⏱️ `setInterval()` e `setTimeout()`

📌 Introdução aos Timers no JavaScript

JavaScript permite executar código **com atraso** ou **repetidamente** usando **timers** controlados pelo navegador (`window`).

Os dois principais são:

- `setInterval()` → execução repetida
- `setTimeout()` → execução única após um tempo

🔁 `setInterval()`

📖 O que é

`setInterval()` executa uma função **repetidamente**, respeitando um intervalo de tempo definido em milissegundos.

⏱️ 1 segundo = 1000 milissegundos

A execução **continua indefinidamente** até que:

- `clearInterval()` seja chamado, ou
- a página seja fechada.

🧠 Sintaxe

```
setInterval(funcao, tempo, param1, param2, ...)
```

📌 Parâmetros

- **funcao** (obrigatório): função que será executada
- **tempo** (obrigatório): intervalo entre execuções (em ms)
- **param1, param2...** (opcional): argumentos passados à função
  ⚠️ Parâmetros extras **não funcionam no IE9 ou anterior**

🔄 Valor de Retorno

- Retorna um **ID numérico**
- Esse ID é usado para **parar o intervalo**

🛑 Como parar um `setInterval`

```
const id = setInterval(funcao, 1000);
clearInterval(id);
```

🧪 Exemplos de `setInterval()`

⏰ Relógio digital

```
setInterval(atualizarHora, 1000);

function atualizarHora() {
  const agora = new Date();
  document.getElementById("relogio").innerHTML =
    agora.toLocaleTimeString();
}
```

📌 Atualiza o horário **a cada segundo**

🛑 Parar o relógio

```
const intervalo = setInterval(atualizarHora, 1000);

function parar() {
  clearInterval(intervalo);
}
```

📊 Barra de progresso animada

```
function iniciar() {
  let largura = 0;
  const barra = document.getElementById("barra");
  const id = setInterval(animar, 10);

  function animar() {
    if (largura === 100) {
      clearInterval(id);
    } else {
      largura++;
      barra.style.width = largura + "%";
    }
  }
}
```

📌 Executa até atingir 100%

🎨 Alternar cores do fundo

```
const intervalo = setInterval(trocarCor, 500);

function trocarCor() {
  document.body.style.backgroundColor =
```

```
    document.body.style.backgroundColor === "yellow"
      ? "pink"
      : "yellow";
}

function pararCores() {
  clearInterval(intervalo);
}
```

📥 Passando parâmetros

❌ Forma incompatível com navegadores antigos:

```
setInterval(minhaFuncao, 2000, "A", "B");
```

☑️ Forma segura:

```
setInterval(function () {
  minhaFuncao("A", "B");
}, 2000);
```

⏳ `setTimeout()`

📖 O que é

`setTimeout()` executa uma função **uma única vez**, após um tempo definido.

📌 Ideal para:

- atrasos
- animações simples
- mensagens temporárias

🧠 Sintaxe

```
setTimeout(funcao, tempo, param1, param2, ...)
```

📌 Parâmetros

- **funcao**: função a ser executada
- **tempo**: atraso em milissegundos (padrão = 0)
- **param1, param2...**: argumentos opcionais

🔄 Valor de Retorno

- Retorna um **ID**
- Esse ID permite cancelar o timeout

🛑 Cancelar um `setTimeout`

```
const id = setTimeout(funcao, 3000);
clearTimeout(id);
```

🧪 Exemplos de `setTimeout()`

⏰ Executar após 5 segundos

```
setTimeout(saudacao, 5000);

function saudacao() {
  alert("Olá!");
}
```

❌ Cancelar antes de executar

```
const tempo = setTimeout(saudacao, 5000);

function cancelar() {
  clearTimeout(tempo);
}
```

📝 Texto com atraso progressivo

```
setTimeout(() => campo.value = "2 segundos", 2000);
setTimeout(() => campo.value = "4 segundos", 4000);
setTimeout(() => campo.value = "6 segundos", 6000);
```

🪟 Abrir e fechar janela automaticamente

```
const janela = window.open("", "", "width=200,height=100");
setTimeout(() => janela.close(), 3000);
```

🔁 Relógio usando `setTimeout` (recursivo)

```
function iniciarRelogio() {
  const agora = new Date();
  document.getElementById("hora").innerHTML =
    agora.toLocaleTimeString();

  setTimeout(iniciarRelogio, 1000);
}
```

📌 Simula um `setInterval`, mas com **mais controle**

⚖️ Comparação Final

Característica	setInterval	setTimeout
Execução	Repetida	Única
Cancela com	clearInterval	clearTimeout
Ideal para	Loops temporizados	Atrasos
Risco de loop infinito	Sim	Não

☑ Dicas Importantes

- Sempre **guarde o ID** do timer
- Evite múltiplos `setInterval` sem controle
- Para loops complexos → prefira `setTimeout` recursivo
- Em projetos grandes, timers mal usados causam **memory leak**