**JavaScript**

**MÓDULO 1**

**Como chegamos até aqui?**

1970 – Darpa > Arpanet.

1993

* Timothy Berners-Lee > HTML, HTTP se juntou com TCP/IP, WWW no CERN.
* Mark Andreessen > Mosaic (Primeiro Navegador)

1994

* Mark Andreessen e Jim Clark > Netscape
* Projeto Interno da Netscape > Mozilla

1995

* Brandon Eich > Mocha > Livescript > JavaScript (Na época Java fazia sucesso)
* Microsoft > JScript

1997

* ECMA (Associação Europeia de Fabricantes de Computadores) > Padronização
* ECMAscript

2002

* Mozilla Firefox (Antigo Netscape)

2008

* Google Chrome

2009

* V8 > Motor de Javascript no Chrome > Gera código JIT (Just in Time) > Código Livre

2010

* Node JS > Roda JavaScript fora do navegador

**ECMAScript**

* 1.0 – 1997
* 2.0 – 1998
* 3.0 – 1999
* ES5 – 2009
* ES6 – 2015
* ES2016 – 2016
* ES2017 – 2017
* ES2018 – 2018

**TECNOLOGIAS**

* JQuery
* Angular
* React
* Vue.js
* Electron
* Ionic
* Cordova

**PRIMEIROS PASSOS**

**Bibliografia**

* JavaScript – O’Riley
* JavaScript – Mauricio Samy Silva
* Guia de referência do Mozilla e da ECMA:

developer.mozilla.org > Referências de JavaScript

ecma-international.org > Standard 262

**CRIANDO O PRIMEIRO SCRIPT**

A criação do script sempre vem antes do </body> com a tag <script></script>

Comandos em JS geralmente com letra minúsculas.

Exs.:

* window.alert(‘Minha primeira mensagem’)
* window.confirm(‘Está gostando de JS?’)
* window.prompt(‘Qual o seu nome?’)

Obs.: Ponto e vírgula no final é opcional.

**MÓDULO 2**

**Variáveis e Tipos Primitivos**

**Comentários em JS:** // (em uma única linha) ou /\* \*/ (em mais de uma linha)

Exemplo de variáveis: Um terreno serve de estacionamento **(memória do computador)** e delimita-se o tamanho das vagas **(variáveis)** para veículos específicos.

Vagas a1 (automóveis) ...  
Vagas c1 (caminhões)...  
Vagas m1 (motos)...

A vaga a1 recebe um carro. Fica assim:  
vaga a1 **=** carro1 (sinal de igual é igual “recebe”)

A vaga a1 recebe outro carro. Fica assim:  
vaga a1 **=** carro2 (tiro o carro 1 e coloco o carro 2

A vaga a1 não pode receber nenhum carro. Fica assim:  
vaga a1 **=** null (ficará nula)

No JS atual utiliza-se a palavra “var” ou “let” para criar variáveis.

Além disso precisa-se de identificadores (nome das variáveis).

Além de valores

**Ex.: var n1 = 5  
 var s1 (s significa strings – cadeia de caracteres) = “Java Script”  
 var s2 = ‘Curso em vídeo’  
 var s3 = `Guanabara`**

**\*Utiliza-se três diferentes símbolos para delimitar a string.**

**Características dos Identificadores**

* Podem começar com letra, $ ou \_
* Não podem começar com números
* É possível letras e números
* É possível usar acentos e símbolos (Ex.: média ou símbolo do PI)
* Não podem conter espaços
* Não podem ser palavras reservadas (alert, function (palavras que o JS usam como comandos)

**Dicas para Identificadores**

* Maiúsculas e minúsculas tem diferença
* Escolher nomes coerentes para as variáveis
* Evite se tornar ‘programador alfabeto’ ou ‘contador’

**Testando variáveis**

* Pode ser pelo Node.Js (.exit pra sair)
* Pode ser pelo Visual Studio Code (Terminal > Novo Terminal ou Ctrl + Shift + ‘)

Digitar node e lançar as variáveis. Para sair, primeiro .exit (node) e depois exit (VScode)

Crtl + L ( limpar o terminal )

**Tipos Primitivos de Dados no JS**

* Number ( 5, 18, -12, 0.5, -15.9)

1. Infinity
2. NaN (Not a Number)

* String (“Google” ‘375.989.089-00’)
* Boolean (True e False)
* Null
* Undefined
* Object

1. Array

* Function

typeof = mostra o tipo do dado no prompt

**TRATAMENTO DE DADOS**

**Símbolo de + significa concatenação**

**CONVERSÃO DE TIPO**

String > Number

* Number.parseInt(n) – Número Inteiro
* Number.parseFloat(n) – Número com ponto decimal (Número Real)
* Number(n) – Versões mais atuais. Não anula as de cima.

Number > String

* String(n)
* n.toString( )

**FORMATANDO STRINGS**

Tendo como exemplo: **var s = ‘JavaScript’**

* ‘Estou aprendendo **s**’ //não faz interpolação
* ‘Estou aprendendo’ + **s** //usa concatenação

Utilizando Template Strings

* `Estou aprendendo ${**s**} //usa template strings (crase, não é aspas simples)

**${ } PlaceHolder**

**Facilitador de strings ao invés de utilizar um monte de concatenações ficando assim confuso.**

Outras coisas utilizando strings

* s.length //o tamanho da string, quantos caracteres tem a string
* s.toUpperCase( ) //tudo para MAIÚSCULAS
* s.toLowerCase ( ) //tudo para minúsculas

\*NOVO COMANDO: document.write (mostra o texto na página)

**FORMATANDO NUMBERS**

Tendo com exemplo: **var n1 = 1543.5**

* **n1.toFixed(n)** //Definir em n as casas decimais
* **n1.toFixed(n).replace(‘.’ , ‘,’)**  //trocar uma coisa por outra (ponto por vírgula)
* **n1.toLocaleString('pt-BR', {style: 'currency', currency: 'BRL' })** //colocar em Reais

**OPERADORES**

**OPERADORES ARITMÉTICOS**

**Binários**

* 5 + 2 = 7
* 5 – 2 = 3
* 5 \* 2 = 10
* 5 / 2 = 2.5
* 5 % 2 = 1 (resto da divisão)
* 5 \*\* 2 = 25 (potenciação)

Obs.: Seguir o precedente matemático  
( ) > \*\* > \* / % > + -

**OPERADORES DE ATRIBUIÇÃO**

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

**Auto atribuições (novas atribuições)**

**(Símbolo de igual significa “recebe”)**

Var n = 3

n = n + 4 ( n passa a valer 7)  
n = n – 5 (n deixa de valer 7 e passa a valer 2)  
n = n \* 4 (n deixa de valer 2 e passa a valer 8)  
n = n / 2 (n deixa de valer 8 e passa a valer 4)  
n = n \*\* 2 (n deixa de valer 4 e passa a valer 16)  
n = n % 5 (n deixa de valer 16 e passa a valer 1)

Simplificando (só se for o mesmo atributo)....

n +=4  
n -=5  
n \*= 4  
n /= 2  
n \*\*=2  
n %=5

**OPERADORES DE INCREMENTO**

Quando for n + 1 ou n -1  
var x = 5

**Pós Incremento (++ ou- - significa + 1 ou -1)**

x = x + 1 🡪 x+= 1 🡪 x ++ = 6  
x = x -1 🡪 x -=1 🡪 x – = 5

**Pré Incremento**

var x = 10

++x = 11  
--x = 10

**OPERADORES RELACIONAIS**

**\*Resultados Booleanos**

* **5 > 2 (TRUE)**
* **7 < 4 (FALSE)**
* **8 >= 8 (TRUE)**
* **9 <= 7 (FALSE)**
* **5 == 5 (TRUE) (OPERADOR DE IGUALDADE)**
* **4 != 4 (FALSE) (NÃO IGUAL)**

A ordem de operações sempre se inicia pelas operações aritméticas

Ex.: a < b – 10 (Faz-se primeiro b -10)

**Operadores de Identidade**

**5 == 5 🡪 True  
5 == ‘5’ 🡪 True**

O JS não testa o tipo com sinal de igualdade

**5 === ’5’ 🡪 False (Não Idêntico)  
5 === 5 🡪 True (idêntico)**

**5 !== ‘5’ 🡪 True (Desigual Restrito)  
5 !== 5 🡪 False (Não é desigual restrito)**

**OPERADORES LÓGICOS**

* **! – NEGAÇÃO (NÃO) (Valor Unário)**

1. ! True 🡪 False
2. ! False 🡪 True

* **&& - CONJUNÇÃO (E) (Valor Binário)**

1. True && True 🡪 True
2. True && False 🡪 False
3. False && True 🡪 False
4. False && False 🡪 False

* **|| - DISJUNÇÃO (OU) (Valor Binário)**

1. True || True 🡪 True
2. True || False 🡪 True
3. False || True 🡪 True
4. False || False 🡪 False

\*Ordem: Operadores Aritméticos > Relacionais > Lógicos (! > && > ||)

**OPERADORE TERNÁRIO**

* **?**
* **:**

**TESTE ? TRUE : FALSE**

Var média = 5.5

MÉDIA >=7.0 **?** ‘APROVADO’ **:** ‘REPROVADO’

“REPROVADO”

**ENTENDENDO O DOM**

**O que é DOM?**

Document Object Model

Presente dentro do navegador

**Árvore DOM**

* window (raiz)

1. location (URL)
2. document (documento atual)
   * + - 1. html (parente)

head (filho de html)

meta

title

body (filho de html)

h1

p

Strong

Abbr

a

etc...

div

etc...

1. history (guarda de onde veio, pra onde vai)

**Selecionando Elementos na árvore Dom**

* + **por Tag**

getElementsByTagName( ) (Pode selecionar mais de um elemento com [])

* + **por Id**

getElementsById ( ) (Pode selecionar mais de um elemento com [])

* + **por Nome**

getElementsByName ( ) (Pode selecionar mais de um elemento com [])

* + **por Classe**

getElementsByClassName ( ) (Pode selecionar mais de um elemento com [])

* + **por Seletor**

**Eventos DOM**

**Eventos de Mouse**

* mouseenter (mouse entrou no elemento)
* mousemove (mouse sendo movido no elemento)
* mousedown (mouse clicado e segurado)
* mouseup (soltar o botão do mouse)
* click (apertar e soltar rapidamente)
* mouseout (mover o mouse para fora do elemento)

**Funções**

Conjunto de códigos que vão ser executadas quando o evento ocorrer

Function ação(parâmetros){

Bloco

}

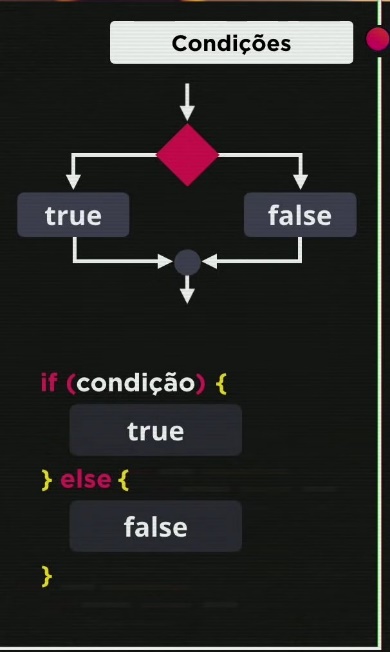
* Os eventos podem ser inseridos fora do script com a adição dos eventos dentro do elemento HTML e usando as funções para chamá-los.  
  Ex.: <div id=”area” **onclick = “clicar()”>**
* Para inserir eventos no script:

area.**addEventListener(‘click’, clicar)**

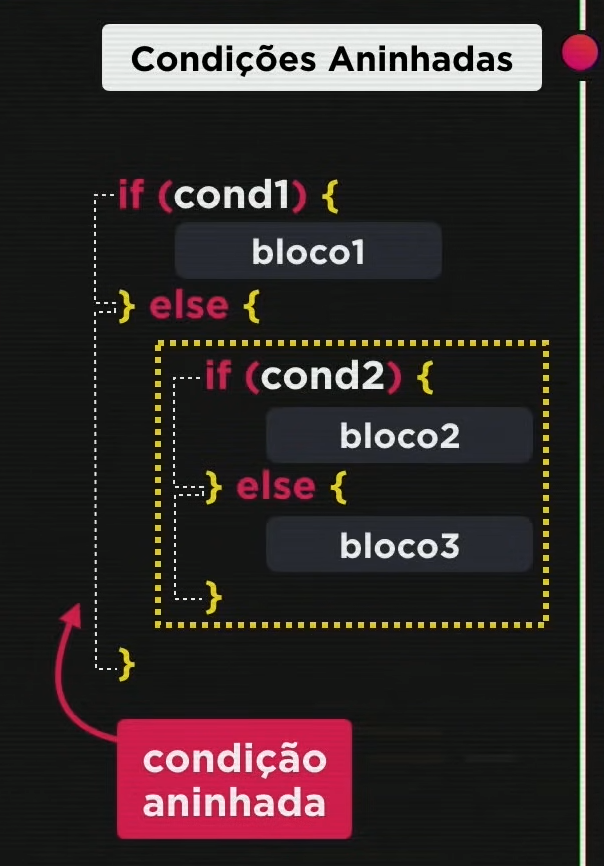
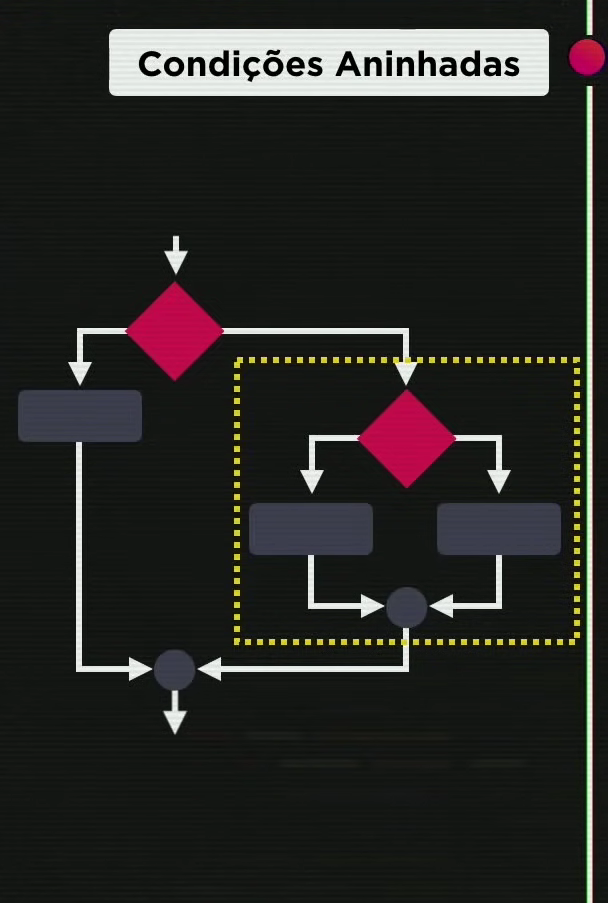
**Detecção de Erros**

* Inspecionar no DevTools

**Condições**

****

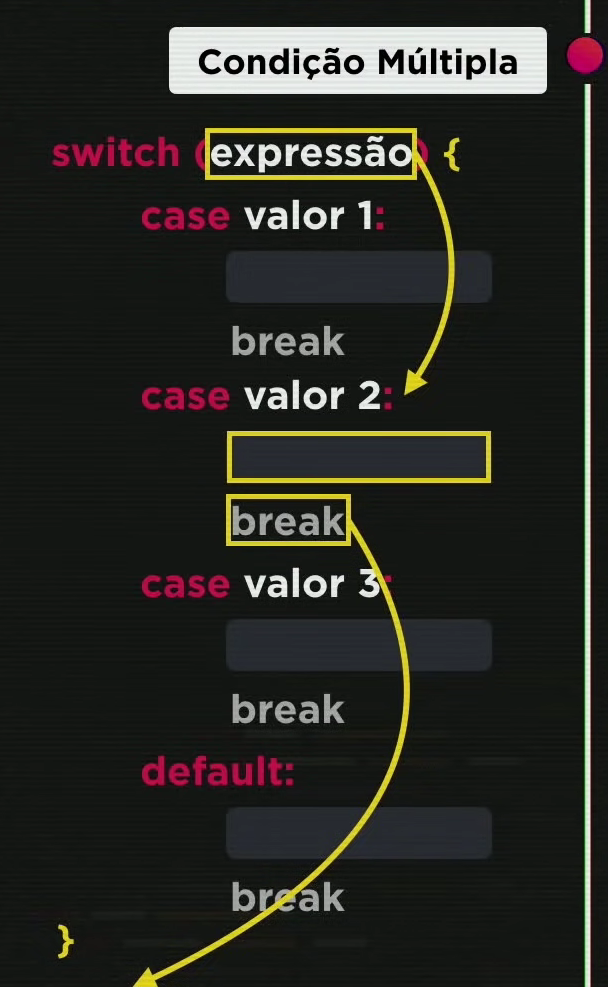
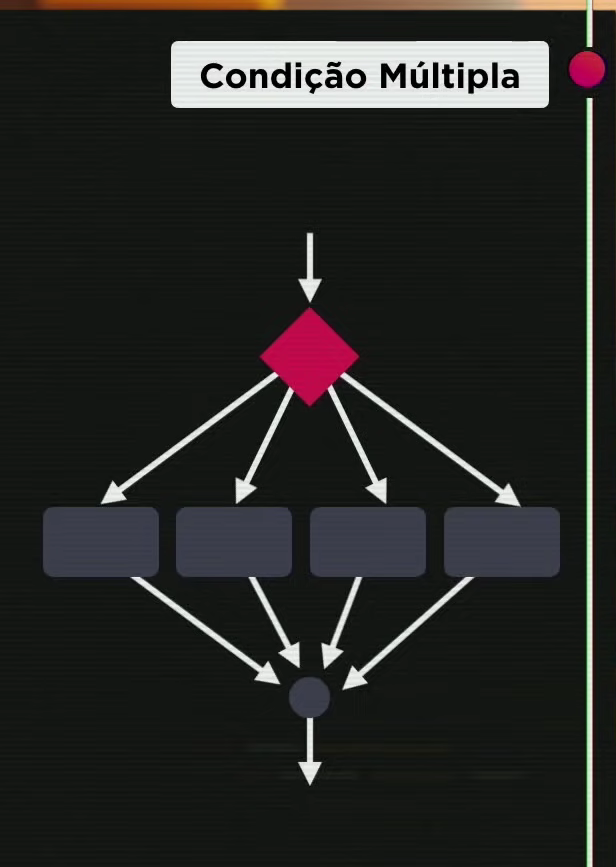
* Condição Simples: Possui apenas o IF
* Condição Composta: Possui IF e Else
* Condições Aninhadas (Uma condição dentro da outra)



(Para enconomizar linhas, utilizar If logo após o Else (Else If)

\*Pegar hora atual do sistema: Criar variável nova “new Date()”

* Condições Múltiplas:



Se a expressão for satisfeita com o valor 2 por exemplo, ele executa os comandos do valor 2, para no “break” e continua fora do bloco (chaves).

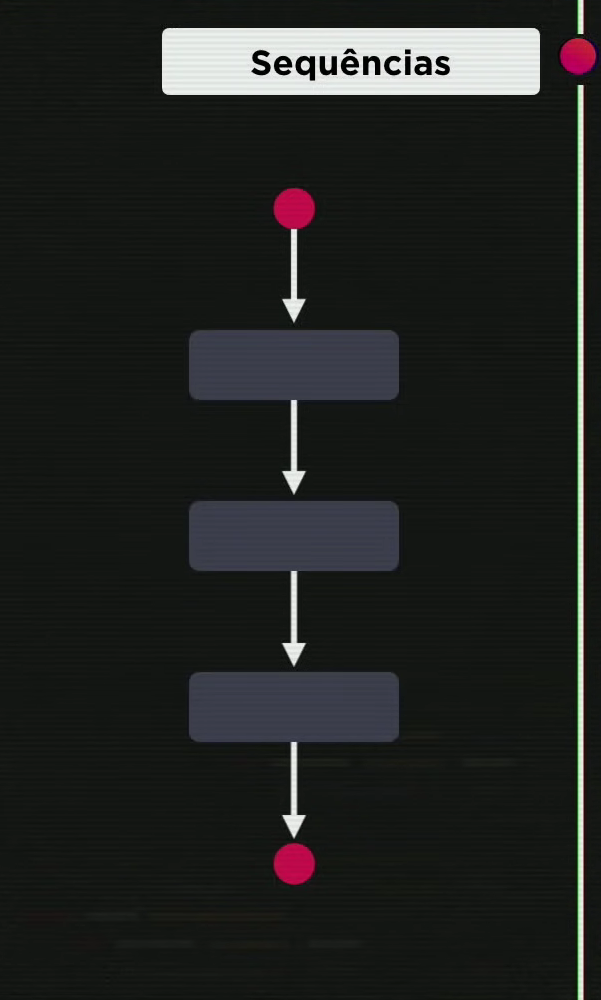
**Novos Comandos**

* **.getHours()**
* **img.src ‘**endereço’
* **.style.background**
* **.setAttribute(**‘src’ , ‘endereço’**)**
* **.appendChild()**
* **.createElement**

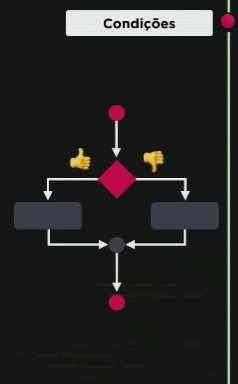
**REPETIÇÕES**

**Estruturas de Controle**

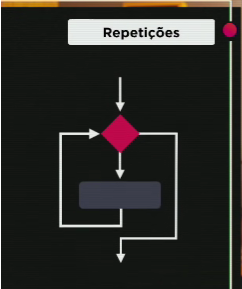
**Sequêncial**

****

**Condicionais**

****

**Estrutura de Repetição com teste Lógico no Início (While)**

****

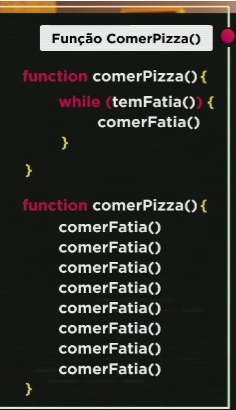
**While (**condição) {

Bloco

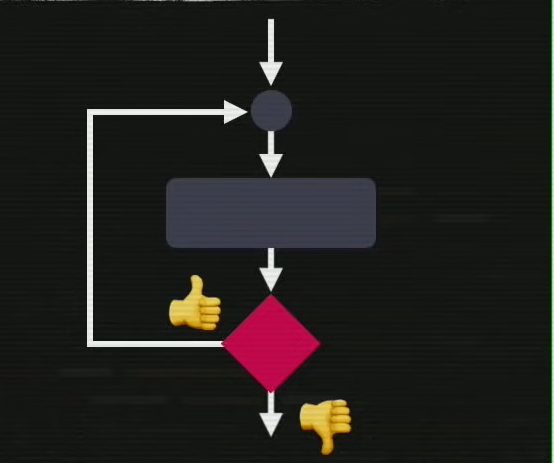
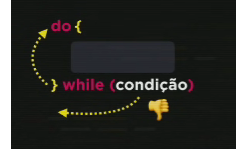
}

Se true, executa o que estiver dentro do bloco, se falsa, continua executando o que está fora

Ex.:



**Estrutura de Repetição com teste Lógico no Final (do / while)**

**Estrutura de Repetição com Variável de Controle (for)**

For (início; teste; incremento) {

Bloco

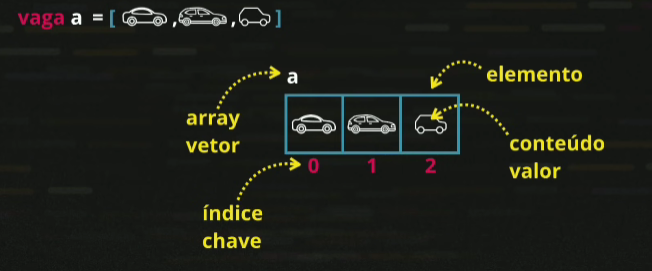
}

**Modo de depuração**

Executar > Iniciar a Depuração > definir a variável em ‘Inspeção’ > Iniciar a Depuração ‘F5’ > Contornar ‘F10’

Analisa cada processo do programa

**VARIÁVEIS COMPOSTAS**



Array é uma variável que possui vários elementos. Cada elemento é composto por seu valor e por sua chave de identificação.  
Let num = [5, 8, 4]  
Num [3] = 6 (coloque o valor 6 na posição 3(que não existia antes))  
num.push(7) (.push adiciona um valor por último)  
num.length (comprimento do array, não possui parênteses)  
num.sort() (coloca em ordem crescente)  
num.indexOf(8) = 2 (está na posição “2”)(quando ele não encontra, retorna o valor -1)

**Para mostrar os vetores sem colchete suar:**

for (let pos = 0; pos < num.length; pos ++){

console.log(num[pos])

}

**Nos ECMAScript mais modernos:**

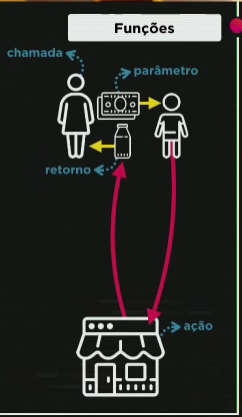
for (let pos **in** num) – para cada **pos** em num{

console.log(num[pos])

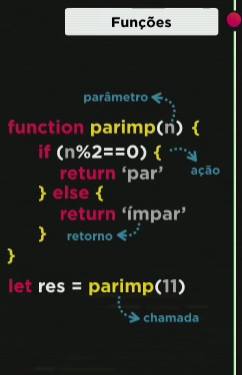
}

**FUNÇÕES**

* **Chamada**
* **Conjunto de Parâmetros**
* **Ação**
* **Retorno**



* **FUNÇÕES SÃO AÇÕES QUE SÃO EXECUTADAS ASSIM QUE SÃO CHAMADAS OU EM DECORRÊNCIA DE ALGUM EVENTO**
* **UMA FUNÇÃO PODE RECEBER PARÂMETROS E RETORNAR UM RESULTADO**



No caso do exemplo de uma function:

function soma (n1, n2) {

return n1 +n2

}

* Dá pra adicionar parâmetros adicionais. Ex.: Se na chamada da função, naõ existir alfum dos parâmetros, pode-se colocar n1=0 e n2=0 para considera 0 um valor.
* Número + Undefined = NaN