

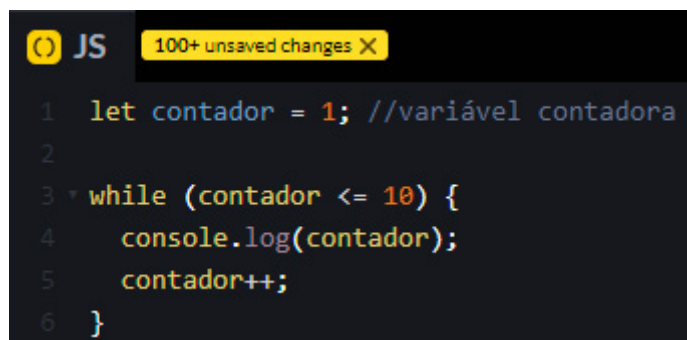
For, For... In e For... Of

Objetivo da Aula

Implementar scripts com laços de repetições.

Apresentação

Na aula anterior aprendemos duas estruturas de repetição muito parecidas: **while** e **do... while**. Nesta aula, aprenderemos a estrutura de repetição **for** e suas variações **for... in** e **for... of**. Dentre todas as estruturas de repetição, ela é a mais utilizada, pois é mais simples que as demais. Por exemplo, na estrutura **while** a variável contadora é inicializada antes da estrutura começar (*linha 1*) e é incrementada antes da estrutura terminar (*linha 5*). Veja:



```

JS 100+ unsaved changes X
1  let contador = 1; //variável contadora
2
3  while (contador <= 10) {
4      console.log(contador);
5      contador++;
6  }
    
```

Na estrutura de repetição **for**, a inicialização, o teste e o incremento é feito numa única linha e por esse motivo acaba sendo a estrutura queridinha dos desenvolvedores.

1. For

A estrutura de repetição **for** é muito utilizada quando temos estruturas incrementais, sendo assim, uma das suas principais aplicações é percorrer objetos ou variáveis do tipo **array**, assunto este que será aprofundado na próxima unidade.

Como já foi dito anteriormente, o **for** concentra tudo numa única linha (inicialização, teste e incremento) e isso facilita “entender o que o laço está fazendo e evita erros, como

esquecer de inicializar ou incrementar a variável de laço”, que é a (variável contadora) (FLANAGAN, 2013). Veja como é a sintaxe desta estrutura:

```
JS
1 * for (inicialização; teste; incremento/decremento ) {
2   instrucao; // um ou vários comandos a serem executados //
3 }
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Onde **inicialização** corresponde a declaração da variável contadora com seu valor inicial; **teste** corresponde a condição que será avaliada para então executar ou sair da estrutura de repetição; e incremento/decremento corresponde a alteração da variável contadora que tanto pode ser incrementada (++) quanto decrementada (--).



Destaque

É importante destacar que a “expressão de inicialização é avaliada apenas UMA vez, antes que o laço comece” (FLANAGAN, 2013). Além disso, a inicialização e o teste não são obrigatórios, ou seja, podem ser omitidos. Porém, é preciso tomar muito cuidado para evitar que a estrutura se torne um laço infinito.

Vamos entender como o **for** funciona, utilizando os mesmos exemplos da aula de **while** e **do... while**. Primeiro vamos ver como escrever os números de 1 a 10 na tela na ordem crescente e decrescente, em seguida vamos fazer o somatório destes números. **Exemplo 1:**

```
JS
1 * for (let contador=1; contador<= 10; contador++) {
2   console.log(contador);
3 }
```

Console
 1
2
3
4
5
6
7
8
9
10

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Repare que na expressão inicial é feita a declaração da variável contador inicializada com o valor 1. No teste é criada uma expressão para avaliar se o valor da variável contador é menor ou igual 10. Por fim, há a atualização (incremento) da expressão inicial representada pelo comando `contador++`, que significa adicionar 1 ao valor de contador. Viu como é simples? Utilizamos apenas 3 linhas para fazer a mesma coisa que o **while** faz em 5 linhas. Observe a diferença:

```

JS
1 for (let contador=1; contador<= 10; contador++) {
2   console.log(contador);
3 }

JS
1 let contador = 1; //variável contadora
2
3 while (contador <= 10) {
4   console.log(contador);
5   contador++;
6 }
  
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Agora vamos decrementar ao invés de incrementar a variável contadora (contador). Veja:

```

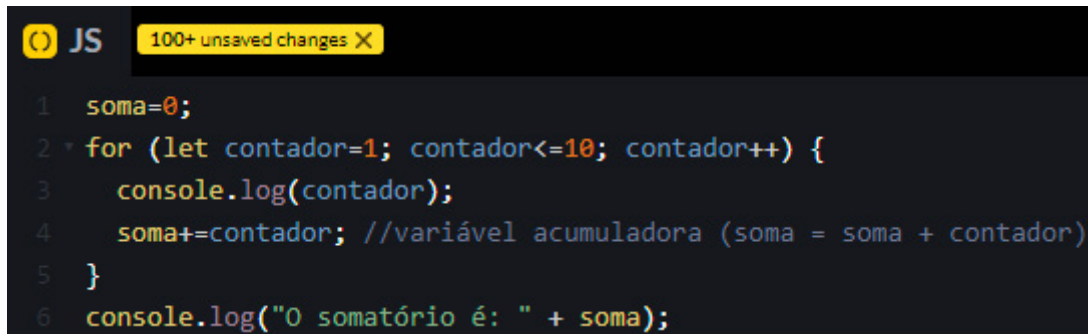
JS
1 for (let contador=10; contador>=1; contador--) {
2   console.log(contador);
3 }
  
```

Console
10
9
8
7
6
5
4
3
2
1

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Repare que na expressão inicial é feita a declaração da variável contador inicializada com o valor 10. No teste é criada uma expressão para avaliar se o valor da variável contador é maior ou igual 1. Por fim, há a atualização (decremento) da expressão inicial representada pelo comando `contador--`, que significa retirar 1 do valor de contador.

Agora vamos ver o nosso exemplo do somatório. **Exemplo 2:**



```

1  soma=0;
2  for (let contador=1; contador<=10; contador++) {
3    console.log(contador);
4    soma+=contador; //variável acumuladora (soma = soma + contador)
5  }
6  console.log("O somatório é: " + soma);

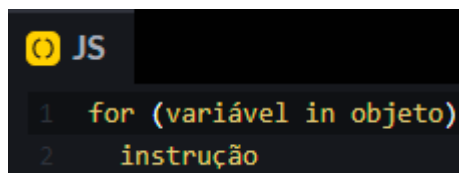
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/9a5c10f46e8567bd539be608ec74f6085be221c5/Unidade2Aula3a>

Mais uma vez economizamos duas linhas se comparado com o mesmo código feito com o *while*.

1.1. For... In

A versão *for... in* é muito utilizada para percorrer as propriedades de um objeto, porém também pode ser utilizado para percorrer *arrays*. Lembra que falamos sobre objeto, propriedade e método na unidade 1? Não! Então vamos relembrar... Dentro do paradigma POO (Programação Orientada a Objetos), um objeto nada mais é que uma representação do mundo real e um possui propriedades e métodos. Para entender melhor, vamos fazer a seguinte analogia: vamos considerar o objeto **PESSOA**. Uma pessoa possui as seguintes **propriedades**: *altura*, *peso*, *sexo* e *naturalidade*. Essas propriedades representam as características deste objeto. Vamos considerar também que este objeto também pode exercer algumas ações (*andar*, *dormir*, *comer*, *trabalhar* etc.), que no JavaScript chamamos de **métodos ou funções**. Para acessar a propriedade de um objeto basta colocarmos um ponto (.) entre o nome do objeto e o nome da propriedade (**pessoa.altura**). Para acessar um método de um objeto também colocamos um ponto (.) entre o nome do objeto e o nome do método, porém o método é seguido de parênteses (**pessoa.andar()**). Agora que já relembramos a diferença entre propriedade e método, vamos ver como é a sintaxe do *for... in*:

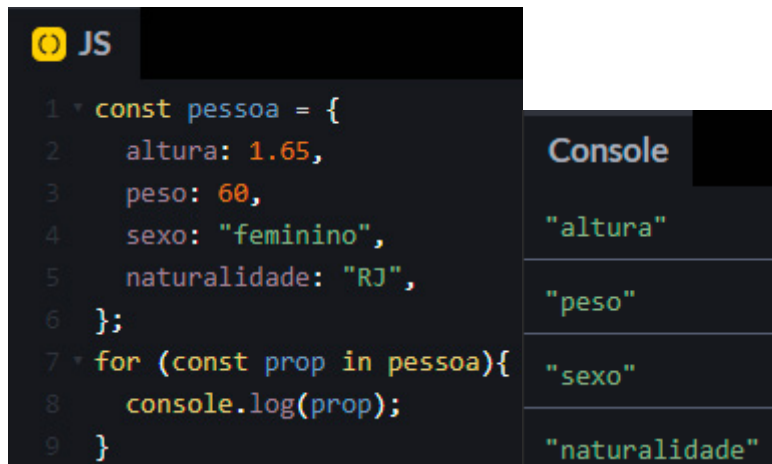


```

1  for (variável in objeto)
2    instrução

```

Onde a **variável** representa a variável que você vai criar para percorrer **objeto** escolhido. Vamos ver como percorrer as propriedades do nosso objeto pessoa:



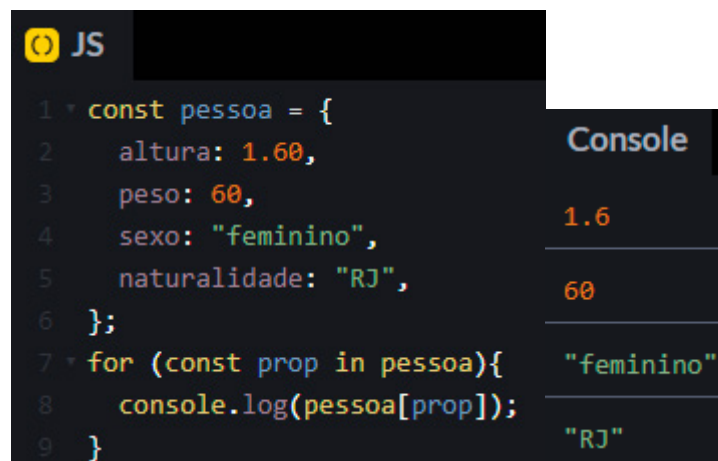
```

1 * const pessoa = {
2   altura: 1.65,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(prop);
9 }
  
```

Console

- "altura"
- "peso"
- "sexo"
- "naturalidade"

Primeiramente, criamos o objeto pessoa (*linha 1 a linha 6*) e depois criamos um **for** para percorrer o objeto criado, para isso, criamos uma variável cujo nome é prop que irá percorrer as propriedades do objeto pessoa. Repare que o **for... in** não percorre o valor que cada propriedade possui, mas, sim, cada propriedade declarada para o objeto pessoa. Se quisermos acessar os valores (atributos) de cada propriedade, basta usarmos prop com índice do nosso objeto (pessoa[prop]). Vejamos:



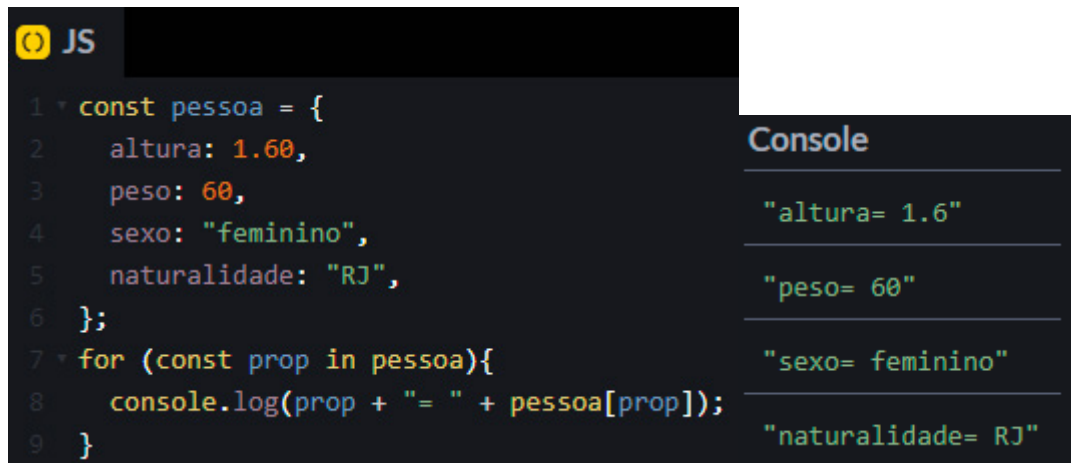
```

1 * const pessoa = {
2   altura: 1.60,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(pessoa[prop]);
9 }
  
```

Console

- 1.6
- 60
- "feminino"
- "RJ"

Se quisermos que o nome da propriedade apareça ao lado do seu respectivo valor, basta montarmos a *string* da seguinte forma:



```

1 * const pessoa = {
2   altura: 1.60,
3   peso: 60,
4   sexo: "feminino",
5   naturalidade: "RJ",
6 };
7 * for (const prop in pessoa){
8   console.log(prop + "= " + pessoa[prop]);
9 }
  
```

Console

```

"altura= 1.6"
"peso= 60"
"sexo= feminino"
"naturalidade= RJ"
  
```

Link para ter acesso ao código: <https://github.com/GRANCodigo/PraticaDeProgramacao/blob/ec06942681daf31c23c466b318e6a72aa232fa2c/Unidade2Aula4a>

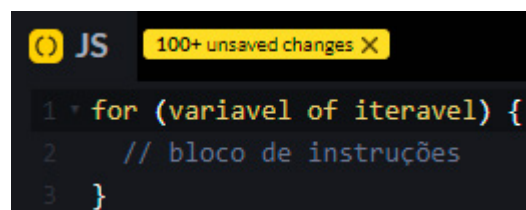
O sinal de "+" foi utilizado para concatenar a propriedade juntamente com o seu valor, separados pelo sinal "=". Acesse o link acima e teste todas os exemplos vistos.

1.2. For... Of

A estrutura **for... of** permite que você percorra um objeto iterativo (*Array*, *Map*, *Set*) e execute um bloco de instruções. É muito utilizado para percorrer *arrays* (vetores). Não se preocupe! Vamos aprender um pouco mais sobre *arrays* na próxima unidade. Saiba que o **for** por si só também percorre *arrays*, porém, com o **for... of**:

(...) você passará diretamente o *array* no qual será percorrido, e também uma variável que conterà o valor de cada posição do *array*. Ele sempre percorrerá o *array* por completo, sem você precisar fazer nenhuma verificação de tamanho do *array* ou colocar alguma condição específica (CERO, 2021).

Além disso, o uso do **for... of** torna o código mais simples e limpo. Veja como é a sua sintaxe:



```

1 * for (variavel of iteravel) {
2   // bloco de instruções
3 }
  
```

No exemplo abaixo, vamos utilizar a estrutura **for... of** para percorrer um vetor chamado vogais:

```
JS
1 const vogais = ['a', 'e', 'i', 'o', 'u'];
2 for (const elemento of vogais) {
3   console.log(elemento);
4 }
```

Na **linha 1**, criamos um **array** chamado vogais e atribuímos os seguintes valores a ele: a, e, i, o, u. Na **linha 2**, criamos um **for... of** com uma variável chamada elemento que irá percorrer o **array** vogais. O método `console.log()` é responsável por exibir na tela os valores assumidos pela variável elemento em cada interação (clico ou rodada).

Também podemos utilizar o **for... of** para percorrer **strings**:

```
JS
1 const curso = "Gran Cursos";
2 for (const letra of curso) {
3   console.log(letra);
4 }
```

Console

```
"G"
"r"
"a"
"n"
" "
"C"
"u"
"r"
"s"
"o"
"s"
```

No exemplo acima conseguimos ter cada letra da string separada, usando o **for... of**.

Considerações Finais

Repetir um bloco de instruções é muito importante e faz parte da vida de um desenvolvedor. Nesta aula aprendemos mais uma estrutura de repetição (**for**) e suas variações. Vimos que o **for** é uma estrutura muito mais simples que o **while**. Podemos utilizar suas variações (**for... in** e

for... of) para deixar o código mais legível e elegante, pois com elas não é necessário inicializar uma variável para controlar o fluxo de repetição e nem a incrementar ou decrementá-la para que a iteração ocorra. O próprio JavaScript se encarrega de saber quantos itens existem dentro do objeto/objeto iterativo a ser percorrido e, com base nisso, repete até que essa lista de itens seja toda percorrida.

Materiais Complementares

Estruturas de repetição *for*, *for... in* e *for... of* Javascript:

<https://youtu.be/VGOhchtuQAc>

Diferenças entre *for... of* e *for... in*:

<https://youtu.be/CRYH34yc99U>

Loops **FOR IN** e **FOR OF** – Curso de Javascript – Aula 99:

https://youtu.be/nGt_9znTQoU

For... of:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...of>

For... in:

<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in>

Referências

CERON, Vitor. *Estruturas de repetição for, for... in e for... of Javascript*, 2021. Disponível em: <https://programandosolucoes.dev.br/2021/03/23/estruturas-de-repeticao/>. Acesso em: 12 de nov. de 2022.

FLANAGAN, David. *JavaScript: O guia definitivo*. Porto Alegre, RS: Bookman, 2013.