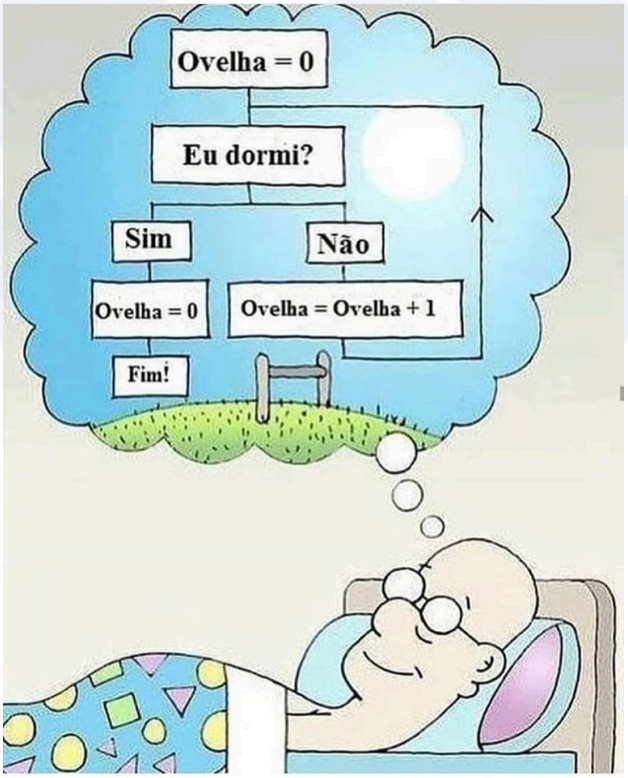
MATERIAL EXTRA

Estruturas de repetição

growdev

ESTRUTURAS DE REPETIÇÃO

Estruturas de repetição, também conhecidas como **loops**, são fundamentais em qualquer linguagem de programação, pois permitem a execução de um bloco de código várias vezes sem a necessidade de duplicação. No JavaScript, assim como em muitas outras linguagens, existem diferentes tipos de estruturas de repetição, cada uma com suas características específicas. Vamos explorar as principais estruturas de repetição de maneira simples e clara.



for

A estrutura `for` é uma das mais comuns em JavaScript e é usada quando você sabe de antemão quantas vezes deseja repetir um bloco de código.

Ela é composta por três partes principais: a inicialização, a condição de execução e o incremento (ou decremento).

Sintaxe:

```
for (inicializacao; condicao; incremento) {
    // bloco de código a ser executado
}
```

Exemplo:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

Este código vai imprimir os números de 0 a 4 no console. Vamos entender como isso funciona:

- **Inicialização**: `let i = 0;` — Aqui, você está criando uma variável `i` que começará com o valor 0.
- **Condição**: `i < 5;` — A repetição continuará enquanto essa condição for verdadeira. No caso, ela vai parar quando a variável `i` for igual ou maior que 5.
- **Incremento**: `i++` — A cada loop, o valor de `i` aumenta em 1.

Vantagens do `for`:

- Útil quando você sabe o número exato de iterações.
- Pode ser adaptado facilmente para trabalhar com arrays e outras coleções.

Para um exemplo mais prático, imagine que temos que imprimir no console de nossa aplicação Javascript a tabuada do 7 (sete).

Solução 1:

```
console.log('7 * 1 = ', 7 * 1)   // 7 * 1 =  7
console.log('7 * 2 = ', 7 * 2)   // 7 * 2 =  14
console.log('7 * 3 = ', 7 * 3)   // 7 * 3 =  21
console.log('7 * 4 = ', 7 * 4)   // 7 * 4 =  28
console.log('7 * 5 = ', 7 * 5)   // 7 * 5 =  35
console.log('7 * 6 = ', 7 * 6)   // 7 * 6 =  42
console.log('7 * 7 = ', 7 * 7)   // 7 * 7 =  49
console.log('7 * 8 = ', 7 * 8)   // 7 * 8 =  56
console.log('7 * 9 = ', 7 * 9)   // 7 * 9 =  63
console.log('7 * 10 = ', 7 * 10) // 7 * 10 =  70
```

Solução 2, utilizando for:

```
for (let indice = 1; indice <= 10; indice++) {
  console.log(`7 * ${indice} = `, 7 * indice)
}
/*
    7 * 1 =  7
    7 * 2 =  14
    7 * 3 =  21
    7 * 4 =  28
    7 * 5 =  35
    7 * 6 =  42
    7 * 7 =  49
    7 * 8 =  56
    7 * 9 =  63
    7 * 10 =  70
*/
```

Com apenas 2 linhas de código obtivemos o mesmo resultado, ou seja, substituímos ações repetidas por um laço **for**. Neste exemplo, nossa expressão inicial, definimos a variável **índice**, inicializando com 1, pois nossa tabuada inicia do 1 (um).

A condição diz que o laço será repetido enquanto o índice seja menor ou igual a 10 (dez). O incremento faz com que a cada interação o índice seja incrementado em + 1 (mais um).

while

A estrutura while é usada quando não se sabe exatamente quantas vezes o loop precisa ser repetido, mas depende de uma condição ser verdadeira. O código dentro do loop while só será executado caso a condição seja verdadeira e continuará a ser executado enquanto a condição permanecer verdadeira.

Sintaxe:

```
while(condicao) {
    // bloco de código a ser executado
}
```

Exemplo:

```
let i = 0

while (i < 5) {
  console.log(i)
  i++
}

/*
    1
    2
    3
    4
*/
```

Neste exemplo:

- O loop começa com a variável i igual a 0.
- O bloco de código dentro do while será repetido enquanto a variável i for menor que 5.
- Após cada execução, a variável i é incrementada em 1.

Aqui, o mesmo resultado do loop for será obtido: os números de 0 a 4 serão impressos.

Vantagens do while:

- Útil quando não se sabe quantas iterações serão necessárias.
- Flexível para condições que podem mudar dinamicamente durante a execução do programa.

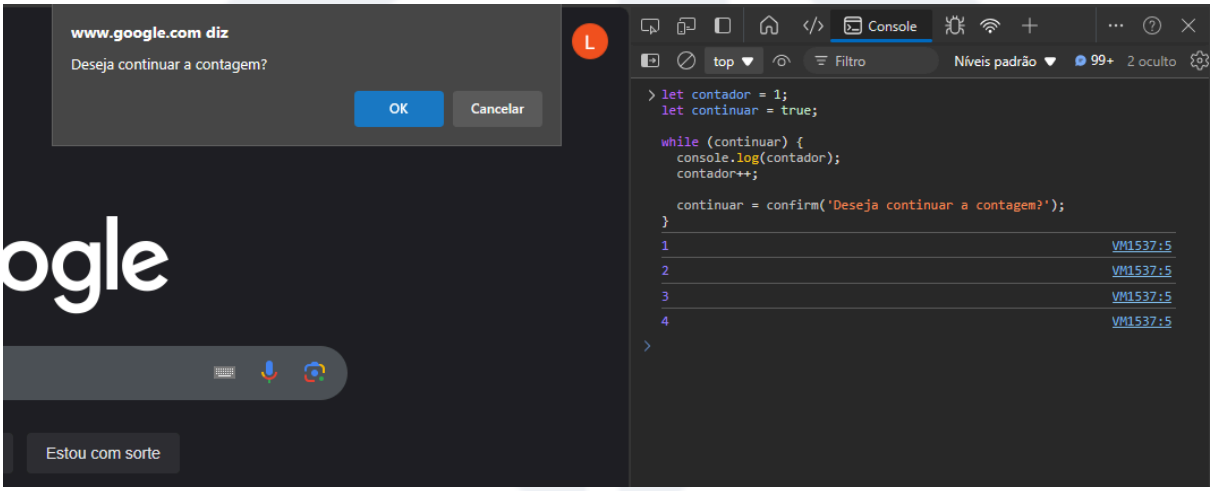
Vamos a um exemplo mais prático de caso de uso do `while`.

```
let contador = 1
let continuar = true

while (continuar) {
  console.log(contador)
  contador++

  continuar = confirm('Deseja continuar a contagem?')
}
```

Neste exemplo temos:

- Declaração da variável `contador`, que irá ser utilizada para mostrar a contagem no `console`.
- Declaração da variável `continuar`, do tipo lógica, que é inicializada com o valor `true`

- Ao iniciar a estrutura de repetição `while`, o valor de `continuar` será verificado para executar ou não o bloco de código dentro do `while`. É iniciado então a contagem.
- Ao final do bloco `while`, é solicitado que o usuário confirme se ele quer continuar vendo a contagem. Desta forma, ele decidirá o momento em que o loop irá parar.
- Enquanto o usuário clicar em "**Ok**" na caixa de diálogo da janela do Browser, a variável `continuar` seguirá tendo o valor `true` e o loop permanecerá sendo executado. O loop só será interrompido quando o usuário clicar em "**Cancelar**" e a variável receber o valor `false`.



do...while

A estrutura `do ... while` é semelhante ao `while`, mas com uma diferença crucial: o código dentro do loop será executado pelo menos uma vez, independentemente da condição. Isso acontece porque a verificação da condição é feita **após** a primeira execução do bloco de código.

Sintaxe:

```
do {
  // bloco de código a ser executado
} while (condicao)
```

Exemplo:

```
let i = 0

do {
  console.log(i)
  i++
} while (i < 5)
```

Nesse exemplo, a variável `i` é incrementada a cada iteração, e o loop para quando `i` atinge 5. O comportamento é praticamente igual ao `while`, com a diferença de que mesmo que `i` já fosse 5 no início, o bloco ainda seria executado uma vez.

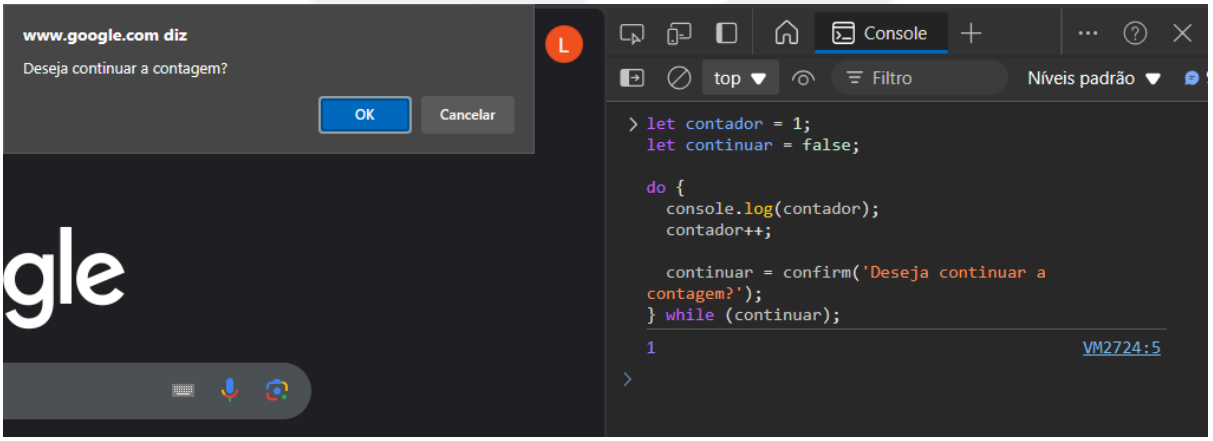
Vantagens do `do ... while`:

- Garante que o código seja executado pelo menos uma vez, independentemente da condição.
- Útil em situações em que a primeira execução precisa ocorrer antes da condição ser avaliada.

Vamos a um exemplo mais prático de caso de uso do `do...while`.

```
let contador = 1
let continuar = false

do {
  console.log(contador)
  contador++

  continuar = confirm('Deseja continuar a contagem?')
} while (continuar)
```

Neste exemplo temos a mesma disposição do exemplo demonstrado no `while`, com a diferença de que agora a variável `continuar` é inicializada com o valor `false` e ainda sim temos a contagem sendo realizada **pelo menos uma vez** antes do usuário decidir se quer continuar a contagem ou não.



for...of

O `for ... of` é uma estrutura mais recente no JavaScript e é muito útil para iterar sobre arrays e outros objetos iteráveis. Ele simplifica o processo de percorrer coleções sem a necessidade de gerenciar contadores como no `for` tradicional.

Sintaxe:

```
for (let valor of lista) {
    // bloco de código que será executado
}
```

Exemplos:

```
const frutas = ['laranja', 'morango', 'uva']

for (let fruta of frutas) {
   console.log(fruta)
}
```

Aqui, o loop percorre cada elemento do array `frutas`, e o valor de cada item é atribuído à variável `fruta` em cada iteração.

```
const nome = 'João'

for (let caractere of nome) {
  console.log(caractere)
}
```

Aqui, o loop percorre cada elemento da string `nome`, e o valor de cada posição é atribuído à variável `caractere` em cada iteração.

Vantagens do `for ... of`:

- Extremamente simples e legível.
- Ideal para iterações sobre arrays e strings.

for...in

O for...in é usado para iterar sobre as propriedades enumeráveis de um objeto. Ele é útil para navegar em objetos em que você deseja acessar as chaves de suas propriedades.

Sintaxe:

```
for(let propriedade in objeto) {
  // bloco de código que será executado
}
```

Exemplo:

```
const carro = {
  marca: 'Toyota',
  modelo: 'Fusca',
  ano: 2022,
}

for (let propriedade in carro) {
  console.log(propriedade, carro[propriedade])
}
```

Nesse caso, o loop percorre as propriedades do objeto `carro` (ou seja, `marca`, `modelo` e `ano`), e dentro do loop podemos acessar o valor correspondente a cada propriedade.

Vantagens do for...in:

- Perfeito para percorrer as propriedades de um objeto.
- Flexível para trabalhar com objetos dinâmicos.

Recapitulando as Principais Estruturas

- **`for`**: Ideal quando sabemos o número exato de iterações necessárias.
- **`while`**: Usado quando a repetição depende de uma condição que pode mudar durante a execução.
- **`do ... while`**: Garante que o bloco de código seja executado ao menos uma vez, independentemente da condição inicial.
- **`for ... of`**: Excelente para iterar sobre arrays e outros iteráveis, simplificando o código.
- **`for ... in`**: Usado para iterar sobre as propriedades de objetos.

Conclusão

Em JavaScript, as estruturas de repetição fornecem flexibilidade para lidar com diferentes tipos de iterações, desde loops simples, como o `for`, até estruturas mais sofisticadas, como o `for ... of` e `for ... in`, que permitem uma navegação eficiente em arrays e objetos.

Ao aprender a usar essas estruturas corretamente, você poderá evitar a duplicação de código e melhorar a legibilidade e a eficiência dos seus programas. Cada estrutura tem sua própria aplicação ideal, dependendo da situação, e dominar todas elas lhe dará mais ferramentas para resolver problemas no dia a dia da programação.

BONS ESTUDOS

growdev