Tipos em JS

Existem 7 tipos dentro do JS que sao: (SNOB N US)

String – para textos

Number – para numeros

Object – tudo no JS eh um objeto, entao basicamente se nao for um dos outros aqui ele sera um objeto.

Boolean – true or false

Null – maneira de dizer nada

Undefined – outra maneira de dizer nada

Symbol – Eh meio que um identificador unico, eh novo no JS

Para saber qual o tipo esta sendo usado em uma variavel por exemplo utilizamos o comando typeof no console, node, ou dentro de algo para ele trazer o tipo e entao utilizarmos para algo.

String:

Temos 3 tipos de maneiras de definir uma string, com aspas simples, aspas duplas ou apostrofe (acento da crase).

Ex:

const name = ‘nomes’

const name = “nomes”

const name = `nomes`

As diferencas sao as seguintes, com uma ou duas aspas sao basicamente iguais, a diferenca eh caso vc precise escrever algo que dentro do texto tera uma aspa simples ou dupla, ai nesse caso vc tera que usar uma diferente no comeco para nao quebrar o JS, ex:

Const sentence = “She’s so cool” – nessa sentenca caso tivessemos utilizado aspas simples no comeco ele iria entender que estavamos terminando a variavel no She, o que estaria errado, e assim funciona para o contrario, caso quisesse colocar o cool entre aspas precisaria ser aspas simples no comeco.

Agora se vc quiser colocar aspas simples e duplas dentro do texto tera que usar a \ antes do item que vc quer que o JS desconsidere ex:

Const sentence = “She\’s so \“cool\”” dessa forma tudo que esta na frente da barra invertida sera considerada como parte do texto e nao como abertura e/ou fechamento da variavel

E o apostrofe a diferenca eh q vc pode usar sem se preocupar com os 2 acima mencionados ex:

Const sentence = `She’s so “cool”` e ira funcionar perfeitamente

Outra vantagem eh caso vc queira mencionar uma variavel dentro de uma string, com o apostrofe vc escreve tudo que quer e quando quiser mencionar a variavel vc usa ${} ex:

Const hello = `Hello my name is ${name}. Nice to meet you`

Chamado de variable interpolation

Number:

Pode ser inteiro (integer), numero com decimais ou Not a Number (NaN)

Podem ser somados, subtraidos, divididos, e multiplicados.

E temos 2 especiais que sao modules (ou resto da divisao {%}) e potencia ou x elevado a..(\*\*)

Caso vc tente fazer numero + string ele ira concatenar os dois ex:

1 + “1” = “11”

Caso vc tente subtrair ou multiplicar com strings ele ira ajustar pra vc, pois os sinais de – e \* nao sao utilizados pra nada alem disso, agora + e / sao utilizados em outras coisas

Alguns exemplos de codigos de matematica utilizados sao:

Math com M maiusculo:

Math.round(20.5) = 21 ( arredonda pra cima ou pra baixo)

Math.floor(20.6) = 20 (arredonda pra baixo)

Math.ceil(20.2) = 21 (arredonda pra cima)

Math.random() = vai te dar um numero aleatorio entre 0 e 1

Obejects:

Basicamente eh uma maneira de agrupar coisas que sao relacionadas e ficam dentro de { } ex:

cont person = {

name: “Rafa”

last: “Moraes”

age: 34

}

Se chamarmos a variavel person ele ira trazer os 3 itens de dentro, para chamar de forma especifica o que criamos dentro utilizamos da seguinte forma:

person.name

Rafa

person.last

Moraes

person.age

34

Null and Undefined:

O undefined eh quando vc cria uma variavel por exemplo e nao define um atrituto a ela, ex:

let dog

dog

undefined

Porque vc criou a variavel dog mas nao definiu nada pra ela, eh diferente de nao definido como por exemplo se vc nao criou nem a variavel vc vai ter um erro de not defined, pq vc nao definiu nada com esse nome. Ex:

nome

error bla bla bla not defined

O Null eh quando vc especifica que algo tem um valor nulo, ou seja nao tem valor nennhum ex:

let dog = Null

dog

null

Bollean and Equality:

Tipos bolleans sao true or false podem ser definidor ou calculados.

Diferencas de iguais:

= recebe ou define um valor ex: let age = 10

== compara se os valores sao iguais ex: “10” == 10 (true pq os 2 numeros sao 10, independente do tipo)

=== compara se os valores e os tipos sao iguais ex: “10” === 10 (false, pq o primeiro 10 eh uma string e nao number como o segundo 10)

Function:

Statement eh uma declaracao de algo, tipo uma variavel

Argument sao os valores definidos dentro do statement (no css sao os valores)

Uma funcao eh uma maneira de agrupar varios statements para que executem quando uma determinada acao ocorrer.

Tipos de msg no console:

Console.log() – Escreve no console algo que vc queira

Console.error() – Da uma mensagem de erro (vermelho com um x) no console caso algo aconteca ou vc queira

Console.warn() – Da uma msg de alerta (amarelo)

Console.table – cria uma tabela caso existam itens que estao em um objeto e tenham o mesmo nome. Ex:

Const people = [

{Name: “Rafa” , country: “Brasil”, cool: true},

{Name: “Rods”, country: “Irlanda”, cool: true},

]

Console.count() – Vai contar quantas vezes aquele comando foi executado.

Console.groupCollapsed ou Console.group() e console.groupEnd() – vc coloca no comeco e no final os comandos e ai ele vai agrupar todos os console. Algo para rodar de uma vez soh.

Eh util caso vc esteja fazendo um loop grande de algo, e ai vc pode pedir para trazer as informacoes de cada parte mas juntas em um grupo. Ex:

Console.groupCollapsed(`${people.name}`);

Console.log(people.country);

Console.log(people.cool);

Console.log(“Done”);

Console.groupEnd(`${people.name}`);

E dessa forma ele vai agrupar por nome de pessoa qual o pais e o cool dela

Callstack ou Stack trace

Quando um erro acontece em uma pagina na aba do console ele vai te informar qual funcao deu problema e qual linha e quem a chamou, e vai informando quem chamou quem ate vc localizar o causador ou a origem.

Grabbind Elements ( Soh funciona no console da pagina)

Quando vc vai inspecionar uma pagina e vc seleciona o item que quer ele vai mostrar qual eh o elemento que vc selecionou, se vc for na aba de console e digitar:

$0 – ele ira trazer para o console o elemento que vc tinha selecionado para vc inspecionar ou editar.

$0 significa o ultimo elemento que vc clicou para inspecionar, ou seja, se vc selecionar outro, o proximo que vc clicar sera o 0 e o primeiro sera o 1 (que sera o penultimo) e assim por diante.

Temos tb a opcao de $ e $$ que funciona como um querySelector, contanto que nao tenha Jquery na pagina.

Ai se colocarmos $(‘p’) ele vai selecionar o primeiro paragrafo que ele encontrar na pagina, se usar o $$(‘p’) ele vai selecionar todos os paragrafos da pagina.

Breakpoints

Caso vc queira fazer com que o JS pause a execucao em um determinado ponto podemos utilizar o comando debbuger em qualquer parte do codigo, mas para que ele funcione o Devtools no browser tem que estar aberto.

A partir dai vc pode olhar na aba de source sobre o que esta sendo feito ate aquele momento e colocar para rodar de 1 em 1 linha ou rodar tudo de uma vez com os botoes no topo da direita.

Parece com o debug do Visual basci do excel que ia executando de um em um

Scope

Basicamente eh saber o que esta e onde esta disponivel, ou seja, quando criamos uma variavel, funcao ou outra coisa dentro de uma funcao ou bloco ( { } ) ela esta disponivel somente dentro daquela funcao e tals, isso eh o scope.

Global scope

Quando vc cria uma variavel por exemplo sem ser dentro de uma funcao, ou modulo, ou seja no corpo do script tag, ele eh uma variavel global e pode ser acessada dentro de qualquer lugar do JS, ate mesmo se vc criar um arquivo separado e quiser acessar dentro do html ele vai funcionar

Function scope

Qualquer variavel criada dentro de uma funcao nao pode ser acessada do lado de fora, ou seja, no global.

Mas dentro de uma funcao vc pode acessar uma variavel que esteja criada no global

Block scope

Toda vez que tivermos {} isso significa um bloco, e as variaveis criadas dentro dele sao restritas a ela, entao caso queira criar uma variavel que sera utilizada dentro de um if, eh melhor criar do lado de fora, ou caso nao va precisar acessar a variavel em nenhum outro lugar, ai pode criar dentro dos { }

A variavel var funciona quase como se fosse criada no global, mesmo sendo criada dentro de um bloco, mas isso tem perigo pq vc pode utilizar a variavel em outro local sem lembrar e pode causar um bug geral.

Usar as variaveis da seguinte forma:

Const como padrao

Let quando precisar alterar o valor da variavel

Var quase nunca

Hoisting

Para funcoes e variaveis eh como se ele jogasse tudo pra cima antes de tudo, entao vc pode por exemplo chamar uma funcao antes mesmo de cria-la e ira funcionar por causa do hoisting.

No caso da variavel eh como se o JS jogasse pra cima somente a criacao do nome da variavel mas nao o valor que vc atribuiu, ficaria como se fosse assim:

Var nome;

Console.log(‘nome’)

Nome = “Rafa”

Mesmo vc tendo criado da seguinte forma:

Console.log(‘nome’)

Var nome = “Rafa”

Closures

Closures eh a habilidade de um child function acessar uma variavel que esta em um parent mesmo depois que a fauncao tenha sido chamada.

No exemplo abaixo ao chamar a funcao o score eh 0, mas por causa do closure ele acessa a funcao child de dentro e altera o valor do score, e com isso se vc for chamando varias vezes a funcao esse numero vai alterando pq ele vai salvando com um valor novo.

Function createGame(gameName){

Const score = 0;

Return function win() {

Score++;

Return `Your game ${gameName} score is ${score}

}

}

Const hockeyGame = createGame(‘Hockey’)

Resposta: Your game Hockey score is 1

The DOM (document Object Model)

Eh tudo relacionado a pagina, o que fica dentro da tag html, e vc pode acessar tudo atraves do JS.

Selecionando elementos

Podemos utilizar o querySelector ou querySelectorALL

A diferenca eh que com o querySelector ele vai selecionar somente o primeiro item que ele encontrar daquele tipo ex:

querySelector(‘p’) – ele vai selecionar o primeiro paragrafo que encontrar na pagina

querySelectorAll(‘p’) – ele vai selecionar todos os paragrafos que encontrar na pagina

Agora se vc colocar um ID ou CLASS ele vai selecionar o que vc especificou, um ou varios itens com o nome definido e com o # ou . no nome

Ele funciona bem parecido com a selecao dentro do CSS onde se vc quiser selecionar uma imagem dentro de uma div ficaria assim:

querySelectorAll(‘.item img’) – e ai ele vai selecionar todas as classes item que tiverem uma imagem de child

Caso vc queira buscar um elemento dentro de outro elemento vc pode usar o modo acima ou criar uma variavel do primeiro elemento e depois utilizar essa variavel e usar o querySelector pra buscar dentro dela. Ex.

Const div = document.querySelector(‘.item’)

Const itemComImagem = div.querySelector(‘img’)

Getter or Setter

Vc pode utilizar commandos para pegar ou consultar algo ou para setar ou determinar o valor de algo.

Ex:

Console.dir(heading.textContent); - isso eh um getter pois vc soh esta consultando o que esta escrito dentro da tag heading

Ou se vc fizer assim:

heading.textContent = ‘wes is cool’; - Isso eh um setter, pois vc esta determinando um novo valor para o heading agora trazer a informacao de wes is cool.

Obs: diferenca de .textContent e .innerText

O innerText vai selecionar ou alterar o valor da propriedade enquanto o textContent ele vai selecionar e ou alterar o valor mas ele tb mostra tudo que estiver relacionado a essa propriedade, como por exemplo se tiver uma tag style dentro do html ou algo escondido ele ira mostrar (no console por exemplo)

Uma maneira mais facil e mais atual de adicionar texto ou conteudo a algo ja existente (antes ou dps do texto sem alterar ele todo) eh utilizar a propriedade element.inserAdjacentText(‘position’, ‘element/text’)

Dentro do position temos 4 opcoes, beforeBegin, beforeEnd,afterBegin e afterEnd.

\*\*Tudo eh um node, ele soh vira um elemento se estiver em volta de uma tag\*\*

Classes

Se utilizarmos a propriedade classList no console ao selecionar um elemento ele vai mostrar tudo que esta relacionado a essa classe.

E a partir dai podemos utilizar outros metodos que podemos adicionar, remover, etc uma classe no elemento.

Por exemplo a variavel pic que selecionou uma classe que eh de uma imagem podemos fazer o seguinte:

pic.classList.add (‘picture’) – ele vai criar uma outra classe nesse elemento com o nome de picture

pic.classList.remove(‘picture) – ele vai remover a classe do elemento que se chama picture

pic.classList.toggle(‘round’)- Nesse exemplo ele criou um css com alguns criterios para a imagem ficar com borda e girar e tal, que ao clicar na imagem iria executar uma funcao que ia utilizar esse toggle que eh basicamente alterar de verdadeiro ou falso, ou seja, ter ou nao ter toda vez q clicasse na imagem.

pic.classList.contain(‘round) – ele vai informar se eh verdadeiro ou falso, ou seja, se existe ou nao existe, eh utilizado para saber se algo esta ativo ou nao , ou se algo esta aberto ou nao, etc....

Data atributes

Um atributo eh qualquer coisa atrelada a uma tag no html, ou seja, classe, src, alt, width, etc...

Ao adicionar ou alterar um atributo podemos selecionar o elemento e utilizar o atributo que queremos ex:

Crio uma variavel para acessar as informacoes do elemento:

Const pic = document.querySelector(‘.custom’)

Depois utilizamos a variavel para adicionar ou alterar um atributo ex:

Pic.width = 200

Pic.alt = “cute pup”

Etc....

Caso vc queira criar um atributo personalizado utilizamos o data-nomeDesejado, pois assim vc consegue acessar pelo JS de forma a que no futuro nao de problema de criarem um atributo q seja o mesmo nome q vc tinha criado. Ex:

<img class = “custom” src =”” data-imagemRafa = “foto” data-nome = “Rafa”>

Ai para acessar esse atributo no JS utilizar assim:

Getter:

Console.log(“pic.dataset.imagemRafa”) ou console.log(“pic.dataset.nome”)

Setter:

Pic.dataset.nome = “Moraes” – Ele vai alterar o valor do atributo data-nome para Moraes.

Usar esses atributos personalizados as vezes sao muito uteis segundo wes bos.

Create HTML

Para criar elementos no html via JS utilizamos o comando createElement(tag do elemento entre aspas) em uma variavel ex:

Const myParagraph = document.createElement(‘p’);

myParagraph.textContent = ‘Hello World’

ou

myParagraph.innerHtml = ‘Hello World’

Apos criarmos a variavel com o elemento ele ficara salvo na memoria e entao temos q atribui-lo a alguma parte do html atraves do comando appendChild. Para facilitar mais pra frente podemos tb dar um atributo de classe a ele usando o classList.add e entao ficaria assim:

myParagraph.classList.add(‘message’)

document.body.appendChild(myParagraph) – Dessa forma ele ira selecionar o body do html e criar um child com o elemento p que vc criou.

De padrao adicionar o comando de onde vc vai imputar as coisas que foram criadas ao final do codigo, nao no comeco, por exemplo, nao colocar primeiro o paragrafo no body e dps colocar o texto, e sim o contrario, pq assim a pagina nao tem que ficar “recarregando” pq algo novo aparaceu na tela.

Outra maneira de adicionar algo na pagina eh com o codigo insertAdjacentElement

Dessa forma temos uma liberdade um pouco maior pois com esse codigo podemos escolher se vamos colocar antes de algo, dps de algo, dps de acontecer ou ao comeco de algo acontecer.

Ou seja, se vc tem uma DIV com coisas dentro e vc quer colocar um texto vc pode definir se o texto vira antes da div, ao comeco da div, ao final da div ou dps da div, com o appendChild vc nao tem toda essa facilidade e tem q selecionar exatamente onde quer q ele va, e se nao der pra selecionar vc vai ter q criar outras maneiras para tal.

Com o insertAdjacentElement caso vc queira criar uma lista e colocar o item 4 antes do item 2, com o insertAdjacentElement vc tera q selecionar o segundo item da lista e definir q o item 4 venha antes do 2 com o beforeBegin por exemplo.

Creating Html from strings – obs existe um risco ao fazer dessa forma q sera explicada na parte de seguranca.

Podemos criar elementos e textos via JS com o codigo innerHtml e escrever como se estivessemos dentro do html colocando divs, classes, h1 etc... isso eh criar via String e nao DOM

Const myHtml = `

<div class = “wrapper”>

<h1> “Hello World” </h1>

</div>

`

Document.body.innerHtml = myHtml – comando para colocar na pagina

O problema de fazer dessa forma eh que caso vc queira selecionar algo que esta dentro disso que vc criou vc tem q criar uma variavel que consiga acessar via tag,class, id e tal....entao eh importante saber q se for fazer dessa forma eh necessario criar formas de acessar o item tb.

Para transformar de String pra DOM utilizar da seguinte forma, criamos uma variavel e iremos acessar o string que foi criado e fazer ele fazer parte do DOM

Const fragment = document.createRange().createContextualFragment(myHtml)

O createRange eh um grupo de coisas que te permite trabalhar com o html

Dentro do metodo acima temos outro metodo que eh o createContextualFragment que eh basicamente falar que eh um html o que vc chamou.

E dai dessa forma vc pode fazer selecionar as coisas criadas nesse html como se estivessem direto na pagina ex.

Const imagem = fragment.querySelector(“img”)

Traversing & remove nodes

A diferenca de um elemento e um node eh basicamente a seguinte, tudo pode ser um node, mas pra ser considerado elemento ele tem q estar dentro de uma tag, <> () [] {} etc....

Codigos relacionados ao elemento, ou seja, os textos nao sao considerados pq eles sao nodes.

children - ira mostrar todos os children elements do item selecionado

firstElementChild – seleciona o primeiro elemento child do item selecionado

lastElementChild - seleciona o ultimoi elemento child do item selecionado

previousElementSibling - seleciona o item anterior do elemento child do item selecionado

nextElementSibling - seleciona o proximo elemento child do item selecionado

parentElement - seleciona o parent element child do item selecionado

Codigos relacionados aos nodes, ou seja, os textos tb sao considerados pq eles sao nodes.

childNodes

firstChild

lastChild

previousSibling

nextSibling

parentNode

Fazem as mesmas coisas que os de cima, mas considerando os nodes dentro dos elementos