09

# Outra forma de repetir

### Transcrição

Quem nunca hesitou ao ter que falar a tabuada do 7? O cálculo das tabuadas é um bom exemplo de processo de repetição. Calculamos a tabuada do 7, indo do 7 vezes 1, até o 7 vezes 10. Inclusive, isso fica mais claro com a forma com a qual alunos de escola escrevem tabuadas:

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

7 x 4 = 28

7 x 5 = 35

7 x 6 = 42

7 x 7 = 49

7 x 8 = 56

7 x 9 = 63

7 x 10 = 70

Esse é mais um caso que conseguimos tranquilamente fazer com que um programa de computador realize todo o trabalho para nós e apenas mostre os resultados que queremos. Então, vamos lá!

O primeiro passo é criar um novo arquivo, que vamos chamar de `tabuadas.html` . Nesse arquivo, começamos escrevendo o código JavaScript através da Tag `<script>` . Teremos:

```
<script>

</script>
```

A tabuada inicia na multiplicação por 1 e vai até o valor 10. Para começar a construir o código vamos criar uma variável que guarde o valor do multiplicador, chamaremos de `multiplicador` e ela iniciará em 1:

```
<script>
    var multiplicador = 1;
</script>
```

**Enquanto** a variável `multiplicador` for menor ou igual a 10, vamos querer calcular a tabuada. Portanto vamos fazer com que nosso programa realize essas multiplicações por 7, enquanto o multiplicador for menor ou igual a 10. Lembre-se de copiar a

função `mostra` para o arquivo `tabuadas.html` , assim, conseguiremos exibir os resultados da tabuada no navegador.

```
<script>
    function mostra(frase) {
        document.write(frase + "<br/>");
    }

    var multiplicador = 1;
    while(multiplicador <= 10) {
        mostra(7 * multiplicador);
    }
</script>
```

Mas se tentarmos abrir esse arquivo no navegador, ele nunca vai parar de fazer as multiplicações. Pior, a multiplicação que ele fará, será sempre 7 vezes 1. Isso acontece pois não aumentamos o número do multiplicador após cada conta realizada. Precisamos que a cada repetição, o multiplicador aumente um. Na programação, costumamos chamar esse processo de **incremento**. Então, precisamos incrementar 1 à variável `multiplicador` . Faremos o seguinte:

```
<script>
    function mostra(frase) {
        document.write(frase + "<br/>");
    }

    var multiplicador = 1;
    while(multiplicador <= 10) {
        mostra(7 * multiplicador);
        multiplicador = multiplicador + 1;
    }
</script>
```

Ficou fácil agora?

Vamos fazer mais um exercício para praticar e melhorar esse exemplo!

Geralmente, quando temos repetições que não são um *loop infinito*, temos 3 características presentes:

- Um valor inicial, que no caso da tabuada é o `multiplicador` de valor 1;
- Uma condição que determina se a repetição deve ser feita ou não, que no caso é quando verificamos que o `multiplicador` ainda é menor que 10;
- Uma modificação no valor que cause o fim da repetição para que, justamente, não sejam feitas repetições infinitas. No caso da tabuada, é quando aumentamos o valor do multiplicador a cada repetição, ou seja, quando utilizamos um incremento. Até agora, realizamos repetições por meio do `while` e sempre escrevemos o código necessário para contemplar essas 3 características, porém não há nada no `while` que nos induza a lembrar desses pontos;

O comando `for` existe com o intuito de nos lembrar de todas as características necessárias antes de montar o código da repetição . A sintaxe, ou seja, o jeito de escrevê-lo pode parecer um pouco estranho, mas vamos entendê-lo aos poucos. Observe:

```
for( ; ; ) {
    alert("mensagem para repetição infinita");
```

```
}
```

Ao abrir um programa, no navegador, com esse código, perceberemos que é feito um *loop* infinito, ou seja, o `for` faz uma repetição. Mas quando ela termina? Nunca! O `for( ; ; )` equivale ao `while(true)` que vimos no começo do capítulo.

Repare que dentro dos parênteses do comando `for` há dois `;` (ponto e vírgula), dando a sensação de que ali existe espaço para que três informações sejam inseridas. O código a seguir não é válido, é apenas para mostrar onde estão esses três espaços:

```
for(espaco 1; espaco 2; espaco 3)
```

Cada um desses espaços espera uma informação para controlar as repetições que acontecerão. Vamos fazer um programa que mostra 10 alertas, cada um com um número de 1 a 10. Novamente, teremos uma repetição, que deve ir de 1 até 10. Você lembra como faríamos isso com o `while`? Seria um código como o seguinte:

```
var numero = 1
while(numero <= 10) {
    alert(numero);
    numero = numero + 1;
}
```

Pois bem, podemos fazer praticamente idêntico, mas utilizando o `for`:

```
var numero = 1;
for(; numero <= 10; ) {
    alert(numero);
    numero = numero + 1;
}
```

Preenchemos o segundo espaço com a condição do `loop`, que será verificada toda vez para sabermos se mais uma repetição deve ocorrer.

Qual é a diferença de usar o `for` para o já conhecido `while`? Dentro do `for` apareceram os pontos-e-vírgulas, que tornam a forma da escrita do código bastante estranha. O resultado final será exatamente o mesmo. Então, pra que outro comando de repetição?

Com o `for`, podemos usar o primeiro espaço antes do ponto-e-vírgula para realizar inicializações. Um exemplo é inicializar a variável `numero`:

```
for(var numero = 1; numero <= 10; ) {
    alert(numero);
    numero = numero + 1;
}
```

O que mudou com essa alteração? Quase nada! Mas, seu código ficou mais legível!

A parte que diz `var numero = 1` é chamada de inicialização do `for` e ela será executada uma única vez. A próxima etapa é verificar se o número é maior ou igual a 10, `numero <= 10`, e assim o programa deve decidir se ele executa ou não a repetição.

O que devemos preencher no terceiro espaço após o segundo ponto-e-vírgula? Lembrando que aquilo que estiver na terceira parte será sempre executada. É muito utilizado, para executar um código que **incrementa**, a variável `loop`. Nesse caso, pra aumentar o valor da variável `numero` vamos escrever um comando que imprima de 1 até 10, teremos o seguinte:

```
for(var numero = 1; numero <= 10; numero = numero + 1) {
    alert(numero);
}
```

O `for` possui 3 espaços/pedaços, no primeiro dizemos qual é o valor inicial que queremos. No segundo definimos qual é a condição que determina se as repetições devem ser feitas ou não e essa parte é igual a que vimos no `while`. E o terceiro é o que modifica o valor de uma variável que influi na finalização das repetições. As três partes são delimitadas pelo `;`. Os três espaços são preenchidos conforme o esquema abaixo:

```
for([inicialização]; [condição para repetir]; [modificação do valor])
```

É importante perceber que o que acabamos de fazer com o `for` é o mesmo que fizemos com o `while`, mas escrito de uma maneira diferente. Enquanto o `for` precisa que escrevamos as 3 características logo no meio dos parênteses do comando, o `while` apenas pede a condição.

Com isso, como ficaria se escrevêssemos o mesmo programa que mostra na tela as tabuadas, mas agora utilizando o `for` ao invés do `while`?

Para começar, vamos nos lembrar de como era o código do programa da tabuada, que utilizamos junto do comando `while`:

```
<script>
function mostra(frase) {
    document.write(frase + "<br/>");
}

var multiplicador = 1;
while(multiplicador <= 10) {
    mostra(7 * multiplicador);
    multiplicador = multiplicador + 1;
}
</script>
```

Agora, podemos criar o arquivo `tabuada_com_for.html`, onde vamos criar nosso novo programa que irá fazer exatamente o que o anterior fazia, mas de uma maneira diferente.

Sabendo que o `for`, precisa primeiro do valor inicial, é fácil identificarmos que ele é a variável `multiplicador`, que começa com o valor 1.

```
<script>
for(var multiplicador = 1; ;) {
}
</script>
```

Agora precisamos preencher as outras duas informações que o `for` precisa. A próxima é até quando as repetições devem continuar, no caso, é a condição que indica que repetições devem ocorrer enquanto o multiplicador for menor que 10.

```
<script>
for(var multiplicador = 1; multiplicador <= 10; ) {
}
</script>
```

Pronto, só falta dizer quando o `multiplicador` aumentará o valor, no nosso caso é de 1 em 1. Chamamos isso de **incrementar** a variável. Para isso, estamos fazendo `multiplicador = multiplicador + 1` . Sempre que queremos somar 1 àquela variável, temos que repetir o nome dela indicando que nela queremos atribuir o valor dela mais 1.

No dia a dia, muitos programadores vão preferir fazer isso de forma abreviada, pois a **sintaxe** - as regras da linguagem - permitem isso. Por exemplo, essa linha pode ser abreviada utilizando-se o `multiplicador++` . Os dois sinais de soma ao lado da variável nada mais são do que uma maneira abreviada que o JavaScript entende como: "você quer aumentar o valor da variável em 1". Vamos usar essa nova forma abreviada em nosso programa.

```
<script>
for(var multiplicador = 1; multiplicador <= 10; multiplicador++) {
}
</script>
```

Pronto, com isso dizemos que o `multiplicador` irá iniciar em 1, assim as repetições vão acontecer enquanto o `multiplicador` for menor que 10 e a cada repetição o valor do `multiplicador` aumentará em 1.

Agora só falta adicionarmos o comportamento necessário para mostrar o resultado da tabuada do 7, já que já temos o `multiplicador` indo de 1 a 10.

```
<script>
for(var multiplicador = 1; multiplicador <= 10; multiplicador++) {
    mostra(multiplicador * 7);
}
</script>
```

Lembre-se que a função `mostra` deve constar no arquivo `tabuada_com_for.html` . Ao abrir o arquivo em um navegador, o resultado será exatamente igual ao que tínhamos antes, quando utilizamos o `while` .