



COMPUTAÇÃO 1 – AULA 8

Estrutura de repetição FOR

Prof. Cesar Raitz

1. Introdução

Para repetir código com **while** você deve lembrar de:

1. Inicializar um contador.
2. Uma condição para continuar repetindo.
3. Incrementar o contador.

Esqueça uma delas e seu programa não funcionará corretamente! 🧐

Além disso, numa reunião informal, alguns programadores perceberam que estavam usando muito os contadores para indexar listas ou strings, como neste exemplo:

```
1. # Mostra números pares de uma lista
2. lista = [3, 22, 15, 8]
3. i = 0
4. while i < len(lista):
5.     if lista[i] % 2 == 0:
6.         print(lista[i])
7.     i += 1
```

- Utilizamos várias vezes o contador quando, o que realmente interessa: *são os números da lista!*
- Nesta aula, veremos uma forma mais fácil de expressar repetição, usando a palavra-chave **for**.



2. Definição e sintaxe

- *for* significa para
- O algoritmo acima, em pseudo-código, poderia ser expresso desta forma:

Algoritmo: Mostra números pares de uma lista.

1. Para cada número n em [3, 22, 15, 8]:
1a. Se n for par:
1ai. Mostrar n

- Percebeu que o algoritmo não tem nenhum contador? 🤔
- Também não há indexação!
- As linhas depois do **Para** são repetidas uma vez para cada número da lista:
 - Na primeira iteração: $n = 3$
 - Na segunda iteração: $n = 22$
 - Na terceira iteração: $n = 15$
 - Na quarta iteração: $n = 8$
- Agora vamos fazer a "mágica" de traduzir diretamente para Pythonês:

```
1. # Mostra números pares de uma lista  
2. for n in [3, 22, 15, 8]:  
3.     if n % 2 == 0:  
4.         print(n)
```



- Você está vendo pela primeira vez uma estrutura de repetição chamada de **laço for**. (loop)

Exercício 1. Olhando uma lista de números.

Ao executar o seguinte código, quais números aparecem na tela?

```
1. for n in [3, 22, 15, 8, 21, 32]:  
2.     if n % 3 != 0:  
3.         print(n)
```

Bem mais simples que usar o **while** né? Vamos olhar a sintaxe:

```
<instruções antes>  
for <variável> in <lista>:  
    <instruções dentro do for>  
<instruções depois>
```

Variáveis caminhantes

- Nos laços `while`, precisamos sempre de um contador.
A variável do `for` é um contador?
- Contador não descreve bem o que a variável do `for` faz. Seu **papel** é *caminhar pelos elementos da lista*, então chamamos essa variável de **caminhante**.

```
for n in [3, 22, 15, 8, 21, 32]:
```

Exercício 2. Olhando uma lista de palavras.

A lista usada no `for` pode conter qualquer tipo de variável. Responda de cabeça: o que acontece no código abaixo?

```
1. palavras = ["estudar", "surfar", "dormir"]  
2. for palavra in palavras:  
3.     print("Eu amo", palavra, "!")
```



Note que o nome escolhido para a variável, `palavra`, está relacionado com os elementos da lista.

Olhando caracteres

- Na aula sobre `while`, vimos alguns algoritmos que usavam repetição para olhar caractere por caractere, também com indexação.
- Veja como é mais simples construir um algoritmo com `for`:

```
1. # Passa somente as vogais para maiúscula  
2. nova_str = ""  
3. for c in "O rato roeu":  
4.     # Verifica se c é uma vogal minúscula  
5.     if c in "aeiou":  
6.         # Passa para maiúscula  
7.         c = str.upper(c)  
8.         # Coloca o caractere na nova string  
9.         nova_str += c  
10.    # Vejamos como ficou  
11.    print(nova_str)
```



Iteráveis

- Vamos tentar executar um laço para um único elemento:

```
for n in 5:  
    print(n)
```

- Imediatamente o Python reclama dizendo:

TypeError: 'int' object is not iterable

- Ou seja: não faz sentido ter uma repetição para um elemento singular!
- Elementos que *podem ser divididos em partes menores*, como listas ou strings, são chamados de **iteráveis**.
- Assim como listas, tuplas também são iteráveis.
- Na próxima aula veremos uma estrutura de dados chamada **dicionário**. Também podemos usá-los em laços `for`.

Exercício 3. Mas pode isso?

É possível iterar em listas, tuplas ou strings vazias? E se tiverem um único elemento/caractere?

3. O iterável range

- Já está com saudades dos contadores?
- Esse iterável especial `range` (significa intervalo) funciona como um contador, sua sintaxe é

```
range(vi, vf, passo)
```

- `vi` é o valor inicial (0 se omitido)
- `vf` é o valor final (não entra no intervalo)
- `passo` é o valor do incremento/decremento (1 se omitido)

- Um pequeno exemplo:

```

1. # Soma os números de 3 a 20, de 3 em 3
2. # i.e. 3, 6, 9, 12, 15, 18
3. soma = 0
4. for n in range(3, 21, 3):
5.     soma += n
6. print(soma)

```

- Na verdade, como `sum` recebe um iterável como entrada, podemos fazer esta soma diretamente:

`sum(range(3, 21, 3))`

- Uma dica interessante, é que você pode transformar um `range` em uma lista, com a função `list`:

`list(range(3, 21, 3))`

Exercício 4. Intervalos iteráveis.

Em quais números o laço `for` vai iterar, se usarmos:

- | | |
|----------------------------------|----------------------------------|
| a) <code>range(5, 10, 1)</code> | d) <code>range(5, 10, 5)</code> |
| b) <code>range(5, 10, -1)</code> | e) <code>range(10, 5, -1)</code> |
| c) <code>range(5)</code> | f) <code>range(-5)</code> |

Se necessário, teste no computador.

Exercício 5. Fibonacci de novo!

Crie uma função usando laço FOR que retorna uma sequência de Fibonacci, na forma de lista. A sequência deve conter n elementos, onde n será o argumento da função. Lembre-se que, como a lista já começa com [1, 1], então você vai precisar de mais n-2 elementos, ou seja, um iterável que vai de 2 a n (fora).

4. Juntando iteráveis

Às vezes é necessário juntar mais de um iterável, como por exemplo, duas listas. Neste caso, usa-se a função `zip`, que junta um elemento de cada lista. Você consegue adivinhar a saída do código abaixo?

```
1. nomes = ["Xiquinho", "Luquinhas", "Ritinha"]  
2. idades = [5, 4, 6]  
3. for nome, idade in zip(nomes, idades):  
4.     print(f"{nome} tem {idade} anos")
```

- `zip` pode juntar quantos iteráveis quiser!
- Os iteráveis não precisam ser de mesma natureza, por exemplo:

```
1. nomes = ["Xiquinho", "Luquinhas", "Ritinha"]  
2. idades = [5, 4, 6]  
3. for num, nome, idade in zip(range(1,4), nomes, idades):  
4.     print(f"{num}. {nome} tem {idade} anos")
```

Para ficar mais claro, vamos sumarizar as repetições do laço `for`:

iteração	num	nome	idade
1	1	"Xiquinho"	5
2	2	"Luquinhas"	4
3	3	"Ritinha"	6

Exercício 6. Soma de vetores.

Usamos muitos exemplos lúdicos mas não podemos esquecer o principal motivo pelo qual os computadores foram inventados: *processamento de dados!* E uma das tarefas mais comuns é a multiplicação de vetores. Agora, imagine vetores representados por tuplas, e.g.

$$\begin{aligned} a &= (1, 2, 3, 4) \\ b &= (-5, -6, -7, -8) \end{aligned}$$

Escreva uma função, usando laços `for` e `zip`, para retornar a soma de dois vetores na forma de tupla. O resultado da soma de `a` com `b` será:

$$(1+(-5), 2+(-6), 4+(-7), 5+(-8)) = (-4, -4, -4, -4)$$

Há duas formas de criar tuplas:

- 1) Crie uma lista da forma que já vimos, usando `list.append()` para anexar elementos novos à lista, e então, converta a lista para tupla usando a função `tuple()`.
- 2) Comece com uma tupla vazia: `t = ()`; depois, em cada iteração, acrescente uma tupla com o elemento novo, i.e. `t += (n,)`.

Coloque dois casos de teste com o valor esperado em comentários.

5. Enumerando os itens

Outras vezes, precisamos somente enumerar os elementos do iterável. Para isto, usamos a função `enumerate`. Mas lembre-se de que `enumerate` sempre começa de 0.

```
1. tarefas = [  
2.     "Arrumar a cama",  
3.     "Varrer o quarto",  
4.     "Lavar a louça"  
5. ]  
6. for i, tarefa in enumerate(tarefas):  
7.     print(f"{i+1}. {tarefa}")
```

Qual é o valor das variáveis em cada iteração?

iteração	i	tarefa
1	0	"Arrumar a cama"
2	1	"Varrer o quarto"
3	2	"Lavar a louça"

6. Interrompendo o laço ou voltando

- **break** e **continue** continuam funcionando com `for`
- **break** termina imediatamente o laço mais próximo
- **continue** volta ao início do laço (próxima iteração)

```

1. palavroes = # Lista de palavrões
2. texto = """
3.     quando digo que deixei de te amar.
4.     quando digo que eu não quero mais você.
5.     REFRÃO:
6.         e nessa loucura, de dizer que não te quero.
7. """
8. for palavra in str.split(texto):
9.
10.    if palavra == "REFRÃO":
11.        continue
12.
13.    if palavra in palavroes:
14.        print(f"Não pode falar {palavra}!")
15.        break
16.
17.    print(f"{palavra} ", end="")
18.
19. print(f"A última palavra foi {palavra}")

```

7. Mais exercícios

Exercício 7. Estilizando o texto.

Uma frase estilizada causa impacto! Crie uma função que pegue uma string e passe todas as vogais para minúscula e consoantes para maiúscula. A função deve retornar a string estilizada. Apresente um caso de teste.

Exercício 8. Produto escalar.

Na última aula, você criou uma função para calcular o produto escalar de dois vetores com duas componentes cada. Agora crie, usando `for` com `zip`, uma função `prod_esc` para calcular o produto escalar de vetores com mais componentes:

$$(a_0, a_1, a_2, \dots) \cdot (b_0, b_1, b_2, \dots) \Rightarrow a_0b_0 + a_1b_1 + a_2b_2 + \dots$$

A melhor forma de fazer isso é multiplicar cada par de componentes e somar as componentes ao acumulador. Utilize os seguintes casos de teste para ver se funcionou:

```
# CASOS DE TESTE
a = (3, 9, -1, 2)
b = (-5, -5, 0, 0)
c = (1, 1, 1, 1)
print( prod_esc(a, b) ) # -60
print( prod_esc(a, c) ) # 13
print( prod_esc(b, c) ) # -10
```

Essa função pode ser útil para calcular média ponderada e modelos lineares para medidas experimentais, entre outras coisas.