

# JavaScript

A linguagem JavaScript complementa o html, faz o mais elaborado que ele não consegue.

Stacktrace = mensagem de erro que aparece no Java.

SOLID = S, cada classe deve ter uma única responsabilidade; O, tem que poder acrescentar, não alterar o que já tem; L, o que funciona no geral precisa funcionar no específico; I, só deve ter o necessário; D, o geral não deve depender do específico

Vercel = site para colocar o código no ar, disponibilizar o site.

Síncrono = acontece em uma sequência certinha, na ordem que deve ser.

Assíncrono = acontecem várias coisas ao mesmo tempo, sem uma ordem exata.

Event Loop = é quem diz para o código se ele vai rodar agora ou mais tarde, manda pro call stack ou pro test queue.

Call Stack = é para onde vão os códigos que vão rodar agora, nesse momento.

Test Queue = é para onde vão os códigos que dependem de algo pra rodar, estão em espera.

Callback = funções que usam outras funções, que chamam de volta outras funções.

API - Interface de de Programação de Aplicações = funciona como um intermediário entre o cliente e o servidor.

## Código:

```
<meta charset="UTF-8">

<script> //Diz pro navegador que o conteúdo é javascript, não html.

    // É dinâmica, não precisa escrever se as variáveis são números, strings ou
    booleanos. Ele vê sozinho, e dá pra mudar o tipo se quiser.
    // Node JS é um aplicativo, instalação para rodar o JavaScript fora do navegador
    de um site, como no próprio editor de código.
    // Orientação a Objetos é um paradigma da programação que lida com as
    características (cor, peso, idade) e comportamentos (sorrir, amar, ler) dos objetos.
    // elemento._proto_ gera uma lista de todas as ações possíveis com aquele
    elemento.
    // Hoisting significa içar, é a capacidade de nomear elementos antes de declarar
    a função.
    // == é o igual do java, compara o conteúdo.
    // === compara o conteúdo e o formato.
    // != é o diferente do java, compara o conteúdo.
    // !== compara o conteúdo e o formato.
    // && é o E do java.
    // || é o OU do java.
    // !valor é a negação do valor, ele falso.
    // Linhas e colunas são entendidas no java. Se eu pedir apenas 10 colunas, o 11º
    caractere não vai ser aceito.
    // * e / transformam textos em números. Por exemplo "5" * 3 = 15.
    // incremento++ = o mesmo que "incremento = incremento + 1", é um atalho.
    // 0 e strings vazias ("") são consideradas falsas.
```

```

// Undefined aparece quando a variável não foi declarada.
// Null aparece quando a variável foi declarada como vazia.
// Control + C para o looping infinito.

tipo-elemento._proto_ //No navegador, abre uma lista com todos os métodos
disponíveis para aquele tipo de dado
console.log("texto"); //Exibe o texto no console.
document.write = `texto ${variavel} texto ${variavel}`; //Concatena textos e
variáveis de um modo muito mais simples.

document.write("texto"); //Cria um texto dentro do java.
document.write("<br>"); //Quebra a linha, como se desse enter.
document.write(3 + 3); //Escreve 6, os números fazem operações.
document.write("3" + "3"); //Escreve 33, os números são strings.
document.write("texto " + 3 + 3); //Escreve texto 33, concatena.
document.write("texto " + (3 + 3)); //Escreve texto 6, faz operações.

Math.round(numero); //Arredonda o resultado para o número mais próximo.
Math.ceil(numero); //Arredonda o resultado para o número mais alto.
Math.floor(numero); //Arredonda o resultado para o número mais baixo.
Math.trunc(numero); //Desconsidera as casas decimais, pega só o número inteiro.
Math.random(); //Gera um número aleatório, como 0.3415... Para um número inteiro,
basta multiplicar e arredondar.
Math.PI //O mesmo que 3.14, para usar em círculos.
nome.toUpperCase(); //Deixa todas as letras maiúsculas.

new Date(); //Mostra a data atual.
data.getUTCFullYear(); //Mostra o ano atual.
data.getUTCMonth(); //Mostra o mês atual.
data.getUTCDate(); //Mostra o dia atual.

parseInt("3"); //Transforma o texto em número, torna apto pra operações.
parseFloat("3.3"); //Transforma o texto em número decimal, torna apto pra
operações.

number(elemento); //Transforma o elemento em número, deixa de ser string.
string(elemento); //Transforma o elemento em string, deixa de ser número.
replace(/\D/g, "valor"); //Troca os dígitos pelo valor que eu quiser.
elemento.substring(numero_inicial, numero_final); //Exibe apenas um pedaço da
string.

isNaN(variavel); //Diz se é verdadeiro ou falso que a variável é um não número.
typeof elemento === tipo_dado; //Analisa se o elemento é do tipo pedido, seja
função, string, objeto...

elemento.value; //Pega o valor de algo.
elemento.validity.valid; //Diz a validade do elemento, se está de acordo com os
parâmetros estabelecidos (true) ou não (false).

```

```
    elemento.validity.valueMissing; //Diz o preenchimento do elemento se ele está
vazio (true) ou preenchido (false).

    elemento.onclick = acontecimento; //Faz o acontecimento quando o elemento é
clicado.

    elemento.onkeydown = acontecimento; //Faz o acontecimento quando a tecla está
pressionada.

    elemento.onkeyup = acontecimento; //Faz o acontecimento quando a tecla deixa de
ser pressionada.

    elemento.focusout = acontecimento; //Faz o acontecimento quando o mouse vai para
outro campo.

    elemento.addEventListener(parametro, acontecimento); //Pede pra "escutar" o
elemento, e quando houver o parâmetro gera o acontecimento.

    setTimeout(acontecimento, tempo_milissegundos); //Faz o acontecimento após o
tempo determinado.

    elemento.focus(); //Deixa o elemento em destaque.

    elemento.add[caracteristica]; //Adiciona uma característica no elemento.

    elemento.remove[caracteristica]; //Remove uma característica no elemento.

    return parametro; //Mostra o que foi pedido, como uma variável ou função.

    break; //Sai do looping.

    document.createElement("tr"); //Cria um novo elemento, nesse caso uma nova linha
numa tabela.

alert("texto"); //Cria um alerta, uma caixinha na página com o texto.

prompt("texto"); //Cria uma proposta, uma caixinha na página com a pergunta pro
usuário e espaço pra resposta, que é guardada como texto.

document.querySelector("elemento"); //Traz pro java um elemento do html, como
input ou botton. O elemento é .classe ou #id ou [data].

document.querySelectorAll("elemento"); //Traz todos os elementos pro java, uma
lista.

var nome = "parametro"; //Cria uma variável. Sempre que chamar pelo nome, vai
exibir o parâmetro dado. Pra chamar: document.write(nome);

nome = "parametro novo"; //Redefine a variável, dá a ela um novo valor.

const nome = "parametro"; //Cria uma variável que não pode ser mudada, precisa
dar o valor quando criar.

let nome = "parametro"; //Cria uma variável que pode ser mudada.

var nome = [3, 4, 5,]; //Cria uma variável com vários parâmetros, uma array.

nome.length; //Indica o número de elementos dentro do array.

nome.classList[numero]; //Acessa o elemento do número, dentro da lista.

nome.push(elemento); //Adiciona um elemento no fim do array.

nome.pop(); //Remove o último elemento do array.

nome.slice(primeiro_elemento, ultimo_elemento); //Pega um pedaço de um array. Se
passar só um parâmetro, vai começar nele e ir até o final do array.
```

```
        nome.splice(numero_primeiro_removido, quantidade_removidos,
elementos_substitutos); //Remove elementos do array e substitui por outros. Pode só
remover ou só adicionar, também.

primeiro_array.concat(segundo_array); //Cria um novo array concatenando ambos.

function nome() //Cria uma função que faz o que eu quiser, como pular linhas, por
exemplo. Pra chamar: nome();

    {acontecimento;}

function nome(parametro) //Cria uma função adaptável que faz o que eu quiser. Pra
chamar: nome(resultado);

    {acontecimento(parametro);}

function nome(parametro1, parametro2) //Cria uma função com 2 parâmetros, que faz
o que eu quiser. Pra chamar: nome(resultado);

    {acontecimento(parametro1 + parametro2);}

function nome(parametro = estepe) //Cria uma função com estepe. Caso não seja
declarado parâmetro, ele aparecerá.

    {acontecimento(parametro);}

nome = function (parametro) {acontecimento}; //Expressão de função, ela
simplificada. Não pode ser chamada anteriormente no código.

nome = parametro => {acontecimento}; //Arrow function, ela simplificada. Não pode
ser chamada anteriormente no código.

setInterval(nomefunção, 1000) //Chama a função de tempos em tempos, nesse caso
1000 milisegundos, ou 1 segundo.

parametro ? "acontecimento_true" : "acontecimento_false"; //O mesmo que o if, só
que mais simples. É o operador ternário.

if(parametro) //Faz o acontecimento, se o parâmetro for verdadeiro. Se for falso,
não acontece nada.

    {acontecimento}

else //Faz o acontecimento, se o parâmetro do if for falso.

    {acontecimento}

while(parametro) //Faz o acontecimento enquanto o parâmetro for verdadeiro. Se
for falso, para de acontecer.

    {acontecimento}

var nome = "valor" //Faz o acontecimento enquanto o parâmetro for verdadeiro. O
valor vai aumentando com o incremento, até tornar o parâmetro falso.
```

```
while(parametro)
    {acontecimento;
    valor++;}

for (variavel; parametro; incremento) //Faz o acontecimento enquanto o parâmetro
for verdadeiro. O incremento vai alterando a variável, até tornar o parâmetro falso.
    {acontecimento}

for (var nome in objeto) //Faz o acontecimento enquanto há dados no objeto,
percorre ele inteiro.
    {acontecimento}

nome_array.forEach(nome_funcao => {acontecimento}); //Faz um for simplificado.
nome_array.map(acontecimento); //Cria um acontecimento com cada um dos elementos
do array.
nome_array.filter(parametro); //Filtra o array de acordo com o parâmetro. Se o
parametro for verdadeiro, o filter exibe.
nome_array.reduce((acum, atual) => atual + acum, 0); //Soma todos os elementos de
um array.

var nome = //Cria um objeto, como um array só que com vários tipos de dados.
    {elemento_1:"valor_elemento_1",
    elemento_2:"valor_elemento_2",
    elemento_3:"valor_elemento_3"}

nome_objeto.nome_elemento; //Acessa um elemento no objeto.
nome_objeto.novo_elemento = valor; //Adiciona um elemento no objeto. Se já
existir algum com esse nome, o valor é atualizado.
delete nome_objeto.nome_elemento; //Deleta um elemento no objeto.
Object.keys(objeto); //Lista em um array todas as chaves, os elementos do objeto.
Object.values(objeto); //Lista em um array todos os valores do objeto.
Object.entries(objeto); //Lista em um array vários arrays com todos os elementos
e valores do objeto.
[...objeto[numero_indice].elemento]; //Pega os valores do elemento e coloca num
array, espalhadinho. Pode ser usado pra concatenação.

var nome = //Cria uma função dentro do objeto, nesse caso que incrementa o
parametro no valor.
    {elemento: valor,
    nome_acao: function(parametro)
        {this.elemento += parametro}}

function nome(elemento_1, elemento_2, elemento_3) //Cria um protótipo de objeto,
para ser preenchido posteriormente e várias vezes.
    {this.elemento_1 = elemento_1
```

```

    this.elemento_2 = elemento_2
    this.elemento_3 = elemento_3};

    function nome(elemento_novo) //Cria um objeto que aproveita os elementos do
protótipo, e adiciona novos.
    {nome_prototipo.call(this, elemento_1, elemento_2, elemento_3)
    this.elemento_novo = elemento_novo};

    var nome = Object.create(nome_prototipo) //Cria um objeto que aproveita os
elementos do protótipo, e adiciona novos.
    console.log(nome.nome_elemento("valor_elemento"));

    var nome = new nome_prototipo("valor_1", "valor_2", "valor_3"); //Preenche o
protótipo com as informações.
    nome_prototipo.prototype.nome_acao = {acontecimento}; //Adiciona algo no
protótipo, de fora.
    Object.setPrototypeOf(objeto_herdeiro, objeto_original); //Transfere para o
herdeiro as propriedades do original.

class nome //Cria uma classe, que é como um objeto só que melhorado.
{constructor(elemento_1, elemento_2, elemento_3)
    {this.elemento_1 = elemento_1
    this.elemento_2 = elemento_2
    this.elemento_3 = elemento_3}
    acao
    {acontecimento}};

class nome_classe_nova extends nome_classe_antiga //Cria uma nova classe que
aproveita os elementos da antiga, e adiciona novos.
{constructor(elemento_1, elemento_2, elemento_3, elemento_novo)
    {super(elemento_1, elemento_2, elemento_3)
    this.elemento_novo = elemento_novo}};

class nome //Cria uma classe com atributo privado. O elemento_1, depois de
declarado, não pode mais ser alterado.
    #elemento_1
    {constructor(elemento_1, elemento_2, elemento_3)
        {this.#elemento_1 = elemento_1
        this.elemento_2 = elemento_2
        this.elemento_3 = elemento_3}
    acao
    {acontecimento}};

class nome //Permite acessar o elemento privado, mostrá-lo em outras classes.
    #elemento_1

```

```

{constructor(elemento_1, elemento_2, elemento_3)
  {this.#elemento_1 = elemento_1
  this.elemento_2 = elemento_2
  this.elemento_3 = elemento_3}
get elemento_1()
  {return this.#elemento_1}};

```

`class nome` //Permite acessar o elemento privado, modificá-lo.

```

#elemento_1
{constructor(elemento_1, elemento_2, elemento_3)
  {this.#elemento_1 = elemento_1
  this.elemento_2 = elemento_2
  this.elemento_3 = elemento_3}
set elemento_1(novo_elemento_1)
  {this.#elemento_1 = novo_elemento_1}};

```

`nome_funcao.bind(nome_objeto);` //Fornece um contexto para o `this`, faz com que ele saiba a que o `this` se refere.

`const cep = input.value.replace(/\D/g, "")` //Transforma os dígitos em espaços vazios, ignora eles.

`fetch("url");` //Consulta os dados da API e retorna uma promise, que ou é resolve (deu certo) ou é reject (deu errado). Sempre colocar o `https://`.

`.then(response => response.json());` //Acessa a promise entregue ao `fetch` e converte ela de bytes pra json, transforma num objeto.

`.then(nome_funcao => acontecimento);` //Usa os dados do objeto.

`.then(nome_funcao => //A promise só retorna reject se o formato for inválido.` Caso o CEP não exista, ela retorna resolve. Aqui se não existir aparece a mensagem, senão é impresso no console.

```

{if (r.erro)
  {throw Error("mensagem") }
else
  {console.log(r)}});

```

`.catch(erro => acontecimento);` //Caso a promise retorne reject, gera o acontecimento.

`.finally(nome_funcao => acontecimento);` //Acontece no final de todo o `fetch`, tenha dado certo ou não.

`async function nome_funcao(parametro)` //Faz uma função assíncrona com parâmetro.

`{try` //Pede pra tentar encontrar o resultado. Se conseguir, acontece o `try`.

`{var nome_var = await fetch(`url${parametro}url`);` //Pega os dados da API, retornando uma promise (resolve se der certo, reject se não). Sempre colocar `https://`.

`var nome_convertido = await nome_var.json();` //Converte os dados da API de bytes pra json, em um objeto.

```

        if (nome_convertido.erro) //Caso não exista o pedido, mostra a mensagem.
            {throw Error("mensagem")}

        var nome_var = document.getElementById("nome_id"); //Pega um campo vazio,
nesse caso pelo id.

        nome_var.value = nome_convertido.nome_no_objeto; //E dá a ele o valor do
objeto, como por exemplo o nome de uma rua ou estado.

        acontecimento; //Caso esteja tudo certo, faz o acontecimento com eles,
convertidos.

        return nome_convertido} //E gera o retorno.

    catch (erro) //Caso não encontre o resultado, acontece o catch.
        {acontecimento}}

    let parametros = ["parametro_1", "parametro_2"]; //Passa os parâmetros que serão
usados na url.

    let varios_parametros = parametros.map(valores => nome_funcao(valores)); //Pega
as promisses retornadas dos parâmetros.

    Promise.all(varios_parametros).then(respostas => acontecimento); //E faz o
acontecimento com todas elas.

    nome_funcao();

    new HMLHttpRequest(); //Objeto que permite transferência de dados entre cliente e
servidor.

    http.open("acao", "http://local_servidor"); //Diz pra abrir o http e realizar a
ação com os dados do servidor.

    http.send(); //Diz pra enviar os dados pro http.

    http.onload = () => //Ao abrir a página, acontece o pedido com a resposta do
servidor.

    {const = JSON.parse(http.response); //Transforma a resposta do servidor em um
objeto.

        acontecimento}

    const nome_1 = (parametro_1, parametro_2) => //Cria uma função conectada a uma
API, com parâmetros.

    {return fetch ("url"), //Pega os dados da API, retornando uma promise (resolve se
der certo, reject se não). Sempre colocar https//.

        method: "POST", //Pede para enviar os dados na API.

        headers: {"Content-Type": "application/json"}, //Informa o tipo de dado que
está sendo enviado.

        body: JSON.stringify //Transforma o conteúdo que será enviado em string
            ({dado_1: parametro_1, dado_2: parametro_2}); //Diz o que vai ser
enviado.

        .then(resposta => //Retorna a resposta obtida.

            {return resposta.body})}

    const nome_2 = document.querySelector("[data-nome]"); //Conecta o botão com o
envio dos dados.

    nome_2.addEventListener("submit", (evento) => //Quando nome_2 for clicado,

```



acontece o pedido.

```
{const dado_1 = evento.target.querySelector("[data-dado_1]").value //Envia para a API o parâmetro passado pro dado 1.
```

```
const dado_2 = evento.target.querySelector("[data-dado_2]").value //Envia para a API o parâmetro passado pro dado 2.
```

```
local_servidor.nome_1(dado_1, dado_2)) //Conecta a função com o servidor.
```

```
const nome_1 = (id, parametro_1, parametro_2) => //Cria uma função conectada a uma API, com parâmetros.
```

```
{return fetch (`url${id}`), //Pega os dados da API, retornando uma promise (resolve se der certo, reject se não). Sempre colocar https//.
```

```
method: "PUT", //Pede para atualizar os dados na API.
```

```
headers: {"Content-Type": "application/json"}, //Informa o tipo de dado que está sendo enviado.
```

```
body: JSON.stringify //Transforma o conteúdo que será enviado em string
```

```
(({dado_1: parametro_1, dado_2: parametro_2})); //Diz o que vai ser enviado.
```

```
.then(resposta => //Retorna a resposta obtida.
```

```
{return resposta.json()}}}
```

```
const nome_2 = document.querySelector("[data-nome]"); //Conecta o botão com o envio dos dados.
```

```
const nome_1 = (id) => //Cria uma função conectada a uma API, com parâmetro.
```

```
{return fetch (`url${id}`), //Pega os dados da API, retornando uma promise (resolve se der certo, reject se não). Sempre colocar https//.
```

```
method: "DELETE"} //Pede para deletar os dados na API.
```

```
<script  
src="https://github.com/codermarcos/simple-mask-money/releases/download/v3.0.0/simple-mask-money.js"></script> //Para usar uma máscara monetária, precisa colocar o link do script embaixo do link do java, no html.
```

```
nome_mascara.setMask(input, {argumentos}) //No java, chama ela.
```

```
data.logradouro; //Valor da rua no retorno da API.
```

```
data.localidade; //Valor da cidade no retorno da API.
```

```
data.uf; //Valor do estado no retorno da API.
```

```
var nome_desenho = variavel_do_canvas.getContext("2d"); //Diz o tipo de desenho que será posto na tela em branco, no canvas.
```

```
nome_desenho.fillStyle = "cor"; //Diz a cor do desenho.
```

```
nome_desenho.strokeStyle = "cor"; //Diz a cor da borda.
```

```
nome_desenho.strokeRect(x_inicial, y_inicial, x_final, y_final); //Diz o espaço da borda, nesse caso de um retângulo.
```

```
nome_desenho.clearRect(x_inicial, y_inicial, x_final, y_final); //Limpa a tela, nesse caso retângulo, no espaço definido.
```

```
nome_desenho.fillRect(x_inicial, y_inicial, x_final, y_final); //Diz o espaço do
desenho, nesse caso retângulo, a partir do ponto 0x e 0y até 300x e 400y.
nome_desenho.arc(x_inicial, y_inicial, raio, angulo_inicial, angulo_final *
3.14); //Desenha um círculo. Os ângulos precisam estar em radiano.

nome_desenho.begin.Path(); //Inicia um desenho livre, linha por linha ou
circular.
nome_desenho.moveTo(x_inicial, y_inicial); //Diz o ponto inicial desse desenho.
nome_desenho.lineTo(x_seguinte, y_seguinte); //Diz o ponto seguinte desse
desenho.
nome_desenho.fill(); //Diz para preencher o espaço desse desenho.

parametro.pageX; // Diz o x da página.
parametro.pageY; // Diz o y da página.
parametro.offsetTop; //Diz respeito ao topo da tela, a parte branca.
parametro.offsetBottom; //Diz respeito ao embaixo da tela, a parte branca.
parametro.offsetRight; //Diz respeito ao lado direito da tela, a parte branca.
parametro.offsetLeft; //Diz respeito ao lado esquerdo da tela, a parte branca.

</script>
```