História:

O JavaScript foi criado na década de 90 por Brendan Eich a serviço da Netscape. Essa década foi um período de revolução, pois os browsers ainda eram estáticos. O navegador mais popular dessa época era o Mosaic, da NCSA. A Netscape foi fundada em 1994 para explorar a Web que estava surgindo. Foi então criado o Netscape Navigator. Em pouco tempo, este se tornou o browser dominante nessa década. Muitos desenvolvedores da NCSA foram designados no projeto do Navigator. A Netscape chegou à conclusão que a web teria que se tornar mais dinâmica, pois o Navigator tinha sempre que fazer uma requisição ao servidor para obter uma resposta no navegador. Em 1995, a Netscape contratou Brendan Eich para criar uma linguagem que proporcionasse isso.

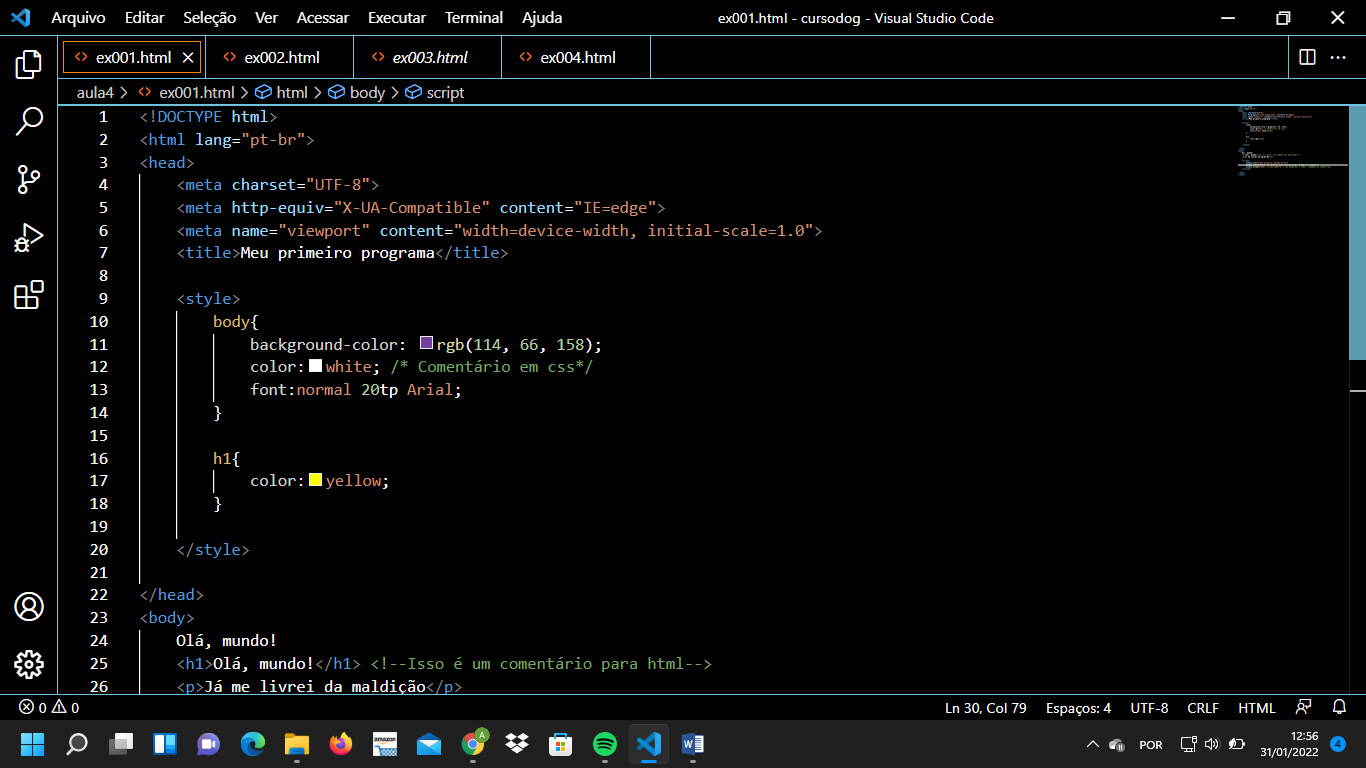
Aula 1:

Para desenvolver o JavaScript você deve ter pronto uma estrutura baseadas em HTML, e CSS. HTML é uma linguagem de marcação utilizada na construção de uma página web, e o CSS é um mecanismo para adicionar estilo a um documento web. Que através de tags e atributos, é possível criar um página web.



Exemplo de um documento HTML

Com a abreviação “!” no editor de código-fonte, Visual Studio Code criado pela Microsoft, é possível criar esse modelo base automaticamente.

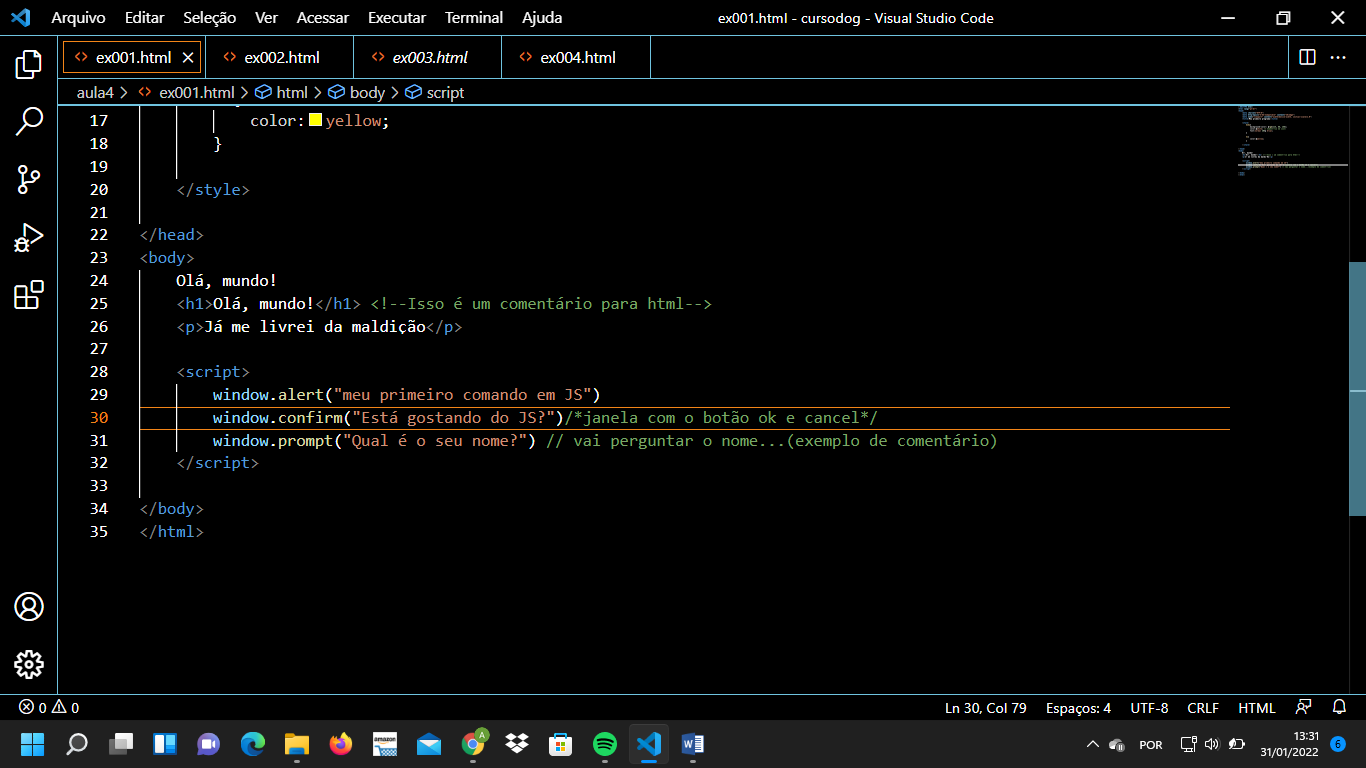


Então para começar criei uma pasta para armazenar meus documentos criados daqui em diante, pois a base para se tornar um bom programador é:

* Ser organizado.
* Treinar bastante.
* Tentar dar nomes curtos e relevantes, aos documentos, “ids”, var, com e intensificadores no geral...
* Não esquecer das aspas nas strings do node!

Nesse primeiro documento adicionei a tag “h1” que é uma tag designada para títulos, e a tag “p” designada para parágrafos, esses textos serão apresentados na minha página web assim que for salva e carregada.

Depois estilizei-a com algumas propriedades CSS, para então testar as ferramentas: “alert”, “confirm”, “pronpt”.



* Alert: você pode usar uma caixa de alerta para exibir dados.
* Confirm: você pode usar a caixa confirm, para confirmar a interação de uma pergunta com o usuário.
* Prompt: você pode usar a caixa de prompt, para coletar a resposta de uma interação com o usuário.

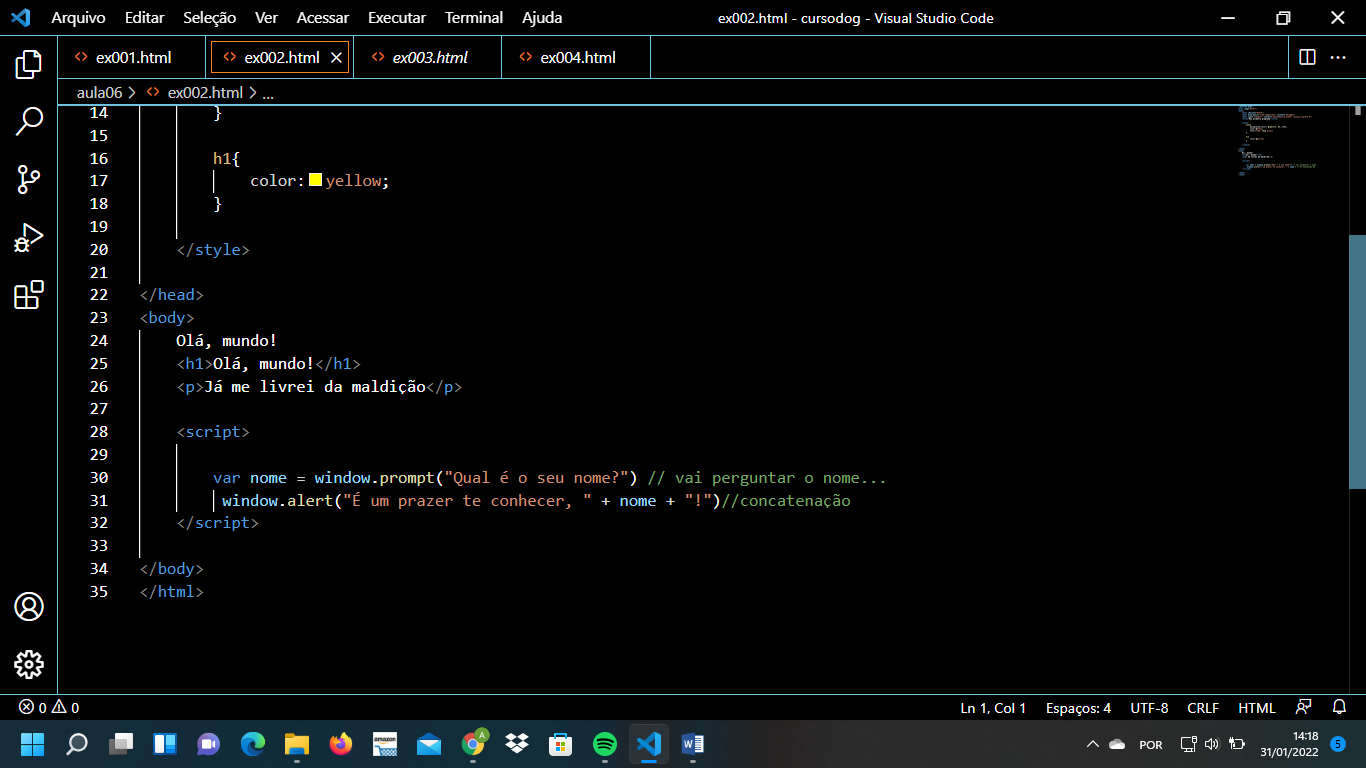
Aula2:

Variáveis e Tipos primitivos

No JavaScript Moderno, além de utilizar a palavra “var”, também podemos usar a palavra “let”.

O nome de uma variável “var ou let” é chamado de identificador

* Podem começar com letra, $ ou \*\_
* Não podem começar com números
* É possível usar letras e/ou números
* É possível usar acentos e símbolos
* Não podem conter espaços
* Não podem ser palavras reservadas



Aqui aplicamos à uma variável “nome”, uma caixa de prompt. Pegamos este valor e o apresentamos em uma caixa de alert, concatenado com uma frase.

Node.js

CTRL + L = Limpa o terminal “node”

Dicas:

* Maiúsculas e minúsculas fazem diferença
* Tente escolher nomes coerentes
* Evite se tornar um programador alfabeto

ou um programador contador

Para sair do terminal = .exit

Variáveis são usadas para armazenar dados!

Tipos primitivos primordiais

Number

Boolen

String

Data Types

* Number
* Infinity
* NaN
* String
* Boolen
* Null
* Underfined
* Object
* Arry
* Funtion

Valores primitivos

Typeof

É um operador para determinar o tipo de dado de uma variável!

Se colocar uma informação entre ”...”, ela é “...”.

Aspas / String

Colchetes ou chaves/ object

Nada / number

Manipulação de dados:

Para converter de:

String para number:

* Number.parseInt(n) => de um número para, um número inteiro.
* Number.parseFloat(n) => para um número real.

Number/String:

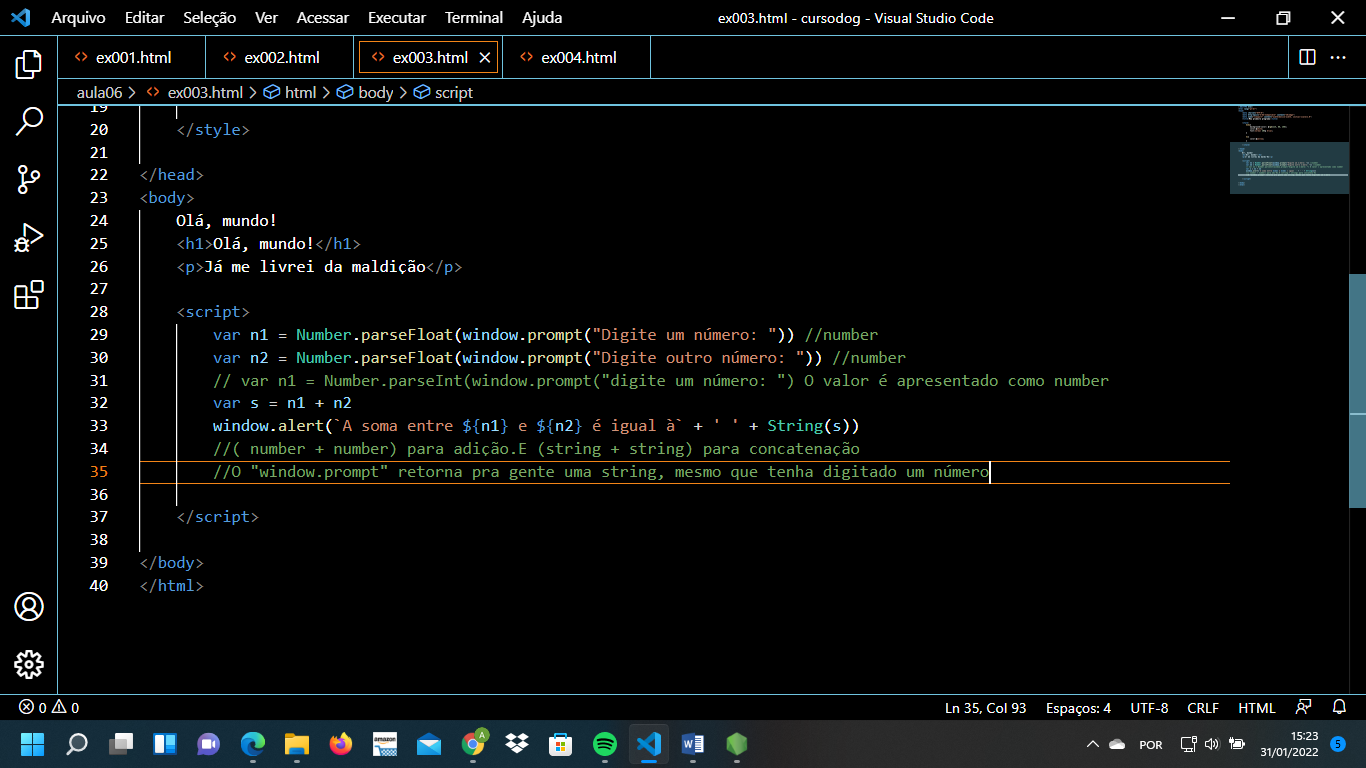
* String > number

ou

* n.toString( )

(``) > é a definição de templin String!

É um string interpolada



//Formatando uma string

Var s = ‘JavaScript’

‘Eu estou aprendendo s’ => não faz interpolação

‘Eu estou aprendendo’ + s => usa concatenação

`Eu estou aprendendo $ {s}` => usa template string

Entre crases

Placeholder

s.length //quantos caracteres a string tem

s.toUpperCase () //tudo para maiúsculo

Não esquecer de abrir e fechar parênteses

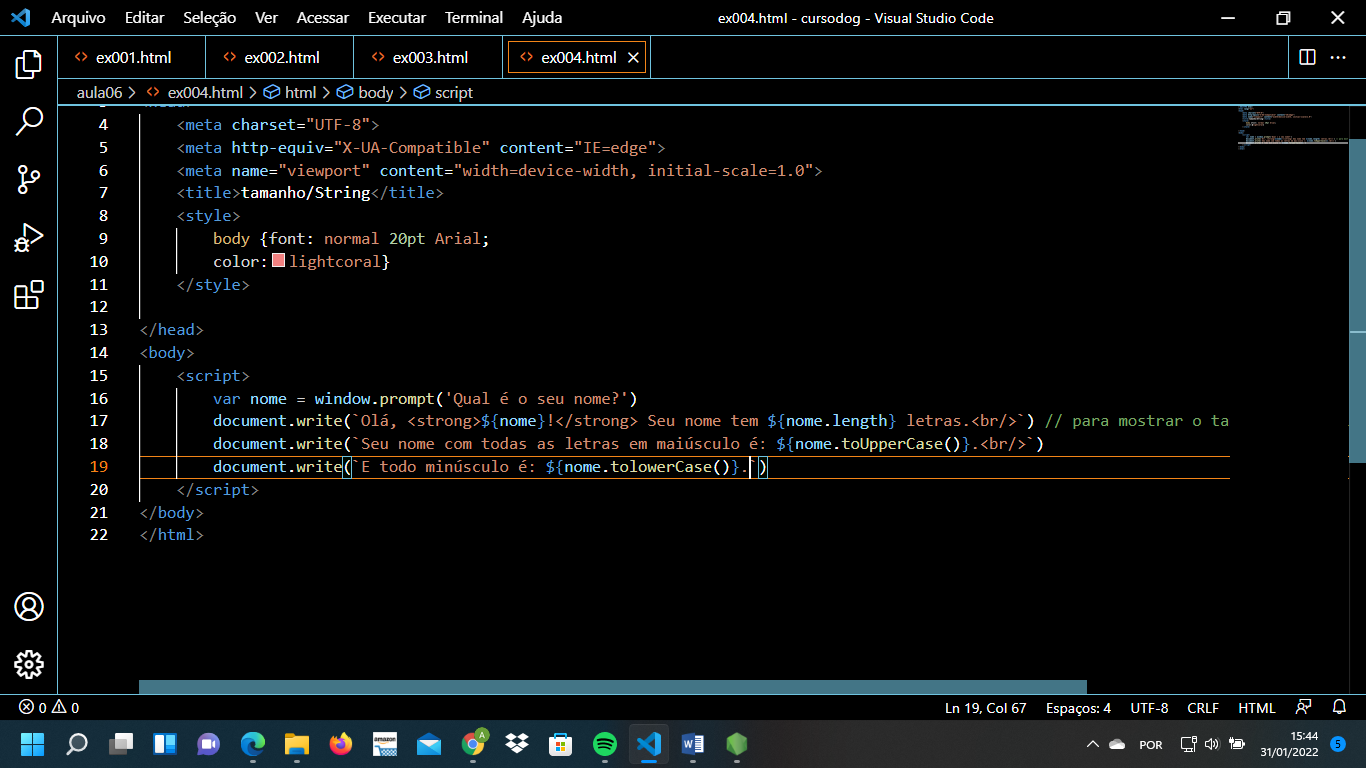
s.tolowerCase() //tudo para minúsculo

document.write( ) //escrever na tela

documet.Writeln( ) //escrever e pular para linha de baixo

ou

<br/> //pular para linha de baixo



//formatação de numbers

Numa variável qualquer var n1 = 1541.5

* Pra adicionar uma casa decimal depois do ponto/virgula,basta: n1.toFixed(2)

À variável vai rodar da seguinte maneira: 1541.50

* Para colocar uma virgula no lugar do ponto, basta: n1.toFixed(2).replace(‘.’,’,’)

A variável n1 vai ficar: 1541,50

* Para adicionar o símbolo de real ou de dólar...,basta: n1toLocaleString(‘pt-br’,{style:’currency’,currency:’BRL’})

n1 vai ficar: R$ 1,541.50

* Para dólar: n1toLocaleString(‘pt-br’,{style:’currency’,currency:’USD’})

Para euro basta mudar o final para EUR

Operadores

O JavaScript possui vária famílias de operadores:

* Aritiméticos
* Atribuição
* Relacionais
* Lógicos
* Ternários

Aritimétricos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | 5 | + | 2 | => | 7 |
| - | 5 | - | 2 | => | 3 |
| \* | 5 | \* | 2 | => | 10 |
| / | 5 | / | 2 | => | 2,5 |
| % | 5 | % | 2 | => | 1 |
| \*\* | 5 | \*\* | 2 | => | 25 |

Todos esses operadores, são operadores binários (que precisão de 2 operários para funcionar).

% => é um operador de resto da divisão inteira

\*\* é potência

Cuidado!

5 + 3 / 2 = 6,5 e não 4

Isso por que em JS sempre se começa pela divisão.

* Para muda a ordem de aparência basta adicionar parênteses “()”

ex: (5 + 3) / 2 = 4

Ordem de aparência:

|  |
| --- |
| ( ) |
| \*\* |
| \* / % |
| + - |

Quem aparecer primeiro

Atribuição

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| var | a | = | 5 + 3 | 8 |
| var | b | = | a % 5 | 3 |
| var | c | = | 5 \* b \*\* 2 | 45 |
| var | d | = | 10 – a / 2 | 6 |
| var | e | = | 6 \* 2 / d | 2 |
| var | f | = | b % e + 4 / e | 3 |

Alto-atribuição:

var n = 3

n = n + 4 => 7

então “n” vai deixar de valer 3, e valerá 7.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| n | = | n | - | 5 | => | 2 | n -= 5 |
| n | = | n | \* | 4 | => | 8 | n \*= 4 |
| n | = | n | / | 2 | => | 4 | n /= 2 |
| n | = | n | \*\* | 2 | => | 16 | n \*\*= 2 |
| n | = | n | % | 5 | => | 1 | n %= 5 |

SIMPLIFICANDO!

Se a mesma variável receber ela + 4 ( n = n + 4 ). Pode-se reescrever isso de uma maneira encurtada. (n += 4)

Operador de incremento

Var x = 5

++x

--x

Pré incremento você soma ou subtrai uma unidade, e a resposta aparece de imediato.

X = x + 1 x++

X = x – 1 x--

Pós incremento você soma ou subtrai uma

unidade, mas só aparece depois quando você

chama a “var”.

Relacionais

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| > | 5 | > | 2 | => | True |
| < | 7 | < | 4 | => | False |
| >= | 8 | >= | 8 | => | True |
| <= | 9 | <= | 7 | => | False |
| == | 5 | == | 5 | => | True |
| != | 4 | != | 4 | => | False |

* Para toda expressão que tenha um operador relacional ligado á ela, o resultado dessa expressão vai ser sempre um valor BOOLEANO (DO TIPO VERDADEIRO OU ERRADO).

!=

quer dizer ‘não é igual”, é diferente.

ex:

preço >= 200.50 //O preço é maior ou igual a 200.50?

idade <= 18 //A idade é menor do que 18?

curso == ‘JavaScript’ //O curso é JavaScript?

n1 != n2 //O primeiro número é diferente do segundo?

Identidade

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | == | 5 | => | True |
| 5 | == | ‘5’ | => | True |
| 5 | === | ‘5’ | => | False |
| 5 | === | 5 | => | True |

// ==: o valor de igualdade nesse caso analisa se tem o mesmo valor que o outro, não o tipo (number, or string).

// ===: nesse caso o JS analisa se os dois “cincos” são idênticos (do mesmo valor e do mesmo tipo).

// !==: significa desigual restrito de tipos diferentes.

Lógicos

! exclamação // significa negação

&& dois ês comerciais //conjunção (“e” lógico)

|| dois pipes //disjunção (“ou” lógico)

Negação

! o é tratado como operador unário, isso é, ele só tem um operador. Ou seja, depois da exclamação ou eu boto “true” ou eu boto “false”.

Comjunção

&& é um operador binário, isto é, eu tenho dois valores lógicos, um de cada lado.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | && | True | => | True |
| True | False | => | False |
| False | True | => | False |
| False | False | => | False |

Só satisfaz se for as duas operações verdadeiras!

Disjunção

|| ( representa o “ou”) Também é um operador binário (que tem dois lógicos). Dependendo da posição deles e do valor lógico, eu tenho também o valor lógico de resultado.

Na disjunção basta que um deles seja verdadeiro para o resultado ser verdadeiro.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True | || | True | => | True |
| True | False | => | True |
| False | True | => | True |
| False | False | => | False |

A ordem de execução desses operadores é sempre:

1. Aritiméticos
2. Relacionais
3. Lógicos

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

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

A ordem de execução dos operadores lógicos é sempre!

1. !
2. &&
3. ||

ex: idade >= 15 && idade <= 17 // a idade está entre 15 e 17?

estado == ‘RJ’ || estado == ‘SP’ //O estado é RJ ou SP?

salário > 1500 && sexo != ‘M’ // O salario é acima de 1500 e não é homem?

Precedência

( ) \*\* /

> < >=

!

&&

||

Ternário

Teste ? true : false

3°

2°

1°

E chamado de ternário pois junta 3 operadores.

1. Teste Lógico que tem como resultado true or false.
2. O que acontece se o teste for verdadeiro.
3. O que acontece se o teste for falso.

ex: A media >= 7,0 ? ‘A’ = ‘R’

A == Aprovado

R == Reprovado

>var x = 8

>var res = % 2 == 0? 5 : 9

>x

8

>res

5

>var r = idade >= 18 ? ‘Maior’ : ‘Menor’

A minha variável “r” vai receber ou maior ou menor. Vai depender se a minha idade é >= 18.

Entendendo o **DOM!**

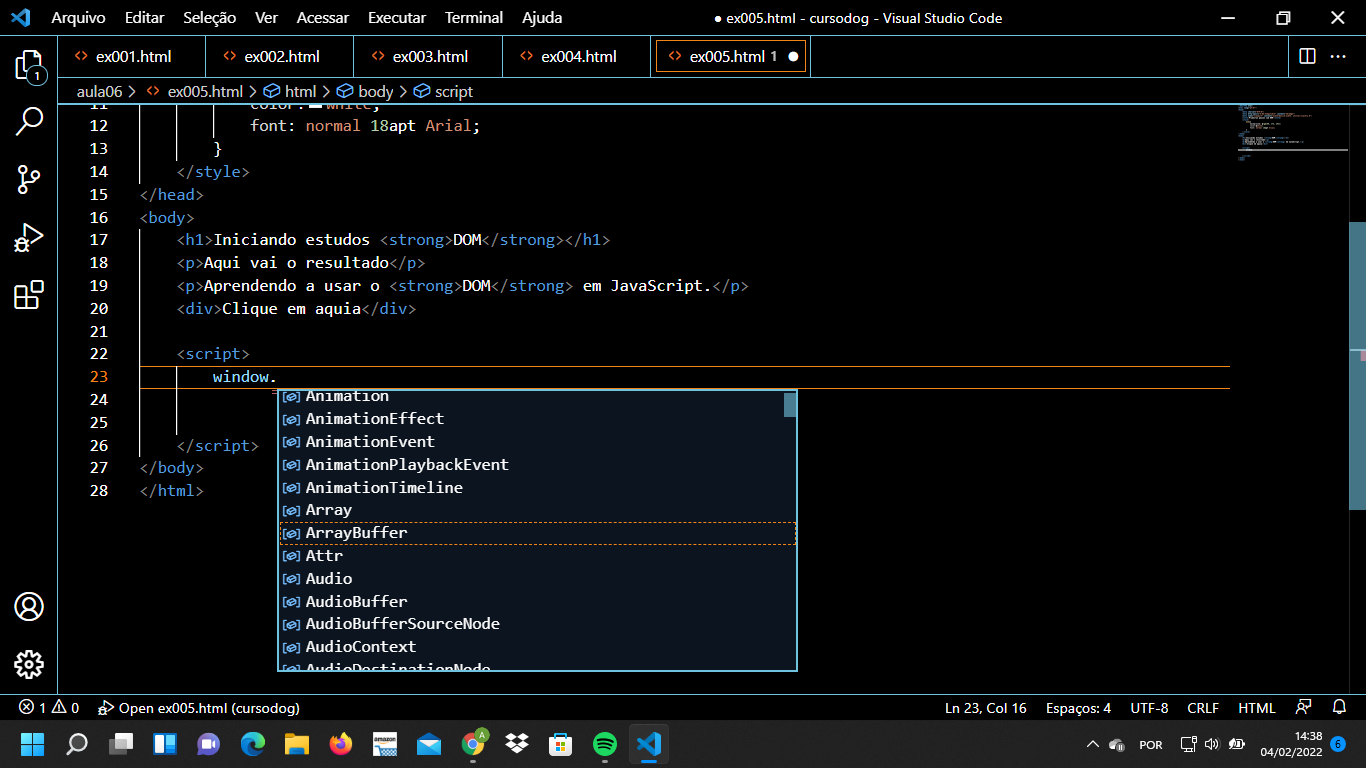
É um acrônimo para

Document Object Model

“Modelo de objetos para documentos”

* Um conjunto de objetos dentro do seu navegador que vai dar acesso aos componentes internos do seu website.

O DOM está presente quando estou rodando JavaScript dento dentro do me navegador.



Árvore DOM

Começa da raiz, e essa raiz, dentro do navegador, é chamada de Window

Tudo do JavaScript está

dentro de um objeto

chamado window.

Aquela “janela” do seu

computador é um objeto

Dom e ele se chama

Window

Árvore DOM

Dentro de window eu tenho vários outros objetos:

Location: que diz qual a localização do site, qual é a url, qual é a página atual, e qual foi a página anterior.

Document: que é o documento atual.

History: que vai guardando de onde você veio, para onde você vai. Isso facilita a navegação dentro do seu site.

Dentro de “documnet” eu tenho o objeto html, que é exatamente a parte html do seu site. E dentro de html, tem basicamente dois “filhos”, ou childs, que é o head e o body.

Que é a parte de cabeçalho, e a parte do corpo.

Window

Location document history

Que é: child de “document”, e parent de body.

Html

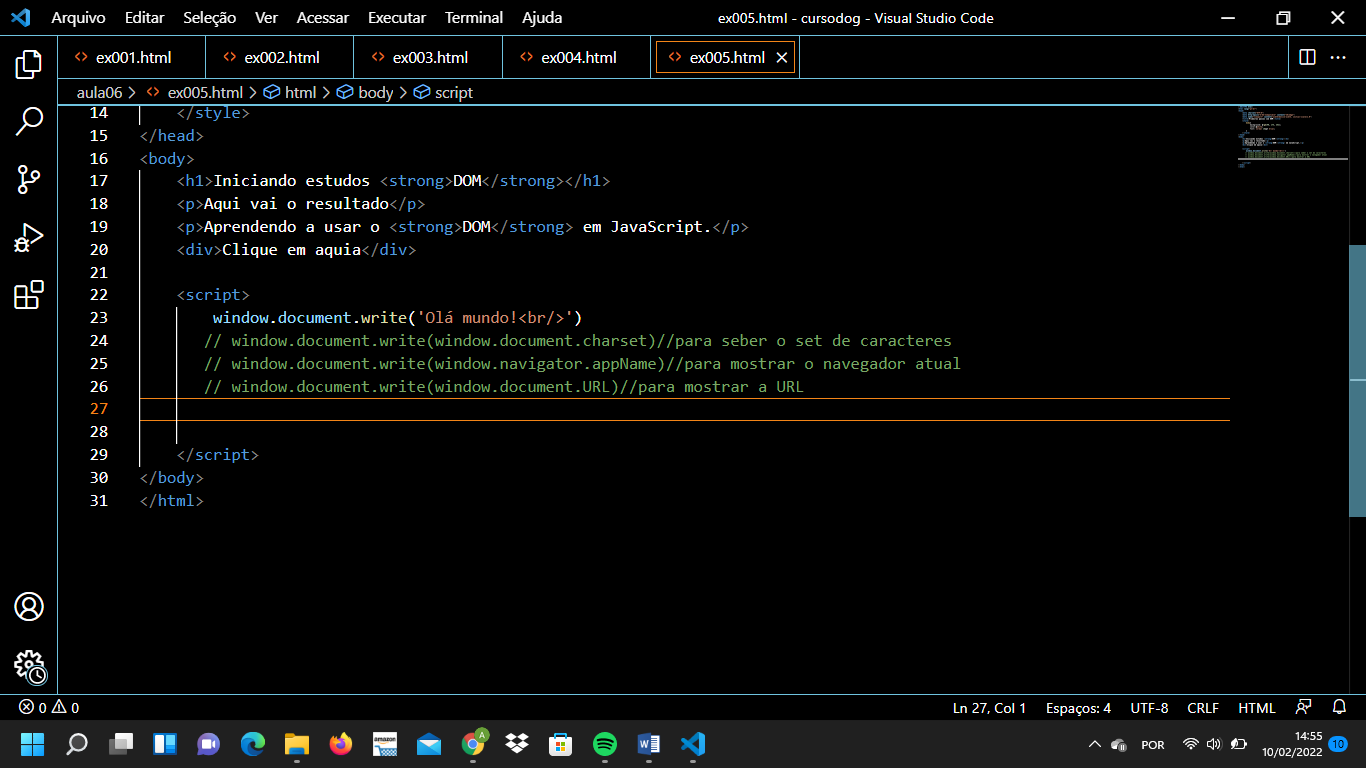
elementos

Head body

Meta title h1 p p div

Strong

Através de algumas propriedades/comandos é possível buscar algumas configurações da página:



Como selecionar elementos para navegar dento do navegador da árvore DOM?

Dentro todos os métodos de acesso, 5 deles são:

* Por marca
* Por ID
* Por nome
* Por classe
* Por seletor

Por marca ou por tag name:

getElementsBytagName( )

está dentro de documents

Quando é utilizado é possível selecionar mais de um objeto. Por que existem vários objetos do mesmo tipo, da mesma tag.

Ex: nosso código tem 2 parágrafos ( duas tags “p”). Mas eu tenho só uma tag body, só uma tag div, e só tenho uma tag h1!

<body>

    <h1>Iniciando estudos <strong>DOM</strong></h1>

    <p>Aqui vai o resultado</p>

    <p>Aprendendo a usar o <strong>DOM</strong> em JavaScript.</p>

    <div id="msg">Clique aqui!</div>

    <script>

       var corpo = window.document.body

       var p1 = window.document.getElementsByTagName('p')[0] //para pegar à primeira tag "p"

       var p2 = window.document.getElementsByTagName('p')[1] //para pegar à segunda teg "p" e assim sucessivamente.

        window.document.write('Está escrito assim: ' + p1.innerText + '<br/>')//innertext é o texto que está dentro do 1° parágrafo

       p1.style.color = 'blue'//para mudar a cor da tag "p,0"

       corpo.style.background = 'orange'//para mudar sempre que quiser a cor de fundo

       window.document.write(p2.innerHTML + '<br/>')//para pegar a formatação do texto também

       window.alert(p2.innerHTML)

    </script>

</body>

Quando se tem por exemplo:

Da para acessar todos os componentes/tags iguais

getElementsBytagName

Com um nome no plural pode-se usar colchetes “[...]”, com um número de 0 á quantos componentes iguais tiver, sendo o 0 o representativo do 1°, para representar qual tag de mesmo tipo você vai usa.

Por ID:

Dá para identificar a “div”, e outros componentes, com o id = “...”.

E usar o: getElementsById( )

<body>

    <h1>Iniciando estudos <strong>DOM</strong></h1>

    <p>Aqui vai o resultado</p>

    <p>Aprendendo a usar o <strong>DOM</strong> em JavaScript.</p>

    <div id="msg">Clique aqui!</div>

<script>

       var d = window.document.getElementById('msg')

       d.style.background = 'green'//para mudar a cor de fundo da "div"

       //d.innerText = 'Estou aguardando...'

       window.document.getElementById('msg').innerText = 'Óla!'//para mudar oque esta escrito dentro de 'msg'\*/

    </script>

</body>

Por nome:

Também pode-se utilizar o nome do objeto se tiver uma propriedade “name”. Então eu vou utilizar:

getElementsByName( )

<body>

    <h1>Iniciando estudos <strong>DOM</strong></h1>

    <p>Aqui vai o resultado</p>

    <p>Aprendendo a usar o <strong>DOM</strong> em JavaScript.</p>

    <div name = "msg">Clique aqui!</div>

    <script>

       var d = window.document.getElementsByName('msg')[0]

       d.innertext = 'Óla!'\*/

    </script>

</body>

Por classe:

Também posso utilizar, como conjunto, por classse:

<body>

    <h1>Iniciando estudos <strong>DOM</strong></h1>

    <p>Aqui vai o resultado</p>

    <p>Aprendendo a usar o <strong>DOM</strong> em JavaScript.</p>

    <div class = "msg">Clique aqui!</div>-->

<script>

 var d = window.document.getElementsByClassName('msg')[0]

       d.innerHTML = 'Óla!'\*/

    </script>

</body>

Por selector:

Recomendada na maioria dos manuais utilizando o:

querySelector( )

Sendo o plural da coisa

querySelectorAll( )

<body>

    <h1>Iniciando estudos <strong>DOM</strong></h1>

    <p>Aqui vai o resultado</p>

    <p>Aprendendo a usar o <strong>DOM</strong> em JavaScript.</p>

    <div id="msg">Clique aqui!</div>

<script>

 var d = window.document.querySelector('div#msg')

d.style.color = 'blue'

    </script>

</body>

Em cada tipo de selector CSS (id, class...) o querySelector se comporta de forma diferente:

*/\* todos os parágrafos\*/*

**p** {}

*/\* todos os parágrafos e todos os títulos \*/*

**p**, **h1** {}

*/\* qualquer elemento com 'warning' em seu atributo 'class' \*/*

**.warning** {}

*/\* um elemento com id="nav"\*/*

**#nav** {}

*/\* qualquer elemento span com 'fatal' e 'error' em sua classe \*/*

**span.fatal.error** {}

A funcao querySelectorAll() permite que os seletores CSS sejam usados no JavaScript, veja:

**var** elem = document.querySelectorAll('p');

**var** elem = document.querySelectorAll('p, h1');

**var** elem = document.querySelectorAll('.warning');

**var** elem = document.querySelectorAll('#nav');

**var** elem = document.querySelectorAll('span.fatal.error ');

Document.createElement()

Em um documento [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML), o método Document.createElement() cria o elemento HTML especificado ou um [HTMLUnknownElement (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/HTMLUnknownElement" \o "Currently only available in English (US)) se o nome do elemento dado não for conhecido.

Em um documento [XUL](https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL), o elemento XUL especificado é criado.

Em outros documentos, ele cria um elemento com um namespace URI null.

[Sintaxe](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#sintaxe)

var elemento = document.createElement(nomeDaTag);

* elemento é o objeto [Element](https://developer.mozilla.org/pt-BR/docs/Web/API/Element) criado.
* nomeDaTag é uma string que especifica o tipo do elemento a ser criado. O [nodeName (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeName" \o "Currently only available in English (US)) do elemento criado é inicializado com o valor da nomeDaTag. Não use nomes qualificados (como "html:a") com este método.

## [Exemplo](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#exemplo)

Este código cria uma nova <div> e a insere antes do elemento com ID "div1".

### [HTML](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#html)

<!DOCTYPE html>

<html>

<head>

<title>||Trabalhando com elementos||</title>

</head>

<body>

<div id="div1">O texto acima foi criado dinamicamente.</div>

</body>

</html>

### [JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#javascript)

document.body.onload = adcElemento;

function adcElemento () {

// cria um novo elemento div

// e dá à ele conteúdo

var divNova = document.createElement("div");

var conteudoNovo = document.createTextNode("Olá, cumprimentos!");

divNova.appendChild(conteudoNovo); //adiciona o nó de texto à nova div criada

// adiciona o novo elemento criado e seu conteúdo ao DOM

var divAtual = document.getElementById("div1");

document.body.insertBefore(divNova, divAtual);

}

Evetos DOM

São coisas que eu posso fazer com os elemento ex: div

Digamos que você tenha criado uma div qualquer:

Com elementos CSS e tudo mais

Interaja...

Os “eventos” é tudo aquilo que possa acontecer com essa div, como qualquer elemento.

O que pode acontecer com essa div?

O mais comum – Eventos de mouse!

* mouseenter = quando você mover o mouse até encostar na div, disparando um evento.
* mousemove = quando você continua movendo o mouse por dentro da div, disparando, várias vezes, um outro evento, o evento mousemove.
* mosedown = quando você clicar e segurar o mouse, disparando o evento mousedown.
* mouseup = quando você soltar o mouse vai disparar outro evento, o evento mouse up
* click = e também, o movimento de click inteiro (clicar e soltar rapidamente), disparando o evento click.
* mouseout = quando você mover o mouse para fora da div, disparando o evento mouseout.

Esses são 6 movimentos/eventos que podem ser disparados só utilizando o mouse!

Existem outros, esses não são os únicos!

Nem todos os elementos são compatíveis, com todos, os eventos!

Existem eventos específicos para cada situação.

Curiosidade!

Agora com o advento

dos celulares:

você tem o

“touch events”

Que são diferentes dos

“mouse events”.

Funções

É um conjunto de códigos/linhas que vão ser

executadas só quando o “evento” ocorrer.

ex: Você vai programar 10 linhas

essas 10 linhas serão chamadas de bloco.

Essa 10 linhas não serão executadas automaticamente, como

todos os nossos códigos até agora no momento.

Agora esse bloco ele só vai ocorrer por exemplo:

quando eu clicar dentro da div. Então essas 10 linhas só vão ser disparadas quando o evento ocorrer (quando clicar dentro da div).

Para executar as 10 linhas somente quando o evento acontecer:

1° = coloca-las dentro de um bloco “{...}”

Um bloco em javascript

É delimitado pelo sinal

De chaves ({...})

Esse bloco deve ser nomeado de FUNCTION!

Então você coloca a palavra function, que significa função, antes do bloco.

E para o método funcionar, eu preciso dar um nome pra essa função.

Geralmente o nome das funções são ações que a gente pode fazer/funções de eventos (nomes de ação).

Você vai colocar o nome da ação que vai acontecer. Abrir e fechar parênteses.

Opcionalmente você pode colocar também dentro desses parênteses alguns parâmetros, que podem ser um ou mais.

function ação (parâmetro) {

Bloco

}

Então primeiro eu tenho que disparar

que é adicionar um evento, e dar um nome a ação dele, ou seja, configurar o modelo.

E esse método só vai ser disparado quando você chamar a function:

O método de adicionar os eventos pelo html:

 <div id="area" onclick="clicar()" onmouseenter="entrar()" onmouseout="sair()"> <!--Se lê: quando eu (nome da ação) na div, vai disparar o nome da function.-->

        interaja...

    </div>

Além de disparar diretamente pelo html, dá pra disparar os eventos utilizando listeners, que são ouvidores, eles ficam prestando atenção dentro do próprio JS.

ao invés de poluir o html com os eventos nele

você adiciona os eventos no javascript por meio do:

addEventListener(“evento”, nome da ação)

isso é, ele vai ficar prestando atenção em um determinado evento.

O método de adicionar os eventos direto pelo JavaScript:

<script>

        var a = window.document.getElementById('area')//dessa forma o objeto "a" esta ligado diretamente com a dive "area".

        a.addEventListener("click", clicar)

        a.addEventListener("mouseenter", entrar)

        a.addEventListener("mouseout", sair)

        function clicar(){

            a.innerText = 'Clicou!'//Se lê: e esse objeto vai mudar o conteudo dele paea clicou.

            a.style.background = "red"

        }

        //Tudo que tiver aqui dentro só vai ser executado quando eu chamar a ação "clicar", na frente fe function.

        function entrar(){

            a.innerText = 'Entrou!'

        }

        function sair(){

            a.innerText = 'Saiu!'

            a.style.background = "green"

        }

    </script>

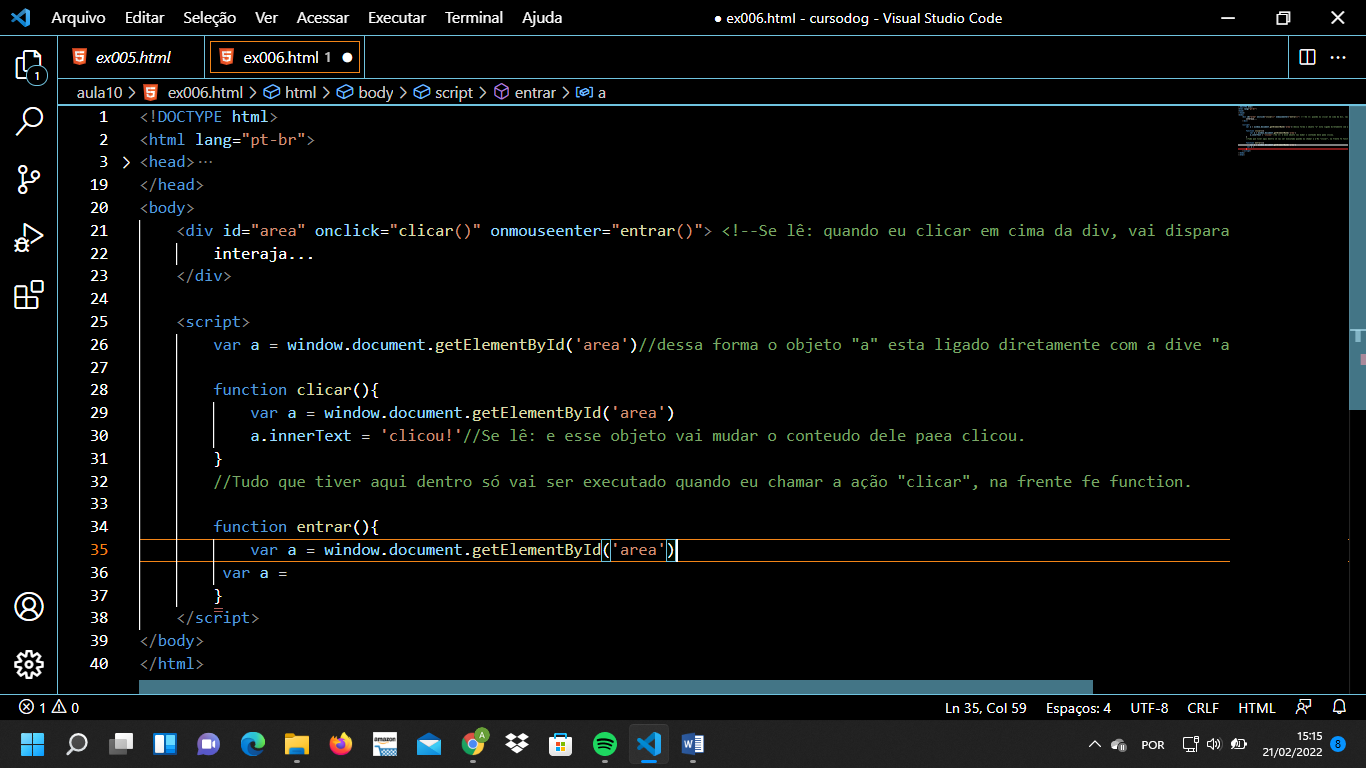
E dentro do bloco da function vai configurar oque vai acontecer quanto “tal” evento ocorrer.

A interatividade que se tem com a div, ou com qualquer outro elemento é muito interessante!

Escopo

Não é uma boa ideia fica interligando os objetos aos elementos a cada função que você tiver, pois isso causa problemas na memória.

É sempre melhor interligar do lado de fora das functions e etc...



Código completo:

<!DOCTYPE html>

<html lang="pt-br">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Eventos DOM</title>

    <style>

        #area{

            font: normal 20pt Arial;

            background-color: green;

            color: white;

            width:200px;

            height: 200px;

            line-height: 200px;

            text-align: center;

        }

    </style>

</head>

<body>

    <div id="area">

        interaja...

    </div>

    <script>

        var a = window.document.getElementById('area')//dessa forma o objeto "a" esta ligado diretamente com a dive "area".

        a.addEventListener("click", clicar)

        a.addEventListener("mouseenter", entrar)

        a.addEventListener("mouseout", sair)

        function clicar(){

            a.innerText = 'Clicou!'//Se lê: e esse objeto vai mudar o conteudo dele paea clicou.

            a.style.background = "red"

        }

        //Tudo que tiver aqui dentro só vai ser executado quando eu chamar a ação "clicar", na frente fe function.

        function entrar(){

            a.innerText = 'Entrou!'

        }

        function sair(){

            a.innerText = 'Saiu!'

            a.style.background = "green"

        }

    </script>

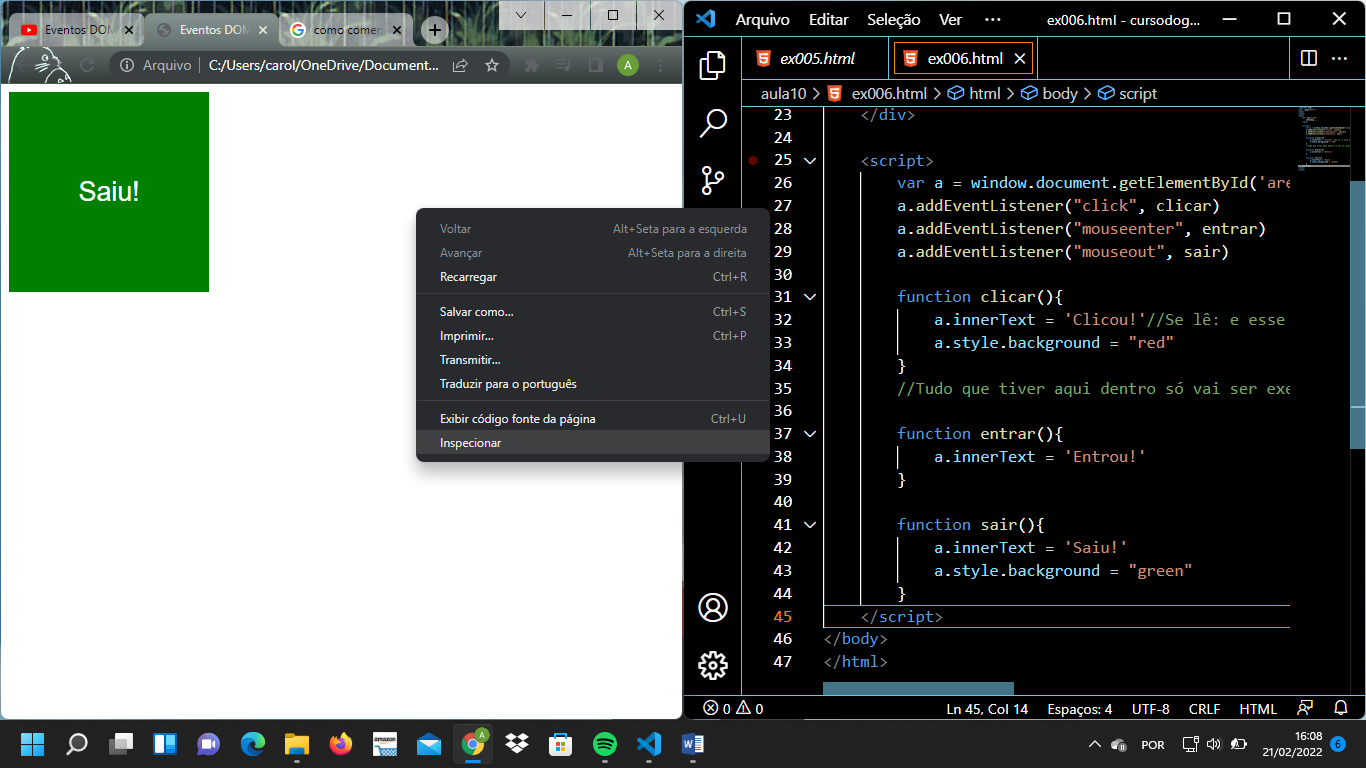
</body>

</html>

Guia de como descobrir um erro no javaScript:

Como descobrir um erro que pode ter no código?

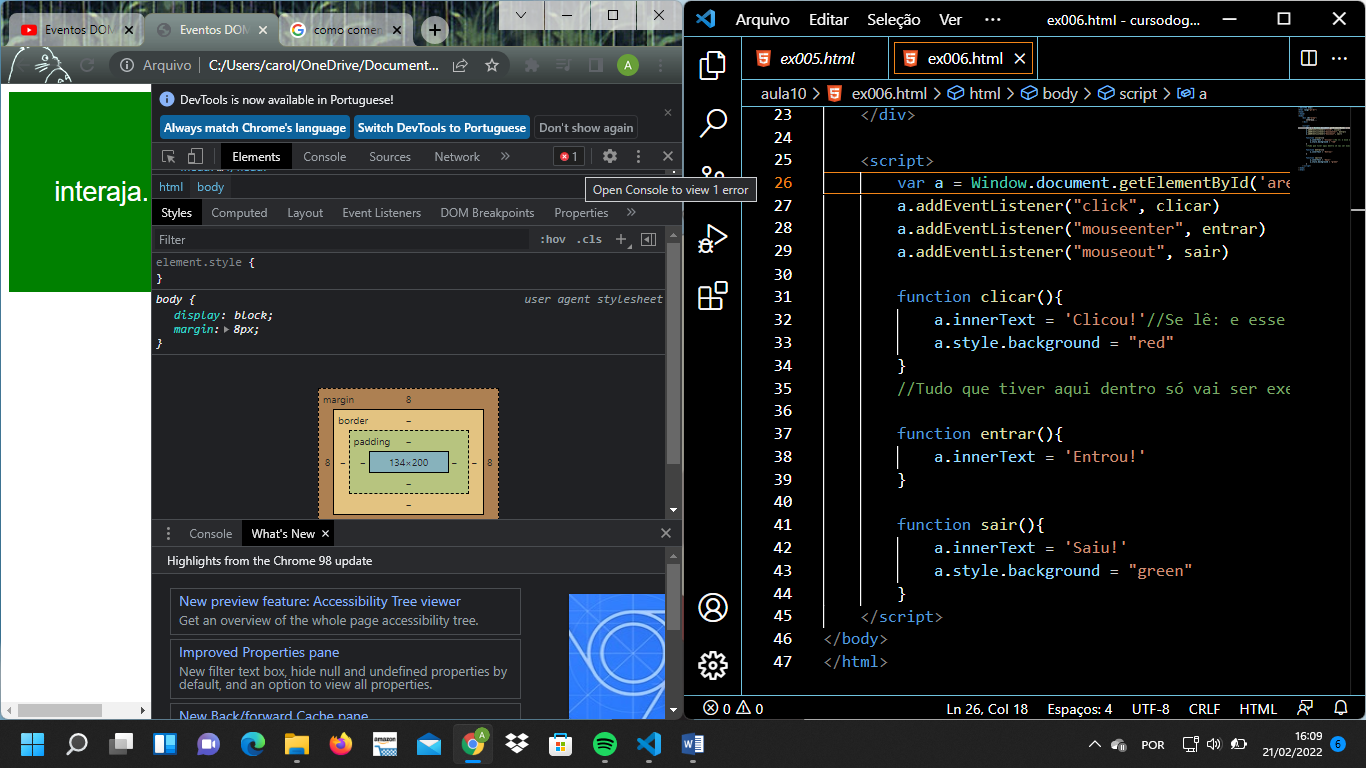
Basta clicar com o botão direito no browser, clicar em inspecionar/inspect, aí será aberto um inspetor o “Dev tools”



geralmente a Dev tools vai aparecer dentro do código, ou em cima, ou em baixo, ou do lado...

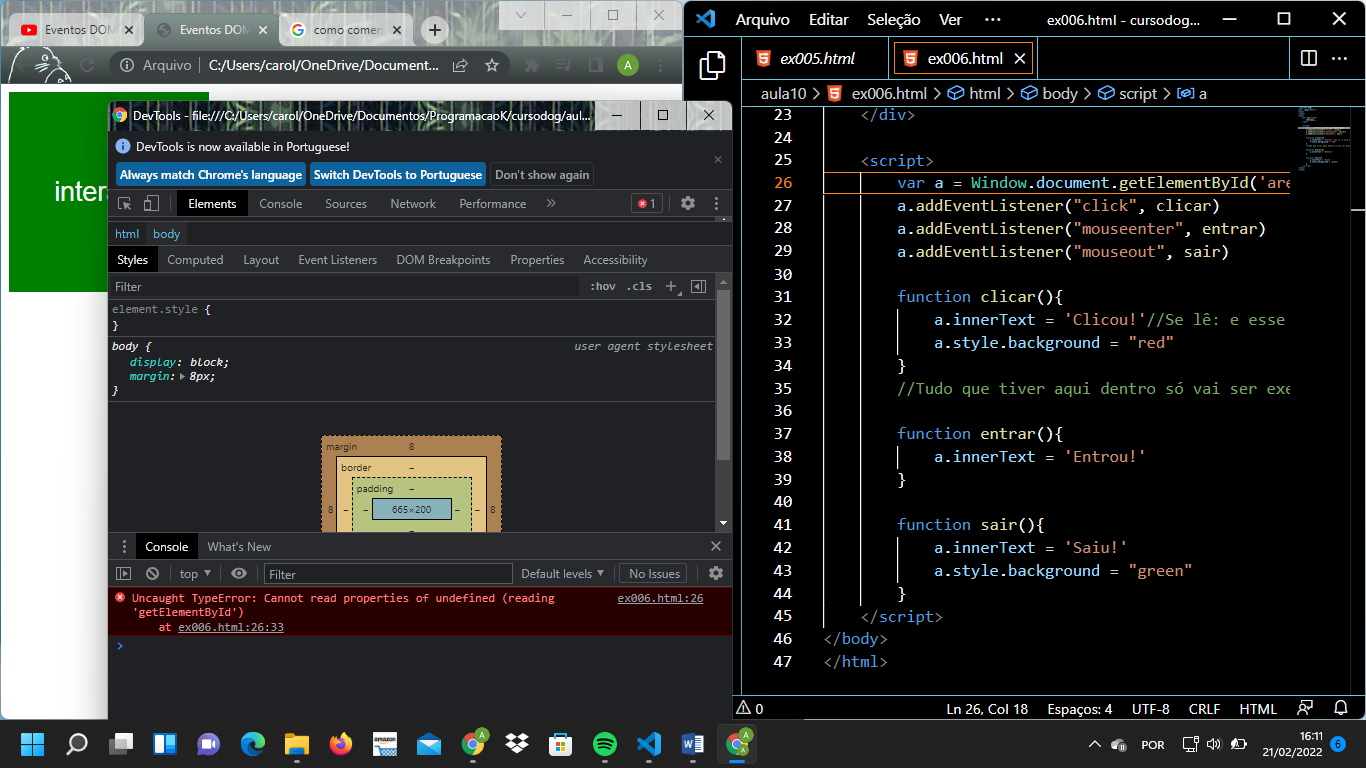
se clicar nas redescendias e ir em ondoc ele fica no centro.

Um bolinha vermelha aparecerce no canto superior direito da Dev tools indicando que há um erro.



Em baixo, no console, você consegue ver qual foi o erro...

geralmente o número da linha do erro apresentado, significa que o erro está daquela linha para frente.



Condições JavaScript

Condição: é uma estrutura controle muito importante na programação.

para entender condições primeiro preciamos aprender SEQUÊNCIA que aparece, e apareceu, em todos os códigos anteriores.

Sendo uma sequência,

serão executados na ordem

Var n = 3

em que forem dispostos.

mas vai ter um momento em que você vai querer executar o comando 2, e tem momento que não!

n += 2

window.alert(n)

Quando isso acontece e que vemos as condições sendo usadas. Elas nada mais são do que uma bifurcação no meio de uma sequência que recebe blocos cheios de comandos a serem ativados se caso for confirmado um valor verdadeiro (true), ou falso (false) para não serem ativados.

If (condição) {

} else {

}

true

bloco

Então se o “if” for verdadeiro certos comandos serão executados, e se for falso outros comandos serão executados.

false

bloco

else

if

bloco

true

false

Tipos de condição:

1° tipo: condição simples com um tipo de bloco:

If (condição) {

true

}

Ex:

var vel = 60.5

console.log(`A velocidade do seu carro é de ${vel}km/h.`)

if (vel > 60){

    console.log('Sua velocidade ultrapassou o limite, MULTADO!')

} //condição simples!

console.log('Dirija sempre com o sinto de segurança!')

Podemos ver que só há o bloco de verdadeiro (if), o bloco de falso não existe.

2° tipo: Condição composta nessa aparece o if e o else e os seus blocos relacionados.

If (condição) {

No primeiro tipo se a condição for falsa nada especial irá acontecer o fluxo continuará normalmente.

true

} else {

Já no segundo tipo, caso a condição seja verdadeiro tem uma coisa para fazer, e caso seja falsa tem outra coisa pra fazer.

false

}

Ex:

var pais = "Japão"

console.log(`Vivendo em ${pais}`)

if(pais != 'Brasil'){

    console.log('você é estrangeiro!')

}else{

    console.log('você é Brasileiro!')

}

Em javascript não funciona o “document.write” em js se utiliza: console.log( )

O espaçamento dos elementos de um bloco em relação a outros elementos de identação que significa recuo.

Usando += com uma string, ao ser executado, não será somado, será concatendado já que é uma string.

3°Tipo: Condição aninhada é como pegar uma condição composta e colocar outras condições dentro dela.

bloco

bloco

bloco

Se a condição 1 for falsa, e a condição 2 for verdadeira ele faz o bloco 2.

Se a condição 1 for falsa e a 2 também, e a 3 for verdadeira ele faz o bloco 3.

E nada impede de no lugar do bloco 3 adicionar uma outra condição (eu estaria subindo de nível no meu aninhamento).

bloco3

bloco2

bloco1

If (condição1) {

} else {

If (condição2) {

}else {

}

}

}

Ex:

var agora = new Date()

var hora = agora.getHours()

console.log(`Agora são ${hora}horas.`)

if(hora > 1 && hora < 12){

    console.log('Bom Dia!')

}else if(hora < 18){

    console.log('Boa Tarde!')

}else{

    console.log('Boa Noite!')

}

3° tipo: Condição múltipla se tem uma expressão. E dessa expressão eu texto vários valores e depois volto para o fluxo normal do meu programa.

Valor4

Valor3

Valor2

Valor1

Esse tipo de condição não serve para todo tipo de situação, mas se representa da seguinte maneira:

O “default” seria o mesmo que o “else” no comando if, ele é executado quando nenhum dos casos deu como verdadeiro.

CUIDADO!

Se o “break” não for adicionado, o comando executa todos os casos até encontrar um break, ou até acabar os comandos.

O break serve, para que, se caso o valor der verdadeiro ele jogar esse resultado no fluxo para que o mesmo possa prosseguir

bloco2

bloco2

bloco2

bloco2

switch(expresão) {

case valor 1:

break

case valor 2:

break

case valor 3:

break

default:

break

O swith só funciona com números inteiros ou strings.

Ex:

var agora = new Date()

var diaSem = agora.getDay()

/\*

    0 = Domingo

    1 = Segunda

    2 = Terça

    3 = Quarta

    4 = Quinta

    5 = Sexta

    6 = Sábado

\*/

/\*console.log(diaSem)\*/

switch(diaSem){

    case 0:

        console.log('Domingo')

        break

    case 1:

        console.log('Segunda')

        break

    case 2:

        console.log('Terça')

        break

    case 3:

        console.log('Quarta')

        break

    case 4:

        console.log('Quinta')

        break

    case 5:

        console.log('Sexta')

        break

    case 6:

        console.log('Sábado')

        break

    default:

        console.log('[Erro] Dia invalido.')

        break

}

# Document.createElement()

Em um documento [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML), o método **Document.createElement()** cria o elemento HTML especificado ou um [HTMLUnknownElement (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/HTMLUnknownElement" \o "Currently only available in English (US)) se o nome do elemento dado não for conhecido.

Em um documento [XUL](https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL), o elemento XUL especificado é criado.

Em outros documentos, ele cria um elemento com um namespace URI null.

## [Sintaxe](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#sintaxe)

var elemento = document.createElement(nomeDaTag);

* elemento é o objeto [Element](https://developer.mozilla.org/pt-BR/docs/Web/API/Element) criado.
* nomeDaTag é uma string que especifica o tipo do elemento a ser criado. O [nodeName (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeName" \o "Currently only available in English (US)) do elemento criado é inicializado com o valor da nomeDaTag. Não use nomes qualificados (como "html:a") com este método.

## [Exemplo](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#exemplo)

Este código cria uma nova <div> e a insere antes do elemento com ID "div1".

### [HTML](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#html)

<!DOCTYPE html>

<html>

<head>

<title>||Trabalhando com elementos||</title>

</head>

<body>

<div id="div1">O texto acima foi criado dinamicamente.</div>

</body>

</html>

### [JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/API/Document/createElement#javascript)

document.body.onload = adcElemento;

function adcElemento () {

// cria um novo elemento div

// e dá à ele conteúdo

var divNova = document.createElement("div");

var conteudoNovo = document.createTextNode("Olá, cumprimentos!");

divNova.appendChild(conteudoNovo); //adiciona o nó de texto à nova div criada

// adiciona o novo elemento criado e seu conteúdo ao DOM

var divAtual = document.getElementById("div1");

document.body.insertBefore(divNova, divAtual);

}

Modulo E

Repetições em JavaScript (laços)

E uma estrutura de controle, e começa exatamente como uma condição: ele vai testar uma expressão, e assim como acontecia nas expressões de condição, onde se tem duas possibilidades: ou teste é verdadeiro, ou ele é falso, só que o comportamento aqui é um pouquinho diferente.

Se essa condição, se esse teste lógico, for verdade ele vai executar um bloco, só que ele não vai seguir desse ponto para o ponto final, que está aqui em baixo, ele vai voltar e vai fazer um looping (um laço/repetição).

Voltando para o losango, ele vai testar de novo essa condição, se essa condição for verdadeira de novo, ele vai executar a tarefa, e fazer outro laço, como acabou de acontecer e vai testar mais uma vez.