**Introdução ao Javascript**

Com sintaxe semelhante ao Java, Javascript domina a web permitindo que sites carreguem conteúdos mais dinâmicos nas suas páginas. Através do NodeJs, Javascript se tornou uma linguagem tanto para Front-end quanto para Back-end.

JS é de tipagem dinâmica e fraca, ou seja, dinâmica por suas variáveis não serem definidas de forma explicita no código e não conhecidas/checadas em tempo de execução/compilação e fraca por não ter o tipo da variável bem definido e a própria linguagem poder alterar o seu tipo de dado sem a intervenção direta do programador.

Tipagem Fraca exemplo em JS: const numberOne = "5"

const numberTwo = 5

console.log(numberOne + numberTwo)

// 55

Tipagem Estática – o tipo de variável deve ser declarado antes de se atribuir valores a ela e seu tipo não pode ser alterado.

Exemplo em C: int numero = 5

Tipagem Dinâmica – declara-se a variável e não o seu tipo. Não há a necessidade de declarar que é uma variável do tipo String, por exemplo. Pode-se atribuir outro tipo de dado a variável também sem necessidade de alterar antes o seu tipo.

Exemplo em JS: var numero = “cinco”

var numero = 5

Tipagem Forte – os tipos das variável são bem definidos e só podem ser alterados com intervenção do programador.

Exemplo em Python: numero1 = “5”

numero2 = 5

print( int(numero1) + numero2)

#10

Por ser de tipagem dinâmica e fraca, JS possui uma linguagem simples e flexível o que compromete um pouco sua clareza e manutenção de código.

Em JS funções são valores, portanto ela se torna uma linguagem própria para a programação funcional. Permite também Orientação a Objetos(OO).

Atualmente por rodar em navegadores, servidores(Node) e mobile(React Native) o JS faz parte do único stack(com adesão) que possui a capacidade de fazer front-end, back-end e mobile com apenas uma linguagem.

**Input**

let input = prompt("Digite um número");

alert("O número é: "+input);

**Variáveis**

Linguagem estática – a variável é de um tipo exclusivo de informação, não podendo ser alterada para outro tipo, por exemplo uma String não pode se tornar do tipo Inteiro, etc.

Linguagem dinâmica – a memória só aloca a variável se ela possuir um valor e seu tipo pode ser alterado ao longo do código.

Tipos de variáveis em JS

Primitivos:

undefined -> Não há tipo nem valor

boolean -> verdadeiro ou falso

string -> textos

number -> números

Complexos:

function -> funções

object -> objetos

Em JS variáveis podem armazenar funções e objetos. Para uma padronização iremos utilizar daqui pra frente na declaração de variáveis *let* e na declaração de constantes *const*. Para conhecimento *null* é diferente de *undefined,* onde *undefined* é uma variável indefinida e null ao ser atribuído declara a variável com valor vazio.

HOISTING

Ocorre quando o código JS é compilado, todas as variáveis declaradas são movidas para o topo do escopo local (se declaradas como função) e para o escopo global (se declaradas fora de uma função). Nesse processo declarações de funções e classes também são movidas para o topo ficando acima das declarações de variáveis.

Em termos de hoisting temos diferenças significativas entre VAR, LET e CONST:

VAR – pode ser declarado sem ser inicializado e está disponível fora de seu escopo

LET – pode ser declarado sem ser inicializado porém não está disponível fora de seu escopo

CONST – não pode ser declarada sem ser inicializada e não está disponível fora de seu escopo

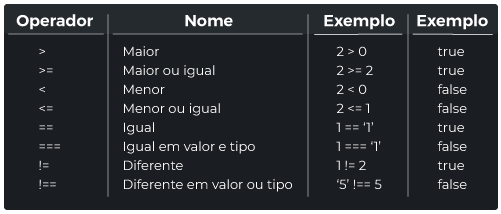
Ao utilizarmos *VAR* essas variáveis declaradas através dele não respeitaram seu escopo, ficando disponível fora dele, ou seja, uma variável declarada dessa forma dentro de uma função também estará disponível fora dessa função. Utilizando *LET* isso não ocorre, ou seja, o escopo é respeitado, tornando essa a principal diferença entre eles. Já o *CONST* impede que o valor seja alterado após ser iniciado, ao tentar muda-lo um erro irá aparecer.

Condicionais

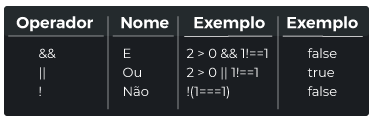
Operadores Lógicos

O código é executado de diferentes formas dependendo das condicionais inseridas dentro dele, que serão interpretadas como verdadeiras ou falsas dependendo de sua interpretação. É aí que surge o tipo de dado BOOLEANO que possui o valor *true* ou *false*.

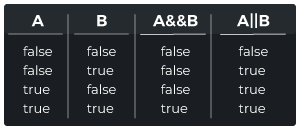
Portanto CONDIÇÃO é uma operação lógica cujo resultado é um valor booleano. Em JS temos os seguintes operadores de comparação:



Os operadores lógicos em JS são:



Os operadores de comparação retornam um booleano dependendo do resultado de comparação. Já operadores lógicos fazem operações sobre valores booleanos. Por exemplo && só retorna TRUE se ambas as condições forem verdadeiras. Já || retorna verdadeiro se uma das condições for verdadeira. Diante disse temos a seguinte tabela lógica.



If/Else

if(condição){ if(condição){

} }

else{ Ou else if(condição){

} }

else{

}

Operador Tenário (if/else)

É uma espécie de simplificação do if/else

Por exemplo: Há um cliente que com 100 pontos é um cliente premium, abaixo disso é um cliente comum.

let pontos = 200;

let tipo = pontos > 100 ? ‘premium’ : ‘comum’;

console.log(tipo);

Switch

Estrutura condicional que recebe um valor e executa o código conforme os casos que a variável receber.

let hoje = new Date().getDay();

let dia;

switch (hoje){

case 0:

dia = “Domingo”;

break;

case 1:

dia = “Segunda”;

break;

case 2:

dia = “Terça”;

break;

case 3:

dia = “Quarta”;

break;

case 4:

dia = “Quinta”;

break;

case 5:

dia = “Sexta”;

break;

case 6:

dia = “Sábado”;

break;

}

Truthy e Falsy

São casos que não há necessidade de testagem com valores específicos, eles já nos retornaram um True ou False. São exemplos de falsy variáveis que retornam FALSE as que contenham os seguintes valores:

* 0 (ZERO)
* -0 (menos Zero)
* '' (aspas simples vazias) ou "" (aspas duplas vazias) (string vazia)
* null
* undefined
* NaN (not a number)

Todos os demais retornam TRUE. São exemplos de truthy:

* []
* {}
* function(){}

Esses conceitos são utilizados para verificar se as variáveis são vazias, se possuem valores UNDEFINE, NULL, NaN, etc.

Laços de Repetição

Aqui temos os chamados Laços ou *loops,* que são estruturas utilizadas dentro da programação para eventos que poderão ocorrer mais de uma vez na execução/compilação do código.

While

Estrutura que executa o código enquanto a condição for verdadeira.

While (condição) {

Código a ser executado

}

Do-While

Há casos em que é necessário executar o *while* pelo menos uma vez. Para isso temos a estrutura *do-while*, onde primeiro se executa o código e depois é verificado a condição para continuar executando ou não.

**Bibliografia**

<https://app.becas-santander.com/pt/program/bolsas-santander-tecnologia-santander-coders-web-full-stack-2021>

<https://natahouse.com/pt/hoisting>

<https://dev.to/joaoava/tipagem-fraca-forte-dinamica-e-estatica-g8k>

<https://robsoncastilho.com.br/2019/11/16/linguagens-estaticamente-ou-dinamicamente-tipadas/>

<https://youtu.be/Mbwg0YIZwYo?list=PLnNURxKyyLIIyJ_XKZHzU8SQyGu7yuqhn>

<https://www.youtube.com/watch?v=q5T_VMpvQyk>