

## Aula 6 - Java Script

### Docupedia Export

Author:Gouveia Raissa (CtP/ETS)

Date:15-Mar-2024 17:23

## Table of Contents

<b>1</b>	<b>Como funciona</b>	<b>5</b>
<b>2</b>	<b>Vantagens</b>	<b>6</b>
<b>3</b>	<b>Desvantagens</b>	<b>7</b>
<b>4</b>	<b>Comentários</b>	<b>9</b>
<b>5</b>	<b>Tipos de Saída</b>	<b>10</b>
5.1	Console.log	10
5.2	innerHTML	10
5.3	Write	11
5.4	Alert	12
<b>6</b>	<b>Declarações</b>	<b>13</b>
6.1	Variáveis	13
<b>7</b>	<b>Tipos de dados</b>	<b>15</b>
<b>8</b>	<b>Operadores</b>	<b>16</b>
8.1	Operadores Aritméticos	16
8.2	Operadores de comparação	18
8.3	Operador Condicional (Ternário)	18
8.4	Operador Lógicos	19
<b>9</b>	<b>Funções</b>	<b>20</b>
<b>10</b>	<b>Objetos</b>	<b>21</b>
<b>11</b>	<b>Eventos</b>	<b>22</b>
<b>12</b>	<b>Arrays</b>	<b>24</b>
12.1	MÉTODOS	24
<b>13</b>	<b>IF E ELSE</b>	<b>27</b>

## 13.1 ELSE IF

28

**14 Laço de repetição For****31****15 Evento de Tempo****32****16 Classes****36****17 Manipulação de Datas****38**

## O que é o JavaScript?



É uma linguagem de programação interpretada estruturada, de script em alto nível com tipagem dinâmica fraca e multiparadigma.

Segundo a Mozilla Foundation, atual nome da antiga Netscape Communications Corporations, empresa responsável pela criação do JS, *"JavaScript é uma linguagem de programação, leve, interpretada, orientada a objetos, baseada em protótipos e em first-class functions (funções de primeira classe), mais conhecida como a linguagem de script da Internet."*

O JavaScript é uma das mais importantes tecnologias voltadas para o front-end e, unindo-se ao trio HTML, CSS e PHP, formam um grupo de linguagens que abrangem praticamente todas as exigências do desenvolvimento de uma página completa, dinâmica e com boa performance.

Alguns exemplos de sites que utilizam JS em seu front e back-end hoje em dia são **Ebay, LinkedIn e Yahoo**.

Mas o JS não se restringe mais apenas às páginas e aos navegadores, como foi durante vários anos: com o advento de diversos frameworks, APIs, melhorias e criação de centenas de funções, hoje já é possível utilizar JavaScript em aplicativos mobile, softwares para desktop e até mesmo em back-end.

# 1 Como funciona

É uma linguagem de programação client-side, ou seja, é executada do lado do usuário, mais especificamente pelo navegador utilizado por este usuário. Em outras palavras, isso significa que todas as suas ações são processadas na máquina de quem as utiliza, sem a necessidade de enviá-las a nenhum outro ambiente.

Como nada é enviado a nenhum servidor externo para processamento, as respostas são imediatas.

## 2 Vantagens

- Versatilidade da linguagem;
- Rapidez de leitura e, portanto, rapidez de execução;
- Sintaxe acessível;
- Não precisa ser compilada — ou seja, os navegadores são capazes de interpretá-la por conta própria;
- Ótima linguagem para iniciantes em programação;
- Compatível com uma grande variedade de navegadores e plataformas;
- Código leve;
- Curva de aprendizagem rápida;
- Grande comunidade ao redor do mundo.

### 3 Desvantagens

Como tudo tem dois lados, o JS também conta com algumas desvantagens.

Entre elas, estão:

- Poucos recursos voltados à segurança;
- Pode conter brechas para a execução de ações maliciosas;
- O Node.js está propenso a mais vazamento de memória em processos de execução longa.

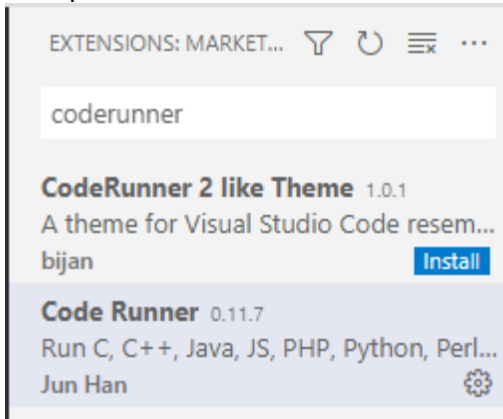
Fundamentos do JS

Ferramentas:



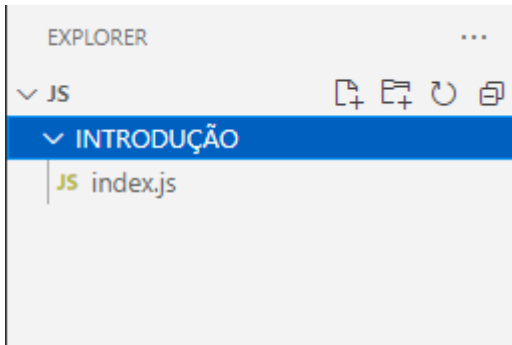
Para iniciarmos abra o VS Code.

Verifique se a extensão CodeRunner está instalada:



Caso não esteja instale para que seja possível executar o código diretamente com node.js.

Abra ou crie uma pasta para o curso, abra esta no VS Code, dentro dela uma crie um arquivo index.js





## 4 Comentários

```
1 //Duas Barras para comentar uma linha
2
3 /* Barra
4  e asterisco
5  para bloco de comentário*/
```


---

## 5 Tipos de Saída

### 5.1 Console.log

É uma função para saída de dados no console, muito utilizada para debugs, não interfere na execução do conteúdo no navegador.

```
console.log('Curso JS');  
console.log("Curso JS");  
console.log(`Curso JS`);  
console.log(55,48.9,'Curso JS');
```

Para executar o código clique no ícone  ou Ctrl + Alt + N

```
[Running] node "u:\JS\INTRODUÇÃO\index.js"
```

```
Curso JS
```

```
Curso JS
```

```
Curso JS
```

```
55 48.9 Curso JS
```

```
[Done] exited with code=0 in 0.494 seconds
```

### 5.2 innerHTML

É uma função para saída de dados na página HTML.

index.html

script.js

```
document.getElementById("texto").innerHTML = "Meu texto <b>JS</b>!";
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
  <p id="texto"></p>

  <script src="./script.js"></script>
</body>
</html>
```

## 5.3 Write

Função para saída de dados na página HTML, geralmente utilizada para testes.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
  <p>
    <script>
      document.write('Texto com document.write');
    </script>
  </p>

</body>
</html>
```

## 5.4 Alert

Função para saída de dados como um pop-up.

```
alert('Meu alerta!');
```

dex.html

**Essa página diz**

Meu aleta!

OK

## 6 Declarações

Em JavaScript, uma declaração é uma instrução que define uma ação a ser realizada. Existem vários tipos de declarações em JavaScript, incluindo declarações de variáveis, declarações de funções, declarações de loops, declarações de condições, entre outros.

### 6.1 Variáveis

**var:**

- Antes do ECMAScript 6 (ES6), `var` era a única maneira de declarar variáveis em JavaScript.
- As variáveis declaradas com `var` têm escopo de função ou global. Isso significa que elas são acessíveis dentro da função na qual foram declaradas ou globalmente, fora de qualquer função.
- As variáveis `var` podem ser redeclaradas e atualizadas.
- Atualmente é recomendado utilizar `let` ou `const` no lugar de `var`, pois ao **usar var**, as variáveis têm escopo de função, o que significa que elas são visíveis em todo o escopo da função em que foram declaradas, independentemente de blocos condicionais ou loops. Isso pode levar a bugs quando você espera que uma variável seja local a um bloco específico.

```
var x = 5;  
var x = 10; // Reatribuição permitida  
console.log(x); // Saída: 10
```

**let:**

- Introduzido no ECMAScript 6 (ES6), `let` permite a declaração de variáveis com escopo de bloco.
- As variáveis declaradas com `let` têm escopo de bloco, o que significa que são acessíveis apenas dentro do bloco no qual foram declaradas.
- As variáveis `let` podem ser atualizadas, mas não redeclaradas no mesmo escopo.

```
let y = 5;  
y = 10; // Reatribuição permitida  
console.log(y); // Saída: 10  
let y = 15; // Erro: redeclaração não permitida
```

**const:**

- Assim como `let`, `const` também foi introduzido no ECMAScript 6 (ES6). Ele é usado para declarar constantes.
- As variáveis declaradas com `const` têm escopo de bloco e não podem ser reatribuídas nem redeclaradas.
- No entanto, se a constante é um objeto ou um array, suas propriedades ou elementos podem ser modificados.

```
const z = 5;  
z = 10; // Erro: reatribuição não permitida  
  
const w = {value: 5};  
w.value = 10; // Atribuição de propriedade permitida  
console.log(w.value); // Saída: 10
```

---

## 7 Tipos de dados



## 8 Operadores

Os operadores JavaScript são usados para atribuir valores, comparar valores, executar operações aritméticas e muito mais.

### 8.1 Operadores Aritméticos

Soma:

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = valor1 + valor2;  
console.log(total)
```

Subtração:

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = valor1 - valor2;  
console.log(total)
```

Multiplicação:

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = valor1 * valor2;  
console.log(total)
```

Divisão:

+	Soma valores
-	Subtrai valores
*	Multiplica valores
/	Divide valores
%	Resto da divisão
++	Incremento de 1
--	Decremento de 1

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\tempCodeRunnerFile.js"  
7
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
3
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\tempCodeRunnerFile.js"  
10
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
2.5
```



```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = valor1 / valor2;  
console.log(total)
```

Incremento: Adiciona +1.

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = ++valor1;  
console.log(total)
```

Decremento: Diminui -1.

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = --valor1;  
console.log(total)
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
6
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
4
```

## 8.2 Operadores de comparação

<code>==</code>	Igual
<code>!=</code>	Diferente
<code>&lt;&gt;</code>	Diferente
<code>===</code>	Idêntico
<code>!==</code>	Não idêntico
<code>&lt;</code>	Menor que
<code>&gt;</code>	Maior que
<code>&lt;=</code>	Menor ou igual
<code>&gt;=</code>	Maior ou igual

```
let valor1, valor2, total;  
valor1 = 5;  
valor2 = 2;  
  
total = (valor1 == valor2); //retorna true ou false  
console.log(total)
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
false
```

## 8.3 Operador Condicional (Ternário)

`condição ? expressão_se_verdadeira : expressão_se_falsa;`

Ou seja, se a condição for verdadeira, a expressão antes do `:` é executada, caso contrário, a expressão após o `:` é executada

```
let idade, eleitor;  
idade = 15;  
eleitor = (idade < 18) ? "Não, Eleitor" : "Sim, eleitor";  
console.log(eleitor);
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
Não, Eleitor
```

## 8.4 Operador Lógicos

São utilizados quando temos mais de uma condição a ser seguida.

	or
&&	and
!	not

```
let idade, resultado1, resultado2, resultado3;  
idade = 68;  
  
resultado1= ( idade > 60 && idade < 70);  
resultado2 = (idade === 65 || idade === 72);  
resultado3= (idade !== 68);  
  
console.log("and: ", resultado1, "\nor:", resultado2, "\nnot:", resultado3);
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
and: true  
or: false  
not: false
```

## 9 Funções

Funções em JavaScript são blocos de código reutilizáveis que podem ser chamados/executados em diferentes partes de um programa. Elas permitem encapsular lógica e dados, promovendo a modularidade e a organização do código. As funções podem receber parâmetros e retornar valores.

Uma função só é executada quando "algo" a chama.

```
function soma(valor1, valor2){  
  let total;  
  total = valor1 + valor2;  
  return total;  
}  
  
console.log(soma(10,20));
```

Podemos fazer também, funções sem parâmetros .

```
function alertaHello(){  
  alert('Hello World');  
}  
  
alertaHello();
```

tml

Essa página diz

Hello World

OK

## 10 Objetos

Objetos são basicamente variáveis com muitos valores dentro.

EX: const carro = {marca: "ford", modelo: "ka", ano: 2009}

Os valores dentro de um objeto são chamados de propriedades.

Objetos também podem ser métodos. Um método é uma função colocada dentro de uma propriedade.

```
const carro = {  
  marca: "ford",  
  modelo: "ka",  
  ano: 2009,  
  placa: "ABC-1234",  
  buzina: function(){ alert('BIBIBI') },  
  completo: function(){  
    return "A marca é ", this.marca;  
  }  
};  
  
console.log("Objeto: ", carro);  
console.log("\nPlaca: ", carro.placa);  
console.log("\nCompleto:", carro.completo());
```

```
[Running] node "c:\Users\GORICT\Desktop\aula js\script.js"
```

```
Objeto: {  
  marca: 'ford',  
  modelo: 'ka',  
  ano: 2009,  
  placa: 'ABC-1234',  
  buzina: [Function: buzina],  
  completo: [Function: completo]  
}
```

```
Placa: ABC-1234
```

```
Completo: ford
```

# 11 Eventos

Eventos são ações disparadas pela interação dos usuários na página. É o correto manejo desses eventos que tornam as páginas interativas e dinâmicas. Existem muitos eventos, mas vamos ver os mais utilizados:

- **ondblclick** -> Disparado quando ocorre um clique duplo.
- **onmouseover** -> Disparado quando o mouse passa por cima do elemento.
- **onmouseout** -> Disparado quando o mouse deixa a área do elemento.
- **onmousemove** -> Disparado quando o mouse é movido sobre o elemento.
- **onmousedown** -> Disparado quando o botão do mouse é pressionado.
- **onmouseup** -> Disparado quando o botão do mouse é solto.
- **onfocus** -> Disparado quando o elemento recebe o foco (válido para elementos como <input>).
- **onchange** -> Disparado quando ocorre uma alteração no conteúdo do elemento.
- **onblur** -> Disparado quando o elemento perde o foco.
- **onkeydown** -> Disparado quando uma tecla do teclado é pressionada.
- **onkeypress** -> Disparado quando uma tecla do teclado é pressionada e solta.
- **onkeyup** -> Disparado quando uma tecla do teclado é solta após ser pressionada.
- **onload** -> Disparado quando a página terminou de ser carregada (normalmente utilizado no elemento <body>).
- **onresize** -> Disparado quando a janela do navegador é redimensionada

index.html

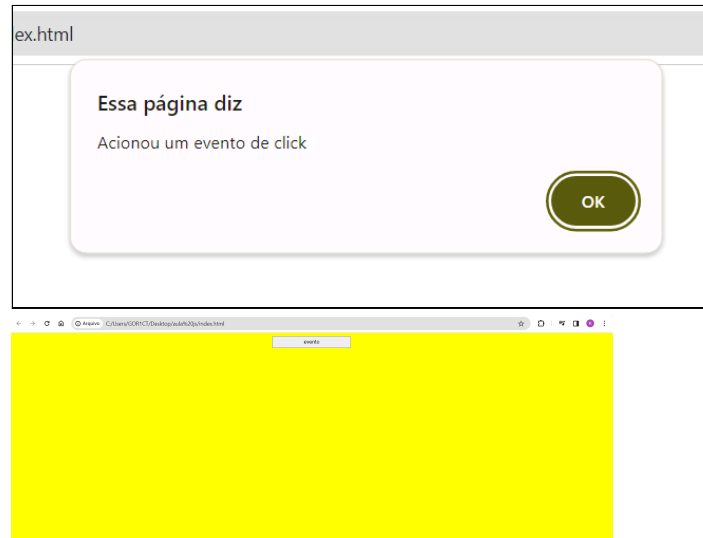
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
  <button onclick="eventoClick()" > evento </button>
  <script src="./script.js"></script>
</body>
</html>
```



E ao clicar no botão, ele acionará a nossa função 'eventoClick'.

## script.js

```
function eventoClick(){  
    alert('Acionou um evento de click');  
    document.body.style.backgroundColor = "yellow";  
}
```



## 12 Arrays

Os arrays, são usados para armazenar vários alores em uma única variável. Diferente dos objetos, que funcionam com propriedades ou 'nomes que você dá para para os itens dentro dele, os arrays não possuem propriedades. O item dentro dele é encontrado pela posição.

Ex: const lista = ["arroz", "feijão", "macarrão", "leite"];

A lista[0] (lista na posição 0) vai conter o valor "arroz".

A lista[1] (lista na posição 1) vai conter o valor "feijão".

E assim por diante.

### Criando um array

```
const mercado = [  
  "arroz",  
  "feijão",  
  "macarrão",  
  "leite"  
];
```

### Adicionando novo item

```
mercado[4] = "batata";  
mercado.push("Óleo");
```

### Pegando o valor da posição 3, que é "leite"

```
console.log(mercado[3]);
```

### Acessando o conjunto completo

```
console.log(mercado);
```

### Retornando quantos itens há no array

```
console.log(mercado.length);
```

### Mostrando o último item do array

```
console.log(mercado[mercado.length - 1]);
```

## 12.1 MÉTODOS



**Join: trocamos o separador do array**

```
console.log(mercado.join(" * "));
```

**Pop: remove o último item do array**

```
mercado.pop();  
console.log(mercado);
```

**Shift: remove o primeiro item do array**

```
mercado.shift();  
console.log(mercado);
```

**unShift: adiciona um item na primeira posição do array**

```
mercado.unshift("limão");  
console.log(mercado);
```

**Splice: substitui ou adiciona um item em uma posição específica**

**Parâmetros:** posição, quantidade de itens a substituir, novo item(s)

```
mercado.splice(3, 1, "cenoura", "xuxu");  
console.log(mercado);
```

**Concatenando arrays**

```
const lista1 = ["A", "B", "C", "D"];  
const lista2 = ["E", "F", "G", "H"];  
const superLista = lista1.concat(lista2);  
console.log(superLista);
```

**Slice: mostra uma parte específica do array**

```
const legumes = mercado.slice(3, 6);  
console.log(legumes);
```

**Sort: Ordenação alfabética**

```
console.log(mercado.sort());
```

**Reverse: Ordem reversa**

```
console.log(mercado.reverse());
```

### Ordenação numérica

```
const numeros = [40, 50, 54, 30, 11, 20];  
console.log(numeros.sort(function(a, b) { return a - b; }));
```

### Ordenação numérica decrescente

```
const numeros = [40, 50, 54, 30, 11, 20];  
console.log(numeros.sort(function(a, b) { return b - a; }));
```

### Maior número

```
function Maior(array) {  
  return Math.max.apply(null, array);  
}  
console.log(Maior(numeros));
```

### Menor número

```
function Menor(array) {  
  return Math.min.apply(null, array);  
}  
console.log(Menor(numeros));
```

## 13 IF E ELSE

Para entendermos a utilização do if/else no JavaScript, vamos criar um sistema de interruptor de luz.

script.js

```
const button = document.getElementById("button");
let interruptor = "on";

function lampada(){
  if (interruptor === "on"){
    document.body.style.backgroundColor = "yellow";
    interruptor = "off";
    button.innerText = "Apagar Luz";
  }else{
    document.body.style.backgroundColor = "black";
    interruptor = "on";
    button.innerText = "Acender Luz";
  }
}
```

index.html



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
  body{
    text-align: center;
    background-color: black;
  }
  button{
    width: 200px;
    height: 30px;
    background-color: aqua;
  }
</style>
</head>
<body>
  <button id="button" onclick="lampada()" > Acender Luz </button>

  <script src="./script.js"></script>
</body>
</html>
```

## 13.1 ELSE IF

Nesse exemplo vamos fazer uma validação para o input de nome.  
script.js

```
function validar(){  
  let nome = document.getElementById("nome").value;  
  let p = document.getElementById("teste");  
  
  if(nome == "" || nome == null){  
    p.innerText = "O campo nome não pode ser vazio!";  
    p.style.color="red";  
  }else if(nome.length < 3){  
    p.innerText = "Insira um nome válido!";  
    p.style.color="orange";  
  }else{  
    p.innerText = "Enviado com sucesso!";  
    p.style.color="green";  
  }  
}
```

index.html

---

O campo nome não pode ser vazio!

---

Insira um nome válido!

---

Enviado com sucesso!

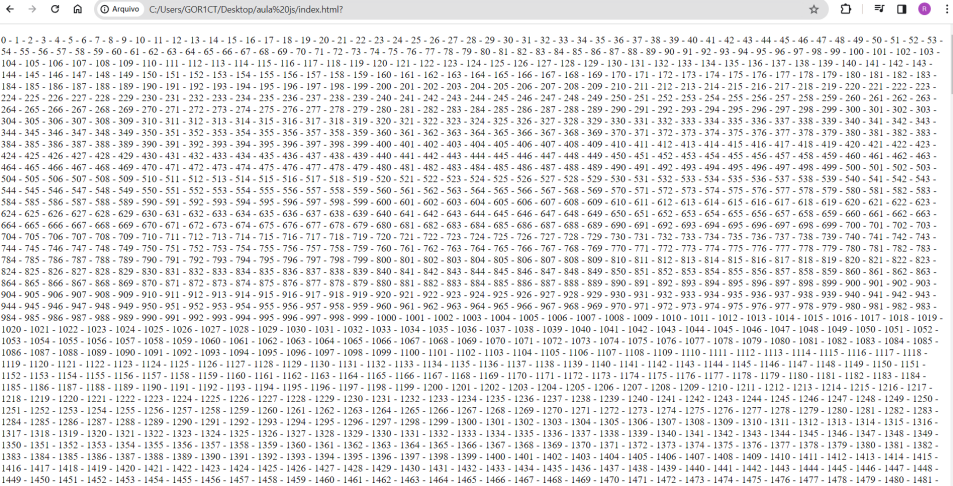
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
  body{
    text-align: center;
  }
</style>
</head>
<body>
  <p id="teste"></p>
  <input type="text" id="nome" placeholder="nome"/>
  <button type="submit" onclick="validar()">Enviar</button>

  <script src="./script.js"></script>
</body>
</html>
```

# 14 Laço de repetição For

Laços oferecem um jeito rápido e fácil de executar uma ação repetidas vezes.

```
for (let i = 0; i < 10000; i++){
  document.getElementById("teste").innerHTML += i + " - ";
}
```



## 15 Evento de Tempo

Os eventos de tempo permitem a execução do código em intervalos de tempo especificados. Esses intervalos de tempo são chamados de eventos de cronometragem. Os dois métodos-chave para usar com JavaScript são:

### **setTimeout(function, tempo em milisegundos)**

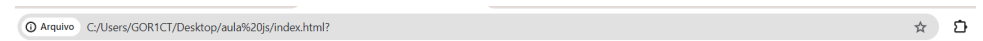
→ Executa uma função, depois de esperar um número especificado de milisegundos.

### **setInterval(function, milisegundos)**

→ É o mesmo que setTimeout(), mas repete a execução da função continuamente.

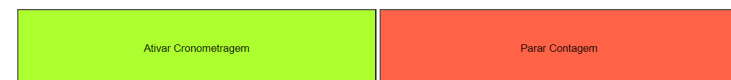
Vamos criar um cronômetro de exemplo para melhor compreensão dessas funções.

index.html



### **Controle de Tempo**

0





```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
  body{
    text-align: center;
  }
  .start{
    width: 30%;
    height: 100px;
    background: greenyellow;
  }
  .stop{
    width: 30%;
    height: 100px;
    background: tomato;
  }
</style>
</head>
<body>
  <h1>Controle de Tempo</h1>
  <h2 id="tempo">0</h2>
  <button onclick="ativarContagem()" class="start">Ativar Cronometragem</
button>
  <button onclick="pararContagem()" class="stop">Parar Contagem</button>

  <script src="./script.js"></script>
</body>
</html>
```

scrip.js

**SetTimeout(function, tempo em milisegundos)**

```
function ativarContagem(){
  document.getElementById('tempo').innerHTML = "Começou a contar!";
  //Ativa a função apenas uma vez após o tempo determinado
  tempo = setTimeout(function(){
    document.getElementById('tempo').innerHTML = "Executou o setTimeout";
  }, 5000);
}

//Para a função antes do tempo determinado
function pararContagem(){
  clearTimeout(tempo);
  document.getElementById('tempo').innerHTML = "Parou a contagem";
}
```

**SetInterval(function, milisegundos)**

R1CT/Desktop/aula%20js/index.html?

**Controle de Tempo****Começou a contar!**

Ativar Cronometragem

Parar Contagem

/GOR1CT/Desktop/aula%20js/index.html?

**Controle de Tempo****Executou o setTimeout**

Ativar Cronometragem

Parar Contagem

sers/GOR1CT/Desktop/aula%20js/index.html?

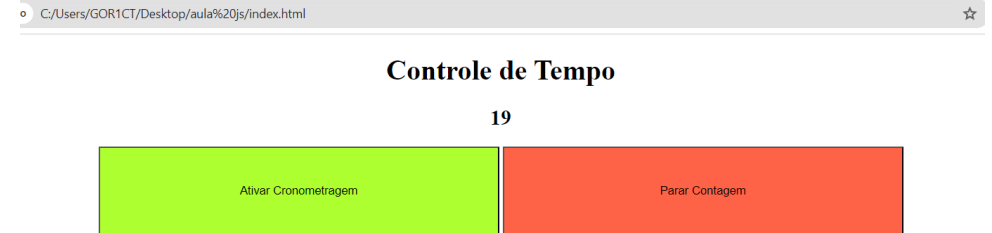
**Controle de Tempo****Parou a contagem**

Ativar Cronometragem

Parar Contagem

```
function ativarContagem(){
  document.getElementById('tempo').innerHTML = "Começou a contar!";
  //Ativa a função apenas uma vez após o tempo determinado
  tempo = setTimeout(function(){
    document.getElementById('tempo').innerHTML = "Executou o setTimeout";
  }, 5000);
}

//Para a função antes do tempo determinado
function pararContagem(){
  clearTimeout(tempo);
  document.getElementById('tempo').innerHTML = "Parou a contagem";
}
```



## 16 Classes

Em 2015 foi introduzido no JavaScript as classes. As classes são um conceito antigo em programação e várias linguagens utilizam elas. Mas no JavaScript , isso é relativamente novo.

Basicamente, as classes são "funções especiais" para criação de objetos.

Assim como uma fábrica da vida real precisa das máquinas para construir os objetos, as classes usam um método chamado constructor() para criar objetos.

Por convenção, sempre iremos dar o nome da nossa classe com a primeira letra maiúscula.

```
//Aqui estamos criando a nossa classe e definindo as suas propriedades e parâmetros
```

```
class Carro{  
  constructor(marca, modelo, ano ){  
    this.marca = marca;  
    this.modelo = modelo;  
    this.ano = ano;  
  }  
}
```

```
//Aqui estamos criando o nosso objeto instaciando a nossa classe Carro  
const uno = new Carro("Fiat", "Uno", 2001);  
console.log(uno);
```

PROBLEMS

17

OUTPUT

TERMINAL

PORTS

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"  
Carro { marca: 'Fiat', modelo: 'Uno', ano: 2001 }
```

Podemos criar "ações" para a nossa classe

```
class Carro{
  constructor(marca, modelo, ano ){
    this.marca = marca;
    this.modelo = modelo;
    this.ano = ano;
  }
  buzina(){
    return this.modelo + " buzinou: BIBIBIBI";
  }
}

const gol = new Carro("Volkswagen", "Gol", 2015);
console.log(gol.buzina());
```

```
[Running] node "c:\Users\GOR1CT\Desktop\aula js\script.js"
Gol buzinou: BIBIBIBI
```

# 17 Manipulação de Datas

Comando base para pegar a data

```
let data = new Date();  
console.log(data);
```

Pegar o ano atual com 4 dígitos

```
let ano = data.getFullYear();  
console.log(ano);
```

Pegar o mês atual - De 0 até 11, sendo 0 Janeiro e 11 Dezembro

```
let mes = data.getMonth();  
console.log(mes);
```

Pegar o mês atual pelo nome

```
const meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho", "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"];  
let mesEscrito = meses[data.getMonth()];  
console.log(mesEscrito);
```

Pegar o dia do mês - 1 até 31

```
let diaMes = data.getDate();  
console.log(diaMes);
```

Pegar o dia da semana - De 0 até 6, sendo 0 Domingo e 6 Sábado

```
let diaSemana = data.getDay();  
console.log(diaSemana);
```

Pegar o dia da semana pelo nome

```
const semana = ["Domingo", "Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"];  
let diaSemanaEscrito = semana[data.getDay()];  
console.log(diaSemanaEscrito);
```

Pegar a hora - De 0 até 23

```
let hora= data.getHours();
```

Pegar minutos - De 0 até 59

```
let minutos = data.getMinutes();
```

Pegar segundos - De 0 até 59

```
let segundos = data.getSeconds();
```

Pegar milisegundos - De 0 até 999

```
let milisegundos = data.getMilliseconds();
```

Pegar data no padrão brasileiro - DIA/MÊS/ANO

```
let dataBR = data.toLocaleDateString('pt-BR', {dateStyle: 'short'});  
console.log(dataBR);
```