**Comandos em JS: danilinda**

//

/\* \*/

<script>

**window.alert** (‘este comando emite uma mensagem!')

**window.confirm** (‘este comando faz uma pergunta de confirmação')

**window.prompt** (‘este comando faz um pergunta de resposta!')

</script>

**Variáveis - VAR**

**Como Criar variáveis,**

Ex: **var** nome ou **let** nome

Para uma variável receber um valor usamos var nome = Gustavo (desta forma criamos uma variável e ao mesmo tempo demos uma valor a ela)

nome = Gustavo (desta forma apenas demos um valor para a variável )

**=** Significa “**Recebe”**

**=null** Significa não recebe

var s1 = “ “

var n1 = ‘ ’

var n1 = ` `

Regras dos identificadores (variáveis):

**Podem** começar com letra, $ ou \_

**Não** podem começar com números

É possível usar **letras ou números**

É possível usar **acentos e símbolos**

Não pode conter espaços, substituindo por

Não pode usar palavras que são comandos (Ex= var var)

Dicas para **nomes das variáveis**:

**Maiúsculas** e **Minúsculas** fazem a diferença!

Nomes coerentes a função da variável. Ex: Variável que vai armazenar a idade, coloca o nome dela de “idade”

Evite se tornar um “Programador Alfabeto” ou um “Programador Numérico”. Ex: Não usar os nomes das variáveis como “a”, “b”, “c” etc.; ou “a1”, “a2”, “a3” etc.

**Tipos de Dados das Variáveis**: (Tipos Primitivos, lembrando que existem muitas outras!)

**Numbers**; 1; -2; 4.5; 6.555 -> Basicamente números

**Strings**; Maria, Google, Joao, pedreiro, (seu CPF) -> Basicamente cadeia de caracteres

Boolean; True; False

**Typeof**; Indica o tipo de variável

**+** Significa concatenação

**Transformando uma string em um number**

var n1 = Number.parseInt (window.prompt ('digite aqui um numero!'))

var numero1 = Number.parseFloat (window.prompt ('digite aqui um numero!'))

var numero1 = Number (window.prompt ('digite aqui um numero!’))

// Number.parseInt: Numero Inteiro

// Number.parseFloat: Numero com virgula

// Number: JS vai decidir qual é

**Transformando um number em uma string**

window.alert ('a soma dos numeros é: ' + soma.toString()) // Jeito mais antigo

ou

window.alert ('a soma dos numeros é: ' + String(soma)) // Jeito mais simples

**Formatando Strings:**

var teste = 'java script’

‘eu estou aprendendo’ + teste

**${ }**

`eu estou aprendendo ${teste}` -> não esqueça de usar crase!

teste.**length** // conta quantos caracteres tem na variável

teste.**toUpperCase** // coloca tudo em caixa alta

teste.**toLowerCase** // coloca tudo em minúsculo

**Formatando números:**

Var n1 = 1543.5

n1.**toFixed(**2**)**  // Coloca em duas casas decimais (para colocar em mais ou menos casas troque o numero entre parênteses)

n1.toLocaleString( ‘pt-BR’,{style: ‘currecy’, currency: ‘BRL’} ) // Coloca o R$ na frente do numero (pode trocar entre outras moedas)

n1.replace(‘.’,‘,’) // Troca o ponto pela virgula

n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL'})

**Operadores**

Tipos de operadores que vamos estudar:

**Aritméticos**

**Atribuição**

**Relacionais**

**Lógicos**

**Ternarios**

**Operadores Aritméticos**

**+**  // Somar

**-** // Subtrair

**\*** // Multiplicação

**/** // divisão

**%** // Resto de uma divisão

**\*\*** // Potencia do primeiro numero elevado ao segundo

5 + 3 = 8

5 – 3 = 2

5 \* 3 = 15

5 / 3 = 1,6

5 % 3 = 2

5 \*\* 3 = 125

**Precedência dos operadores aritméticos**

Prioridade nas operações

( )

\*\*

/ \* %

+ -

Mesmo grau na mesma linha, faz da esquerda para a direita

**Auto Atribuições**

var n = 3

n = n + 4 // ele vai somar ele mesmo a 4

n = n - 5 //ele vai pegar ele mesmo e subtrair 5

n = n\*4 //ele vai pegar ele mesmo e multiplicar por 4

n = n / 2 //ele vai pegar ele mesmo e dividir por 2

n = n \*\* 2//ele vai pegar ele mesmo e elevar a 2 potência

n = n % 5 //ele vai pegar ele mesmo, dividir por 5 e dar o resto

**Forma Simplificada**

n += 4

n -= 2

n \*= 5

n /= 2

n \*\*= 2

n %= 5

**Outra simplificação**

n++ // é a mesma coisa que n = n + 1 ou n += 1

n— //é a mesma coisa que n = n - 1 ou n -= 1

++n // ele vai somar antes

—n // ele vai diminuir antes

**Operadores Relacionais**

5 **>** 2 // Maior

7 **<** 4 // Menor

8 **>=** 8 // Maior igual

9 **<=** 7 // Menor igual

5 **==** 5 // Igual

4 **!=** 4 // Não igual

**=** Significa RECEBE

**==** Significa IGUAL

**Operadores Relacionais de Identidade**

5 == 5 → True // Cinco é igual a cinco

5 == ‘5’ → True // == Não testa o tipo (Number ou String)

5 === ‘5’ → False // Operador de **igualdade restrita**; Não são idênticos

**Operadores Lógicos**

**!** // Negação; Quero uma caneta, mas ela **NÃO** pode ser azul

**&&** // Conjunção; Quero uma caneta azul **E** vermelha

**||** // Disjunção; Quero uma caneta azul **OU** vermelha

**Negação**

**!** True → False

False → True

**Conjunção**

**&&**  True + True → True

True + False → False

False + True → False

False + False → False

**Disjunção**

**||**  True + True → True

True + False → True

False + True → True

False + False → False

**Precedência de Operadores**

( ) ; \*\* ; / ; ... Aritméticos

> ; < ; >= ; ... Relacionais

! Lógicos

&&

||

**Operadores Ternarios**

Teste **?** True **:** False

**DOM**

Document Object Model

Árvore DOM

Diagrama

Descrição gerada automaticamente

Diferença entre innerHTML, innerText e textContent

**innerHTML**: Retorna o conteúdo de texto do elemento, **COM formatações** e **COM elementos html**

**innerText**: Retorna **apenas o conteúdo de texto** do elemento e todos os seus filhos, sem formatações ou elementos html, exceto os elementos <script> e <style>.

**textContent**: Retorna o conteúdo de texto do elemento e todos os descendentes, **COM formatações**, mas **sem os elementos html**

**Utilizando DOM**

por **Marca** → getElementsByTagName( )

por **ID** → getElementsById( )

por **Nome** → getElementsByName( )

por **Classe** → getElementsByClassName( )

por **Seletor** → querrySelector( )

querrySelectorAll( )

**Utilizando Selector**

**<div>** é representado por **#**

<div id = ‘msg’> Clique em mim</div>

var d = window.document.querrySelector (’div#msg’)

**Classe** é representado por**. (ponto)**

<div class = ‘msg’> Clique em mim</div>

Var d = window.document.querrySelector(’div.ms’)

**Eventos DOM**

**onmouseenter**=" " → executa alguma FUNÇÃO quando o cliente **ENTRAR COM O CURSOR DENTRO** do elemento.

**onmousemove**=" " → executa alguma FUNÇÃO quando o cliente **MOVER O CURSOR DENTRO** do elemento.

**onmousedown**=" " → executa alguma FUNÇÃO quando o cliente PRESSIONAR O CLICK DENTRO do elemento.

**onmouseup**=" " → executar alguma FUNÇÃO quando o cliente SOLTAR O CLICK DENTRO do elemento.

**onclick**=" " → executa alguma FUNÇÃO quando o cliente **CLICAR** DENTRO do elemento.

**onmouseout**=" " → executa alguma FUNÇÃO quando o cliente **SAIR COM O CURSOR DE DENTRO** do elemento.

**onmouseover**=" " → executa alguma FUNÇÃO enquanto o cliente **MANTER O CURSOR** SOBRE o elemento.

**FUNÇÃO**

Conjunto de códigos que vão ser executados quando determinado evento ocorrer

Precisa estar em um bloco

**function** nomedaFunção ( ) {

bloco de códigos

}

**addEventListener**( ) → Encurtar

element.addEventListener(event, function, useCapture)

**<input>** → Especificar uma ÁREA onde o usuário poderá SUBMETER DADOS.

Assim como o método window.prompt(), a tag <input> também retorna um valor do tipo STRING. Então, para fazer a soma de 2 NUMBERS, é necessário converter o dado padrão de STRING para NUMBER.

**.value** → DEFINIR ou RETORNAR o valor de um atributo.

**Condições**

Estrutura **Condicional Simples**: **IF** (

Se (if) condição1 for VERDADEIRA, executa o Bloco A.

**if (condição1)** {

Bloco A

}

Estrutura **Condicional Composta**: **IF/ELSE**

Se (if) condição1 for VERDADEIRA, executa o Bloco A.

Se não (else), executa o Bloco B.

**if (condição1)** {

Bloco A

} **else** {

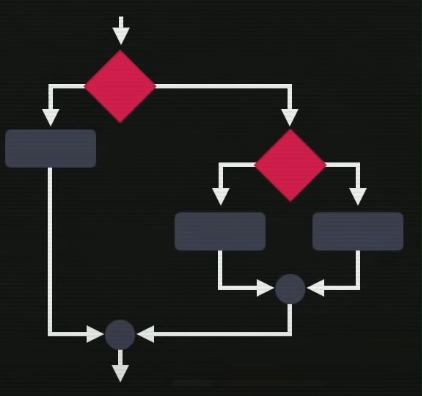
Bloco B

}

**Condicional Aninhada**: IF/ELSE IF/ELSE (se/senão se/senão)

Uma condição dentro da outra

Boa para intervalos de valores



Se (if) condição1 for VERDADEIRA, executa o Bloco A.

Senão (else), se (if) condição2 for VERDADEIRA, executa o Bloco B.

Senão (else), executa o Bloco C.

if (condição1) {

Bloco A

} else if (condição2) {

Bloco B

} else {

Bloco C

}

**Condicional Múltipla**: SWITCH (expressão)

\*tradução: switch = escolha; case = caso

Útil para valores fixos

Uma imagem contendo objeto, relógio

Descrição gerada automaticamente

Switch (expressão) {

case valor 1:

break

case valor 2:

break

case valor 3:

break

default:

break

}

O break é necessário para quebrar a sequência

**new Date**() → DIA da SEMANA, DATA, HORA ATUAL e FUSO HORÁRIO.

Ex:

Fri Nov 04 2022 08:30:47 GMT-0300 (Horário Padrão de Brasília)

data = new Date()

data**.getFullYear**() → ANO atual (yyyy)

data**.getMonth**() → MÊS atual (0-11)

data**.getDate**() → DIA atual (1-31)

data**.getDay**() → DIA da SEMANA atual como um NÚMERO (0-6)

\*0 = domingo; 6 = sábado.

data**.getHours**() → HORA atual (0-23)

data**.getMinutes**() → MINUTOS atuais (0-59)

data**.getSeconds**() → SEGUNDOS atuais (0-59)

data**.getMilliseconds**() → MILISSEGUNDOS atuais (0-999)

data**.getTime**() → MILISSEGUNDOS desde o dia 1 de Janeiro de 1970 (WTF? xD)

**Repetições**

**Com Teste Lógico no Início**

Irá se repetir enquanto(while) a condição for verdadeira

Faz o teste lógico no início e depois executa

**Desenho de pessoa com relógio digital

Descrição gerada automaticamente com confiança baixa**

Estruturas de (Laços || Iterações || Repetições || Looping):

Function comer pizza( ) {

while (temFatia( ) ) {

comerfatia( )

}

}

**Com Teste Lógico no Final**

**do/while** (faça/enquanto)

Esta estrutura faz o teste lógico DEPOIS de rodar o programa.

do {

Bloco

} while (condição)

**Repetição Com Variável de Controle**

**for** (inicialização; condição; expressão final)

console.log('Vai começar')

for ( var c = 1 ; c <= 5 ; c++) {

console.log(c)

}

console.log('Fim!')

**Variáveis Compostas**

"Um **array** é uma **variável composta** que tem **vários elementos**. Cada elemento é composto por **seu valor** e por **uma chave (key) de identificação**". - Guanabara, Gustavo.

**Variável SIMPLES** → armazena **UM VALOR** por vez.

**let** identificador = valor

**Variável COMPOSTA** → armazena **VÁRIOS VALORES** em uma MESMA ESTRUTURA.

**let** identificador = **[**valor, valor, valor**]**

**let num = [5, 8, 4]**

0 1 2 **(posições)**

Nesta array há:

3 ELEMENTOS (elemento é conjunção de: VALOR && POSIÇÃO)

com VALORES 5, 8 e 4, respectivamente.

o valor 5 está na POSIÇÃO 0;

o valor 8 está na POSIÇÃO 1;

o valor 4 está na POSIÇÃO 2.

\*posição == índice == chave == key

**num[3] = 6** → Para ADICIONAR um NOVO VALOR à esta ARRAY em uma POSIÇÃO ESPECÍFICA

Agora num vale: [5, 8, 4, 6]

**num.push(7)** → Para **ADICIONAR** um NOVO VALOR à esta ARRAY **na ÚLTIMA POSIÇÃO**

Agora num vale: [5, 8, 4, 6, 7]

**num.length** → Para **RETORNAR a QUANTIDADE de ELEMENTOS**/KEYS da ARRAY num.

O valor retornado é: 5

**num.sort()** → Para **ORDENAR CRESCENTEMENTE os ELEMENTOS**/VALORES desta ARRAY

Agora num vale: [4, 5, 6, 7, 8]

*/\* É PROBLEMÁTICO tratando-se de NÚMEROS. Pois ele VERIFICA a 1° CASA DECIMAL e, ORGANIZA os ELEMENTOS \*/*

*let numeros = [99, 25, 100]*

*numeros.sort()*

*console.log(numeros)*

*Ele organizaria o 100 antes do 25, devido ao 1 \*/*

**num.indexOf(número)** → Para **BUSCAR um VALOR** dentro desta ARRAY:

RETORNE a CHAVE/ÍNDICE do VALOR 7.

O valor retornado é: 3

\*Caso seja passado um número INEXISTENTE, o valor retornado será: **-1**

**for/in (para/em)**

Faz um **LOOP** ATRAVÉS das PROPRIEDADES de um OBJETO ou ARRAY.

**for (** key **in** object**) {**

**}**

/\* Lê-se: PARA(for) cada POSIÇÃO(x) NO(in) ARRAY(numeros) {execute este bloco} \*/

inseriu-se no 1° placeholder: a CHAVE(x);

TODO ARRAY É UM OBJETO. MAS NÃO TODO OBJETO É UM ARRAY. \*/

**Funções**

São ações executadas assim que são chamadas ou em decorrência de algum evento. (Automaticamente)

Chamada Parâmetros Ação Retorno

Uma função pode receber **parâmetros** e retornar um **resultado**.

**function** ação(parâmetros) {

**return** res

}

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Diagrama

Descrição gerada automaticamente com confiança baixa

src="script.js"

link href="estilo.css"

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

O que estudar?

Functions:

Arrow Functions

Callbacks

Funções Anônimas

iife

JavaScript Funcional

-----

Objetos (orientação a objetos)

Modularização

RegEx (Regular Expressions)

JSON

AJAX

NodeJS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

O que é um Object?

OBJECTS são CONTAINERS de PROPRIEDADES e MÉTODOS.

OBJECTS são VARIÁVEIS COMPOSTAS, assim como os ARRAYS.

Array syntax:

let identificador = [valor, valor, valor]

0 1 2

Object syntax:

let identificador = {nome:valor, nome:valor, nome:valor}

Diferenças:

Para declarar um ARRAY usa-se COLCHETES [ ]

Para declarar um OBJECT usa-se CHAVES { }

\_\_\_\_\_\_\_\_

Nos ARRAYS, os ELEMENTOS (conjunção de: posição e valor) são ACESSADOS por seus ÍNDICES (0, 1, 2).

Nos OBJECTS, as PROPRIEDADES (conjunção de: nome e valor) são ACESSADAS por suas KEYS (nome1, nome2, nome3).

Como fazer este ACESSO (rode este código):

const pessoas = {nome1: "rodolfo", nome2: "rogerin"}

const idades = [20, 35]

// Há 2 maneiras de acessar as PROPRIEDADES de um OBJETO.

// 1) syntax: objectName.propertyName

console.log(`O ${pessoas.nome1} tem 20 anos de idade e o ${pessoas.nome2} têm 35.`)

// 2) syntax: objectName["propertyName"]

console.log(`O ${pessoas["nome1"]} tem 20 anos de idade e o ${pessoas["nome2"]} têm 35.`)

// acessando os ELEMENTOS de um ARRAY através de seus ÍNDICES:

console.log(`O rodolfo tem ${idades[0]} anos de idade e o rogerin têm ${idades[1]}.`)

/\* pessoas é um OBJETO;

nome1 e nome2 são as PROPRIEDADES do OBJETO pessoas;

"rodolfo" e "rogerin" são os VALORES das PROPRIEDADES nome1 e nome2 , respectivamente. \*/

\_\_\_\_\_\_\_\_

Os OBJETOS podem também ter MÉTODOS.

Um MÉTODO é uma FUNÇÃO que é ARMAZENADA como PROPRIEDADE de um OBJETO. Ou seja, MÉTODOS são AÇÕES que podem ser EXECUTADAS em OBJETOS.

Exemplo (rode este código):

const person = {

fName: "Elon",

lName: "Musk",

fullName() {

return `${this.fName} ${this.lName}`

}

}

console.log(person.fullName())

/\* this é uma keyword que refere-se a um OBJETO.

Portanto, this.fName refere-se à propriedade fName DESTE (this) OBJETO (person). \*/

\_\_\_\_\_\_\_\_

O que é this ?

this é uma keyword que refere-se a um OBJETO. this NÃO É uma VARIÁVEL. Você NÃO PODE MUDAR o VALOR de this .

Mas QUAL objeto this refere-se? Isto dependerá de COMO esta keyword foi utilizada ou chamada:

Em um MÉTODO, this refere-se ao OBJETO (exemplo anterior).

Sozinha, this refere-se ao OBJETO GLOBAL.

Em uma FUNÇÃO, this refere-se ao OBJETO GLOBAL.

Em uma FUNÇÃO, em modo estrito (\*strict mode), this é undefined.

Em um EVENTO, this refere-se ao ELEMENTO que RECEBEU o EVENTO.

Métodos como call(), apply() e bind() , podem referir this a QUALQUER OBJETO.

\*O que é Strict Mode?

O comando: "use strict" , indica que um código DEVE SER EXECUTADO em Strict Mode, modo estrito. Este pode ser utilizado no INÍCIO do SCRIPT, agindo GLOBALMENTE, ou DENTRO de uma FUNÇÃO ESPECÍFICA, agindo LOCALMENTE.

Este modo serve para REVELAR MÁS PRÁTICAS de ESCRITA, SYNTAX, ao lançar NOVAS MENSAGENS de ERRO no CONSOLE do NAVEGADOR, que antes passariam desapercebidas. Portanto, este modo auxilia na escrita de um código mais "clean", previnindo que você atribua um valor à uma variável antes de declará-la, por exemplo (rode este código):

// SEM UTILIZAR o Strict Mode:

function serHumano() {

return nome = "rodolfo" // atribuindo um valor à uma variável SEM DECLARÁ-LA.

}

console.log(serHumano()) // "rodolfo" foi exibido normalmente sem mensagens de erro.

console.log('-------')

// UTILIZANDO o Strict Mode:

function pessoa() {

"use strict" // caso APAGUE este "use strict", o código NÃO DARÁ ERRO, mesmo que este exista.

return sobrenome = "da Silva" // atribuindo um valor à uma variável SEM DECLARÁ-LA.

}

console.log(pessoa()) // ReferenceError: sobrenome is not defined

// Para saber mais, Google: "JavaScript Use Strict"