# **Anotações curso udemy**

**Algoritmo**

**Sequência de sentenças de código que vão ser executadas**

**Estruturas de dados**

**Não é o dado em si, mas é algo que define as regras que os dados vão seguir**

**Estruturas de dados básicas**

1. **Inteiro**

**real**

**letra**

**booleanos**

1. **Variáveis e constantes**
2. **Operações - lógicas - relacionais - aritméticas- atributos**

**Sequências de código podem ou não ocupar mais de uma linha**

**Podem ser organizados por blocos - dentro de chaves**

**Podem ou não terminar em ponto e vírgula;**

**Executando JavaScript**

console.log("testando")

**Com a extensão code runner instalada – Ctrl + Alt + n pra executar o código no terminal**

**FORMATAR DOCUMENTO NO VSCODE - identa o texto**

**F12 pra abrir o console no browser**

**Comentários**

**// comentários de uma linha**

**/\* comentários de múltiplas linhas \*/**

**/\* + enter - inicia com \* a cada nova linha**

* **Comentários relevantes**
* **Nomes claros**
* **Comentar tudo que sentir necessidade**
* **Linguagem ubíqua**

**Variáveis e Constantes**

**Como declarar**

var a = 3 // var a recebe 3

var a = 4 // redeclarando a variável a

**var pode redeclarar o valor da variável**

**let não pode**

let b = 4

let b = 5 // o programa não irá executar pois não é possível criar uma let com o mesmo nome e redeclarar o valor

**let não pode redeclarar o valor, mas pode atribuir um novo valor. Por isso é mais indicado se utilizar let, assim saberemos que já tem uma variável com o mesmo identificador**

**Ex:**

let c = 5

c = 6

console.log(c)

**programa irá mostrar ‘6’ na tela**

const d = 10

console.log(d)

**Não podemos alterar o valor de uma constante**

**Evitar nomes genéricos e siglas para nomear dados**

**Tipagem fraca - tipagem dinâmica**

**A variável pode variar de tipos conforme você troca o seu conteúdo**

**String pra number...**

**Number: alguns cuidados**

console.log(7/0)

**Infinity**

console.log("10"/2)

**5**

**O JavaScript vai considerar a string "10" e transferir o seu tipo para Number, pois tem uma tipagem fraca**

console.log("10,2"/2)

**NaN (não é um número)**

**10.2 certo**

console.log('3' + 2)

**32**

**O Js não vai somar, e sim concatenar.**

console.log('3' - 2)

**1**

**Ao contrário do + (que também é um operador de atribuição) ele considera a string como number**

**JavaScript tem alguns problemas com pontos flutuantes**

console.log( 0.1 + 0.7)

**7.999999999**

**Math**

**JavaScript não tem operador de exponenciação então utilizamos o math**

**Math.PI - contante que calcula o valor de PI**

**Math.pow( raio,2 ) // raio elevado a 2 // - calcula uma exponenciação**

**String**

**Cadeia de caracteres**

**Pode ser delimitado por aspas duplas (""), simples ('') ou símbolo da crase ` `**

**const nome = "rayane"**

console.log(nome.charAt(4))

**retorna a quarta letra da string**

**O índice começa com 0, então ele retorna a letra 'n'**

console.log(nome.charCodeAt(3))

**Código da tabela unicode**

**console.log(nome.indexOf('3'))**

**Retorna 'a' que é o índice associado ao digito 3 que está dentro da var**

console.log(nome.substring(1))

ayane - a partir do índice 1 até o final

console.log(nome.substring(0,3))

ray

**- vai do índice 0 ao 3 (sem incluir o 3)**

console.log('Olá '.concat(nome).concat("!"))

**ou**

console.log('Olá '+nome+"!")

**Olá rayane - concatena o Olá + o conteúdo da variável nome**

**Olá - literal: valor direto sem armazenar numa variável**

**Regex**

**console.log(nome.replace(3, 't'))**

**raytne - pegou o índice três e substituiu por t**

**console.log(nome.replace(/\d/, 'e'))**

**Substitui todos os dígitos do texto pela letra 'e'**

console.log(nome.replace(/\w/g, 'e'))

**Substitui tudo pela letra 'e'**

**Template String**

const contatenacao = 'olá ' + nome + '!'

console.log(contatenacao)

olá rayane!

**não permite quebra de linha**

const template= `

Olá

${nome}`

console.log(template)

**considera a quebra de linhas**

**O template string também realiza expressões matemáticas**

console.log(`1 + 1 =  ${1 + 1}`)

1 + 1 = 2

**Boolean**

let boolean = false

console.log(boolean)

**atribui um valor falso a variável, o resultado no terminal resulta em false**

boolean = true

console.log(boolean)

**o valor da variável é alterado para true, o terminal retorna true**

boolean = 1

console.log(!!1)

**A atribuição 1 retorna true pois utiliza as duas exclamações, a primeira nega e a segunda retorna o valor real. Se colocar pra imprimir apenas o 1, o terminal vai imprimir uma string**

boolean = 0

console.log(!!0)

**retorna false**

**Array**

* **Estrutura de dados dinâmica – pode adicionar ou excluir elementos**
* **Estrutura indexada**
* **Estrutura heterogênea – pode misturar dados (não indicado)**

const valores = [7.7, 9.9, 8.7, 8.3]

**declarada entre conchetes, e separadas por vírgulas**

console.log(valores[0], valores[3]) // acessa o índice 0 e o 3

console.log(valores [4]) // retorna undefinedy, pois o índice 4 não existe

valores[10] = 10

console.log(valores)

[ 7.7, 9.9, 8.7, 8.3, <6 empty items>, 10 ]

console.log(valores.length) // quantos itens tem numa array

valores.push(false, null, 'teste') // adiciona itens a array \*\*não é uma boa prática, devemos criar uma array pra cada tipo de dado

console.log(valores.pop()) // retira o último item da array

console.log(valores) // array sem o último item

delete valores [2] // deleta o segundo índice da array

console.log(typeof valores) // tipo object

**Object**

* **Declarado entre chaves**
* **Conjunto de pares chave/valor**
* **Notação literal de objetos**
* **Pode criar um objeto vazio e depois definir como ele é**
* **Dentro de um objeto só pode haver um identificador**
* **Pode utilizar identificadores com espaço \*não é uma boa prática\***

const prod1 = {}

prod1.nome = 'teste' // atribuindo valor a um objeto vazio

prod1.preco = 2090.90

prod1['teste 2'] = 0.40 // evitar atributos com espaço

const prod2 = {

    nome: 'camisa',

    preco: 79.90,

}

**Pode haver outros objetos dentro de um objeto único**

**Null e Undefined**

* **Utilizada pra quando queremos zerar uma variável do tipo referência, usar o Null para isso.**
* **Não é possível acessar um objeto quando o seu valor ta null ou undefined**

**const a = {name: 'teste'}**

**essa constante tem o endereço no qual o objeto está localizado**

**const b = a**

**a constante b recebe o mesmo endereço ao qual a constante a aponta, ou seja, as duas apontam para o mesmo lugar na memória**

**b.name = 'teste2'**

**a partir disso a constante a também vai ser “trocada”**

**Isso é chamado atribuição por referência**

**let c = 3**

**undefined**

**let d = c**

**undefined**

**d++**

**3**

**d**

**4**

**Chamado cópia por valor , ou seja, cada um tem seu valor**

let valor // declarado mas não foi inicializado

**undefined**

valor = null // não recebe nenhum valor, e não aponta pra nenhum endereço de memória

// mas ela foi definida

console.log(valor)

//console.log(valor.toString()) // não tentar acessar dados de uma variável nula

console.log(produto.preco) // retorna undefined pois não consegue localizar o preco, mas como a produto foi definida, mesmo vazia, o programa não dá erro

console.log(produto.preco.a)

**nesse caso o programa vai dar erro, pois não conseguimos acessar ‘a’ porque preco não foi definido – o que vem antes do ponto tem que estar definido**

produto.preco = undefined // evitar atribuir undefined

// isso não vai fazer com que o objeto deixe de ter o atributo

// delete produto.preco - assim consegue se deletar o atributo, retornando um objeto vazio

**Funções**

* **Define um bloco de códigos no qual tem uma sentença de códigos e executa-as**
* **Bloco de código nomeado, que pode ser reusado várias vezes**
* **Uma função pode não receber nenhum parâmetro**
* **Pode não retornar dados**
* **Pode receber dados de entrada e não retorna, etc.**

// Função sem retorno

function imprimirSoma(*a*, *b*) {

    console.log(a+b)

}

imprimirSoma(2, 3) // retorna a soma dos dois

imprimirSoma(2) // undefined

imprimirSoma(2, 3, 4, 5, 6, 7) //retorna o resultado da soma das duas primeiras

//Funcao com retorno

function soma(*a*, *b* = 1) {

    return *a* + *b*

}

console.log(soma(2, 3))

console.log(soma(2)) // passou só um parâmetro, mas como o valor de b foi definido ele soma 2 + 3

// Armazenando uma função em uma variável

const imprimirSoma = function (*a*, *b*) {

    console.log(*a*+ *b*)

}

// Armazenando uma função arrow em uma variável

const soma = (*a*, *b*) => { /// => substitui o nome function

    return *a* + *b*

}

console.log(soma(2,3))

// Retorno implícito

const subtracao = (*a*, *b*) => *a* - *b* // realiza o return sem precisar usar a palavra return

console.log(subtracao(2, 3))

**Declarando Variáveis com VAR**

* **Duas formas: Var global, ou dentro do escopo função. (FUGIR DO ESCOPO GLOBAL)**
* **Recomendado utilizar ela apenas dentro da função, pois apenas pode ser acessada dentro dela**
* **Pode haver duas variáveis iguais dentro do programa, desde que sejam escopos diferentes**
* **NÃO TEM ESCOPO DE BLOCO, só existe escopo de bloco dentro de uma função**

{{{{ var sera = 'Será?'}}}}

console.log(sera)

**Em outras linguagens essa variável apenas ficaria visível dentro do escopo, mas não é recomendado fazer isso**

function teste() {

    var local = 123

    console.log(teste)

}

teste()

// console.log(local)  não consegue acessar a variável,

// apenas dentro do escopo da função

var numero = 1

{

    var numero = 2

    console.log('dentro=', numero)// retorna o conteúdo atribuído abaixo

}

console.log('fora=', numero) // retorna a variável que foi atribuída dentro do escopo,

// pois não foi criada uma função pra armazenar a variável, apenas um bloco entre chaves.

**Declarando Variáveis com LET**

* **Let tem escopo de bloco, não precisa estar dentro de uma função**
* **Têm escopo global, de função e de bloco**
* **É visível dentro do bloco quando estiver uma variável declarada dentro, caso não haja, ele busca fora do bloco**

**EX:**

var numero = 1

{

    let numero = 2

    console.log('dentro', numero) //retorna apenas o que está dentro do escopo

}

console.log('fora', numero) //retorna o que está dentro de var

**EX 2:**

var numero = 1

{

    let numero2 = 2

    console.log('dentro', numero) /\*ele procura se dentro do escopo tem uma varável chamada numero2, como não tem retorna o que está fora\*/

}

**Var e Let em Loop**

for (var i = 0; i < 10; i++) {

    console.log(i)

}

console.log('i=', i) // var é mostrada fora do bloco, pois não tem escopo de bloco

for (let a = 0; a < 10; a++) {

    console.log(a)

}

// console.log(a) a variável let não está visível fora do bloco, pois tem escopo de bloco

for (var i = 0; i < 10; i++) {

    funcs.push(function() { // toda vez que var der uma volta, atribui um novo valor a array

        console.log(i)

    })

}

funcs[2]()

funcs[8]() // imprime em todas situações o valor 10 \*\*COM LET ISSO NÃO ACONTECE\*\*, mostra o número 2 e 8

**Hoisting**

* **‘Joga pra cima’, então mesmo que uma variável seja declarada abaixo da sua chamada o programa retorna undefined**
* **Em outras linguagens de programação isso não seria possível**
* **Não é recomendado fazer isso**
* **Com let não ocorre esse efeito de içamento, então o programa dá erro**

**EX:**

console.log('a=', a) //undefined

var a = 2 // joga pra cima

console.log('a=', a) // retorna o valor da variável a