

AULA 4 – RECURSIVIDADE

OBJETIVO DA AULA

Aplicar o conceito de recursão em *JavaScript*.

APRESENTAÇÃO

Nesta aula vamos falar de recursividade; este novo conceito tem tudo a ver com a aula de funções. Antes mesmo de falar de recursividade em JavaScript, vamos entender o conceito de recursão:

Recursão é um método de resolução de problemas que envolve quebrar um problema em subproblemas menores e menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente. Normalmente recursão envolve uma função que chama a si mesma. Embora possa não parecer muito, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar (MILER, 2014).

Vem comigo para entender melhor este conceito que parece tão complexo.



LINK

Saiba mais sobre o conceito de recursão no nosso blog: <https://blog.grancursosonline.com.br/recursividade-na-logica-de-programacao/>. Acesso em: 05/01/2023.



1. RECURSIVIDADE

Segundo a Wikipedia (2021), recursividade na ciência da computação nada mais é do que “a definição de uma sub-rotina (função ou método) que pode invocar a si mesma”, por exemplo, uma função chamando outra função. Basicamente seria assim:

```
JS 100+ unsaved changes X
1 function recursividade(){
2   //instruções
3   recursividade();
4   //instruções
5 }
```

Livro Eletrônico

A recursividade pode substituir as estruturas de repetição, sua utilização depende do problema a ser resolvido, pois em alguns casos é mais fácil usar a estrutura de repetição e em outros casos é mais fácil usar a recursividade. É importante lembrar que uma função recursiva precisar ter uma condição que interrompa a invocação dela mesma, caso contrário entrará num *loop* infinito. Ficaria assim:

```
JS 100+ unsaved changes X
1 function recursividade(){
2   if (condition){
3     //para de se chamar
4   }
5   else{
6     recursividade();
7   }
```

Um exemplo clássico de recursividade é criar uma função para calcular o fatorial (multiplicação de todos os números de uma sequência de 1 até o limite) de um número. Vamos fazer juntos? Antes de mais nada vamos relembrar como calcular o fatorial de um número, no nosso caso será o número 5:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

ou

$$5 \times 4 = 20$$

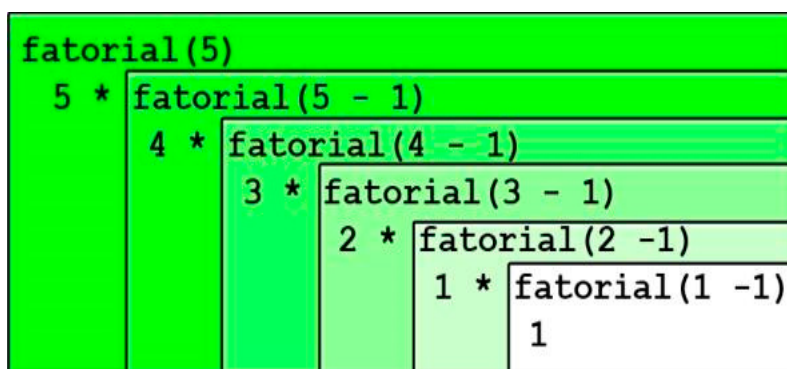
$$20 \times 3 = 60$$

$$60 \times 2 = 120$$

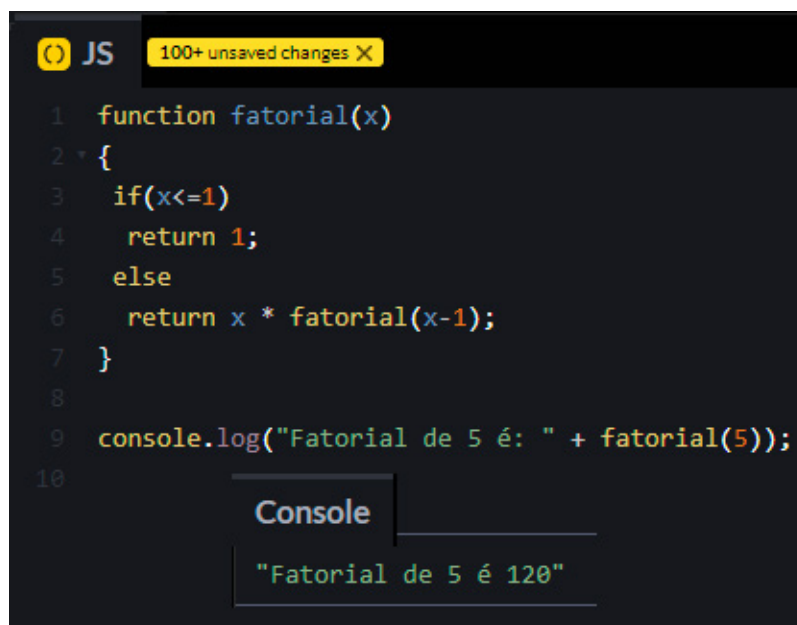
$$120 \times 1 = 120$$

Para criar uma função recursiva para resolver este caso devemos multiplicar o número 5 pelo valor anterior (4). No próximo passo a mesma coisa, multiplicar o resultado obtido (20) pelo valor anterior (3) e assim por diante até que o valor seja um para indicar o fim do processo. Veja:

FIGURA 1 | Exemplo de função recursiva



Agora vamos colocar a mão na massa para ver como ficará o código em JavaScript:



```

1  function fatorial(x)
2  * {
3    if(x<=1)
4      return 1;
5    else
6      return x * fatorial(x-1);
7  }
8
9  console.log("Fatorial de 5 é: " + fatorial(5));
10

```

Console

"Fatorial de 5 é 120"

Neste exemplo criamos uma função chamada *fatorial* que está recebendo um valor como parâmetro "x" (*linha 1*). Em seguida, verificamos se o valor x é menor ou igual a 1 (*linha 3*), caso o resultado seja verdadeiro o valor 1 é retornado para função, caso o resultado seja falso, inicia-se o processo de recursividade. Na *linha 6*, o retorno da função *fatorial* é a multiplicação do valor passado como parâmetro "x" pelo retorno da função *fatorial* com o valor de "x-1" (`return x * fatorial(x-1)`), como observamos no exemplo acima, a primeira iteração (ciclo) é 5 * 4 cujo resultado é 20, logo, 20 será multiplicado por x – 1 e assim sucessivamente.



PRA PRATICAR

Como seria a função *fatorial* sem a recursividade? Coloque a mão na massa e tente fazer sozinho uma função que calcule o fatorial utilizando iteração e não recursão.

Agora vamos ver um mesmo exemplo (contagem regressiva) utilizando a estrutura de repetição *for* e a recursividade. Com o *for* ficaria assim:

The screenshot shows a code editor with a JavaScript file. The code is a `for` loop that decrements a counter from 10 to 1, logging each value to the console. The console output shows the numbers 10, 9, 8, 7, 6, 5, 4, 3, 2, and 1 in descending order.

```

1 for(var cont=10; cont>=1; cont--)
2   console.log(cont);

```

Utilizando a recursividade ficaria assim:

The screenshot shows a code editor with a JavaScript file. The code is a recursive function `regressiva` that takes a counter and a limit as arguments. It logs the counter and calls itself with the decremented counter until it reaches the limit. The console output shows the numbers 10, 9, 8, 7, 6, 5, 4, 3, 2, and 1 in descending order.

```

1 function regressiva(cont,num){
2   console.log(cont);
3   if(num < cont){
4     regressiva(--cont,num);
5   }
6 }
7
8 regressiva(10,1);

```

Neste exemplo criamos uma função chamada **regressiva** que recebeu dois parâmetros de entrada: *cont* que representa o valor inicial e *num* que representa o valor final da contagem regressiva (*linha 1*). Dentro da função, a primeira linha (*linha 2*) mostra no console o primeiro valor da variável *cont* que neste caso é 10. Em seguida (*linha 3*) temos a estrutura condicional *if* que verificará a condição estabelecida para a nova chamada da função **regressiva**. Caso a condição seja verdadeira, a função é chamada novamente (*linha 4*), caso contrário, tudo é encerrado. Repare que a variável *cont* está sendo decrementada como pré-incremento (`--cont`)

e não como pós-incremento (`cont++`), pois neste caso é necessário que a variável `cont` já esteja decrementada. Para dar início ao processo de recursividade precisamos chamar a função (linha 8) regressiva (10, 1) e passar os valores inicial (10) e final (1) da contagem regressiva.

Resumindo, iniciamos a chamada da função com os valores `cont=10` e `num=1`. A função então verifica SE `num` é menor que `cont`, sendo menor este teste retorna `true` e a função **regressiva** é chamada novamente, porém, com o valor de `cont` decrementado (`--`) em um. Então chama a função **regressiva** com os valores `cont=9` e `num=1`, assim o processo é repetido, até que `cont` tenha o valor 1, neste momento o teste irá retornar `false` e não chamará a função novamente, saindo assim da recursividade.

CONSIDERAÇÕES FINAIS

Nesta aula falamos sobre um assunto que espanta muita gente que é a Recursividade. Aprendemos que uma função recursiva é aquela que chama a si mesma até que seja interrompida. No início pode parecer um tanto quanto bizarro, mas tenho certeza de que agora você já está tirando de letra. Lembrando que a prática é fundamental para absorção do conceito. Vimos que a recursividade torna o nosso código mais elegante e tem o poder de substituir uma estrutura de repetição, porém é preciso tomar cuidado para não colocar o seu código em *loop* infinito.

MATERIAIS COMPLEMENTARES

Recursividade/Dicionário de Programação:

<https://youtu.be/NKymAD4pJZI>

Lógica de Programação – Recursividade:

<https://youtu.be/M7c-m2xN9FQ>

REFERÊNCIAS

MILLER, Brad et al. *How to think like a computer scientist: Interactive edition*. Runestone, 2014.

WIKIPEDIA. *Recursividade (ciência da computação)*, 2021. Disponível em: < [https://pt.wikipedia.org/wiki/Recursividade_\(ci%C3%A2ncia_da_computa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Recursividade_(ci%C3%A2ncia_da_computa%C3%A7%C3%A3o))>. Acesso em 19 de nov. de 2022.