

# JavaScript

repetições

**JS**  
JavaScript

parte 7

# aprendendo Javascript

## agenda

até aqui já aprendemos muito sobre a estrutura da linguagem. Aprendemos sobre tipos primitivos, escopo e como criar condições específicas para o nosso algoritmo. Agora é aprender a lidar com instruções repetitivas em JavaScript.




# quando necessito repetir como executar N vezes uma tarefa ?

não será incomum quando você escrever algum programa para computador, se deparar com uma necessidade de fazer uma mesmíssima ação um conjunto finito de vezes. Essencialmente em quase todas as situações teremos algum desafio próximo a este.

por exemplo, você deve escrever um um programa que calcule os 5 primeiros números elevado ao cubo. em matemática, um número elevado ao cubo é a operação na qual ele é multiplicado por si mesmo 3 vezes. O cubo de **6** é  **$6 * 6 * 6 = 216$** .

não seria digamos um pecado se um desenvolvedor escolhesse fazer um programa para resolver este problema segundo o código a seguir.



```
// 5 primeiros números ao cubo
console.log(Math.pow(1,3)); // 1^3
console.log(Math.pow(2,3)); // 2^3
console.log(Math.pow(3,3)); // 3^3
console.log(Math.pow(4,3)); // 4^3
console.log(Math.pow(5,3)); // 5^3
```

o objeto global `Math` possui um método estático chamado `pow` que recebe como parâmetro a base e seu expoente. Aqui como são os 5 primeiros números, nosso programador sem titubear escolheu fazer da maneira mais rápida e simples possível. Agora vamos imaginar os 1.000 primeiros, ou os 10.000 primeiros, sendo apenas os pares.

# quando necessito repetir

## estrutura de repetição FOR

para que não seja necessário escrever 1001 vezes a mesma instrução nós podemos usar uma estrutura de repetição que irá nos ajudar a construir um laço que se repetirá até que uma condição seja verdadeira. Vejamos:

```
// sintaxe do laço for
for (início; condição; incremento) {
  // comandos sendo digitados aqui.
  ...
}
```

com essa construção nós podemos criar um loop de repetição que satisfaça quaisquer condições e enquanto ela for verdadeira, o bloco entre { e } será executado.

```
// sintaxe do laço for
for (let i = 1; i <= 1000; i++) {
  console.log(Math.pow(i,3));
}
```

no `for` acima, declaramos `i` como uma variável usando o operador `let`. Depois colocamos um `;` e declaramos a condição que neste caso é ser menor ou igual a 1.000 e mais um `;` onde incrementamos o valor de `i` para cada iteração. `i++` é o equivalente a escrever `i = i + 1` porém de maneira mais rápida. Nosso **loop** irá acontecer 1.000 vezes e o cubo de cada um dos 1.000 números será ecoado na tela. 3 linhas ao invés de 1.000.



# quando necessito repetir

## estrutura de repetição FOR

agora imagina missão 2 que é elevar ao cubo os 1.000 primeiros números pares. Poderíamos escrever uma primeira versão do código assim.

```
// Elevar ao cubo os 1000 primeiros números pares.
for (let i = 1; i <= 1000; i++) {
  if (i % 2 === 0) {
    console.log(`${i} ao cubo é ${Math.pow(i,3)}`);
  }
}
```

com já aprendemos a criar condições, nós usamos o operador `%` de módulo para testar se o resto da divisão por 2 é 0 e aí sabemos que é um número par, então nesta condição nós exibimos ele ao cubo. Resolvido o assunto.

```
// Elevar ao cubo os 1000 primeiros números pares.
for (let i = 0; i <= 1000; i += 2) {
  console.log(`${i} ao cubo é ${Math.pow(i,3)}`);
}
```

o código acima é uma versão 2 do primeiro que escrevemos. Ele usa melhor a estrutura `for` e ao invés de incrementar de 1 em 1 passou a incrementar de 2 em 2. Agora a condição é `i += 2` que é exatamente a mesma coisa de `i = i + 2`. Ele é mais eficiente que o primeiro em vários aspectos. O primeiro é que irá contar metade das vezes já que a iteração é de 2 em 2 e também não precisará de efetuar uma instrução matemática e outra comparação para decidir se é par. Então nosso novo código é tecnicamente melhor.

# quando necessito repetir regressivamente

## estrutura de repetição FOR

a estrutura `for` é muito flexível na sua construção. O controle de início, e incremento está totalmente nas mãos do programador.

```
// Contagem regressiva.  
for (let i = 100; i >= 1; i--) {  
    console.log(`Contagem regressiva ${i}`);  
}
```

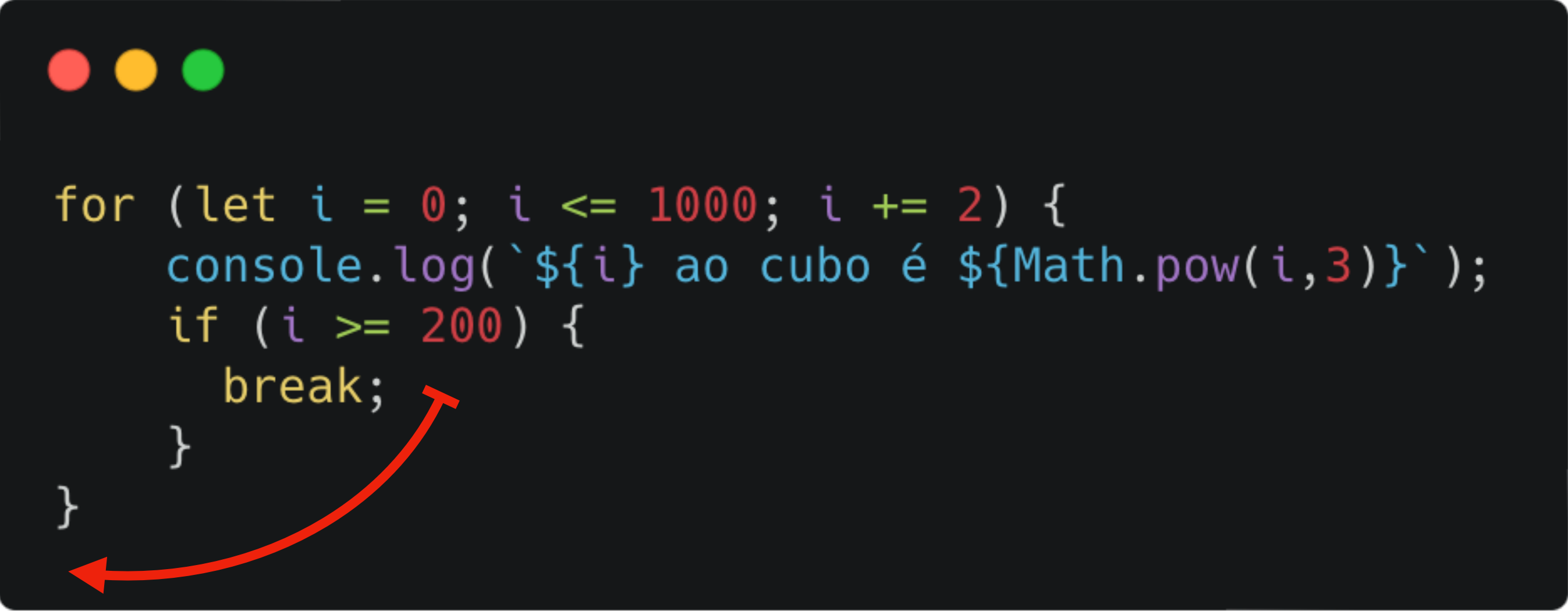
é muito fácil controlar a estrutura para criar uma repetição regressiva para o seu código, sem colocar nenhum operador adicional ou lógica maluca para ficar alterando todo o processo.

```
/**  
 * Contagem regressiva para exibir se o número  
 * é impar ou par, e se for números ímpares  
 * exibir como alerta na console.  
 */  
  
for (let i = 1000; i >= 1; i--) {  
    if (i % 2 === 0) {  
        console.log(`${i} é número par.`);  
    } else {  
        console.warn(`${i} é número ímpar.`);  
    }  
}
```

# dá pra parar antes do fim ?

## é possível encerrar antes

no JavaScript é possível em determinadas situações ter a necessidade de interromper um loop antes da condição acontecer. E para isto temos um operador `break` que ao ser executado sai do loop do `for`. Vejamos.



```
for (let i = 0; i <= 1000; i += 2) {  
  console.log(`${i} ao cubo é ${Math.pow(i,3)}`);  
  if (i >= 200) {  
    break;  
  }  
}
```

A red curved arrow points from the `break;` statement to the closing curly brace of the `for` loop, indicating that the loop is terminated before reaching its end condition.

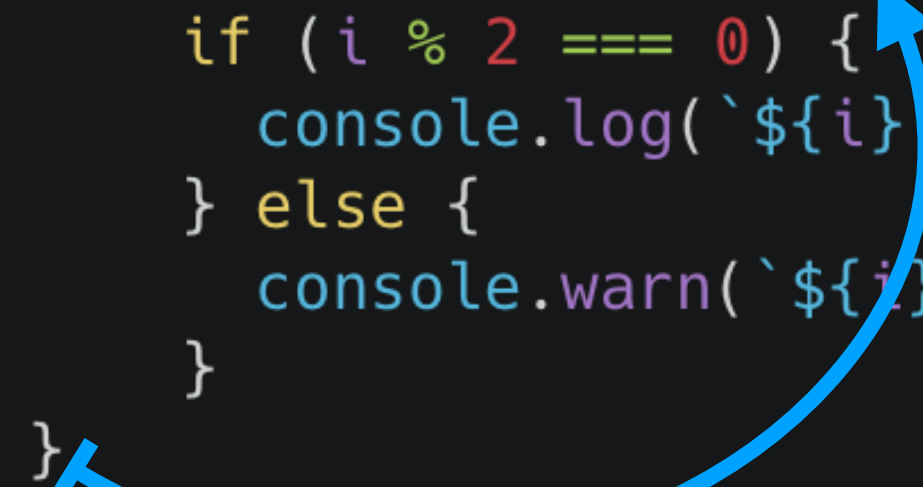
# forçar próxima iteração

## é possível redirecionar antes do fim do bloco

a estrutura de repetição `for` é quem gerencia seu controle de iterações. Basicamente ele testa a condição de mais uma iteração sempre que encontra o final do seu bloco `{ e }` e dispara o incremento. Funciona assim:

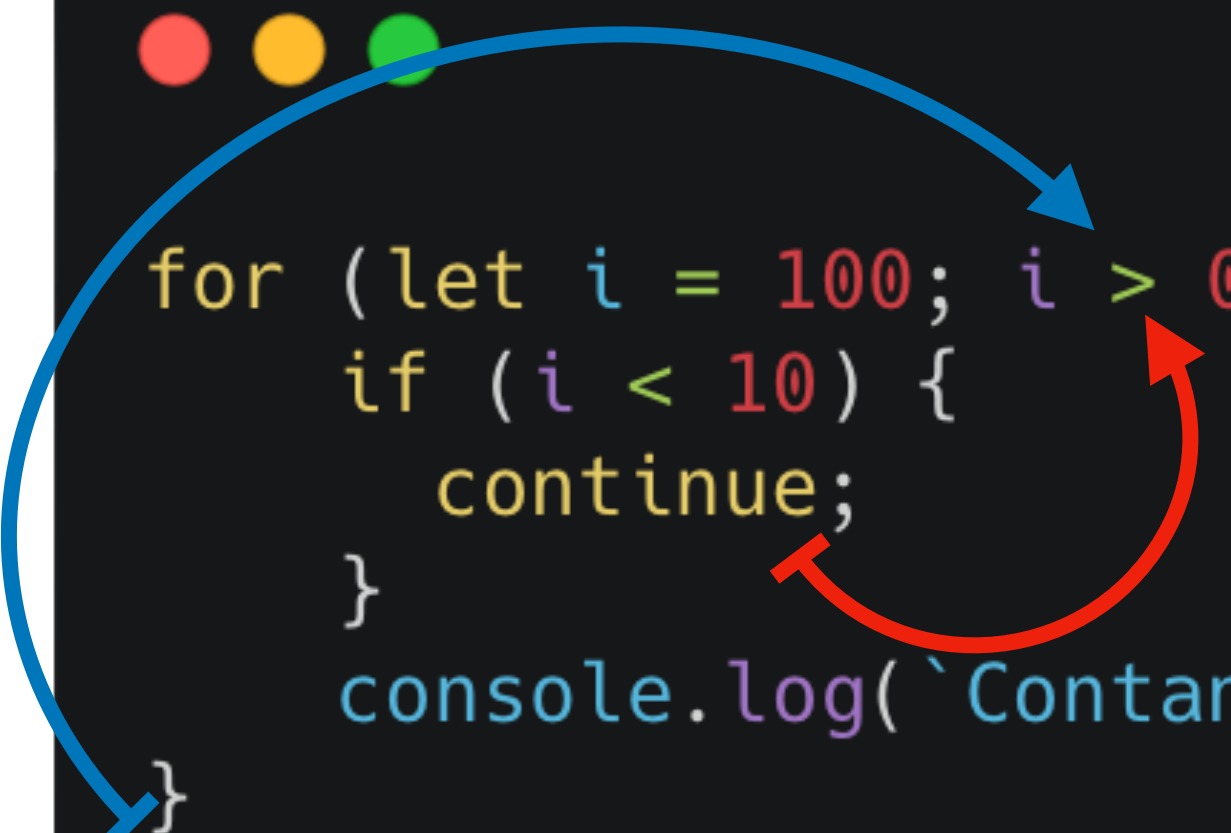
```
/**
 * Contagem regressiva para exibir se o número
 * é ímpar ou par, e se for números ímpares
 * exibir como alerta na console.
 */

for (let i = 1000; i >= 1; i--) {
  if (i % 2 === 0) {
    console.log(`${i} é número par.`);
  } else {
    console.warn(`${i} é número ímpar.`);
  }
}
```



quando encontra o final do bloco de escopo do `for` automaticamente o fluxo é desviado para a estrutura inicial do comando para ele efetuar o teste de condição e caso seja verdadeiro, ele faz mais uma iteração incrementando a variável. É possível forçar isto usando o `continue` dentro do laço. Veja abaixo.

```
for (let i = 100; i > 0; i--) {
  if (i < 10) {
    continue;
  }
  console.log(`Contando regressivo ${i}`);
}
```







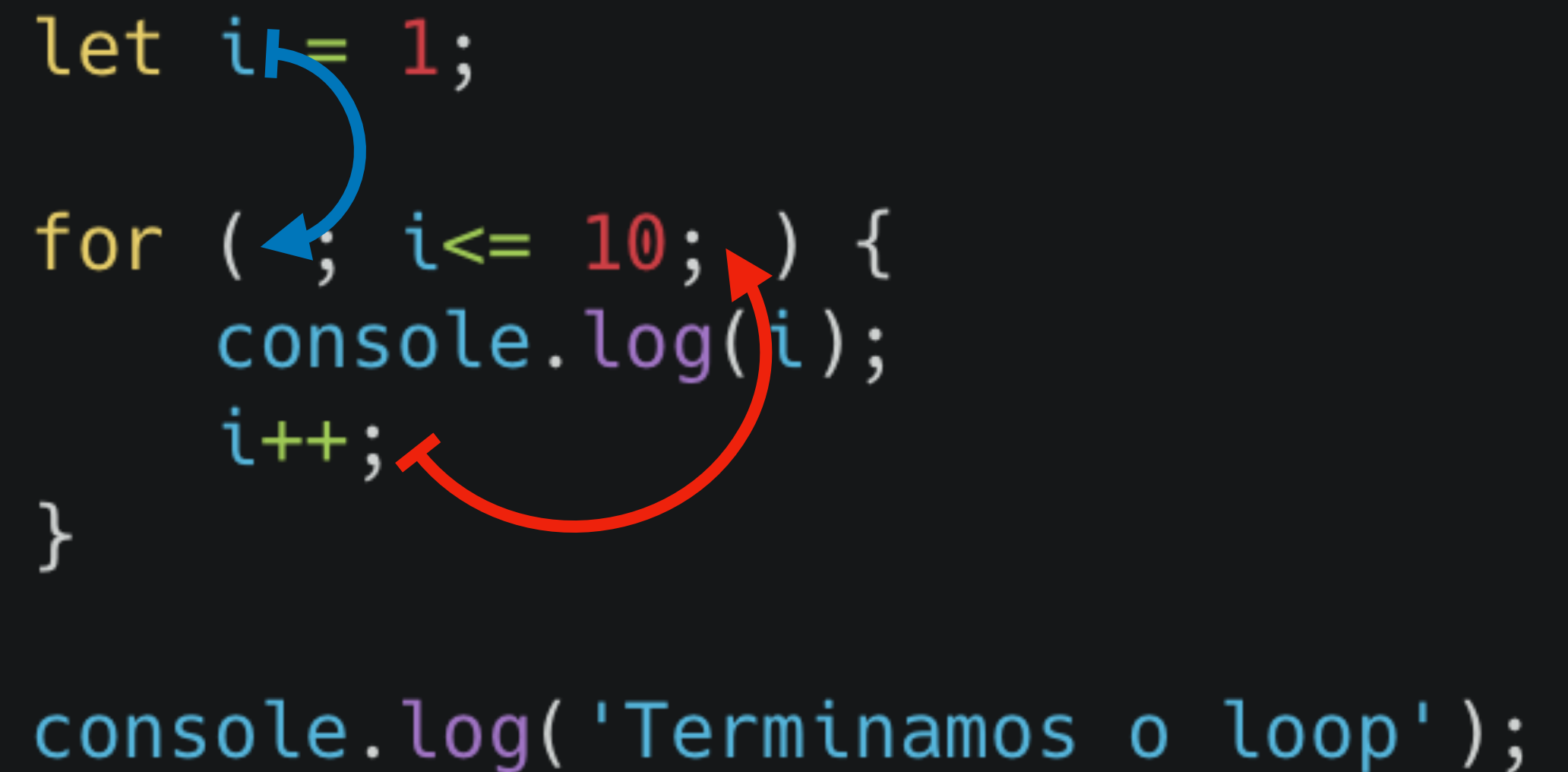
VAMOS PRATICAR

programming

# parece bizarro mas não é

## tudo depende do quanto você é capaz de abstrair

olhando bem as três estruturas de repetição do JavaScript dá para notar que elas tem uma grande similaridade. A diferença do `for` é a vantagem dele fazer o papel de incrementar nossa variável de iteração sempre que chegar ao fim do bloco. Mas na prática ele também pode deixar esse controle nas nossas mãos. Apesar de não ser recomendado fazer isto para não causar má interpretação de outros devs, sim, é possível escrever um `for` onde o incremento seja controlado dentro do bloco de maneira similar ao `do` e `while`. **But ... don't do it**



```
let i = 1;
for ( ; i <= 10; ) {
  console.log(i);
  i++;
}

console.log('Terminamos o loop');
```

# revendo a prática

## princípio básico

1. vimos que JavaScript é possível criar loops usando **for** para quaisquer condições.
2. **for** tem um controle melhor, mas pode ser totalmente controlado pelo programador dentro do bloco.
3. em todos os casos, **break** e **continue** tem uma ação importante que é preciso conhecer.



