Aula 10 – O comando for de repetição

É muito comum usarmos o comando **`while`** para repetir alguns comandos um número definido de vezes. Fazemos isso usando um contador.

Considere os dois problemas abaixo, já resolvidos anteriormente que usam o comando while:

P10.1) Dado N > 0 e uma sequência de n números calcular a soma dos elementos da sequência:

```
def main():
    # entrada de N
    N = int(input("Entre com o valor de N: "))

    # inicia o contador e a soma
    contador = 1
    soma = 0

    # repete N vezes o trecho de programa
    while contador <= N:
        x = int(input("Entre com um valor inteiro: "))
        soma = soma + x
        contador = contador + 1

    # imprima o resultado
    print ("O valor da soma dos", N, "numeros eh", soma)

main()
```

P10.2) Dado n > 0 calcular os divisores de n maiores que 1 e menores que n

```
def main():

   # leia n
   n = int(input("Entre com n: "))

   # primeiro candidato a divisor
   div = 2

   # repete o teste para todos os candidatos de 2 até n // 2
   while div <= n // 2:
      if n % div == 0:
         print(div, "e' um divisor de", n)
      # testa o próximo candidato
      div = div + 1

main()
```

P10.3) Tabelar a função $x^3 - 1$ para x = -10, -8, -6, ..., 6, 8, 10

```
def main():
    x = -10
    while x <= 10:
        print(x, x * x * x - 1.0)
        x = x + 2
```

```
main()
```

O uso dos comandos marcados (inicialização, comparação e incremento) ocorre com frequência e é usada sempre que é necessária a repetição de um trecho de programa certo número de vezes.
O objetivo do contador é controlar o número de vezes que a repetição será efetuada.

Generalizando um pouco, o que queremos é um contador que assuma valores num intervalo ou numa lista de valores e repita os comandos para cada um dos valores assumidos pelo contador.

Esse tipo de construção é muito comum nos algoritmos. Por isso, as linguagens de programação em geral possuem um comando que resume as 3 operações (inicialização, comparação e incremento).

Não é diferente na linguagem Python. O comando **`for`**, associado à função **`range`** realiza exatamente isso.

A função **`range`**

A função **`range`** cria uma lista de valores (o conceito de lista de valores será visto com mais detalhes á frente). Exemplos:

```
range(10) =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(2, 10) =  [2, 3, 4, 5, 6, 7, 8, 9]
range(2, 10, 3) =  [2, 5, 8]
range(2, 10, 2) =  [2, 4, 6, 8]
range(10, 2, -2) =  [10, 8, 6, 4]
range(10, 2) =  []
range(10, 20, -1) =  []
```

Importante: O limite superior do intervalo (ou inferior no caso de contagem regressiva) está excluído da lista.

Importante: A lista precisa ser bem definida. Por isso, listas com incrementos fracionários não são permitidas devido a arredondamentos que são feitos para se representar valores fracionários.

Exemplo: **`range (-1, 1, 0.1)`** não é uma lista válida.

O comando **`for`**

A forma simples do comando **`for`** é as seguinte:

```
for <elemento> in <lista>:
    <comandos>
```

O **`<elemento>`** assume todos os valores da **`<lista>`** e para cada valor repete os **`<comandos>`**.

Veja agora a solução de alguns problemas com esse comando e compare com os mesmos resolvidos anteriormente com o while:

P10.1) Dado N > 0 e uma sequência de n números calcular a soma dos elementos da sequência:

```
def main():
    # entrada de N
    N = int(input("Entre com o valor de N: "))
```

```
    # inicia a soma
    soma = 0

    # repete N vezes o trecho de programa
    # note que contador assume valores 0, 1, 2, ..., N - 1
    for contador in range(N):
        x = int(input("Entre com um valor inteiro: "))
        soma = soma + x

    # imprima o resultado
    print ("O valor da soma dos", N, "numeros eh", soma)

main()
```

O comando **for** no exemplo acima pode ser escrito de outras formas:

```
for contador in range(1, N + 1):
```

ou

```
for contador in range(N, 0, -1):
```

P10.2) Dado n > 0 calcular os divisores de n maiores que 1 e menores que n

```
def main():

   # leia n
   n = int(input("Entre com n: "))

   # repete o teste para todos os candidatos de 2 até n // 2
   # note que queremos testar div = 2, 3, 4, ..., n // 2
   for div in range(2, n // 2 + 1):
      if n % div == 0:
         print(div, "e' um divisor de", n)

main()
```

O comando **for** no exemplo acima pode ser escrito de outra forma:

```
for div in range(n // 2, 1, -1):
```

P10.3) Tabelar a função $x^3 - 1$ para x = -10, -8, -6, ..., 6, 8, 10

```
def main():
    for x in range(-10, 11, 2):
        print(x, x * x * x - 1)

main()
```

Observe que no exemplo acima não podemos usar **range(-10, 10, 2)**. O último valor a ser calculado seria -8.

P10.4) Dado n $\geq$ 0 calcular n!

Como sabemos: n! = n.(n-1).(n-2). ... .2.1 se n > 0 e n! = 1 se n=0.

```
def main():
    n = int(input("Digite o valor de n maior ou igual a zero: "))

    # inicia as variáveis
    fat = 1

    for i in range(2, n + 1): fat *= i

    # resultado
    print("O valor de %d! e' = " %n, fat)

#-------------------------------------------
main()
```

O comando **for** no exemplo acima pode ser escrito na forma:

```
for i in range(n, 1, -1):
```

Alternativa à função range

Quando a lista de valores a serem assumidos pela variável não possui um passo fixo, podemos usar a própria lista de forma explicita no comando for. Por exemplo:

```
for k in [2, -45, 83, 0, -53]: print (k)
for k in [1, -1, 0, -1, 1]: print(k, 2**k)
```

P10.5) Calcular o valor da função $x^3 + x^2 + x + 1$ para x = 1, -1, 2, -2, 3, -3

```
def main():

    # exemplo de for com lista fixa

    for k in [0, 1, -1, 2, -2, 3, -3]:
        print("Valor para x = ", k, "e' ", k * k * k + k * k + k + 1)

main()
```

P10.6) Dado um intervalo de inteiros [a, b] , calcular o valor da funcao $x^3 - x^2 + x - 1$ para $a \leq x \leq b$

```
def main():

    # ler a e b inteiros
    a = int(input("Entre com o valor de a inteiro: "))
    b = int(input("Entre com o valor de b inteiro: "))

    for k in range(a, b + 1):
        print("Valor para x = ", k, "e' ", k * k * k - k * k + k - 1)

main()
```

P10.7) Dado $n \geq 0$ e x diferente de zero, inteiros, calcular x^n. Sem usar a função interna **pow** do Python.

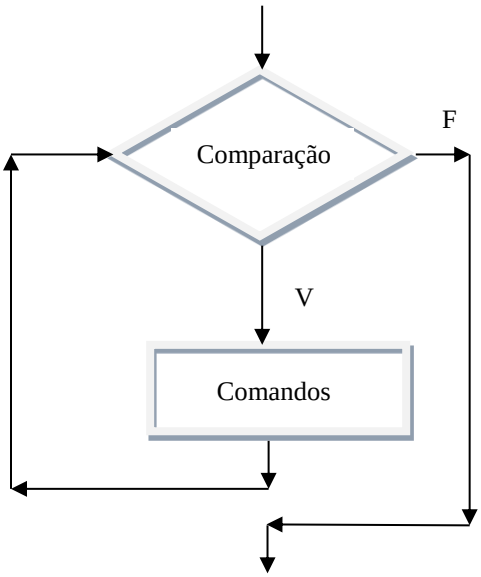
```
def main():
```

```
    # ler n inteiro >= 0 e x float diferente de zero
    n = int(input("Entre com o valor de n inteiro ≥ a zero: "))
    x = float(input("Entre com o valor de x float ≠ de zero: "))

    # inicia o valor de pot
    pot = 1

    # multiplica por x n vezes
    for k in range(0, n):
        pot = pot * x

    # resultado
    print("Valor de %f elevado %d  = " %(x, n), pot)

main()
```

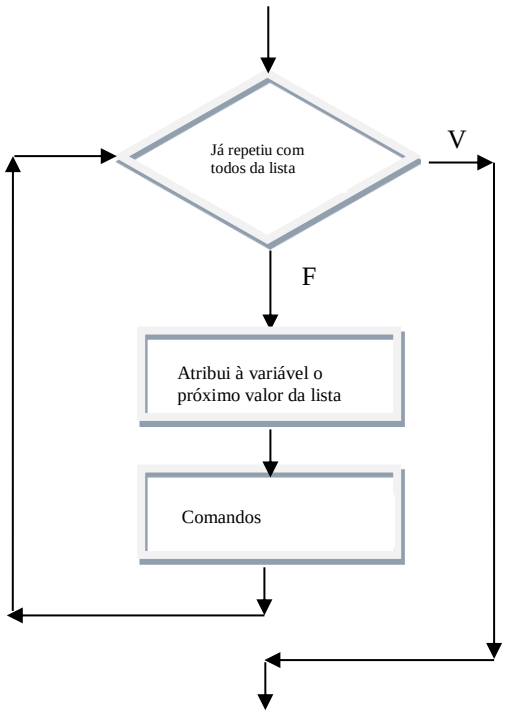
Fluxograma dos comandos while e for

Compare agora o fluxograma do comando while e do comando for.

while



for



São bem semelhantes. Por isso, quase sempre é possível escrever uma repetição tanto com while quanto com for. Em geral, usamos sempre o for quando temos que repetir um número definido de vezes.

Se executado até o final, a variável que controla a repetição do comando for, fica com o último valor da lista. Veja os comandos abaixo e o que é impresso:

```
for i in range(5):
    print (i)

# saiu do for
print("final = ", i)

for x in ["azul", "amarelo", "vermelho"]:
    print (x)

# saiu do for
print("final = ", x)
```

```
0
1
2
3
4
```

```
final =  4
azul
amarelo
vermelho
final =  vermelho
```

A lista de valores a ser percorrida pela variável que controla o for é definida no início, baseada nos valores fornecidos à função range. Possíveis mudanças nos valores feitas dentro da repetição não afetam a lista pré-definida. Veja o exemplo abaixo:

```
start = 1
stop = 10
step = 2
for i in range(start, stop, step):
    print(i)
    # qualquer mudança nos valores não afeta a lista original
    start = 32
    stop = -45
    step = -2
    # não adianta muda o valor da variável de controle
    i = -247
    print(i)
# qual o valor de i na saida do for? -247 ou 9 (último da lista)?
print("valor de i ao sair do for -247 ou 9 (último da lista) = ", i)
```

Será impresso:

```
1
-247
3
-247
5
-247
7
-247
9
-247
valor de i ao sair do for -247 ou 9 (último da lista) =  -247
```

Exercícios:

Resolva com o comando for, os exercícios anteriores que foram resolvidos com while.