

ESTRUTURA DE REPETIÇÃO

INTRODUÇÃO

O QUE É UMA ESTRUTURA DE REPETIÇÃO

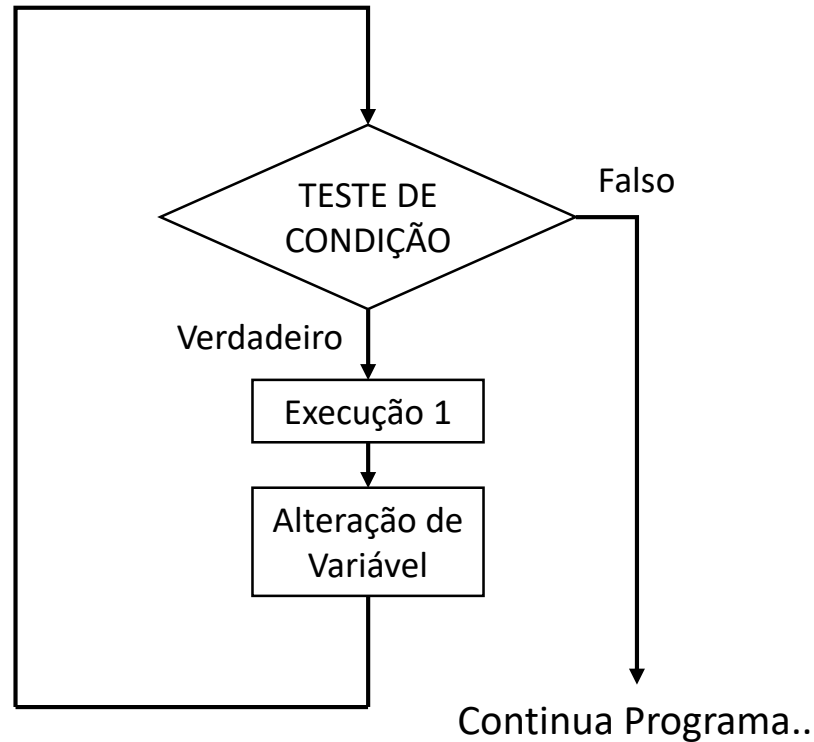
- As **estruturas de repetição** permitem executar mais de uma vez um mesmo trecho de código.
- Trata-se de uma forma de **executar blocos de comandos somente sob determinadas condições**, mas com a opção de **repetir o mesmo bloco quantas vezes for necessário**.
- As estruturas de repetição são úteis, por exemplo, para repetir uma série de operações semelhantes que são executadas para todos os elementos de uma lista ou de uma tabela de dados, ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.



As estruturas de repetições são também conhecidas como **LOOPS**

INTRODUÇÃO

- As estruturas de repetição são úteis, por exemplo, para repetir uma série de operações semelhantes que são executadas para todos os elementos de uma lista ou de uma tabela de dados, ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.



The background of the slide is a blue-tinted image of a complex electronic circuit board. It features various components such as integrated circuits, resistors, and capacitors, connected by a dense network of white and yellow traces. The top half of the image is slightly blurred, while the bottom half is in sharper focus, showing the intricate details of the board's layout.

ESTRUTURA DE REPETIÇÃO WHILE

ESTRUTURA DE REPETIÇÃO – WHILE

- A estrutura de repetição while, repete um bloco enquanto a condição for verdadeira.

SINTAXE:

```
while CONDIÇÃO:  
    BLOCO DE EXECUÇÃO  
    CONTADOR
```

Onde:

- **CONDIÇÃO:** na qual será testada sempre que for executado uma repetição. Enquanto for verdadeiro, o programa será repetido.
- **BLOCO DE EXECUÇÃO:** Comandos que serão executados em cada repetição.
- **CONTADOR:** O contador necessita que seja alterado em cada loop para que essa variável seja utilizada na condição de teste. Caso contrario teremos um loop infinito (sempre verdadeiro) ou loop no qual nunca será executado (sempre falso).

ESTRUTURA DE REPETIÇÃO – WHILE

EXEMPLO: Imprima a tabuada do numero 6 de 0 a 10.

```
prog_teste.py - C:/Users/hel...
File Edit Format Run Options Window Help
# EXEMPLO 1 - Tabuada do 7 de 0 a 10
# Metodo print sem loop
print(f" 6 x 0 = {6*0}")
print(f" 6 x 1 = {6*1}")
print(f" 6 x 2 = {6*2}")
print(f" 6 x 3 = {6*3}")
print(f" 6 x 4 = {6*4}")
print(f" 6 x 5 = {6*5}")
print(f" 6 x 6 = {6*6}")
print(f" 6 x 7 = {6*7}")
print(f" 6 x 8 = {6*8}")
print(f" 6 x 9 = {6*9}")
print(f" 6 x 10 = {6*10}")
Ln: 10 Col: 0
```

```
prog_teste.py - C:/Users/hel...
File Edit Format Run Options Window Help
# EXEMPLO 1 - Tabuada do 7 de 0 a 10
# Metodo COM Loop - WHILE
x = 0
while x <= 10:
    print(f" 6 x {x} = {6*x}")
    x = x + 1
Ln: 7 Col: 0
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
6 x 0 = 0
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
>>>
Ln: 16 Col: 4
```

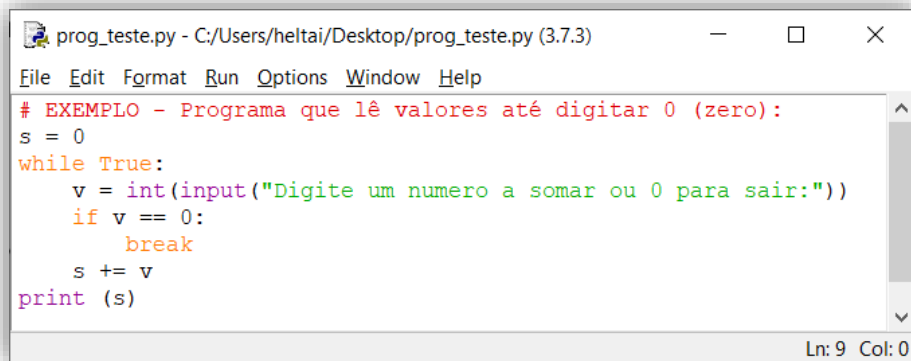



ESTRUTURA DE REPETIÇÃO INTERROMPENDO A REPETIÇÃO

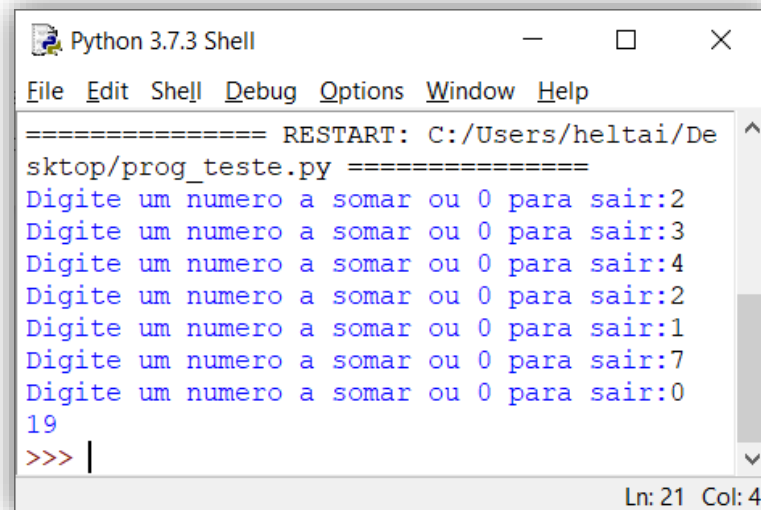
INTERROMPENDO A REPETIÇÃO

- A estrutura **while** só verifica sua condição de parada no início de cada repetição. Dependendo do problema, a habilidade de terminar **while** dentro do bloco a repetir pode ser interessante.

EXEMPLO: Programa que lê valores até digitar 0 (zero):



```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO - Programa que lê valores até digitar 0 (zero):
s = 0
while True:
    v = int(input("Digite um numero a somar ou 0 para sair:"))
    if v == 0:
        break
    s += v
print (s)
Ln: 9 Col: 0
```



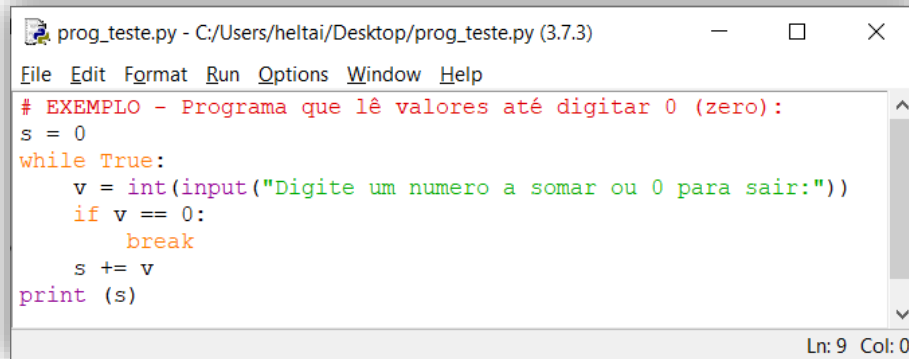
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai/De
sktop/prog_teste.py =====
Digite um numero a somar ou 0 para sair:2
Digite um numero a somar ou 0 para sair:3
Digite um numero a somar ou 0 para sair:4
Digite um numero a somar ou 0 para sair:2
Digite um numero a somar ou 0 para sair:1
Digite um numero a somar ou 0 para sair:7
Digite um numero a somar ou 0 para sair:0
19
>>> |
Ln: 21 Col: 4
```


INTERROMPENDO A REPETIÇÃO

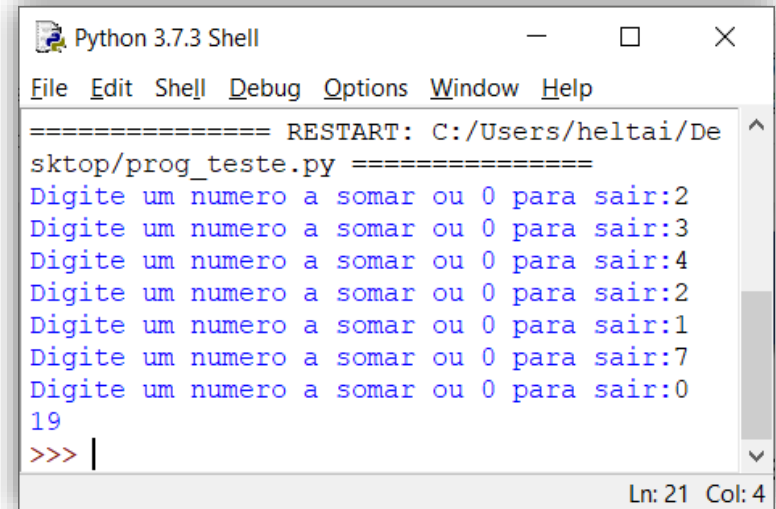
FUNÇÃO BREAK E CONTINUE:

- A função **break** é utilizada para interromper a execução independentemente do valor atual de suas condições.
- A função **continue** por outro lado é utilizada para continuar a execução do programa.

EXEMPLO: Programa que lê valores até digitar 0 (zero):



```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO - Programa que lê valores até digitar 0 (zero):
s = 0
while True:
    v = int(input("Digite um numero a somar ou 0 para sair:"))
    if v == 0:
        break
    s += v
print (s)
Ln: 9 Col: 0
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai/De
sktop/prog_teste.py =====
Digite um numero a somar ou 0 para sair:2
Digite um numero a somar ou 0 para sair:3
Digite um numero a somar ou 0 para sair:4
Digite um numero a somar ou 0 para sair:2
Digite um numero a somar ou 0 para sair:1
Digite um numero a somar ou 0 para sair:7
Digite um numero a somar ou 0 para sair:0
19
>>> |
Ln: 21 Col: 4
```

INTERROMPENDO A REPETIÇÃO

EXEMPLO: Programa que le um valor e imprime a quantidade de cédulas necessárias para pagar o valor. Para simplificar, vamos trabalhar apenas com valores inteiros e com cédulas de R\$ 50, R\$ 20, R\$ 10, R\$ 5 e R\$ 1.

```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO - Programa que diz cédulas necessarias
valor = int(input("Digite o valor a pagar: "))
cedulas = 0
atual = 50
apagar = valor
while True:
    if atual <= apagar:
        apagar -= atual
        cedulas += 1
    else:
        print(f"{cedulas} cédula(s) de R$ {atual}")
        if apagar == 0:
            break
        if atual == 50:
            atual = 20
        elif atual == 20:
            atual = 10
        elif atual == 10:
            atual = 5
        elif atual == 5:
            atual = 1
        cedulas = 0
Ln: 23 Col: 0
```

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
Digite o valor a pagar: 175
3 cédula(s) de R$ 50
1 cédula(s) de R$ 20
0 cédula(s) de R$ 10
1 cédula(s) de R$ 5
>>>
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
Digite o valor a pagar: 225
4 cédula(s) de R$ 50
1 cédula(s) de R$ 20
0 cédula(s) de R$ 10
1 cédula(s) de R$ 5
>>> |
Ln: 17 Col: 4
```



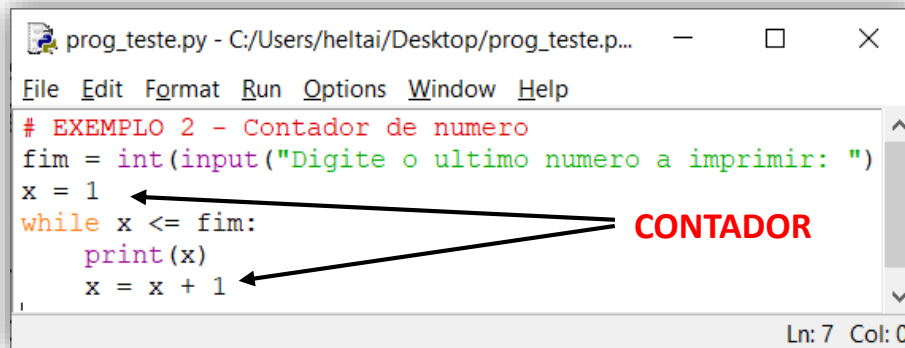
ESTRUTURA DE REPETIÇÃO CONTADOR E ACUMULADOR

CONTADOR E ACUMULADOR

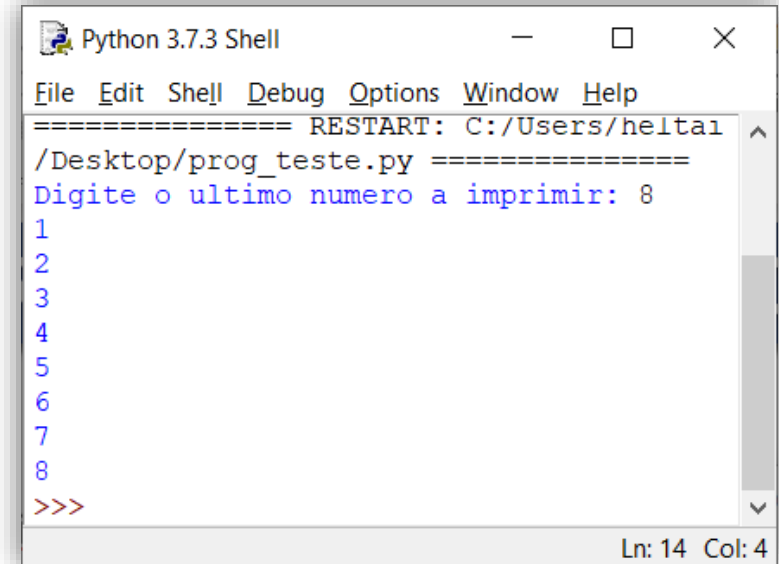
CONTADOR

- A estrutura de repetição while, repete um bloco enquanto a condição for verdadeira. A condição no qual é testado utilizada uma variável no qual deve ser iniciada com um valor e modificada ao longo dos loop.

EXEMPLO: Programa na qual o usuário informa o ultimo numero a ser imprimido e o programa imprime do numero 1 até o numero informado



```
File Edit Format Run Options Window Help
# EXEMPLO 2 - Contador de numero
fim = int(input("Digite o ultimo numero a imprimir: "))
x = 1
while x <= fim:
    print(x)
    x = x + 1
Ln: 7 Col: 0
```

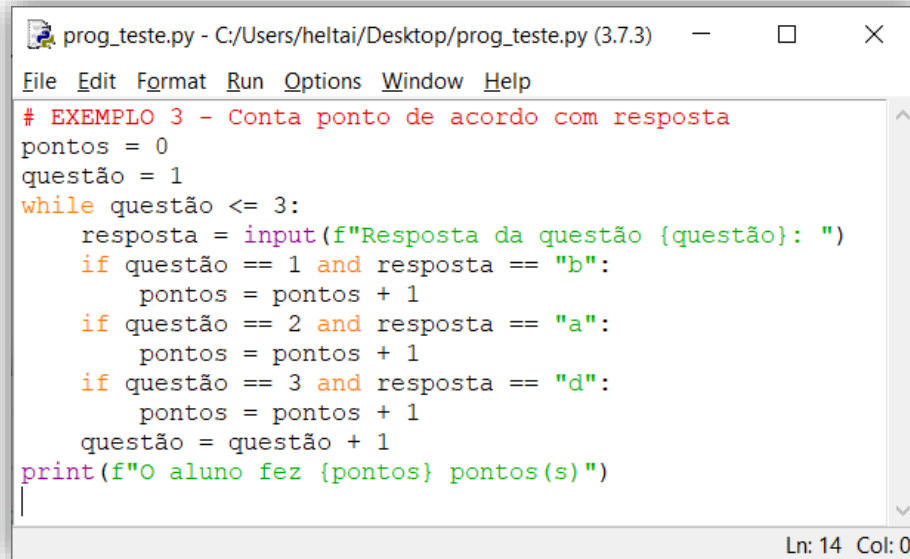


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai /Desktop/prog_teste.py =====
Digite o ultimo numero a imprimir: 8
1
2
3
4
5
6
7
8
>>>
Ln: 14 Col: 4
```

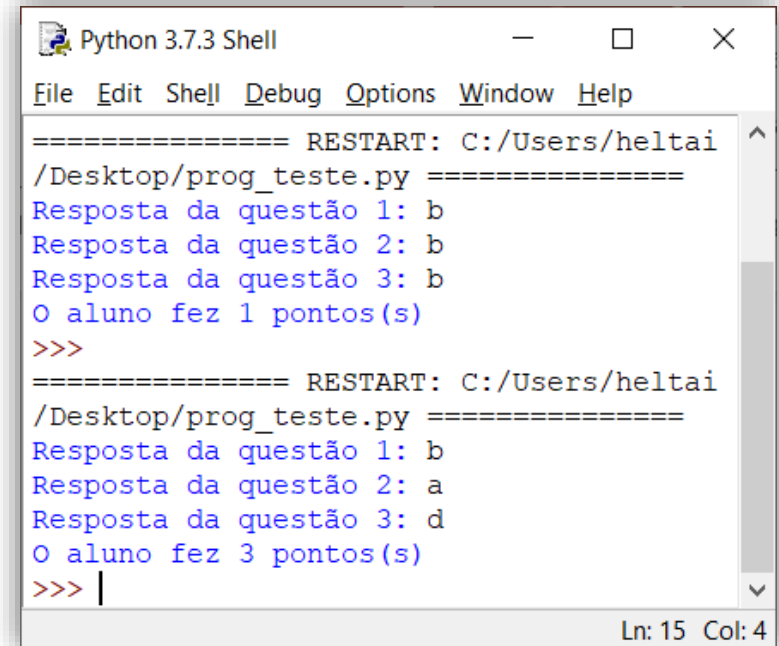
CONTADOR E ACUMULADOR

- Contador pode ser utilizado quando usados em condições dentro dos programas.

EXEMPLO: O programa conta um ponto a cada resposta correta



```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO 3 - Conta ponto de acordo com resposta
pontos = 0
questão = 1
while questão <= 3:
    resposta = input(f"Resposta da questão {questão}: ")
    if questão == 1 and resposta == "b":
        pontos = pontos + 1
    if questão == 2 and resposta == "a":
        pontos = pontos + 1
    if questão == 3 and resposta == "d":
        pontos = pontos + 1
    questão = questão + 1
print(f"O aluno fez {pontos} pontos(s)")
Ln: 14 Col: 0
```



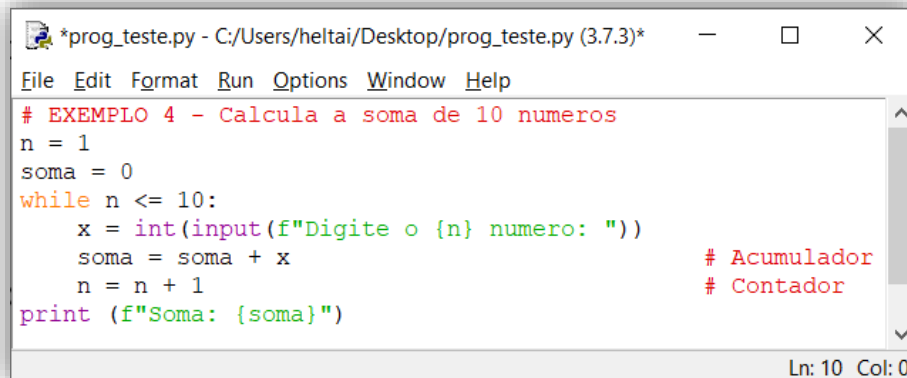
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
Resposta da questão 1: b
Resposta da questão 2: b
Resposta da questão 3: b
O aluno fez 1 pontos(s)
>>>
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
Resposta da questão 1: b
Resposta da questão 2: a
Resposta da questão 3: d
O aluno fez 3 pontos(s)
>>> |
Ln: 15 Col: 4
```

CONTADOR E ACUMULADOR

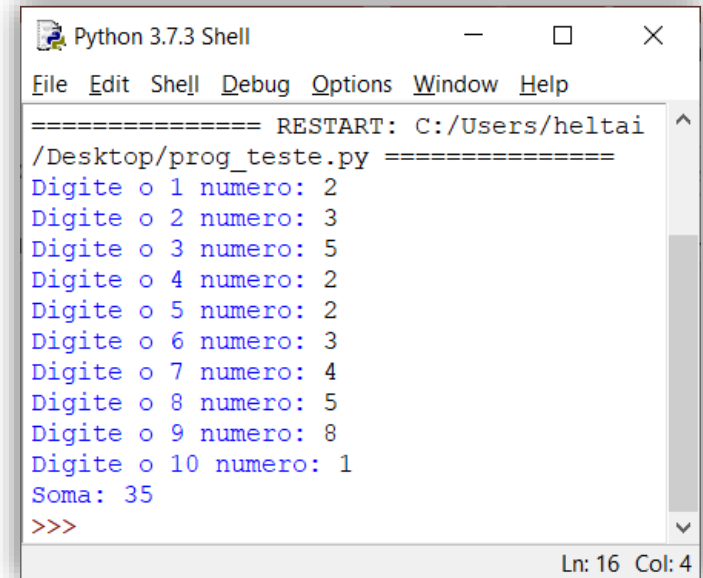
ACUMULADOR

- Os acumuladores são utilizados para acumular valores de operações matemáticas (uma soma por exemplo).
- A diferença entre um contador e um acumulador é que nos contadores o valor adicionado é constante e, nos acumuladores variável.

EXEMPLO: Programa que calcule a soma de 10 números.



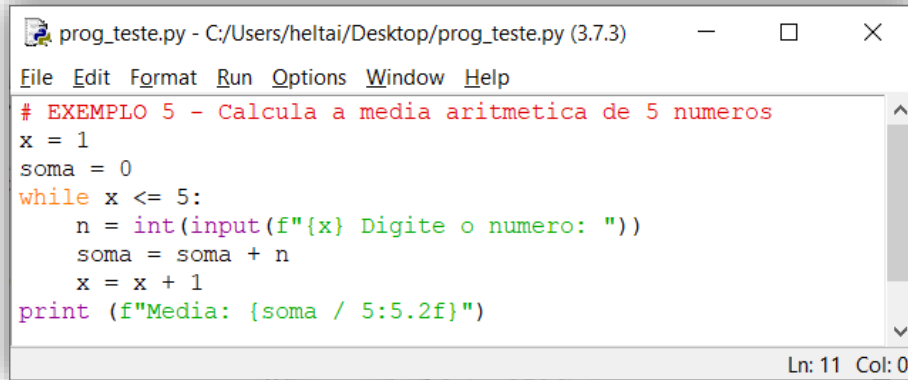
```
*prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)*
File Edit Format Run Options Window Help
# EXEMPLO 4 - Calcula a soma de 10 numeros
n = 1
soma = 0
while n <= 10:
    x = int(input(f"Digite o {n} numero: "))
    soma = soma + x          # Acumulador
    n = n + 1               # Contador
print (f"Soma: {soma}")
Ln: 10 Col: 0
```



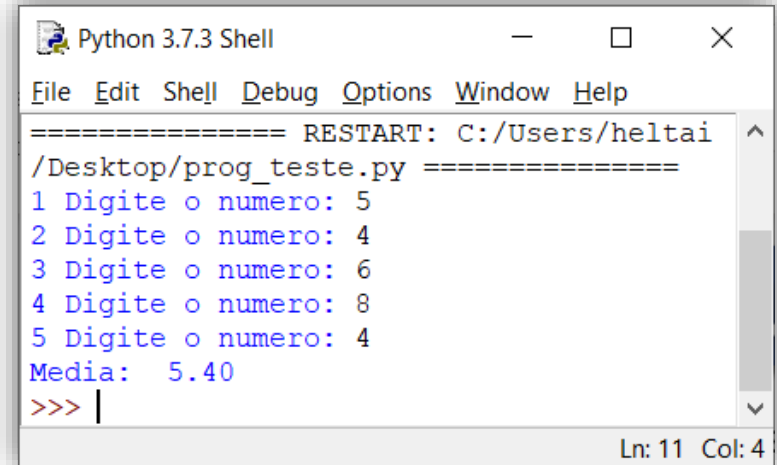
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
Digite o 1 numero: 2
Digite o 2 numero: 3
Digite o 3 numero: 5
Digite o 4 numero: 2
Digite o 5 numero: 2
Digite o 6 numero: 3
Digite o 7 numero: 4
Digite o 8 numero: 5
Digite o 9 numero: 8
Digite o 10 numero: 1
Soma: 35
>>>
Ln: 16 Col: 4
```


CONTADOR E ACUMULADOR

EXEMPLO: Programa que calcula a media aritmética de cinco números digitados pelo usuário.



```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO 5 - Calcula a media aritmetica de 5 numeros
x = 1
soma = 0
while x <= 5:
    n = int(input(f"{x} Digite o numero: "))
    soma = soma + n
    x = x + 1
print (f"Media: {soma / 5:5.2f}")
Ln: 11 Col: 0
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
1 Digite o numero: 5
2 Digite o numero: 4
3 Digite o numero: 6
4 Digite o numero: 8
5 Digite o numero: 4
Media:  5.40
>>> |
Ln: 11 Col: 4
```

CONTADOR E ACUMULADOR

OPERADORES DE ATRIBUIÇÃO ESPECIAL

- O processo de acumuladores, contadores e outras operações podem ser simplificados com os operadores de atribuição especiais, conforme tabela abaixo:

OPERADOR	EXEMPLO	EQUIVALENCIA
+=	$X += 1$	$X = X + 1$
-=	$Y -= 1$	$Y = Y - 1$
*=	$C *= 2$	$C = C * 2$
/=	$D /= 2$	$D = D / 2$
**=	$E **= 2$	$E = E ** 2$
//=	$F //= 4$	$F = F // 4$



ESTRUTURA DE REPETIÇÃO

REPETIÇÃO ANINHADAS

REPETIÇÕES ANINHADAS

- É possível combinar vários **while** de forma a obter resultados mais elaborados.

EXEMPLO: Programa que imprime as tabuadas de multiplicação de 1 a 10.

```
prog_teste.py - C:/Users/heltai/Desktop/prog_teste.py (3.7.3)
File Edit Format Run Options Window Help
# EXEMPLO - Imprime a tabuada de 1 a 10
tabuada = 1
while tabuada <= 10:
    numero = 1
    while numero <= 10:
        print(f"{tabuada} x {numero} = {tabuada * numero}")
        numero += 1
    tabuada += 1
|
Ln: 9 Col: 0
```

```
Python 3.7.3 ...
File Edit Shell Debug Options
Window Help
===== RESTART: C:
/Users/heltai/Desktop/prog_
teste.py =====
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
Ln: 221 Col: 4
```

```
Python 3.7.3 ...
File Edit Shell Debug Options
Window Help
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
Ln: 221 Col: 4
```

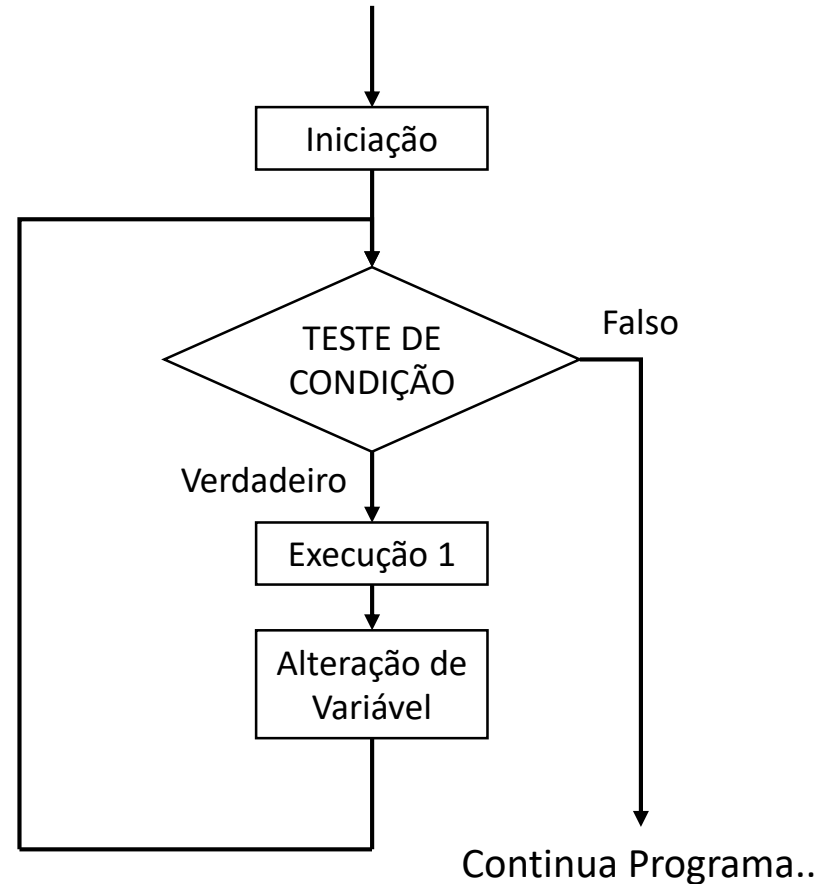
```
Python 3.7.3 ...
File Edit Shell Debug Options
Window Help
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
>>> |
Ln: 221 Col: 4
```



ESTRUTURA DE REPETIÇÃO FOR

ESTRUTURA DE REPETIÇÃO – FOR

- O comando **for** é a estrutura de repetição parecida com o **while**.
- Na instrução é passada uma situação inicial, uma condição e uma ação a ser executada a cada repetição. Uma variável é inicializada com um valor inicial. Essa variável é utilizada para controlar a quantidade de vezes em que o conjunto de comandos será executado.
- E ao final do conjunto de comandos a variável sempre sofrerá uma alteração, aumentando ou diminuindo de acordo com a lógica utilizada.



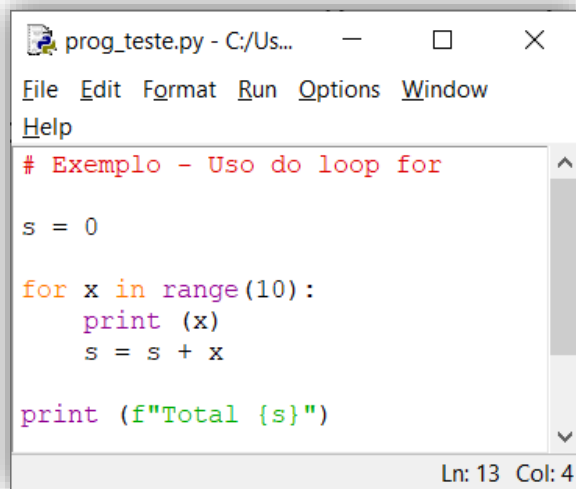
ESTRUTURA DE REPETIÇÃO – FOR

- Durante a execução de um laço **for**, a referência aponta para um elemento da sequência. A cada iteração, a referência é atualizada, para que o bloco de código do **for** processe o elemento correspondente.
- O código dentro do **else** é executado ao final do laço, a não ser que o laço tenha sido interrompido por **break**.

SINTAXE:

for <referencia> **in** <sequencia>
<bloco_de_código>
continue / **break**

ATENÇÃO: O comando **for** é utilizado com **lista** e outros comandos nos quais serão vistos adiante. Idem para **range**



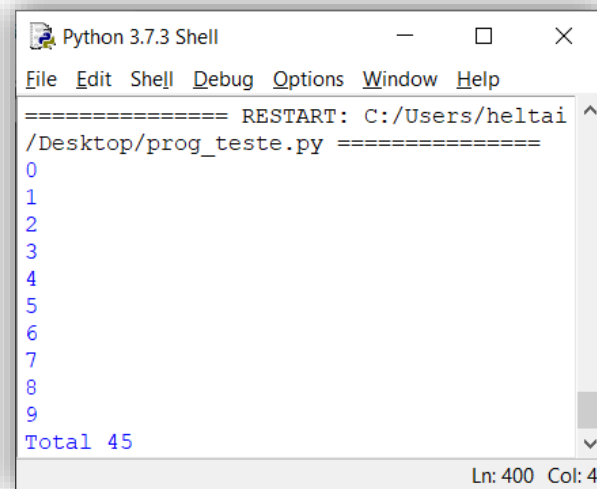
```
File Edit Format Run Options Window Help
# Exemplo - Uso do loop for

s = 0

for x in range(10):
    print (x)
    s = s + x

print (f"Total {s}")
```

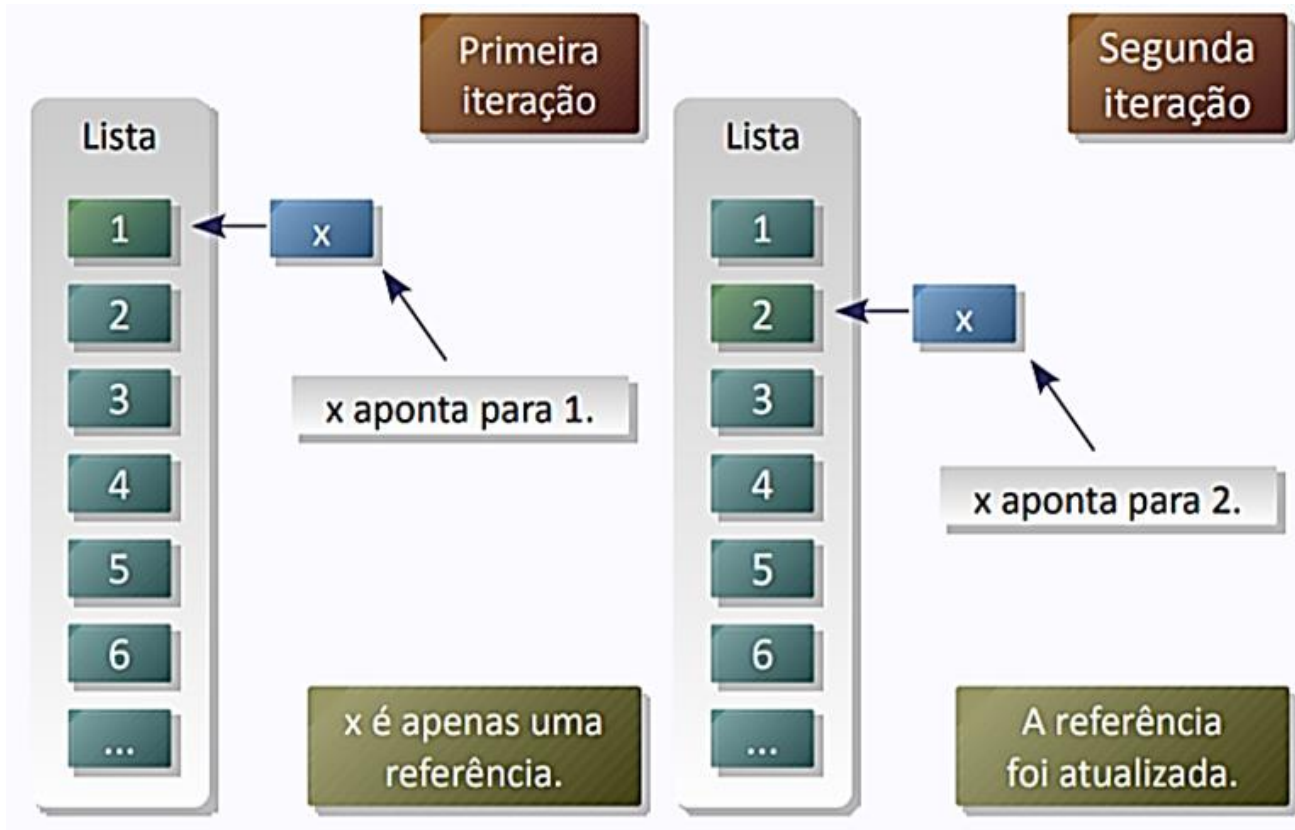
Ln: 13 Col: 4



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/heltai
/Desktop/prog_teste.py =====
0
1
2
3
4
5
6
7
8
9
Total 45
```

Ln: 400 Col: 4

ESTRUTURA DE REPETIÇÃO – FOR



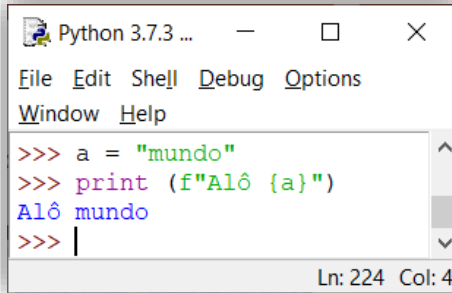


F-STRINGS - COMPLEMENTO

F-STRING – COMPLEMENTO

- Complementando os ensinamento de f-string, no qual foram adicionados na versão Python 3.6 no qual é possível substituir o valor de uma variável ou expressão dentro de uma string.

EXEMPLO:



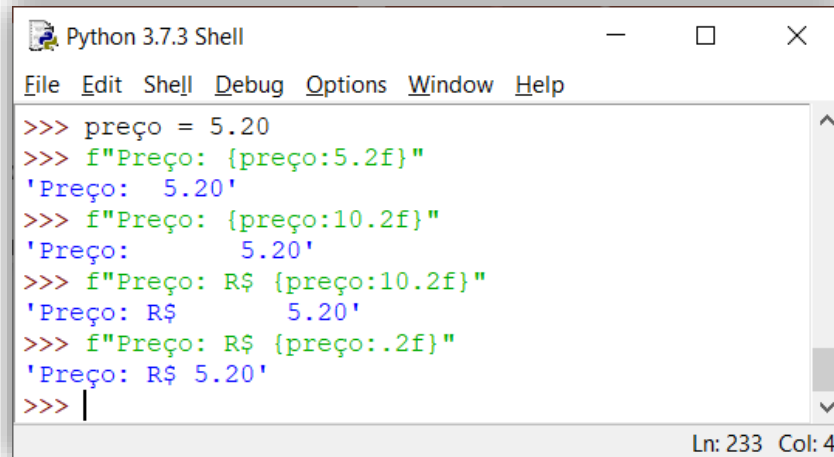
```
Python 3.7.3 ...
File Edit Shell Debug Options
Window Help

>>> a = "mundo"
>>> print (f"Alô {a}")
Alô mundo
>>> |
```

Ln: 224 Col: 4

- É possível formatar f-strings especificando o numero de caracteres após o nome da variável e dos dois pontos

EXEMPLO:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

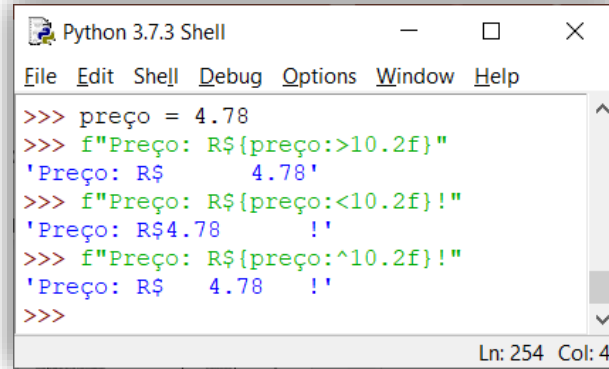
>>> preço = 5.20
>>> f"Preço: {preço:5.2f}"
'Preço:  5.20'
>>> f"Preço: {preço:10.2f}"
'Preço:           5.20'
>>> f"Preço: R$ {preço:10.2f}"
'Preço: R$           5.20'
>>> f"Preço: R$ {preço:.2f}"
'Preço: R$ 5.20'
>>> |
```

Ln: 233 Col: 4

F-STRING – COMPLEMENTO

- É possível formatar f-strings especificando o alinhamento os valores à esquerda, à direita ou ao centro:

EXEMPLO:

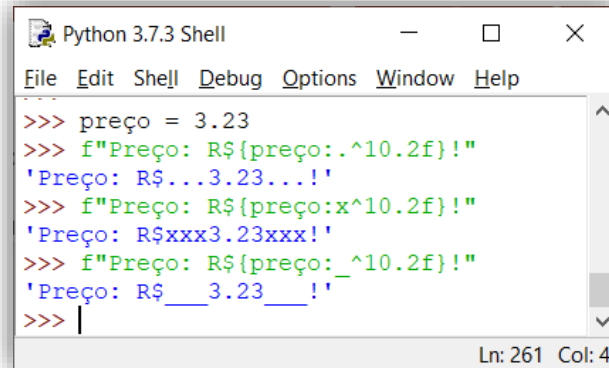


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> preço = 4.78
>>> f"Preço: R${preço:>10.2f}"
'Preço: R$          4.78'
>>> f"Preço: R${preço:<10.2f}!"
'Preço: R$4.78      !'
>>> f"Preço: R${preço:^10.2f}!"
'Preço: R$   4.78   !'
>>>
```

Ln: 254 Col: 4

- É possível formatar f-strings especificando qual caractere deve ser utilizado para preencher os espaços em branco:

EXEMPLO:



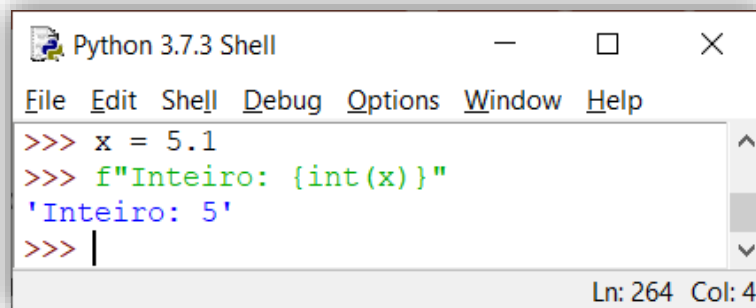
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> preço = 3.23
>>> f"Preço: R${preço:~10.2f}!"
'Preço: R$...3.23...!'
>>> f"Preço: R${preço:x^10.2f}!"
'Preço: R$xxx3.23xxx!'
>>> f"Preço: R${preço:_^10.2f}!"
'Preço: R$__3.23__!'
>>> |
```

Ln: 261 Col: 4

F-STRING – COMPLEMENTO

- É possível chamar funções dentro da f-string (função poderosa no Python)

EXEMPLO:

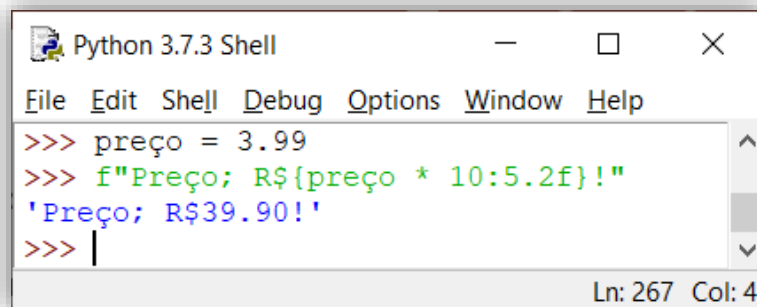


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> x = 5.1
>>> f"Inteiro: {int(x)}"
'Inteiro: 5'
>>> |
```

Ln: 264 Col: 4

- Realizar operações matemáticas

EXEMPLO:



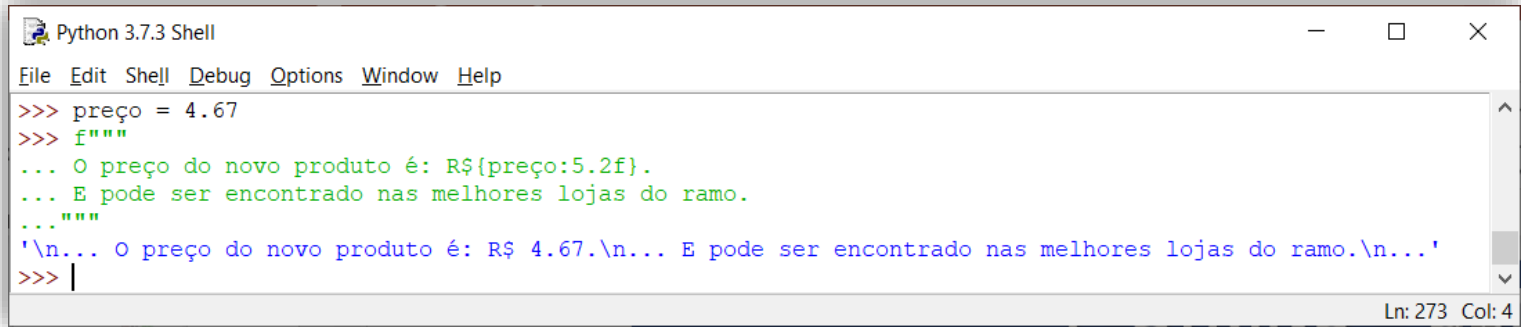
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> preço = 3.99
>>> f"Preço; R${preço * 10:5.2f}!"
'Preço; R$39.90!'
>>> |
```

Ln: 267 Col: 4

F-STRING – COMPLEMENTO

- É possível utilizar strings de múltiplas linhas, usando-se as aspas triplas prefixadas com a letra f:

EXEMPLO:

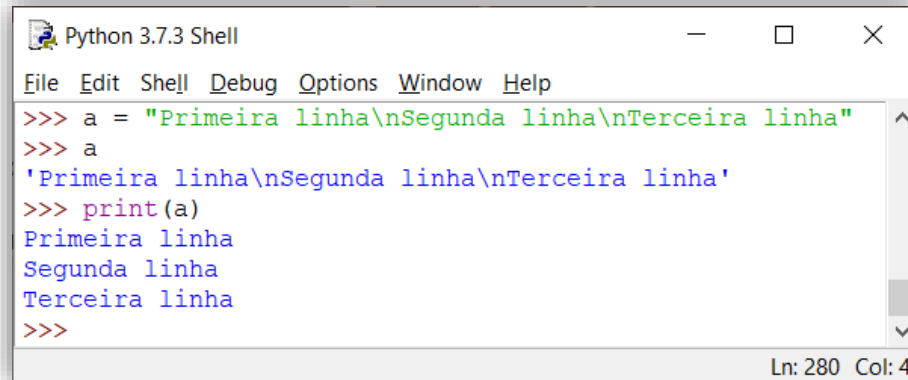


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> preço = 4.67
>>> f"""
... O preço do novo produto é: R${preço:5.2f}.
... E pode ser encontrado nas melhores lojas do ramo.
... """
'\n... O preço do novo produto é: R$ 4.67.\n... E pode ser encontrado nas melhores lojas do ramo.\n...'
>>> |
```

Ln: 273 Col: 4

- As linhas são representadas com `\n`. Esta combinação de caracteres é utilizada para representar uma quebra de linha e pode utiliza-la em em strings.

EXEMPLO:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> a = "Primeira linha\nSegunda linha\nTerceira linha"
>>> a
'Primeira linha\nSegunda linha\nTerceira linha'
>>> print(a)
Primeira linha
Segunda linha
Terceira linha
>>>
```

Ln: 280 Col: 4



PROF. VINICIUS HELTAI

vinicius.pacheco@docente.unip.br

www.heltai.com.br

(11) 98200-3932



/vheltai



@vheltai



/vheltai



@Vinicius Heltai



@vheltai