PROBLEMAS INVERSOS EM PYTHON

Profa.: Josiele da Silva Teixeira

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - CEFET
Campus UNED Nova Friburgo

October 23, 2024

Sumário

Sumário

ESTRUTURAS DE REPETIÇÃO

MOTIVAÇÃO

* Há situações do nosso cotidiano que, às vezes, a repetição de uma atividade é necessária;

* Como em situações do mundo real, em um programa, pode ser necessário repetir um trecho diversas vezes até que uma determinada condição seja satisfeita;

* Em programação, para casos como esse, são usadas estruturas conhecidas como iteração (não é interação!), repetição, laço ou loop.

* Python implementa duas estruturas de repetição: **while** e **for**.

ESTRUTURA **WHILE**

* O comportamento da estrutura de repetição **while** segue a seguinte ideia: enquanto uma condição (ou um conjunto delas) for verdadeira, uma instrução (ou um conjunto de instruções) que está dentro do laço deverá ser executada.

```
1    while <condição (ou um conjunto delas) for verdadeira>:
2        #Instruções a serem executadas
3    #Instruções a serem executadas após o encerramento do loop
```

Exemplo 1 do uso do **while**

* Construir um programa para escrever a saudação **Olá!** quatro mil vezes.

```
1    cont = 1
2    while cont <= 4000:
3       print("Olá!")
4       cont += 1 #o mesmo que cont = cont + 1
5    print("FIM")
```

Exemplo 2 do uso do **while**

* Calcular e exibir a soma dos 10 primeiros números inteiros positivos

```
1    soma = 0
2    termo = 1
3    while termo <= 10:
4        soma += termo
5        termo += 1 #geração do termo da PA, de razão 1
6    print(soma)
```

Função **range()**

* Gera uma sequência de números dentro de uma faixa especificada.

1. `range(<quantidade_de_números_a_serem_gerados>)`
2. `range(<início_da_faixa>, <fim_da_faixa>[,<incremento>])`

Exemplo:

```
list(range(3)) #Saída: [0, 1, 2]
list(range(10, 16))    #Incremento = 1:  [10, 11, 12, 13, 14, 15]
list(range(10, 16, 2)) #Incremento = 2:  [10, 12, 14]
list(range(16, 10, -1))#Incremento = -1: [16, 15, 14, 13, 12, 11]
list(range(16, 10, -2))#Incremento = -2: [16, 14, 12]
```

ESTRUTURA **FOR**

* Em geral, quando a quantidade de iterações é indeterminada, a estrutura **while** é uma boa alternativa. Por outro lado, quando o número de iterações é definido, a estrutura **for** é bastante adequada.

```
1    for <variável> in range([maneira_1|maneira_2]):
2        #Instruções a serem executadas
3    #Instruções a serem executadas após o fim do loop
```

Exemplo:

```
1    soma = 0
2    for termo in range(1, 11):
3        soma += termo
4    print(soma)
```

Exemplo de *loop* infinito

* Em algumas situações é necessária a construção de um trecho cujo comportamento seja semelhante a de um ***loop*** infinito, ou seja, uma situação em que o programa entra em um laço e não consegue escapar. Uma maneira de escrevê-lo é utilizando a palavra reservada **True**.

```
1    while True:
2        print("Execução de um loop infinito")
```

COMANDO **BREAK**

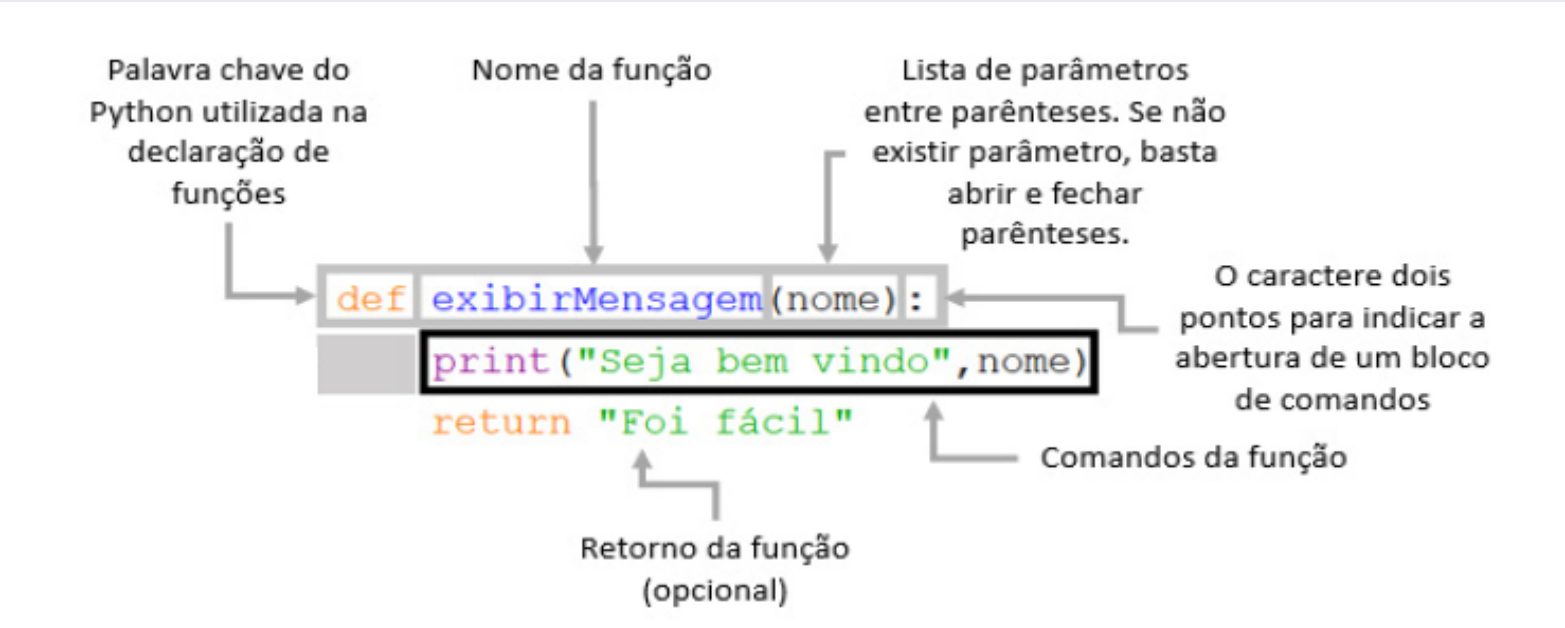
* O papel do comando **break** é quebrar a execução de uma estrutura de repetição, isto é, forçar a saída do fluxo do programa de dentro do laço.

```
1    print("*** Operação de divisão ***")
2    while True:
3        n1 = int(input("Informe o primeiro número: "))
4        n2 = int(input("Informe o segundo número: "))
5        if n2 == 0:
6            print("Divisor não pode ser 0.\nPrograma será encerrado...")
7            break
8        print(f"{n1} / {n2} = {(n1/n2):.2f}")
9    print("*** Fim da operação de divisão ***")
```

FUNÇÕES

* **Função** é um nome para um conjunto de comandos que pode ser chamado várias vezes em pontos diferentes do programa, sem a necessidade de repetição de código.
* Até aqui, já utilizamos várias funções em nossos programas Python. Exemplo: **int()**, **float()**, **input()**, **print()**, **range()**, etc

DECLARANDO FUNÇÕES



FUNÇÕES: Exemplos

```
1    def exibirMensagem():
2        print("Python é fácil")
3    exibirMensagem()
4    exibirMensagem()
5    exibirMensagem()
```

```
1    def exibirMensagemBoasVindas(pessoa, mensagem):
2        print(f"{mensagem}, {pessoa}")
3
4    exibirMensagemBoasVindas("João","Oi") #Saída: Oi, João
```

FUNÇÕES: Exemplos

```
1    def exibirMensagemBoasVindas(pessoa, mensagem):
2        print(f"{mensagem}, {pessoa}")
3
4    exibirMensagemBoasVindas("João","Oi") #Saída: Oi, João
```

* Caso seja necessário, você pode passar os parâmetros nomeados. Nesse caso não importa a ordem dos parâmetros.

```
1    def exibirMensagemBoasVindas(pessoa, mensagem):
2        print(f"{mensagem}, {pessoa}")
3
4    exibirMensagemBoasVindas(mensagem = "Bom dia", pessoa = "Ana")
5    #Saída: Bom dia, Ana
```

LISTAS

* **Lista** é uma estrutura de dados cujos elementos são organizados de forma linear. Cada elemento de uma lista pode ser acessado a partir do índice que representa a sua posição na coleção.
* É importante observar que o primeiro elemento de uma lista é armazenado no índice 0, o segundo no índice 1, e assim sucessivamente.
* Os elementos de uma lista podem ser de diversos tipos, como os primitivos (int, float, string e lógico) ou os mais complexos como listas, dicionários e objetos.

DECLARANDO UMA LISTA

* Durante a declaração de uma variável do tipo lista, não é necessário especificar o tamanho máximo porque ela cresce dinamicamente.
* Uma lista pode ser declarada de duas formas: a) vazia ou b) preenchida:

```
<lista> = []
<lista> = [<elemento1>, <elemento2>, ..., <elementoN>]
```

Exemplo:

```
1    # Lista vazia
2    idades = []
3
4    # Lista com elementos pré-definidos
5    idades = [27, 19, 33, 47]
```

```
dados = ["Caroline Paola Oliveira da Silva", 0, 1.70, True]
```

LISTA

* Para acessar um elemento, basta informar o nome da lista e em qual índice ele se encontra. A sintaxe básica para isso é a seguinte:

```
lista[<índice>]
```

* onde **<índice>** é o índice em que se encontra o elemento desejado.

Exemplo:

dados =	Caroline Paola Oliveira da Silva	0	1.70	True
Índice:	0	1	2	3

LISTA

```
1    dados = ["Caroline Paola Oliveira da Silva", 0, 1.70, True]
2    print(f"Nome.........: {dados[0]}")
3    print(f"Filhos.......: {dados[1]}")
4    print(f"Estatura.....: {dados[2]:.2f}m")
5    if dados[3] == True:
6       print("Usa Instagram: Sim")
7    else:
8       print("Usa Instagram: Não")
```

```
Nome.........: Caroline Paola Oliveira da Silva
Filhos.......: 0
Estatura.....: 1.70m
Usa Instagram: Sim
```

INCLUINDO ELEMENTOS: **append()**

* Para incluir elementos em uma determinada lista é utilizando o método **append()**. Ele faz com que o elemento seja adicionado no final da lista.

```
<lista>.append(<elemento>)
```

```
1    atrizes = ["Paolla de Oliveira"]
2    atrizes.append("Camila Queiroz")
3    while True:
4        nome = input("Digite o nome de uma atriz: ")
5        atrizes.append(nome)
6        resp = input("Deseja continuar [S|N]? ")
7        if resp.upper() == "N":
8            break
9    print(atrizes)
```

INCLUINDO ELEMENTOS: **insert()**

* Outra forma de incluir elementos em uma lista é utilizando o método **insert()**. A diferença desse método para o append() é que você especifica em qual índice deseja incluir o novo elemento.

```
<lista>.insert(<índice>, <elemento>)
```

```
1    atrizes = ["Paolla de Oliveira"]
2    atrizes.append("Sheron Menezes")
3    atrizes.insert(1, "Raquel Nunes")
4    print(atrizes)
```

REMOVENDO ELEMENTOS

* Para remover elementos de uma lista, existem os métodos **remove()** e **pop()** e a função **del**.

```
<lista>.remove(<elemento>)
<lista>.pop(<índice>)
del <lista>[<índice>]
```

- **remove()**: basta informar qual elemento se deseja excluir. Se, na lista, existir mais de um elemento com o valor informado, será removido apenas a primeira ocorrência encontrada.
- **pop()**: o seu argumento de entrada é o índice a ser removido e não o próprio elemento. Todavia, se o índice não for informado, o método removerá o último elemento da lista.
- **del**: informam-se o nome da lista e em qual índice o elemento se encontra. Se não for informado o índice da lista que se deseja remover o elemento, a função del destruirá a variável do tipo lista.

```
1    atrizes = ["Adriana", "Adriana", "Camila",
2               "Danielle", "Fernanda", "Helena",
3               "Paolla", "Raquel", "Viola"]
4
5    print("Removendo a primeira ocorrência do elemento Adriana...")
6    atrizes.remove("Adriana")
7    print(atrizes)
8
9    print("Removendo o elemento de índice 1, que é Camila...")
10   atrizes.pop(1)
11   print(atrizes)
12
13   print("Índice não foi informado. Remove o último elemento: Viola")
14   atrizes.pop()
15   print(atrizes)
16
17   print("Removendo o novo elemento de índice 1, que é Danielle...")
18   del atrizes[1]
19   print(atrizes)
20
21   del atrizes #Destroi a variável atrizes
```

Funções úteis:

* **count()**: <lista>.count(<elemento>). Ex: **ocorrencias = atrizes.count("Adriana")**

- contar quantas ocorrência de um determinado elemento existe em uma lista;

* **len()**: len(<lista>). Ex: **elem = len(atrizes)**

- contar quantos elementos existem em uma lista;

* **min(), max(), sum()**: Ex:

```
1    notas_turma = [4, 7.2, 9, 10, 1.75, 3.5, 8, 6.3]
2
3    menor = min(notas_turma)
4    maior = max(notas_turma)
5    media = sum(notas_turma) / len(notas_turma)
6
7    print(F"Menor nota....: {menor:.2f}")
8    print(F"Maior nota....: {maior:.2f}")
9    print(F"Média da turma: {media:.2f}")
```

RETORNANDO O ÍNDICE DE UM ELEMENTO

* **index()** é utilizado para retornar o índice em que se encontra a primeira ocorrência de um elemento informado.

- Sintaxe: **<lista>.index(<elemento>)**

```
1    atrizes = ["Adriana Prado", "Bárbara Borges", "Camila Queiroz",
2               "Danielle Winits", "Fernanda Paes Leme",
3               "Helena Ranaldi", "Paolla de Oliveira",
4               "Raquel Nunes", "Viola Davis"]
5
6    procurar_por = "Marina Ruy Barbosa"
7    if procurar_por in atrizes:
8        indice = atrizes.index(procurar_por)
9        print(f"{procurar_por} está no índice {indice}.")
10   else:
11       print(f"{procurar_por} não está na lista.")
```

* O operador **in**, além de verificar a existência de um elemento, também pode ser utilizado para percorrer uma lista elemento a elemento.

LISTAS ANINHADAS

* Uma lista aninhada consiste na existência de uma lista como sendo elemento de uma outra lista.

  Ex: **familia = ["João Nascimento", ["João Vitor", "Maria Clara"]]**

* As listas aninhadas podem ser utilizadas para a representação de matrizes.

$$A = \begin{bmatrix} 2 & 1 & -5 \\ 3 & 7 & 0 \\ 6 & 4 & 8 \end{bmatrix},$$

```
A = [
      [2, 1, -5],
      [3, 7, 0],
      [6, 4, 8]
    ]
```

LISTAS ANINHADAS

* Para acessar um elemento da lista aninhada A (matriz), é preciso informar o índice correspondente e o índice da lista interna.
  Sintaxe: **lista [<indiceExterno>][<indiceInterno>]**

```
1    A = [[2, 1, -5], [3, 7, 0], [6, 4, 8]]
2
3    tr_A = A[0][0] + A[1][1] + A[2][2]
4    print(f"tr(A) = {tr_A}")
5
6    # Soma dos elementos
7    soma_A  = A[0][0] + A[0][1] + A[0][2]
8    soma_A += A[1][0] + A[1][1] + A[1][2]
9    soma_A += A[2][0] + A[2][1] + A[2][2]
10   print(f"soma(A) = {soma_A}")
```