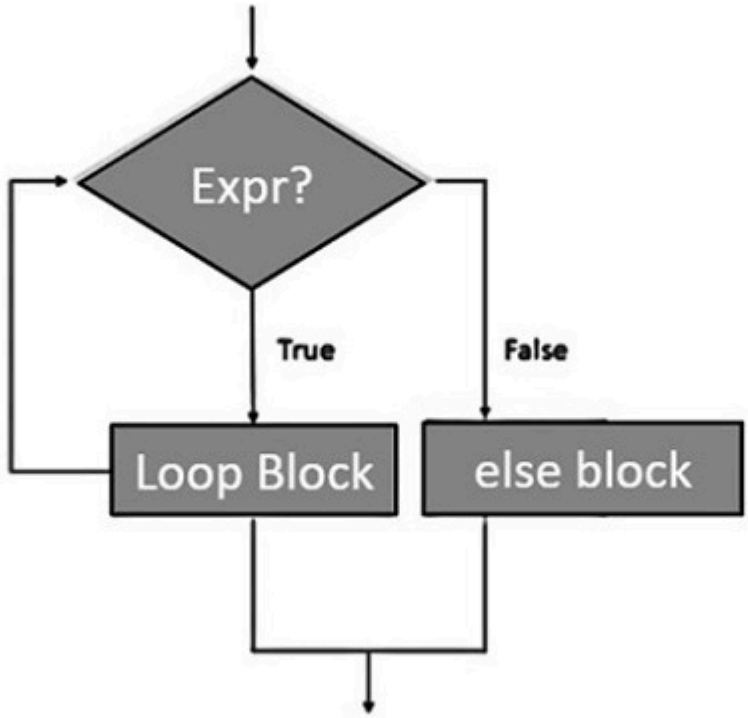
Loops for-else em Python

Python - Loop Para Else

Python suporta ter uma instrução “else” associada a uma instrução de loop “for” . Se a instrução "else" for usada com um loop "for", a instrução "else" será executada quando a sequência se esgotar, antes que o controle mude para a linha principal de execução.

Fluxograma do loop For Else

O diagrama de fluxo a seguir ilustra como usar a instrução **else** com loop **for** -



Sintaxe do loop For Else

A seguir está a sintaxe do loop for com cláusula else opcional -

```
for variable_name in iterable:
 #stmts in the loop
 .
 .
 .
else:
 #stmts in else clause
 .
 .
```

Exemplo de loop For Else

O exemplo a seguir ilustra a combinação de uma instrução else com uma instrução for em Python . Até que a contagem seja inferior a 5, a contagem de iterações é impressa. À medida que se torna 5, a instrução print no bloco else é executada, antes que o controle seja passado para a próxima instrução no programa principal.

```
for count in range(6):
   print ("Iteration no. {}".format(count))
else:
   print ("for loop over. Now in else block")
print ("End of for loop")
```

Ao ser executado, este código produzirá a seguinte **saída** -

```
Iteration no. 1
Iteration no. 2
Iteration no. 3
Iteration no. 4
Iteration no. 5
for loop over. Now in else block
End of for loop
```

Usando a instrução else com loop for em python

Em outras linguagens, a funcionalidade else é fornecida apenas em pares if-else . Mas o Python também nos permite implementar a funcionalidade else com loops for.

A funcionalidade else está disponível para uso somente quando o loop termina normalmente. No caso de encerramento forçado do loop, a instrução else é ignorada pelo intérprete e, portanto, sua execução é ignorada.

.

NOTA : Quando o loop não é finalizado por uma instrução **break** , o bloco else imediatamente após for/while é executado.

Método 1: construção For-Else com terminação normal (sem instrução break)

Exemplo

O programa a seguir mostra como usar a instrução else com loop for -

```
for i in ['T','P']:
   print(i)
else:
   # Loop else statement
   # there is no break statement in for loop, hence else part gets executed directly
   print("ForLoop-else statement successfully executed")
```

Ao ser executado, o programa acima irá gerar a seguinte saída -

```
T
P
ForLoop-else statement successfully executed
```

Método 2: construção For-Else com encerramento forçado (com instrução break)

Exemplo

O programa a seguir mostra como as condições funcionam no caso de uma instrução break -

```
for i in ['T','P']:
   print(i)
   break
else:
   # Loop else statement
   # terminated after 1st iteration due to break statement in for loop
   print("Loop-else statement successfully executed")
```

Ao ser executado, o programa acima irá gerar a seguinte saída -

```
T
```

Explicação

Este tipo de else só é útil se houver uma condição if dentro do loop que dependa de alguma forma da variável do loop.

No **Método 1** , a instrução loop else é executada, pois o loop for termina normalmente após a iteração sobre list['T','P']. No entanto, no **Método 2** , a instrução loop-else não é executada, pois o loop é interrompido forçadamente usando instruções de salto, como break.

Esses **métodos** mostram claramente que quando o loop é encerrado forçadamente, a expressão loop-else não é executada.

Agora considere um exemplo em que a instrução loop-else é executada em algumas condições, mas não em outras.

Método 3: construção For-Else com instrução break e condições if

Exemplo

O programa a seguir mostra como as condições funcionam no caso de instruções break e instruções condicionais -

```
# creating a function to check whether the list item is a positive
# or a negative number
def positive_or_negative():
   # traversing in a list
   for i in [5,6,7]:
   # checking whether the list element is greater than 0
      if i>=0:
         # printing positive number if it is greater than or equal to 0
         print ("Positive number")
      else:
         # Else printing Negative number and breaking the loop
         print ("Negative number")
         break
   # Else statement of the for loop
   else:
      # Statement inside the else block
      print ("Loop-else Executed")
# Calling the above-created function
positive_or_negative()
```

Ao ser executado, o programa acima irá gerar a seguinte saída -

```
Positive number
Positive number
Positive number
Loop-else Executed
```

Usando a instrução else com loop while em python

Else-While sem instrução break

Algoritmo (etapas)

A seguir estão o algoritmo/etapas a serem seguidas para realizar a tarefa desejada -

- Valor k inicializado com 0.
- Usando um loop while para percorrer até que a condição especificada seja verdadeira (até k<8).
- Aumentando o valor k em 1, pois não queremos executar o loop while infinitas vezes.
- Imprima o valor k.
- Else bloco é executado quando a condição falha/se torna falsa, ou seja, quando o valor **k se torna 8.**

Exemplo

O programa a seguir demonstra o uso da instrução else no loop while -

```
k=0
# traversing until the condition is true(k<8)
while k<8:
   # incrementing the k value by 1
   k+=1
   # printing k value
   print("k =",k)
else:
   # else block gets executed when the condition fails(becomes false)
   print("This is an else block")
```

Ao ser executado, o programa acima irá gerar a seguinte saída -

```
k = 1
k = 2
k = 3
k = 4
k = 5
k = 6
k = 7
k = 8
This is an else block
```

Usando a instrução Else no loop While com uma instrução break

Exemplo

```
# creating a function that checks if the
# list passed to it contains an even number
def hasEvenNumber(l):

   # getting the list length
   n = len(l)
   # intializing a variable with 0(index)
   i = 0
   # traversing the loop until the I value is less than the list length
   while i < n:
   # checking whether the corresponding index element of a list is even

      if l[i] % 2 == 0:

         # printing some text, if the condition is true
         print("The input list contains an even number")

         # giving a break statement/break the loop

         break

         # incrementing the I (index) value by 1

      i += 1

   else:
   # Else print "The input list doesn't contain an even number"
   # It executes Only if the break is NEVER met and the loop is terminated after all

      print("The input list doesn't contain an even number")

# calling the hasEvenNumber() function by passing input list 1 as an argument
print("For Input list 1:")
hasEvenNumber([3, 9, 4, 5])
# calling the hasEvenNumber() function by passing input list 2 as an argument
print("For Input list 2:")
hasEvenNumber([7, 3, 5, 1])
```

Ao ser executado, o programa acima irá gerar a seguinte saída -

```
For Input list 1:
The input list contains an even number
```

For Input list 2:
The input list doesn't contain an even number

Loops aninhados em Python

A linguagem de programação Python permite o uso de um loop dentro de outro loop. A seção a seguir mostra alguns exemplos para ilustrar o conceito.

Sintaxe

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statements(s)
      statements(s)
```

A sintaxe para uma instrução de loop **while** aninhada na linguagem de programação Python é a seguinte -

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Uma observação final sobre o aninhamento de loop é que você pode colocar qualquer tipo de loop dentro de qualquer outro tipo de loop. Por exemplo, um loop for pode estar dentro de um loop while ou vice-versa.

Exemplo

O programa a seguir usa um loop for aninhado para exibir tabelas de multiplicação de 1 a 10.

```
for i in range(1,11):
   for j in range(1,11):
      k=i*j
      print ("{:3d}".format(k), end=' ')
   print()
```

O loop interno da função print() tem end=' ' que acrescenta um espaço em vez da nova linha padrão. Portanto, os números aparecerão em uma linha.

O último print() será executado no final do loop **for** interno .

Quando o código acima é executado, ele produz a seguinte **saída** -

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
```