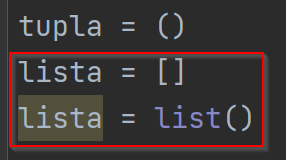
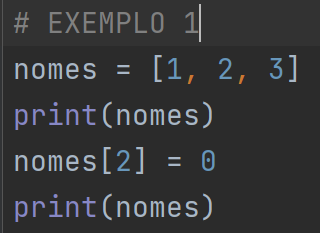
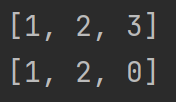
Aula 10

**Variáveis compostas – Listas (Parte I)**



* Tuplas e Lista são muito similares. Da mesma forma como tuplas armazenam vários valores, listas também. Porém, a vantagem que a lista traz é a capacidade de alterar um valor dentro do seu armazenamento.

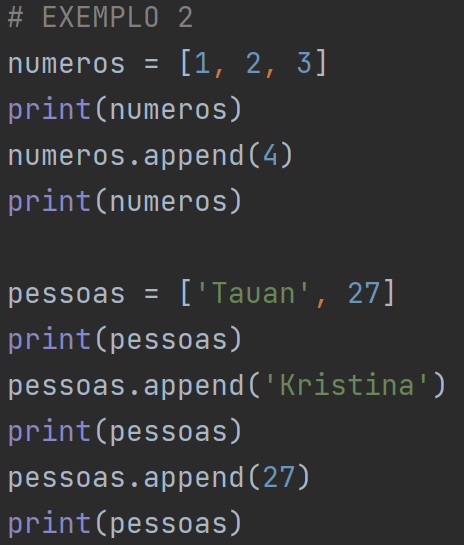


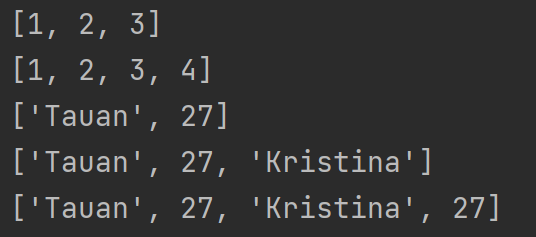


* Tuplas eram imutáveis até no seu tamanho. Uma vez que o usuário definiu a tupla, não era possível adicionar nenhum valor nela, visto que o tamanha era inalterável. Contudo, lista nos permitem fazer isso usando o método **.append()**

**ADICIONANDO ELEMENTOS À UMA LISTA**

1. **.append()**:

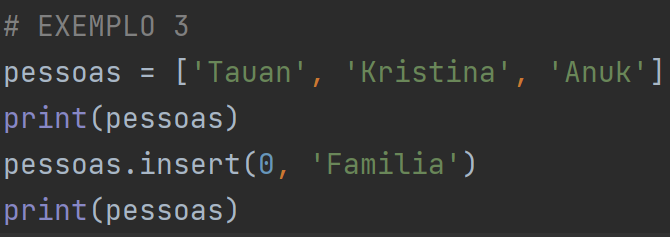


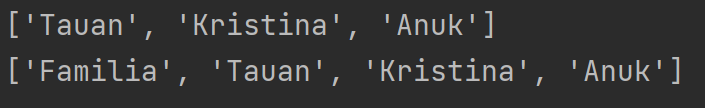


* Lista também podem ser mistas. Ou seja, conter string e númericos como tipos primitivos ao mesmo tempo;
* O método **.append()** aceita apenas um valor como argumento;
* Todos os valores adicionados, seguindo a forma mostrada nos exemplos, vão automáticamente para o final da lista.

1. **.insert()**:

* É possível também adicionar novos valores em uma posição à sua escolha usando o método **.insert()**.

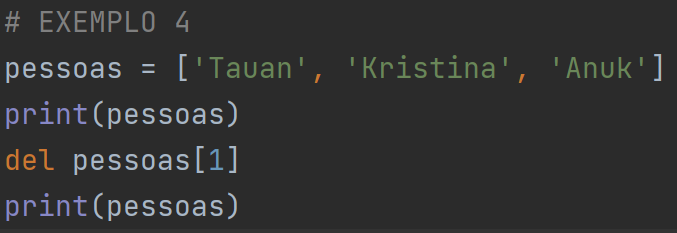


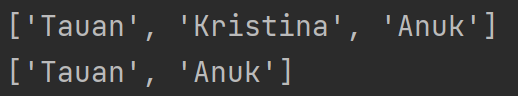


* **ATENÇÃO**! Note que antes ‘Tauan’ ocupava a posição 0, ‘Kristina’ ocupava a posição 1, ‘Anuk’ ocupava a posição 2. Com a adição de um novo elemento, todos os valores serão descolados um valor para direita e o tamanho da lista aumenta em uma unidade;
* Observe também que o método insert recebe dois valores. O primeiro é em qual posição nós gostariamos de adicionar o novo valores, e o segundo o valor em si.

**ELIMINANDO VALORES DE UMA LISTA**

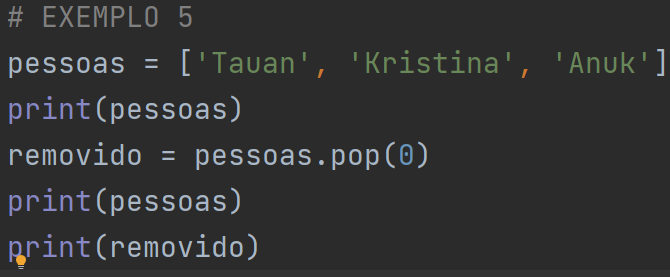
1. **del**:

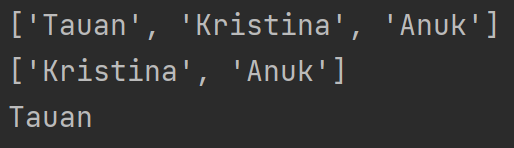




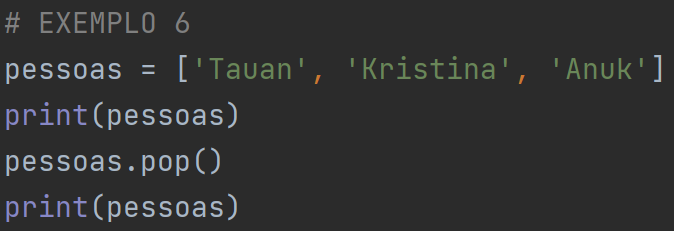
* Lembrando que toda vez que mexemos na estrutura de uma lista, a posição dos elementos muda e o tamanho da lista também.

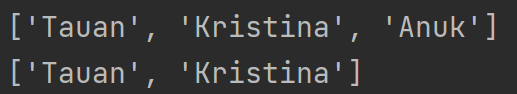
1. **.pop()**:



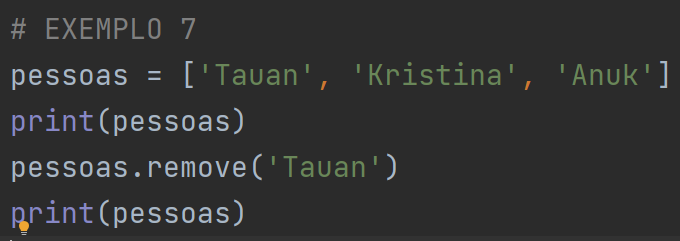


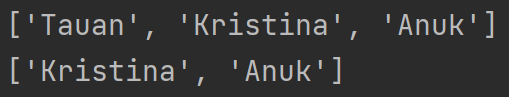
* Observe que o .pop() recebe como argumento a posição da variável que se deseja remover;
* Caso nenhum valor seja passado, ele remove o último por padrão.



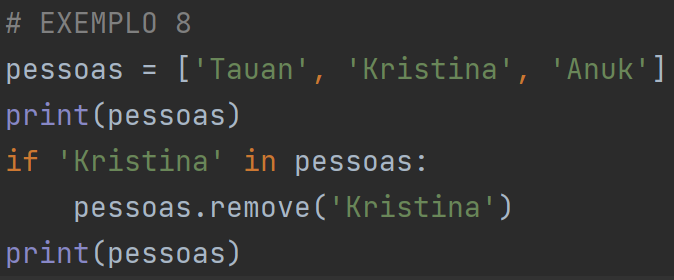


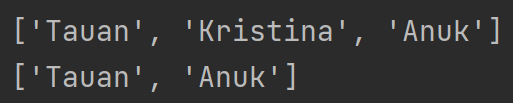
1. **.remove()**:



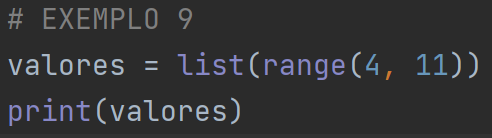


* Usando o método .remove(), passamos como parâmetro o conteúdo que queremos remover.
* Se tertarmos remover um índice que não existe? A linguagem retorna um ERRO. Então, a melhor solução para esse problema é usar a estrutura if.

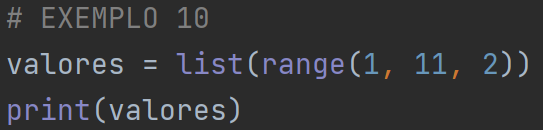




**GERANDO LISTAS**

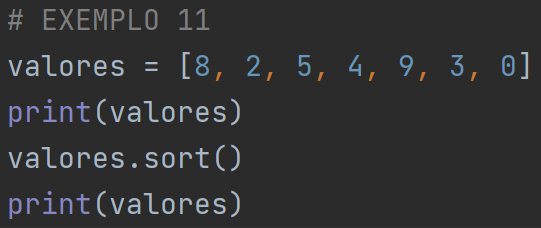


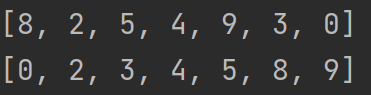
* Observe que usamos a cláusula **range()** para criar os termos dentro da lista.
* Visto que podemos usar o **range()**, isso significa que também podemos usar seus super poderemos, como escolher o salto que vamos dar dentro do **range()**.



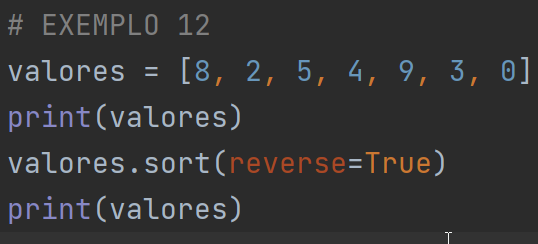


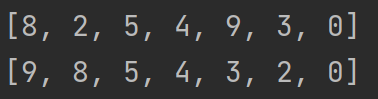
* Note também que o valores já são criados em ordem. Porém, podemos ter uma lista fora de **ordem crescente** e colocar ela em ordem quando desejarmos.



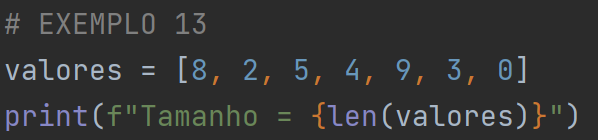


* Note que usamos o **.sort()** fora do argumento do print, pois **.sort()** reajusto todos os elementos dentro da lista. Caso coloque dentro **print()**, receberiamos “None” como output.
* Caso quiséssemos na ordem inversa, ou seja, na **ordem decrescente**?



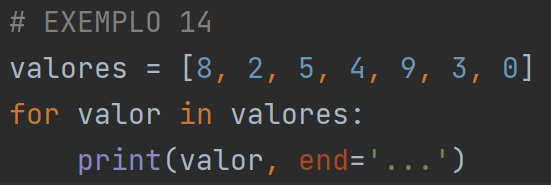


* Assim como vimos anteriormente, nós podemos também saber o tamanho da lista usando **len()**.

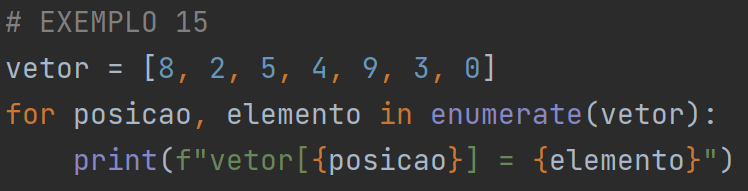


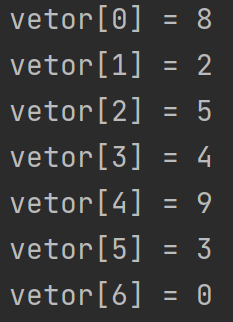


**USANDO ESTRUTURAS DE REPETIÇÃO COM LISTAS**

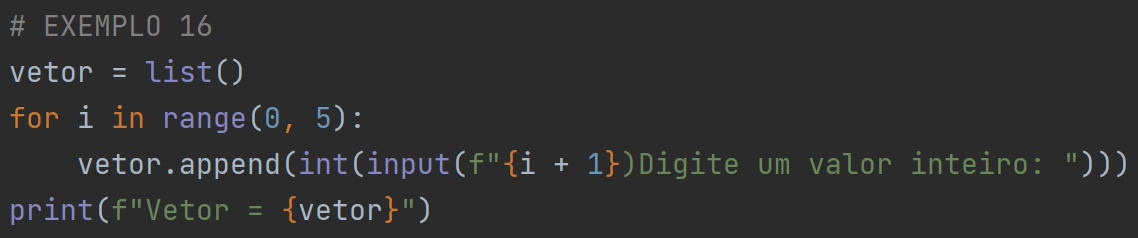


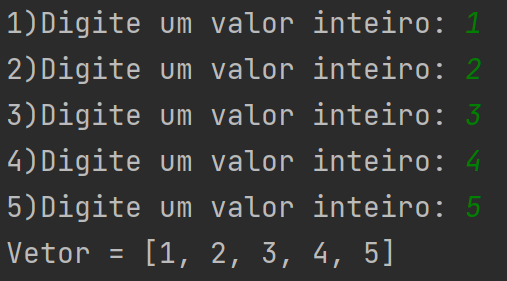




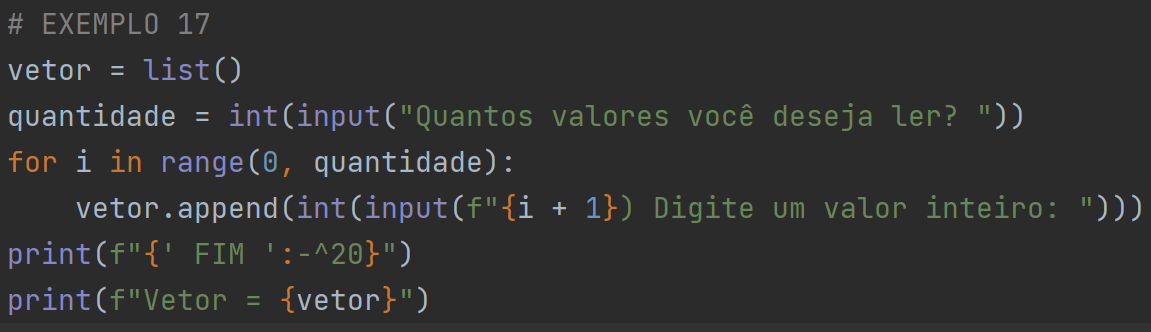


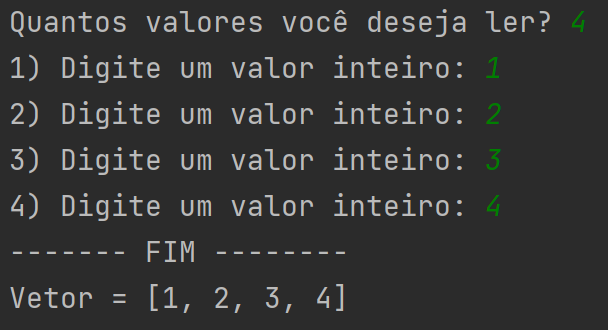
* Nós agora podemos ler valores do usuário e colocar dentro de uma única variável, uma lista! E mais do que isso, podemos usar estruturas de repeticação para fazer isso.



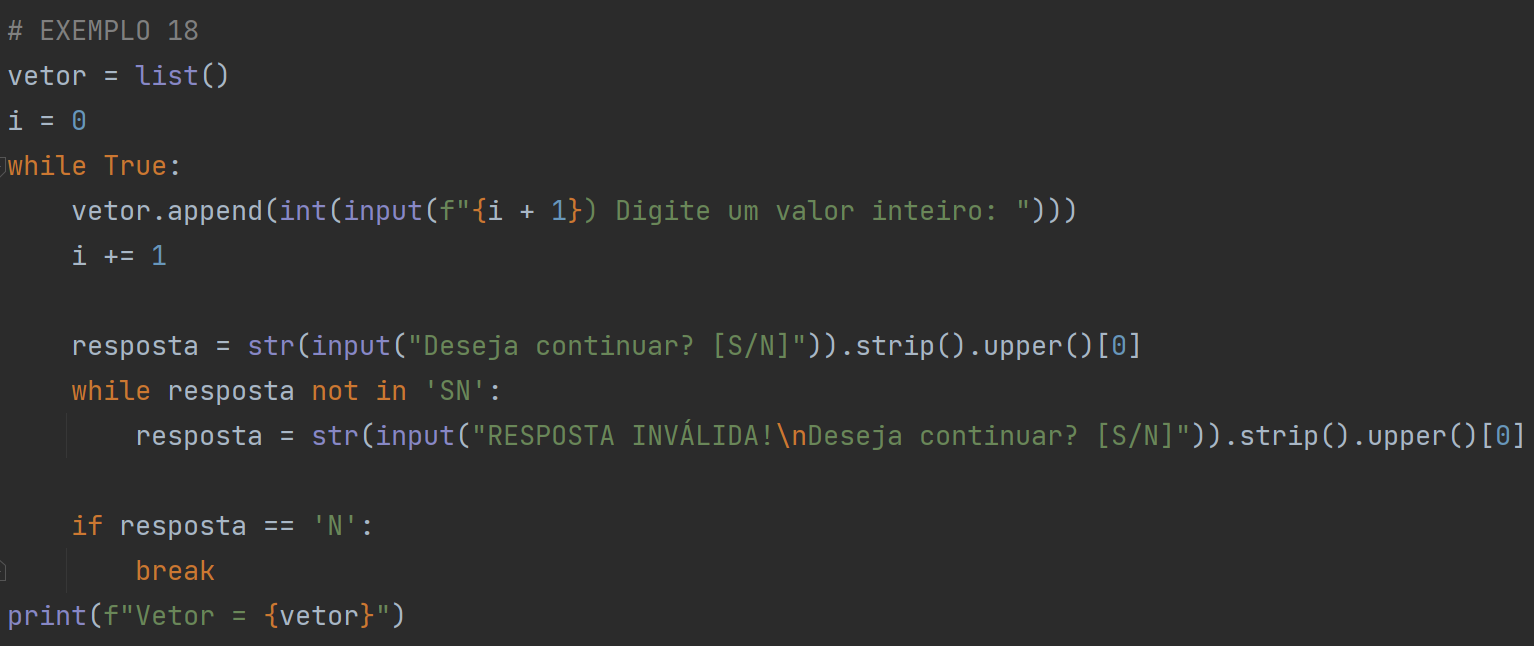


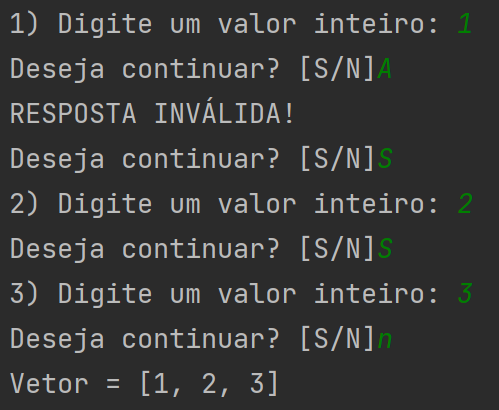
* Podemos também usar estruturas de repetição FOR para perguntar ao usuário quantos valores ele gostaria de ler.



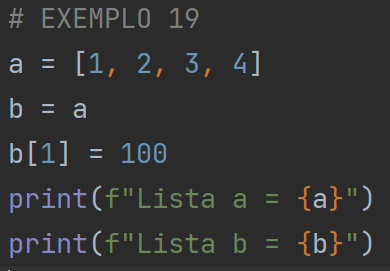


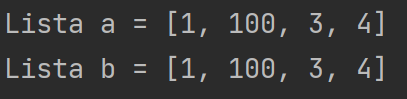
* Entretanto, ainda podemos usar a estrutura WHILE.



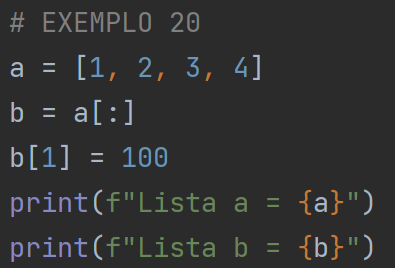


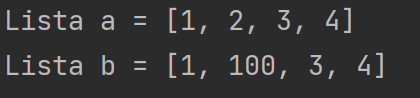
**LIGAÇÃO ENTRE LISTAS**





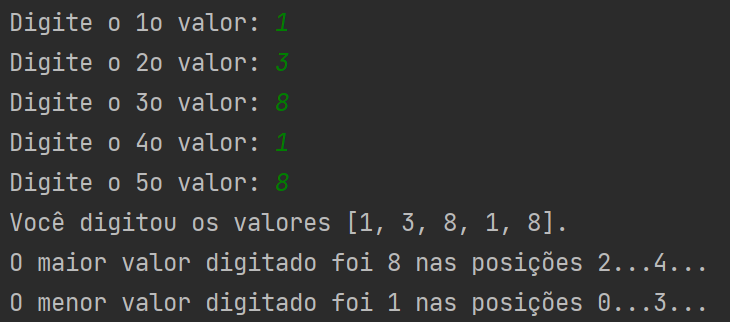
**CÓPIA DE LISTAS**



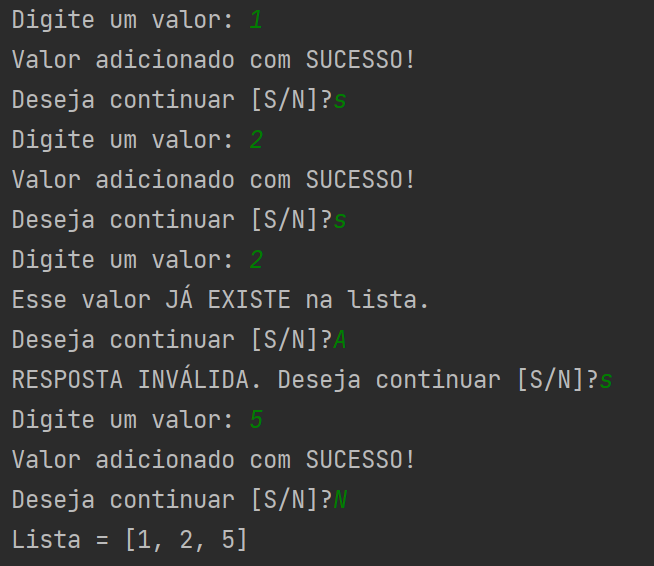


**PRÁTICA**

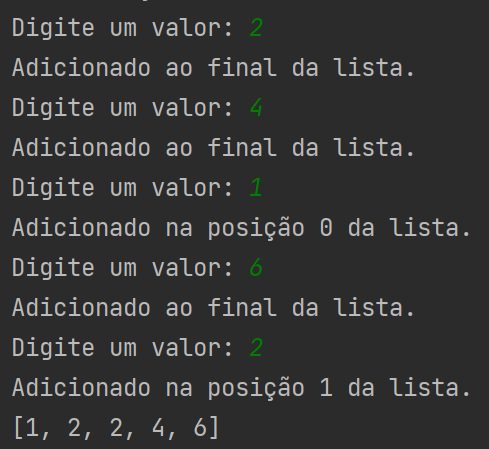
**[Exercício 1]** **[DESAFIO]** Escreva um programa que leia 5 valores e guarde-os dentro de uma lista. No final, mostre qual foi o maior e o menor valor digitado e as suas respectivas posições na lista.



**[Exercício 2]** Escreva um programa que o usuário escreverá vários valores numéricos e cadastre-os em uma lista. Caso o número já exista lá dentro, ele não será adicionado. No final, serão exibidos todos os valores únicos digitados, e em ordem crescente.

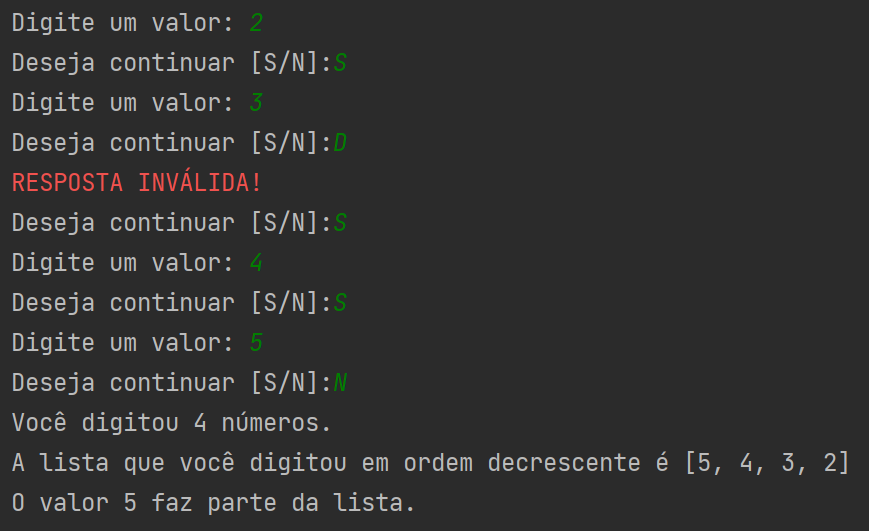


**[Exercício 3]** Escreva um programa que o user digite 5 valores numéricos e cadastre-os em uma lista, já na posição correta de inserção (SEM USAR O **sort()**). No final, mostre a lista ordenada na tela.



**[Exercício 4]** Escreva um programa que vai ler vários valores numéricos e colocar eles em uma lista. No final, mostre:

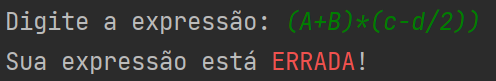
1. Quantos números foram digitados;
2. A lista de valores digitados, ordenada de forma crescente;
3. Se o valor 5 foi digitado e está ou não na lista.

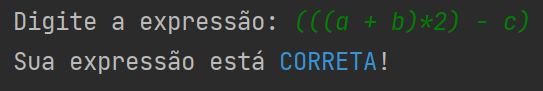


**[Exercício 5]** Escreva um programa que vai ler vários valores numéricos e colocá-los em uma lista. Depois disse, crie duas listas extras que vão conter apenas os valores pares e ímpares digitados, respectivamente. No final, mostre as 3 listas. Caso a lista esteja vazia, informe isso ao usuário.

|  |
| --- |
|  |
|  |
|  |

**[Exercício 6]** **[DESAFIO]** Escreva um programa que o usuário digite uma expressão matemática qualquer que use parênteses. Seu aplicativo deverá analisar se a expressão passada está com os parênteses abertos e fechados na ordem correta.





**Respostas**

