:

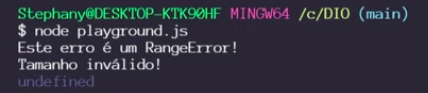


Retorno



Agora vamos ver o RangeError.   
Quero uma array com 5 itens dentro, mas mando uma vazia, ou seja, vai dar RangeError:





Agora vamos colocar uma array válida (com cinco itens, e também solicitando 5 itens):



Retorno sem erro:

