

ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

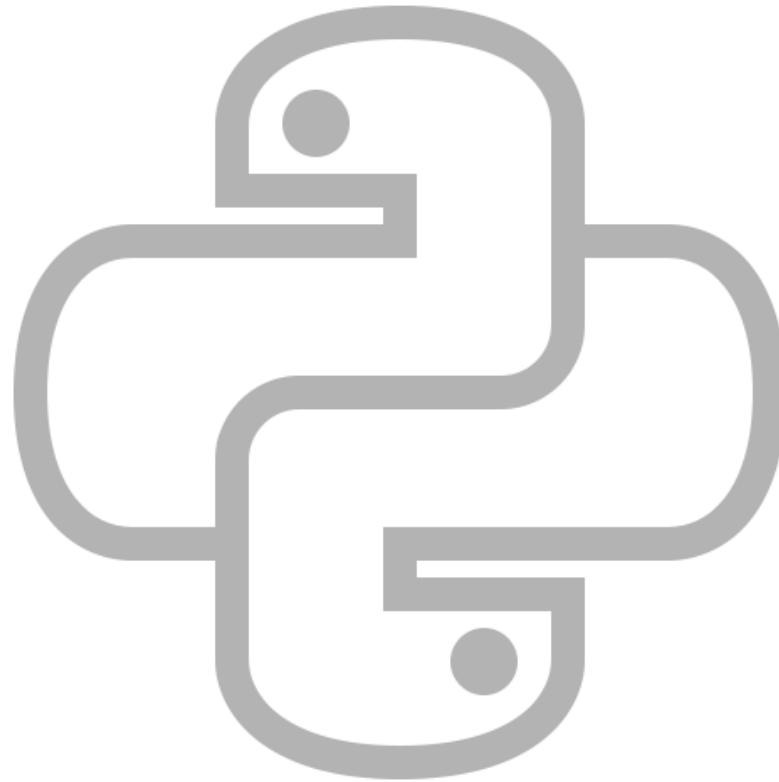
AULA 5: ESTRUTURA DE REPETIÇÃO WHILE

DO QUE VAMOS FALAR

- 1. Estruturas de controle de fluxo
- 2. Estruturas de repetição
- 3. Estrutura de repetição while
- 4. Variável contadora
- 5. Variável acumuladora
- 6. Comando break
- 7. Comando continue
- 8. Representação da estrutura repita... até que...
- 9. Exercícios extras

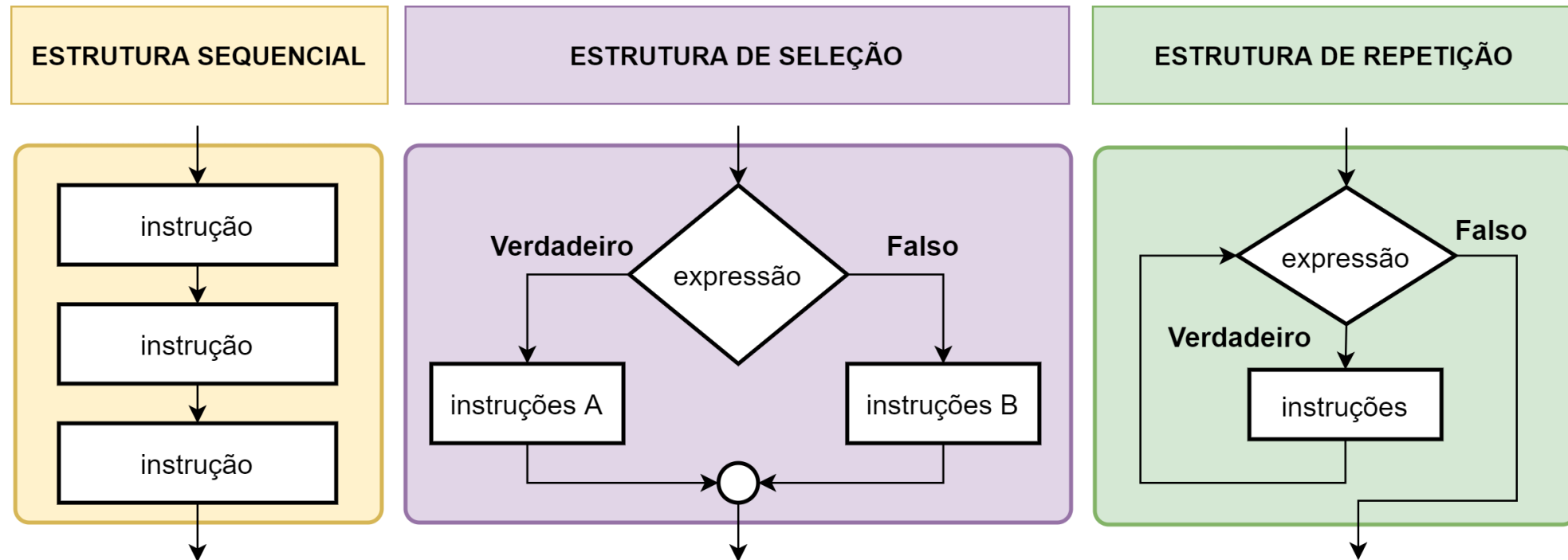


1. ESTRUTURAS DE CONTROLE DE FLUXO

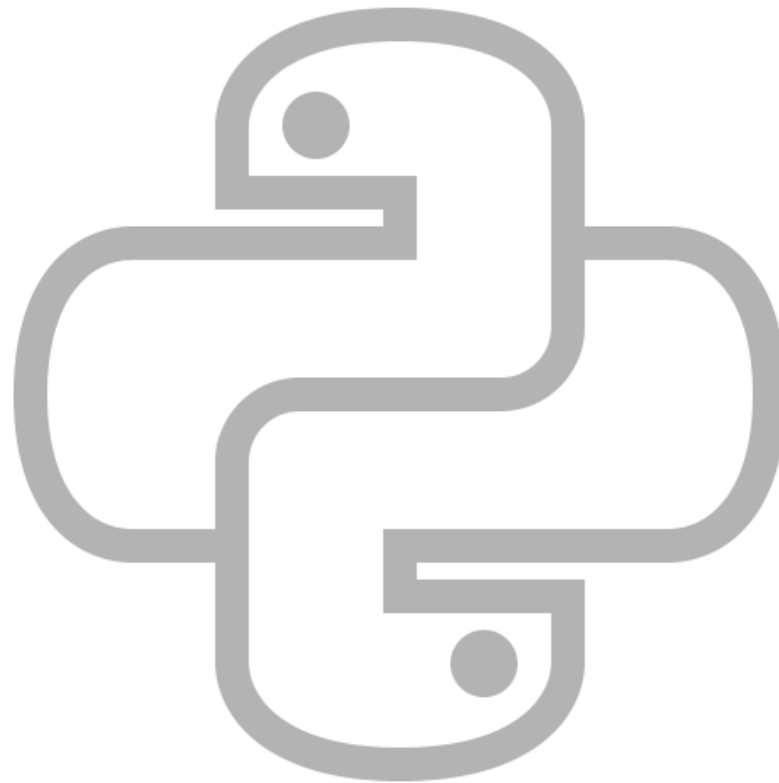


ESTRUTURAS DE CONTROLE DE FLUXO

A ordem em que as instruções são executadas em um programa chama-se fluxo de execução. Existem três estruturas fundamentais para o controle do fluxo de execução:

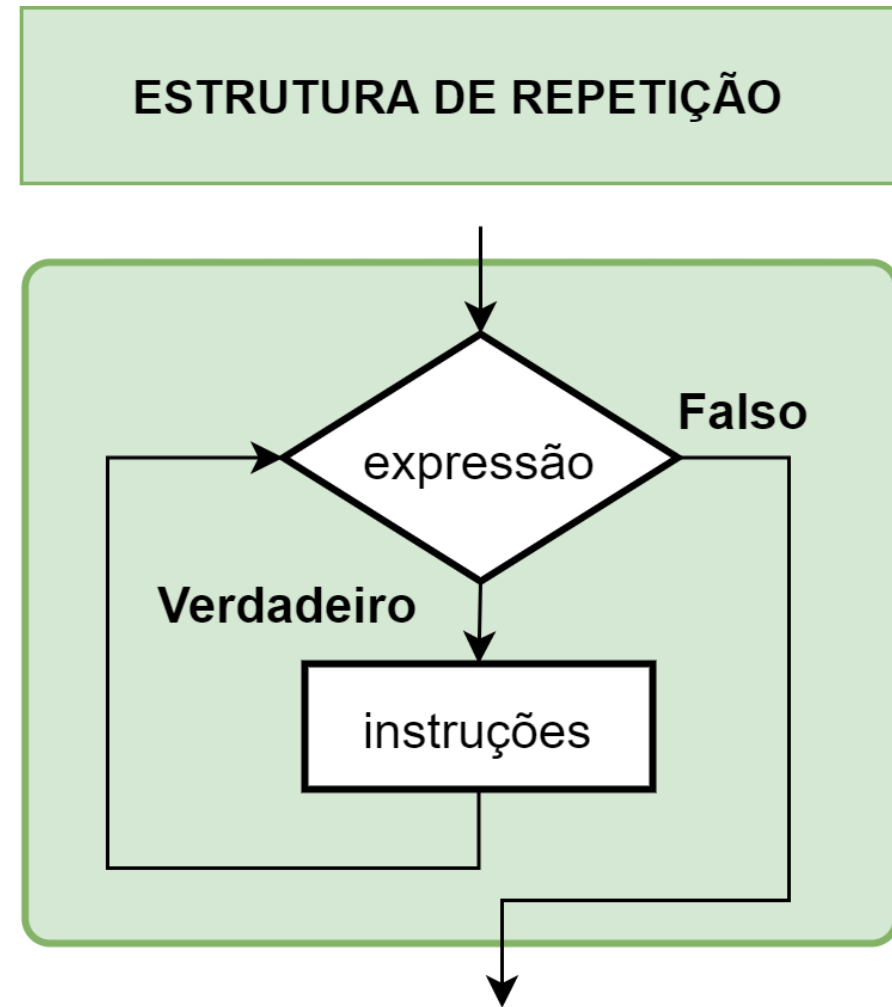


2. ESTRUTURAS DE REPETIÇÃO



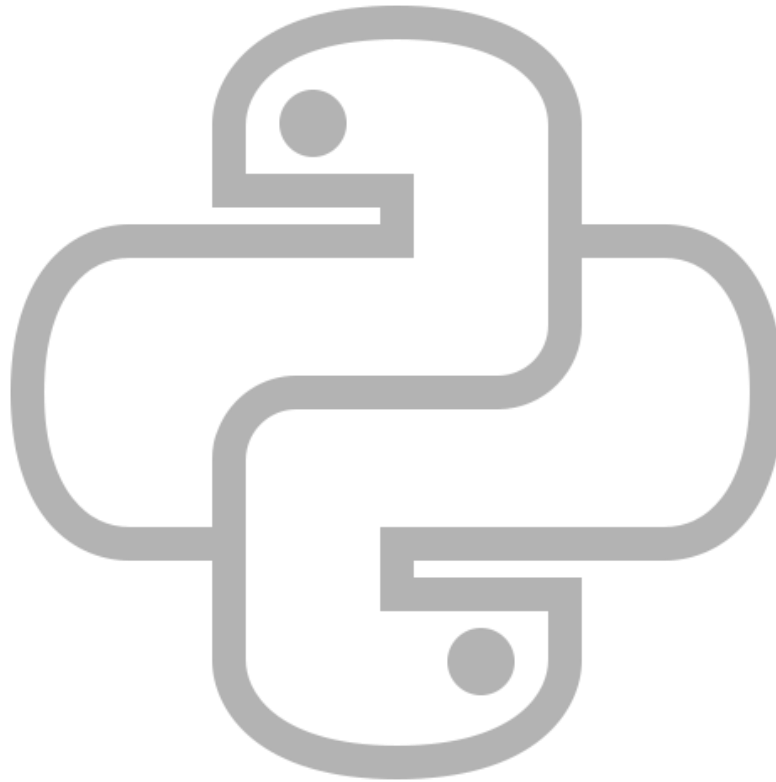
ESTRUTURAS DE REPETIÇÃO

As estruturas de repetição, também conhecidas como estruturas iterativas, **laços de repetição** ou **loops**, permitem que um bloco de instruções seja executado **repetidas vezes** sem que seja necessário escrevê-lo mais do que uma vez.



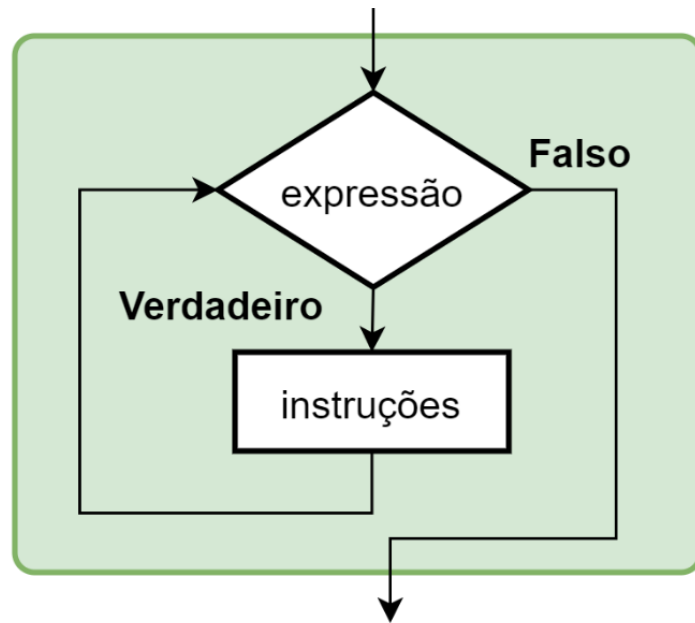
3. ESTRUTURA DE REPETIÇÃO

WHILE



ESTRUTURA DE REPETIÇÃO WHILE

Uma estrutura de repetição **while** executa um bloco de instruções internas repetidamente, enquanto sua expressão de decisão resultar em verdadeiro. Note que a quantidade de repetições pode **não ser conhecida** antecipadamente.




```
...  
while expressão:  
    instruções  
...
```


ESTRUTURA DE REPETIÇÃO WHILE

Quando o laço **while** é executado, sua expressão é avaliada. Se, e somente se, a expressão resultar em verdadeiro o bloco de instruções será executado.

Após a execução do bloco de instruções, o fluxo da execução é deslocado para a expressão que será novamente avaliada. Caso resulte em verdadeiro, o bloco será novamente executado, caso contrário o laço é encerrado.

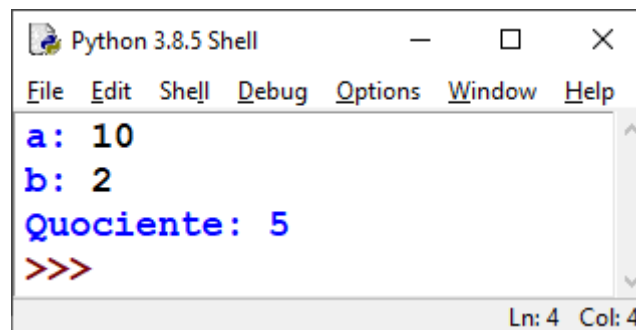
```
instrução A
while expressão:
    instrução X
    instrução Y
    instrução Z
instrução B
```



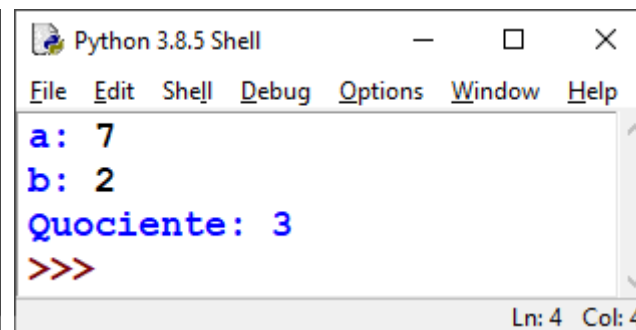
ESTRUTURA DE REPETIÇÃO WHILE

[EXEMPLO 1] Crie um programa que receba dois números inteiros a ($a \geq 0$) e b ($b > 0$) e calcule o quociente da divisão inteira a por b . Observação, não use operadores de divisão, produto e resto.

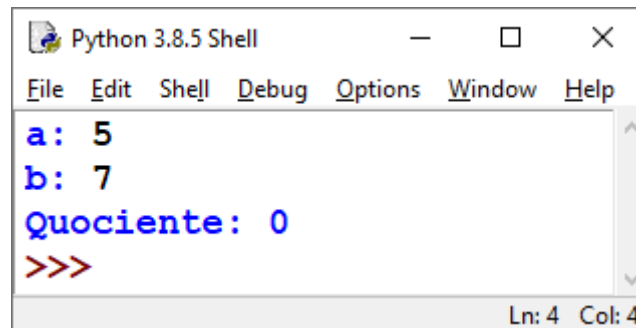
```
a = int(input('a: '))
b = int(input('b: '))
q = 0
while a >= b:
    a -= b
    q += 1
print('Quociente:', q)
```



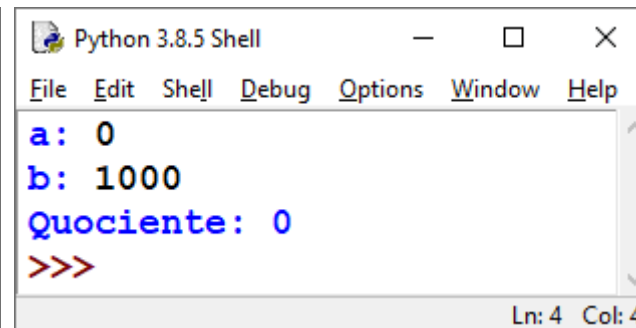
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 10
b: 2
Quociente: 5
>>>
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 7
b: 2
Quociente: 3
>>>
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 5
b: 7
Quociente: 0
>>>
```

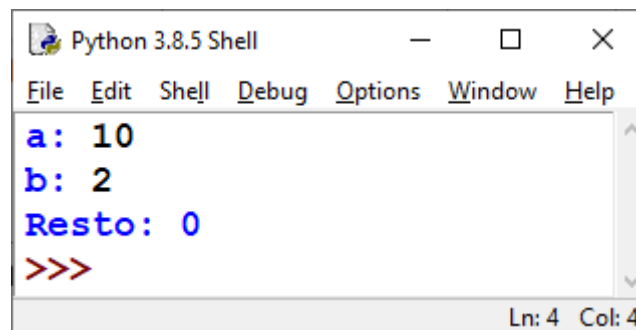
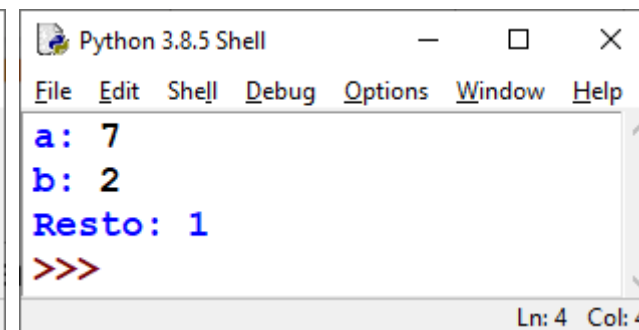
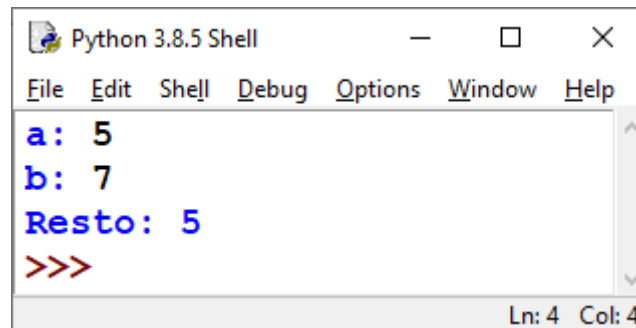
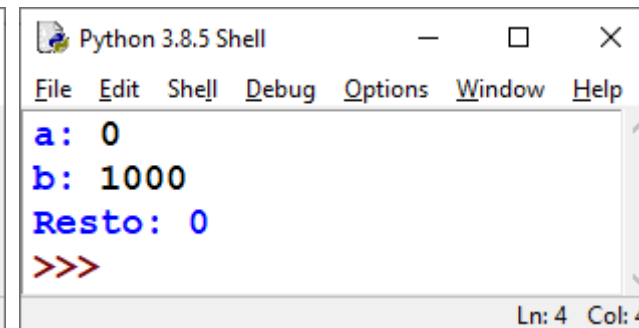


```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 0
b: 1000
Quociente: 0
>>>
```

ESTRUTURA DE REPETIÇÃO WHILE

[EXERCÍCIO 1] Crie um programa que receba dois números inteiros a ($a \geq 0$) e b ($b > 0$) e calcule o resto da divisão inteira a por b . Observação, não use operadores de divisão, produto e resto.

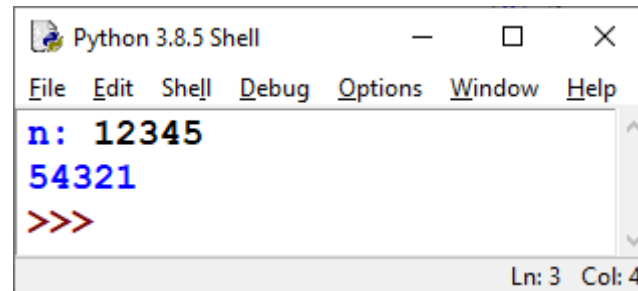
```
a = int(input('a: '))
b = int(input('b: '))
while a >= b:
    a -= b
print('Resto:', a)
```

A screenshot of a Python 3.8.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following text: 'a: 10', 'b: 2', 'Resto: 0', and '>>>' on separate lines. The status bar at the bottom right shows 'Ln: 4 Col: 4'.A screenshot of a Python 3.8.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following text: 'a: 7', 'b: 2', 'Resto: 1', and '>>>' on separate lines. The status bar at the bottom right shows 'Ln: 4 Col: 4'.A screenshot of a Python 3.8.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following text: 'a: 5', 'b: 7', 'Resto: 5', and '>>>' on separate lines. The status bar at the bottom right shows 'Ln: 4 Col: 4'.A screenshot of a Python 3.8.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area contains the following text: 'a: 0', 'b: 1000', 'Resto: 0', and '>>>' on separate lines. The status bar at the bottom right shows 'Ln: 4 Col: 4'.

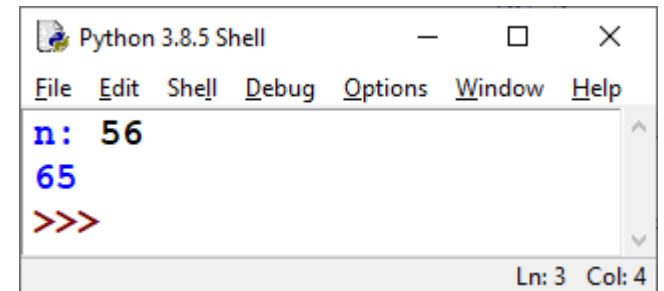
ESTRUTURA DE REPETIÇÃO WHILE

[EXERCÍCIO 2] Crie um programa que receba um número inteiro n ($n > 0$) e exiba os dígitos de n da direita para a esquerda.

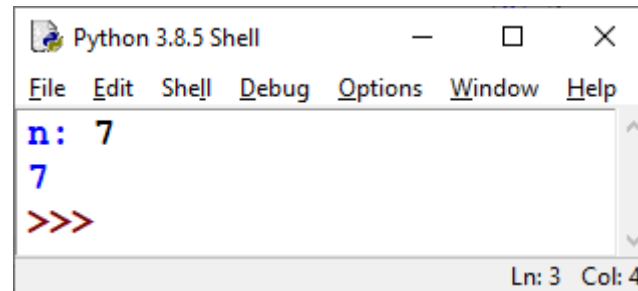
```
n = int(input('n: '))
while n > 0:
    print(n % 10, end=' ')
    n = n // 10
```



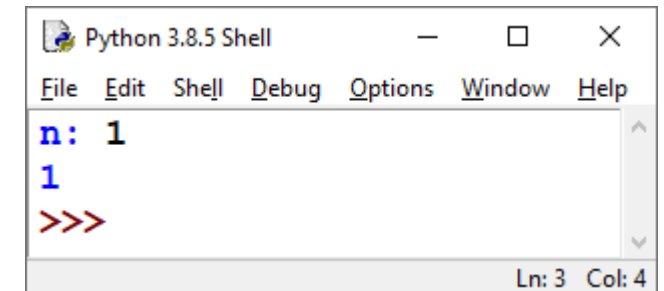
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 12345
54321
>>>
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 56
65
>>>
```

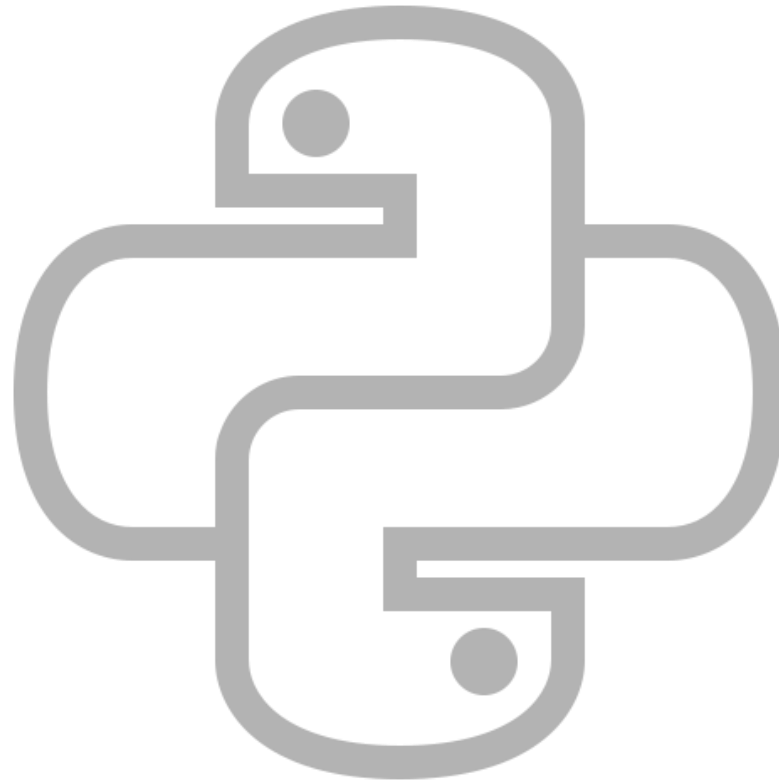


```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 7
7
>>>
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 1
1
>>>
```

4. VARIÁVEL CONTADORA

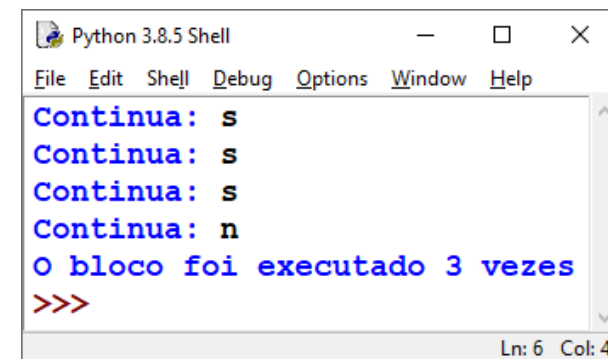


VARIÁVEL CONTADORA

Pode ser usada para **contar quantas vezes** o bloco de instruções do laço foi executado e/ou para **controlar quantas vezes** será executado.

```
continua = input('Continua: ')
contador = 0
while continua == 's':
    contador += 1
    continua = input('Continua: ')
print('O bloco foi executado %d vezes' % contador)
```

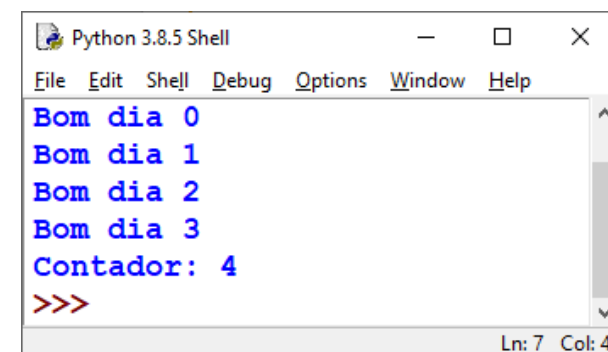
APENAS CONTAR



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Continua: s
Continua: s
Continua: s
Continua: n
O bloco foi executado 3 vezes
>>>
```

```
contador = 0
while contador < 4:
    print('Bom dia %d' % contador)
    contador += 1
print('Contador:', contador)
```

CONTROLAR

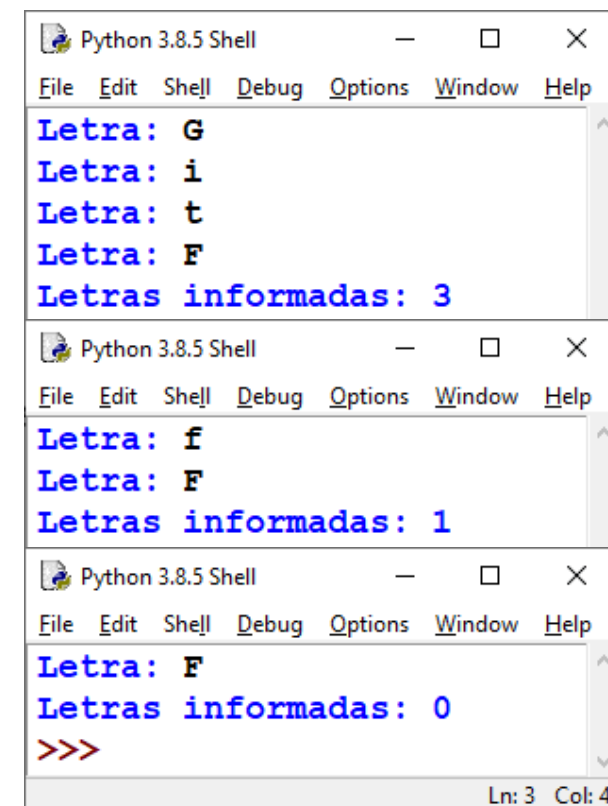


```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Bom dia 0
Bom dia 1
Bom dia 2
Bom dia 3
Contador: 4
>>>
```

VARIÁVEL CONTADORA

[EXEMPLO 2] Crie um programa que receba apenas letras maiúsculas e minúsculas e exiba a quantidade de letras lidas. Observação: as leituras serão encerradas quando a letra maiúscula 'F' for inserida (não deve ser contabilizada).

```
contador = 0
L = input('Letra: ')
while L != 'F':
    contador += 1
    L = input('Letra: ')
print('Letras informadas: %d' % contador)
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Letra: G
Letra: i
Letra: t
Letra: F
Letras informadas: 3

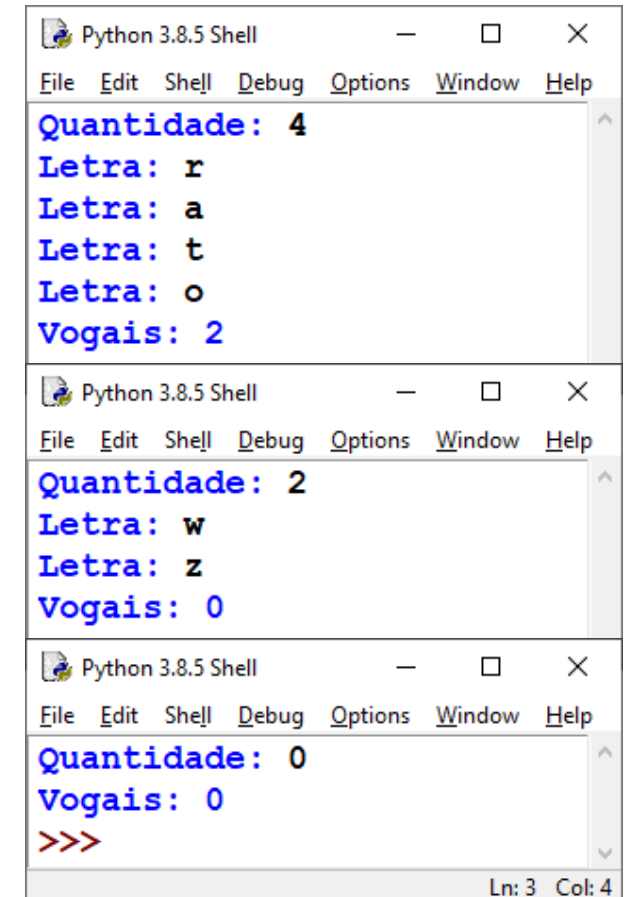
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Letra: f
Letra: F
Letras informadas: 1

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Letra: F
Letras informadas: 0
>>>
Ln: 3 Col: 4
```

VARIÁVEL CONTADORA

[EXEMPLO 3] Crie um programa que receba um natural n seguido de n letras minúsculas e exiba, ao final, a quantidade de vogais lidas.

```
n = int(input('Quantidade: '))
vogais = 0
contador = 0
while contador < n:
    L = input('Letra: ')
    if L=='a' or L=='e' or L=='i' or L=='o' or L=='u':
        vogais += 1
    contador += 1
print('Vogais: %d' % vogais)
```



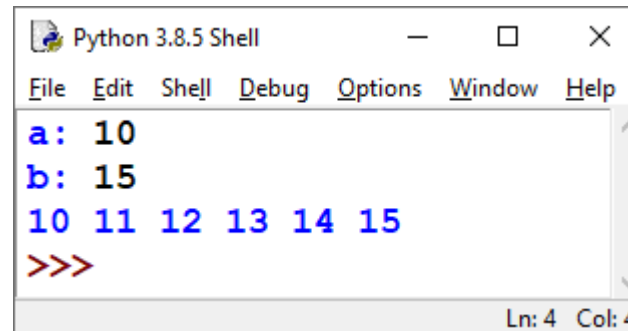
The image shows three sequential screenshots of a Python 3.8.5 Shell window, demonstrating the program's execution for different inputs.

- First Screenshot:** The user enters '4' for the quantity and then the letters 'r', 'a', 't', and 'o'. The program outputs 'Vogais: 2'.
- Second Screenshot:** The user enters '2' for the quantity and then the letters 'w' and 'z'. The program outputs 'Vogais: 0'.
- Third Screenshot:** The user enters '0' for the quantity. The program outputs 'Vogais: 0' and ends with the prompt '>>>>'.

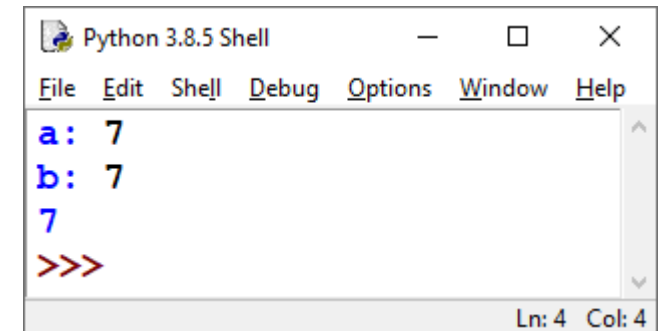
VARIÁVEL CONTADORA

[EXERCÍCIO 3] Crie um programa que receba dois números inteiros a e b e exiba todos os inteiros do intervalo crescente $[a..b]$.

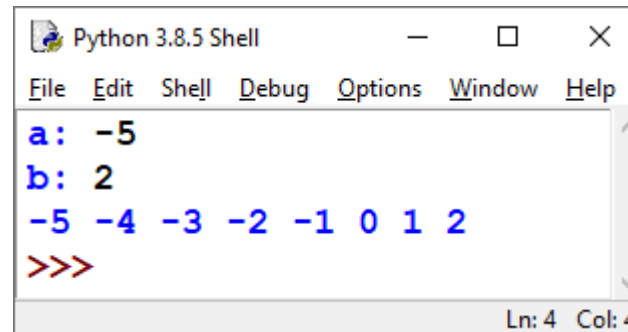
```
a = int(input('a: '))
b = int(input('b: '))
i = a
while i<=b:
    print(i, end=' ')
    i += 1
```



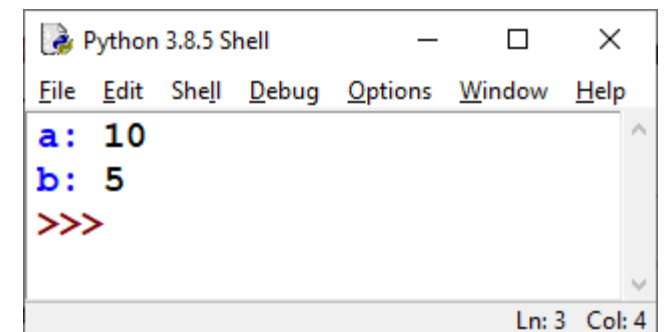
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 10
b: 15
10 11 12 13 14 15
>>>
Ln: 4 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 7
b: 7
7
>>>
Ln: 4 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: -5
b: 2
-5 -4 -3 -2 -1 0 1 2
>>>
Ln: 4 Col: 4
```

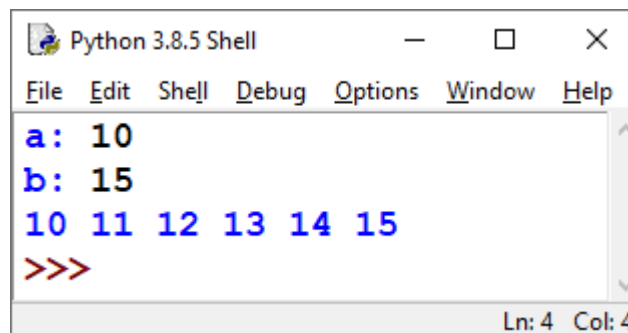


```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 10
b: 5
>>>
Ln: 3 Col: 4
```

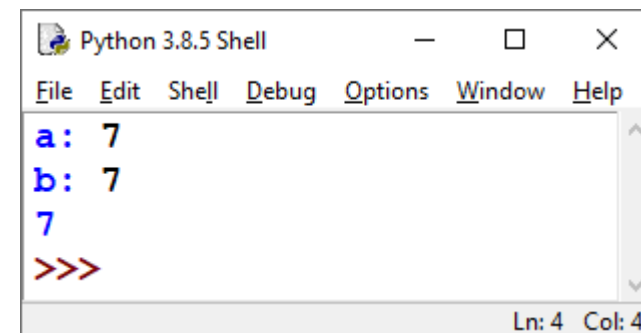
VARIÁVEL CONTADORA

[EXERCÍCIO 4] Crie um programa que receba dois números inteiros a e b e exiba todos os inteiros do intervalo crescente $[a..b]$. Use apenas duas variáveis.

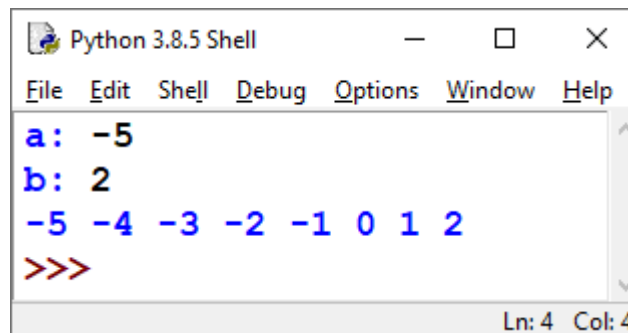
```
a = int(input('a: '))
b = int(input('b: '))
while a <= b:
    print(a, end=' ')
    a += 1
```



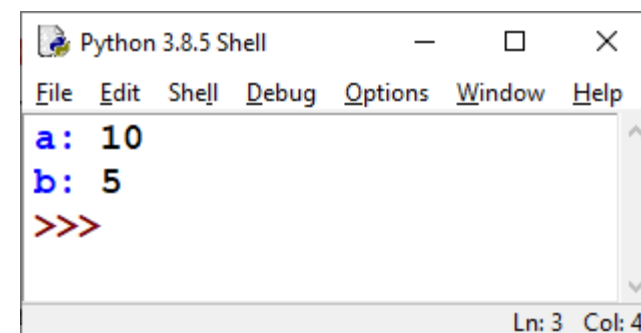
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 10
b: 15
10 11 12 13 14 15
>>>
Ln: 4 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 7
b: 7
7
>>>
Ln: 4 Col: 4
```

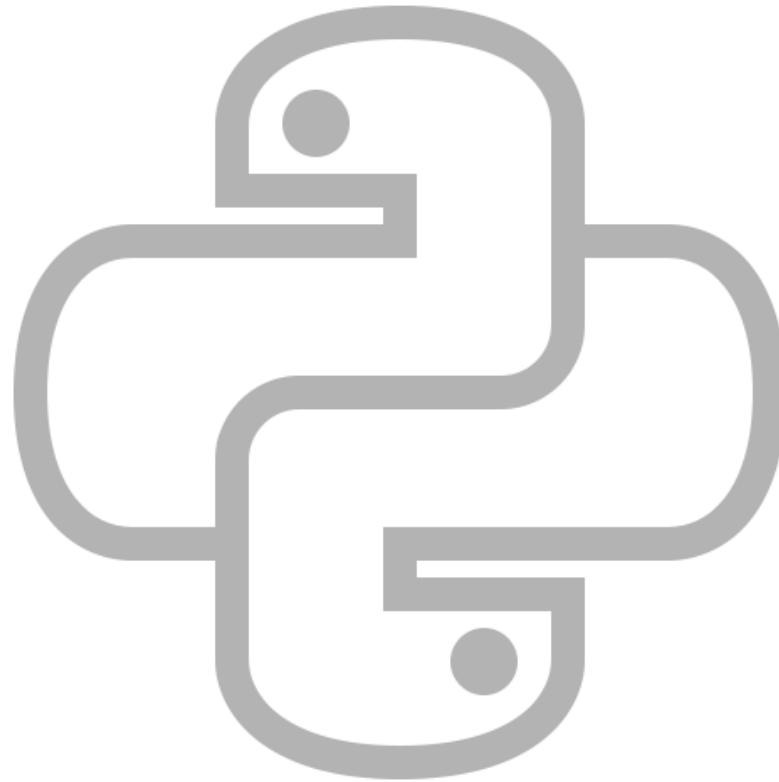


```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: -5
b: 2
-5 -4 -3 -2 -1 0 1 2
>>>
Ln: 4 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
a: 10
b: 5
>>>
Ln: 3 Col: 4
```

5. VARIÁVEL ACUMULADORA

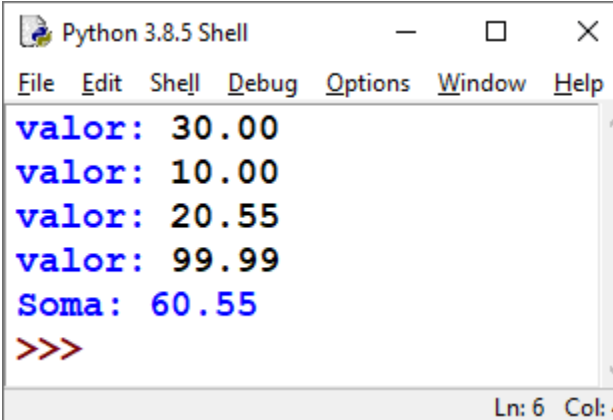


VARIÁVEL ACUMULADORA

Uma variável acumuladora é geralmente usada para **acumular valores** que **não são constantes**, ou seja, seu incremento é variável.

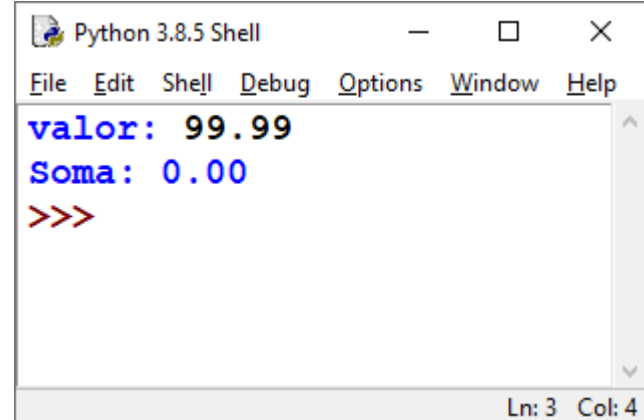
[EXEMPLO 4] Crie um programa que receba valores reais e ao final exiba a soma dos valores lidos. Observação: o valor 99.99 encerra as entradas e não deve ser contabilizado.

```
acumulador = 0.0
x = float(input('valor: '))
while x != 99.99:
    acumulador += x
    x = float(input('valor: '))
print('Soma: %.2f' % acumulador)
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
valor: 30.00
valor: 10.00
valor: 20.55
valor: 99.99
Soma: 60.55
>>>
```

Ln: 6 Col: 4



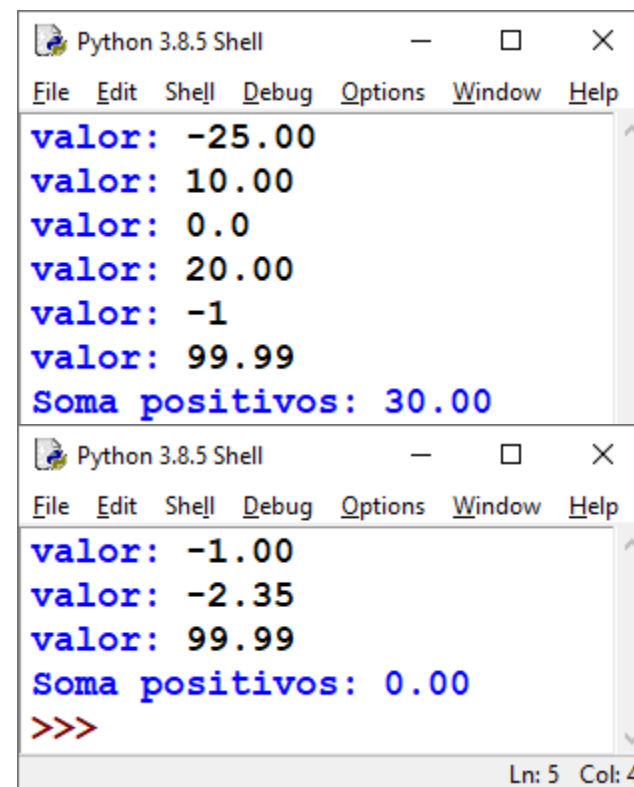
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
valor: 99.99
Soma: 0.00
>>>
```

Ln: 3 Col: 4

VARIÁVEL ACUMULADORA

[EXERCÍCIO 5] Crie um programa que receba valores reais e ao final exiba a soma dos valores positivos lidos. Observação: o valor 99.99 encerra as entradas e não deve ser contabilizado.

```
acumulador = 0
x = float(input('valor: '))
while x != 99.99:
    if x > 0:
        acumulador += x
    x = float(input('valor: '))
print('Soma positivos: %.2f' % acumulador)
```



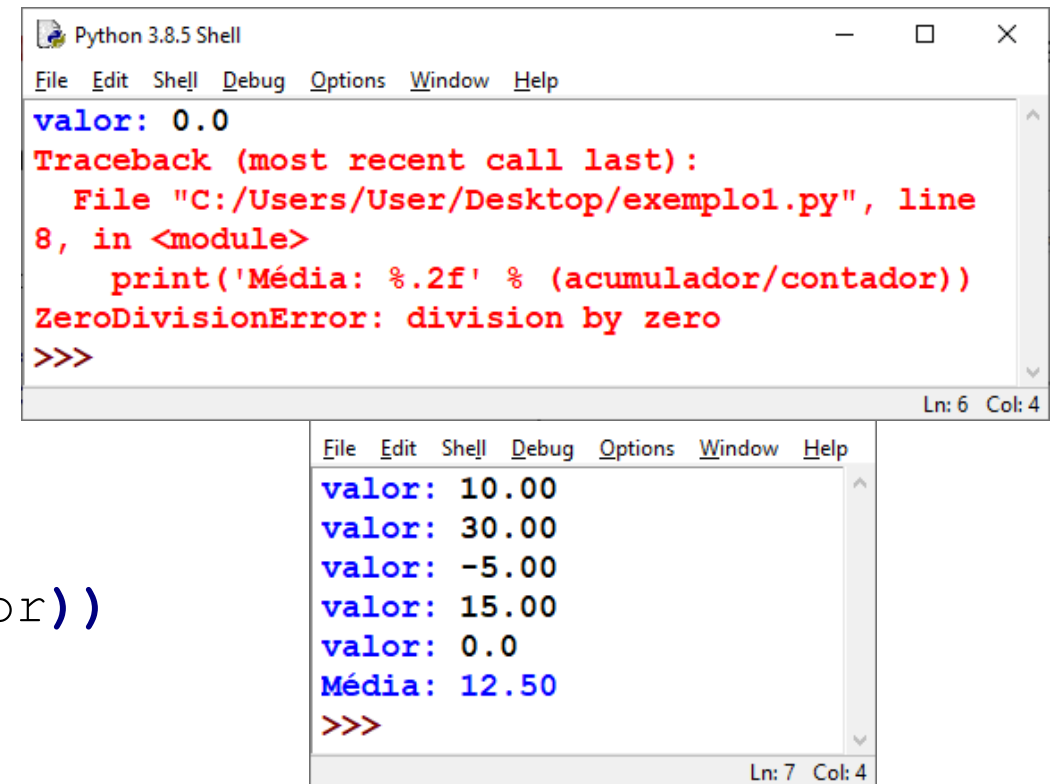
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
valor: -25.00
valor: 10.00
valor: 0.0
valor: 20.00
valor: -1
valor: 99.99
Soma positivos: 30.00

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
valor: -1.00
valor: -2.35
valor: 99.99
Soma positivos: 0.00
>>>
Ln: 5 Col: 4
```

VARIÁVEL ACUMULADORA

[EXERCÍCIO 6] Crie um programa que receba valores reais e ao final exiba a média aritmética simples dos valores lidos. Observação: o valor 0.0 encerra as entradas e não deve ser contabilizado.

```
acumulador = 0
contador = 0
x = float(input('valor: '))
while x != 0.0:
    acumulador += x
    contador += 1
    x = float(input('valor: '))
print('Média: %.2f' % (acumulador/contador))
```



Python 3.8.5 Shell

File Edit Shell Debug Options Window Help

valor: 0.0

Traceback (most recent call last):

File "C:/Users/User/Desktop/exemplo1.py", line 8, in <module>

print('Média: %.2f' % (acumulador/contador))

ZeroDivisionError: division by zero

>>>

Ln: 6 Col: 4

File Edit Shell Debug Options Window Help

valor: 10.00

valor: 30.00

valor: -5.00

valor: 15.00

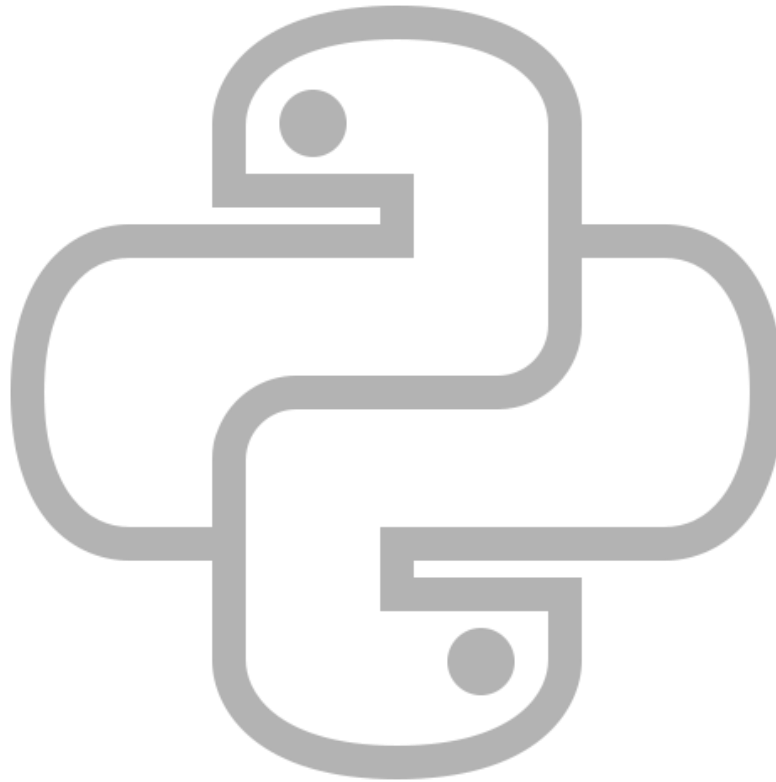
valor: 0.0

Média: 12.50

>>>

Ln: 7 Col: 4

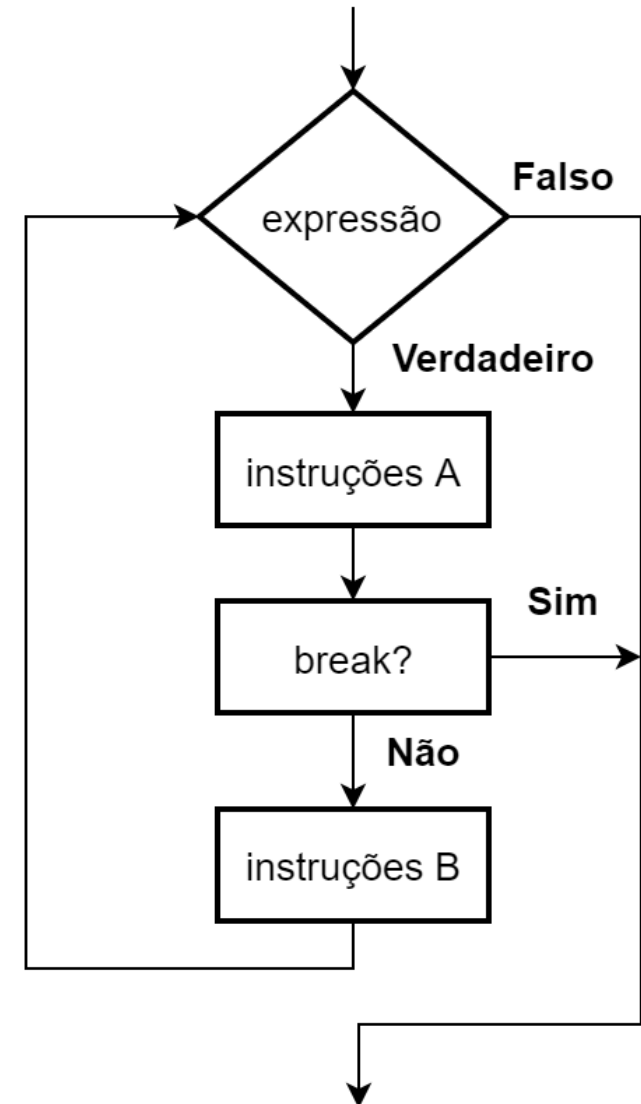
6. COMANDO BREAK



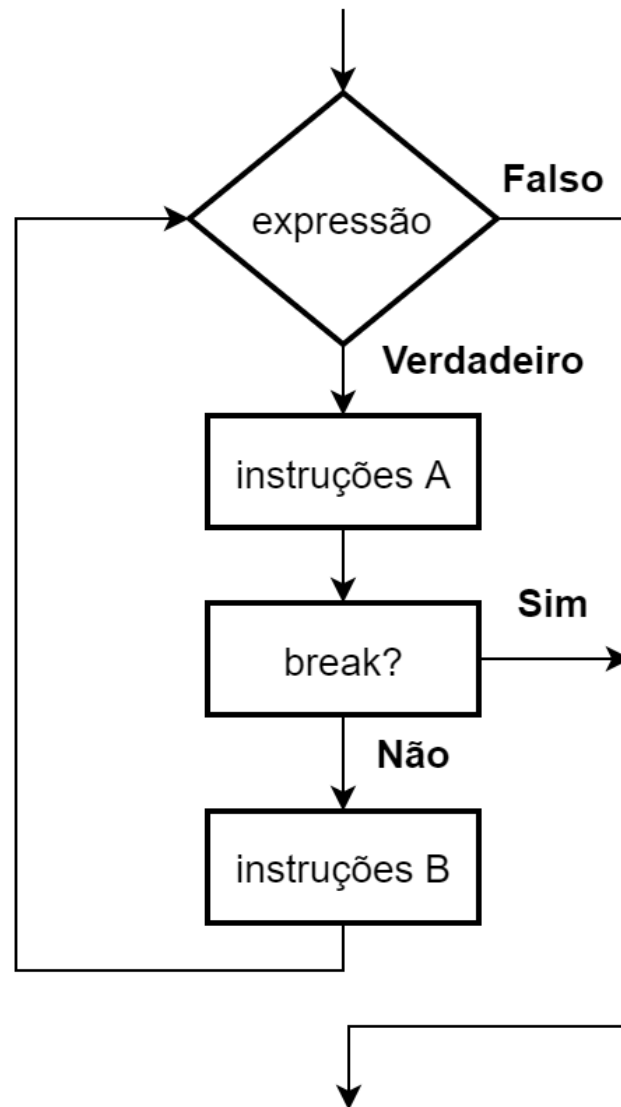
COMANDO BREAK

O comando **break** encerra a execução do laço de repetição mais interno em que está contido.

```
...  
while expressão:  
    instruções A  
    break?  
    instruções B  
...
```



COMANDO BREAK



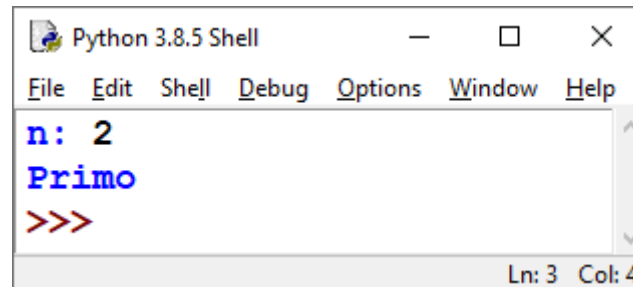
Caso o comando **break** seja executado o laço é encerrado e o fluxo de execução será direcionado para a próxima instrução fora dele.

Portanto, é comum que o comando **break** esteja dentro do bloco de instruções de uma estrutura de seleção, de modo que só seja executado se uma condição específica for satisfeita.

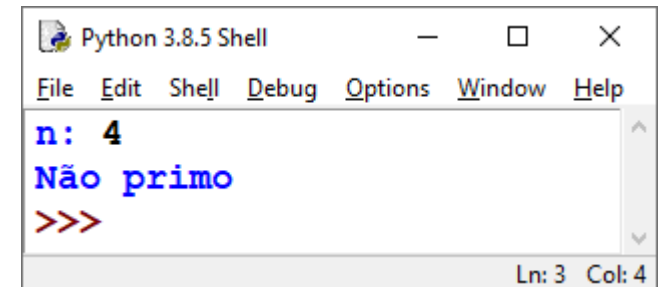
COMANDO BREAK

[EXEMPLO 5] Crie um programa que receba um número natural n ($n > 1$) e exiba uma mensagem indicando se n é primo.

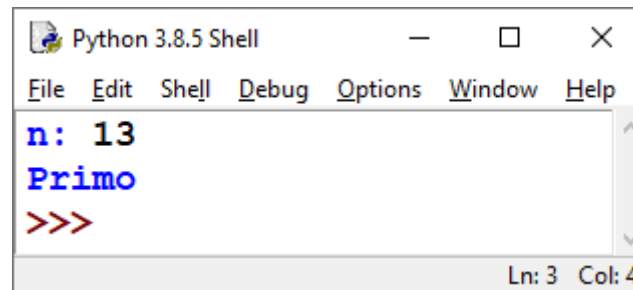
```
n = int(input('n: '))
d = 2
while d < n:
    if n%d == 0:
        break
    d += 1
if d==n:
    print('Primo')
else:
    print('Não primo')
```



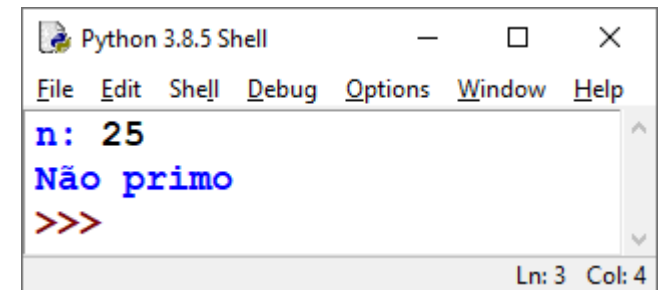
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 2
Primo
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 4
Não primo
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 13
Primo
>>>
Ln: 3 Col: 4
```



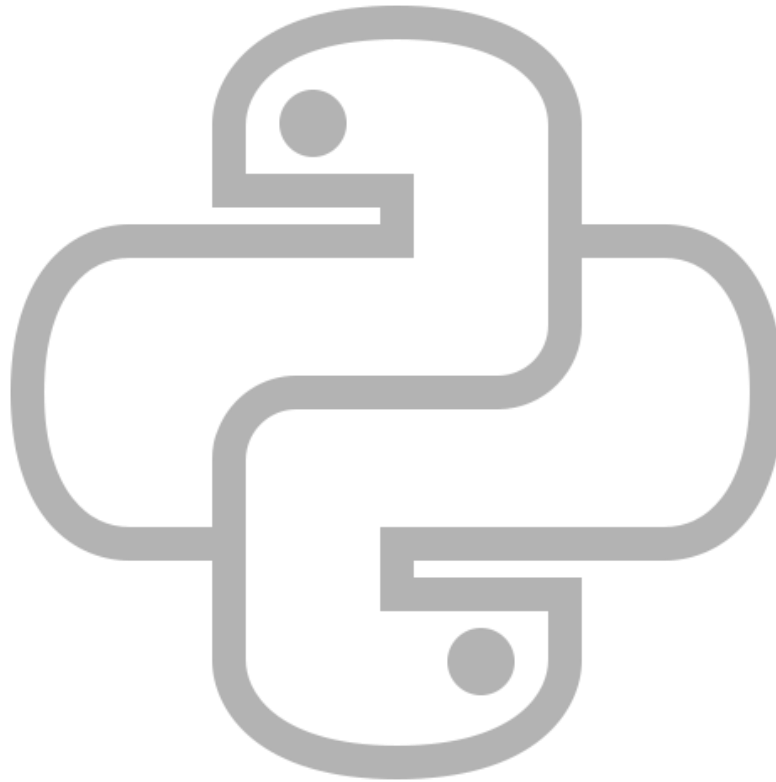
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 25
Não primo
>>>
Ln: 3 Col: 4
```

COMANDO BREAK

[EXEMPLO 6] Crie um programa que receba dois números inteiros *a* e *b* e um caractere *op* representando um operador aritmético (+, -, * ou /) e exiba o inteiro resultante da expressão *a op b*. O programa deve executar o procedimento enquanto o usuário desejar ou até a ocorrência de uma divisão inválida.

```
e = input('Calcular? ')
while e=='s':
    a = int(input('a: '))
    b = int(input('b: '))
    op = input('operador: ')
    if op=='+': r = a+b
    elif op=='-': r = a-b
    elif op=='*': r = a*b
    elif op=='/':
        if b==0:
            print('Divisão por zero!')
            break
        else:
            r = a/b
    print('%d %c %d = %d\n' % (a,op,b,r))
    e = input('Calcular? ')
```

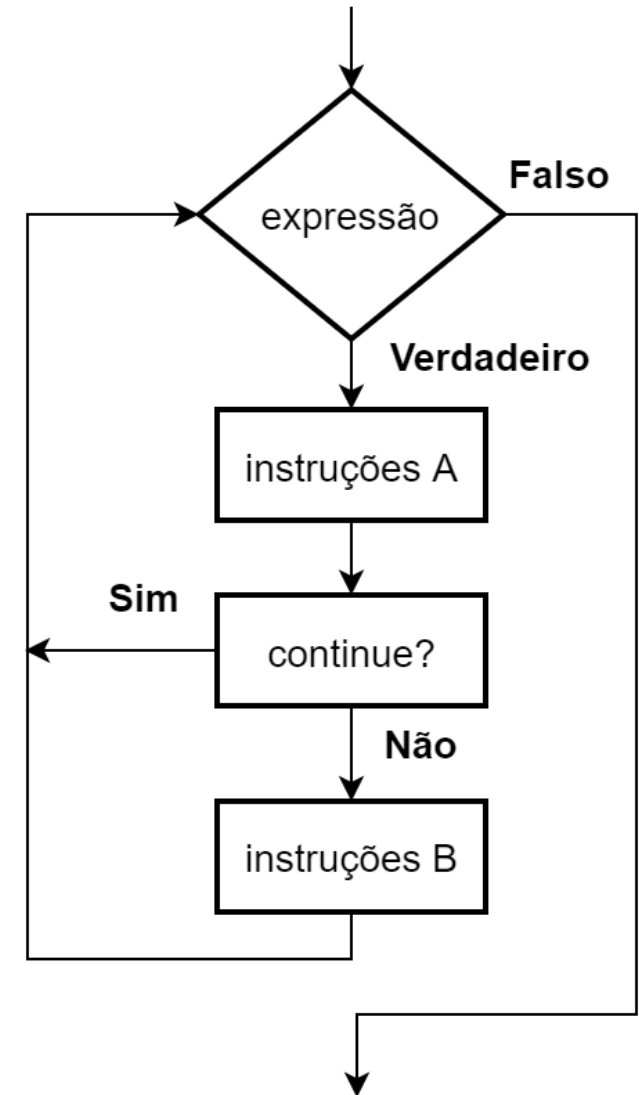
7. COMANDO CONTINUE



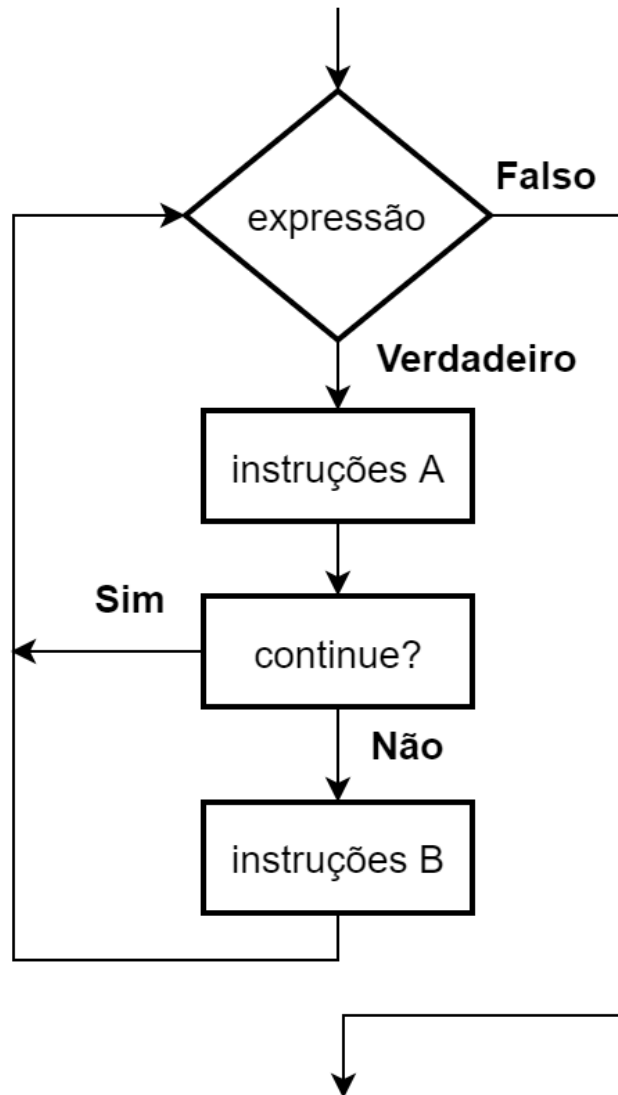
COMANDO CONTINUE

O comando **continue** desloca o fluxo de execução para a expressão de decisão do laço de repetição mais interno em que está contido. Ou seja, quando executado, encerra a repetição atual e passa para a próxima.

```
...  
while expressão:  
    instruções A  
    continue?  
    instruções B  
...
```



COMANDO CONTINUE

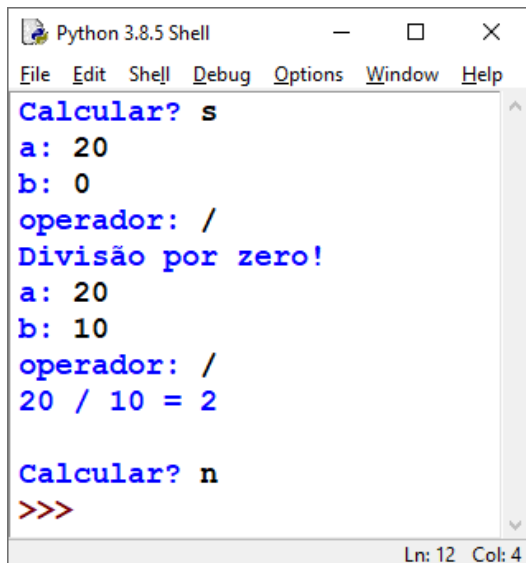


Caso o comando **continue** seja executado a rodada atual do laço é interrompida e o fluxo de execução será direcionado para a condição do *loop*.

Portanto, é comum que o comando **continue** esteja dentro do bloco de instruções de uma estrutura de seleção, de modo que só seja executado se uma condição específica for satisfeita.

COMANDO CONTINUE

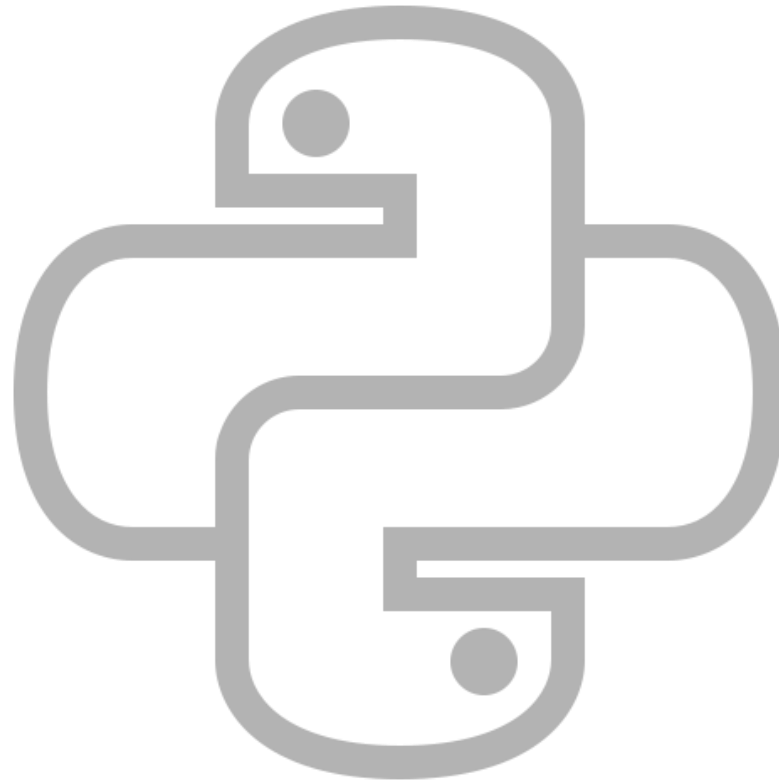
[EXERCÍCIO 7] Altere o programa do Exemplo 6 para que caso ocorra uma divisão inválida o usuário possa inserir novos operandos e continuar a execução.



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Calcular? s
a: 20
b: 0
operador: /
Divisão por zero!
a: 20
b: 10
operador: /
20 / 10 = 2
Calcular? n
>>>
Ln: 12 Col: 4
```

```
e = input('Calcular? ')
while e=='s':
    a = int(input('a: '))
    b = int(input('b: '))
    op = input('operador: ')
    if op=='+': r = a+b
    elif op=='-': r = a-b
    elif op=='*': r = a*b
    elif op=='/':
        if b==0:
            print('Divisão por zero!')
            continue
        else:
            r = a/b
    print('%d %c %d = %d\n' % (a,op,b,r))
    e = input('Calcular? ')
```

8. REPRESENTAÇÃO DA ESTRUTURA REPITA... ATÉ QUE...



REPRESENTAÇÃO DA ESTRUTURA REPITA... ATÉ QUE...

Há situações em que um bloco de instruções de um laço deve ser executado **pelo menos uma vez** e que a verificação que decidirá se ele será executado novamente deve ficar no **final do bloco** e não antes do começo.

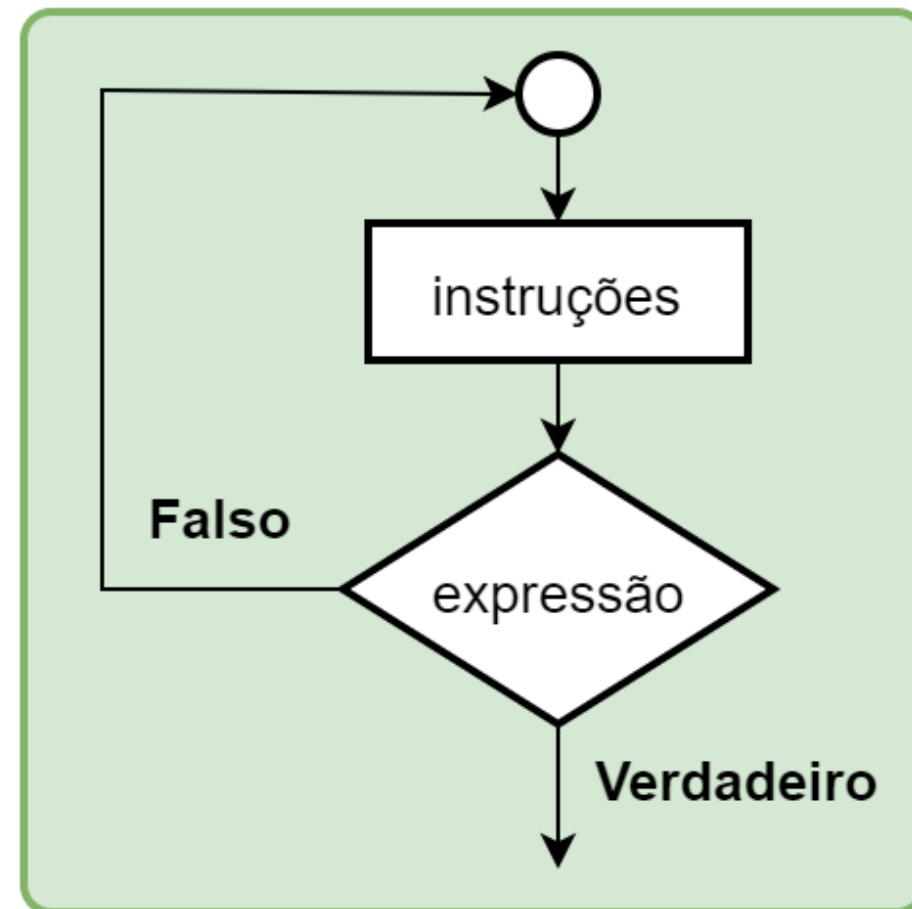
...

```
while True:
```

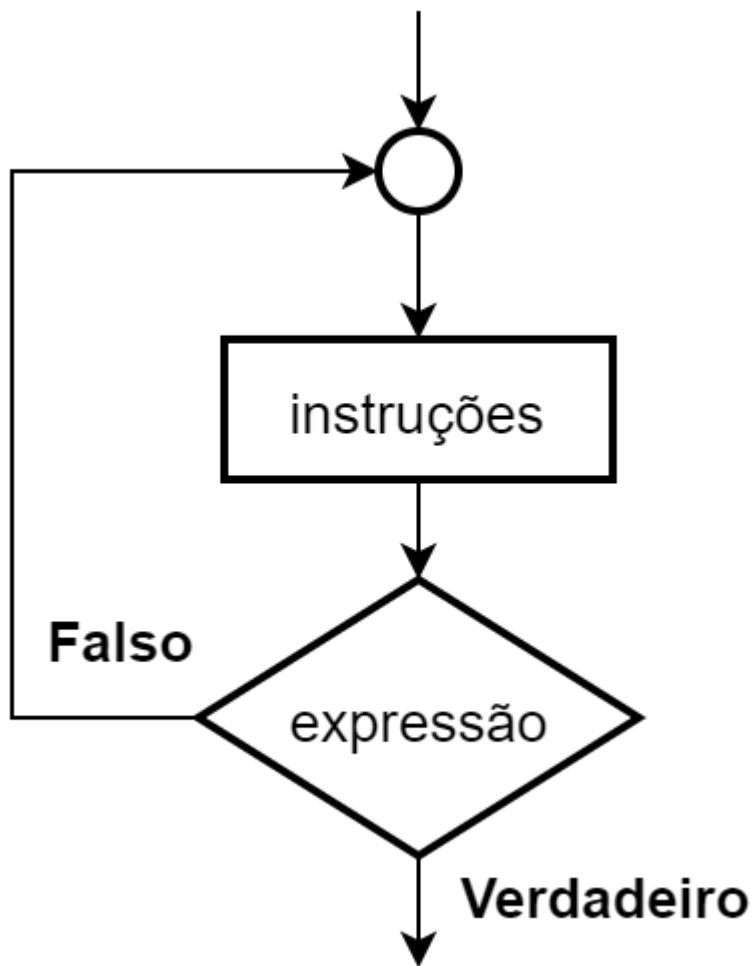
```
    instruções
```

```
    if expressão: break
```

...



REPRESENTAÇÃO DA ESTRUTURA REPITA... ATÉ QUE...



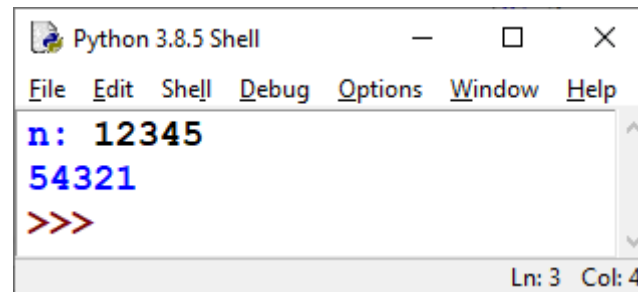
O laço **repeat...until** possui uma demarcação inicial, para indicar seu início, seguida por um bloco de instruções e, por fim, sua expressão de decisão, ou seja, a condição do *loop*, que também indica seu fim.

Após a primeira execução do bloco de instruções, a condição é avaliada, caso resulte em **falso** o fluxo de execução é direcionado para a demarcação inicial, caso resulte em **verdadeiro**, o laço é encerrado.

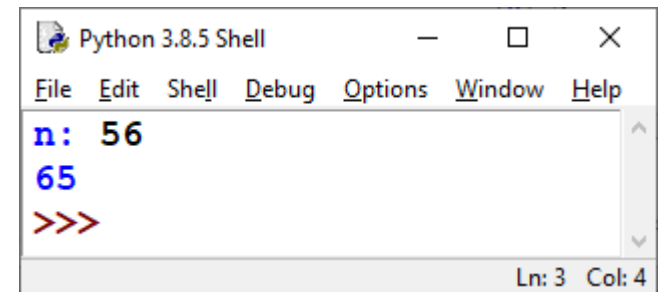
REPRESENTAÇÃO DA ESTRUTURA REPITA... ATÉ QUE...

[EXEMPLO 7] Crie um programa que receba um número inteiro n ($n \geq 0$) e exiba os dígitos de n da direita para a esquerda.

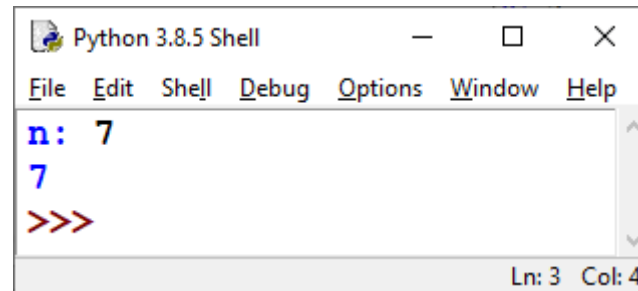
```
n = int(input('n: '))
while True:
    print(n % 10, end=' ')
    n = n // 10
    if n==0: break
```



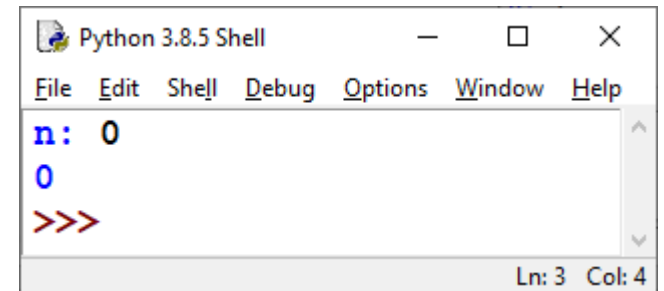
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 12345
54321
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 56
65
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 7
7
>>>
Ln: 3 Col: 4
```



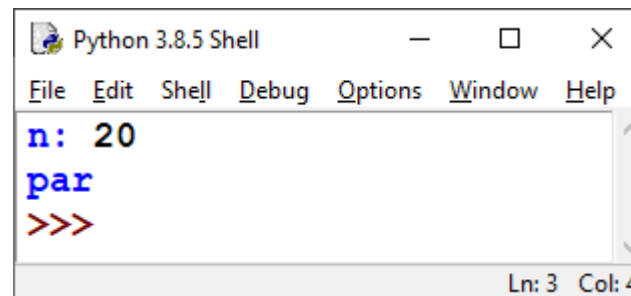
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 0
0
>>>
Ln: 3 Col: 4
```

REPRESENTAÇÃO DA ESTRUTURA REPITA... ATÉ QUE...

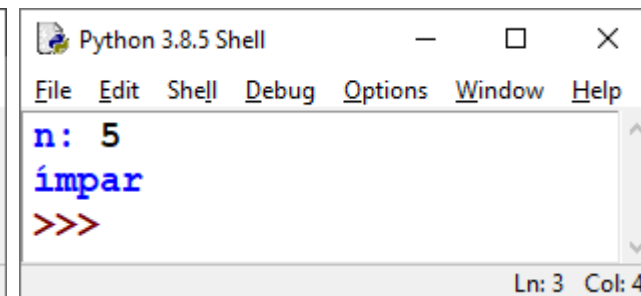
[EXERCÍCIO 8] Crie um programa que receba um número inteiro n e, caso n seja um número natural, exiba uma mensagem indicando se n é par ou ímpar. Enquanto n for um número negativo, repita a solicitação de entrada.

```
while True:
    n = int(input('n: '))
    if n >= 0: break

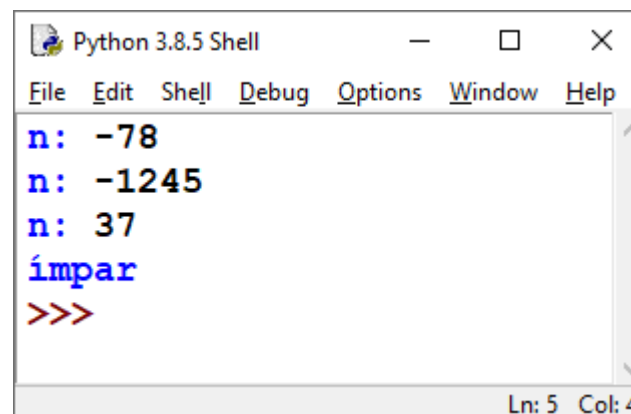
if n % 2 == 0:
    print('par')
else:
    print('ímpar')
```



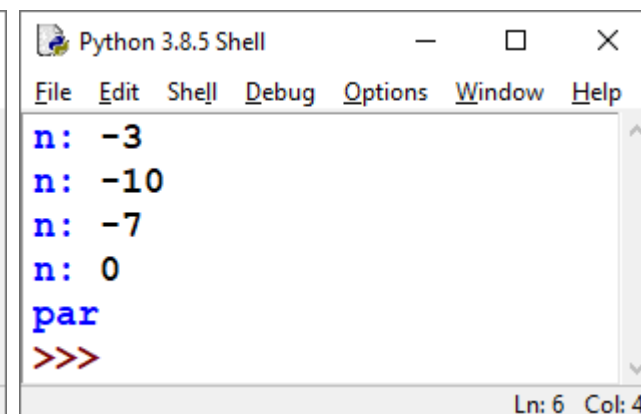
```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 20
par
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: 5
ímpar
>>>
Ln: 3 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: -78
n: -1245
n: 37
ímpar
>>>
Ln: 5 Col: 4
```



```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
n: -3
n: -10
n: -7
n: 0
par
>>>
Ln: 6 Col: 4
```

9. EXERCÍCIOS EXTRAS

