Bem-vindo ao roteiro de Introdução ao Python - Parte 3

*Confeccionado por Marcílio F. Oliveira Neto (marciliofoneto@gmail.com)
Github: github.com/marcilioliveira

Neste roteiro você irá aprender sobre a sintaxe do Python. Este roteiro tem o intuito de seguir o conceito *do-it-yourself*.

Neste roteiro você encontrará os seguintes tópicos sobre Python:

- for
- while
- for/else
- while/else

Instruções para utilizar este roteiro:

- Importar o arquivo deste roteiro no Jupyter;
- Clicar no menu superior **Cell**;
- Clicar na opção **Run All**;
- Para executar célular individualmente utilizar:
  - Selecionar a célula
  - Clicar no botão **Run** - menu superior - ou;
  - Clicar na opção **Cell** e **Run Cells** - ou;
  - Utilizar o atalho **Ctrl + Enter**.

O *for*

Este comando é, em linhas gerais, um laço de execução. Entenda por laço algo que facilita sua vida, que possui execução por ciclos determinados - por você! Isso garante produtividade, uma vez que podemos reexecutar códigos escritos apenas uma única vez.

Image que sua tarefa seja exibir na tela 1000 frases *"Olá, estou aprendendo Python"*, como você faria? Mil "copiar + colar" seria uma alternativa, mas nada produtivo e elegante, concorda? Eis que o **for** salvará sua pele.

Veja como sua sintaxe é simples e elegante:

In [1]:

```
for i in range(3): #define que o laço será executado 3x
    print("Olá, estou aprendendo Python")
```

```
Olá, estou aprendendo Python
Olá, estou aprendendo Python
Olá, estou aprendendo Python
```

Note no quadro acima que com apenas um comando **print** ocorreram 3 exibições da frase em questão. Como isso é possível? Com o **for** e alguns outros detalhes.

O comando **range(x)** tem por característica informar ao **for** quantas vezes os comandos abaixo dele deverão ser executados, o *x* é a quantidade de vezes que deseja. Lembre-se da indentação, os comandos abaixo do **for** possuem maior recuo, o que os permite estar na hierarquia deste comando.

Mas e o **i**? Vou te responder com código!

In [2]:

```
for i in range(3): #define que o laço será executado 3x
    print("Olá, estou aprendendo Python - frase:", i) #A variável i é responsáve
l por dizer qual execução estamos!
```

```
Olá, estou aprendendo Python - frase: 0
Olá, estou aprendendo Python - frase: 1
Olá, estou aprendendo Python - frase: 2
```

A variável ***i*** - na verdade pode ser o nome que você desejar - guarda o número da execução que o **for** está! Isso é muito importante, pois poderemos utilizá-lo futuramente em outros conceitos.

Veja alguns exemplos práticos do **for**:

In [3]:

```
#Exemplo que realiza a leitura e exibição de 5 possíveis números informados pelo
usuário
for i in range(5): #define que o laço será executado 5x
    numero = int(input("Digite um número: "))
    print("O número digitado foi:", numero)
```

```
Digite um número: 12
O número digitado foi: 12
Digite um número: 12
O número digitado foi: 12
Digite um número: 12
O número digitado foi: 12
Digite um número: 12
O número digitado foi: 12
Digite um número: 12
O número digitado foi: 12
```

Perceba que podemos colocar qualquer código abaixo do **for**! Veja abaixo outra aplicação.

In [4]:

```
#Exemplo que realiza a leitura e exibição de 5 possíveis números informados pelo
usuário
#Além disso, se o número digitado for 10, ele informa mensagem de "ACERTO"
for i in range(5): #define que o laço será executado 5x
    numero = int(input("Digite um número: "))
    print("O número digitado foi:", numero)
    if numero == 10:
        print("ACERTO")
```

```
Digite um número: 1
O número digitado foi: 1
Digite um número: 1
O número digitado foi: 1
Digite um número: 3
O número digitado foi: 3
Digite um número: 2
O número digitado foi: 2
Digite um número: 1
O número digitado foi: 1
```

Do-it-yourself

Escreva um script capaz de simular um validador de senha. O usuário tem 3 chances de acerto.

Suponha que a senha correta você já tenha definido em uma variável.

Se o usuário acertar a senha, mostre uma mensagem amigável *Bem-vindo!*, senão, mostre *Errou a tentativa!*.

O *while*

O comando é mais um laço e execução, porém, possui alguns detalhes diferentes quando comparado ao **for**, por exemplo.

Para que o **while** seja executado, uma condição - definida por você - deve ser satisfeita. Assim, os testes dessas condições levam consigo os mesmos detalhes dos testes utilizados com o comando **if**.

Veja como sua sintaxe é simples e elegante:

In [5]:

```
#Exemplo de execução do while enquanto o número digitado pelo usuário for difere
nte de 1
numero_digitado = 0 #variável que guardará o número a ser digitado - inicialment
e é 0
while numero_digitado != 1: #enquanto a variáel numero_digitado é diferente de
 1, faça:
    numero_digitado = int(input("Digite um numero qualquer - para sair, digite
 1: "))
```

```
Digite um numero qualquer - para sair, digite 1: 1
```

Note que não houve a necessidade de limitar - ou informar - quantas vezes o laço deve ser executado. Uma grande diferença quando comparado ao **for**.

Do-it-yourself

Modifique o **do-it-yourself** acima, possibilitando ao usuário digitar a senha - incontáveis vezes - enquanto estiver errada, ou seja, enquanto a senha for incorreta, permita-o digitar novamente.

Para que o **while** tenha um comportamento parecido ao do **for**, podemos utilizar a seguinte estrutura, com a ajuda de uma variável auxiliar.

In [6]:

```
#Exemplo do comando while com variável auxiliar (contadora) - igual ao do for
aux = 0
while aux < 3:
    print(aux)
    aux = aux + 1
```

```
0
1
2
```

O for e a cláusula else

Como deve ter ficado claro para você nos tópicos acima, o ***for*** é um laço de execução que a cada passada realiza um teste, é assim que ele sabe que deve executar de 1 a 5, por exemplo.. Tomando como base o comando **range**, esse gera uma lista e para cada número contido nela o **for** é executado. Porém, quando não houver mais elementos, o laço é então interrompido, mas podemos tratar essa interrupção e capturá-la numa cláusula **else**, no maior estilo if-else

Veja um exemplo prático:

In [7]:

```
#Exemplo de for-else, onde o else captura a saída do comando for
for j in range(4):
    print(j)
else:
    print("Neste caso, tratamos a saída do for, com j =",j)
```

```
0
1
2
3
Neste caso, tratamos a saída do for, com j = 3
```

Bem louco, não é? Note, acima, que a cláusula **else** é executada SEMPRE quando laço **for** TERMINA! Ou seja, se por algum motivo o **for** não chegar ao seu final, o **else** não será executado, vou mostrar logo a seguir isso.

Ok, mas aí você me pergunta, por que isso é bom?

Te respondo com um exemplo, o qual visa percorrer uma lista e buscar um número inserido pelo usuário:

In [8]:

```
#lista numérica
lst_numero = [1,2,3,4,5]
#número a ser pesquisado - digitado
pesquisa = int(input("Digite um número a ser pesquisado:"))
#para cada número da lista, testar se pesquisa é igual pesquisa
for i in lst_numero:
    if i == pesquisa:
        print("Número",i,"encontrado")
        break; #caso o número seja encontrado, o break interrompe o laço for!
else:
    print("O número não existe!")
```

```
Digite um número a ser pesquisado:4
Número 4 encontrado
```

Interessante, não é? Se o número EXISTE na lista, o for é interrompido pelo comando **break** e o **else** não é executado, caso contrário, o **for** é executado até o final e a cláusula **else** tratará o que não foi encontrado na lista.

Basicamente, o **else** tratará uma condição não satisfeita em um dado momento no **for**, ou como verá a seguir, no **while**.

Isso é uma funcionalidade que o Python te fornece, diferente de outras linguagens. Caso contrário, você provavelmente usaria outras variáveis de controle e vários outros testes com **if** para representar a lógica acima.

O while e a cláusula else

O **while** é um laço de execução que sempre realiza testes para executar e, enquanto o resultado do teste for verdade, a execução continua. Com o **else** podemos capturar e tratar o cenário dessa condição ser falsa.

Veja um exemplo prático:

In [9]:

```
i = 0
#enquanto o valor de i é menor que 10, faça:
while i < 10:
    i += 1
else: #caso i seja = ou > que 10, trate:
    print("O valor de i =", i)
```

```
O valor de i = 10
```

Note que o **while** é executado e responsável por incrementar a variável i, inicialmente com 0. Ao passo que a variável torna-se igual a 10, o **while** é encerrado, portanto, sua condição é FALSA e capturada pelo **else**.

Do-it-yourself

Utilizando o comando **for**, verifique se um número informado pelo usuário é primo ou não. Utilize um intervalor de [1..NUM], onde NUM é o número informado. Trate com a cláusula **else** caso o número for

Do-it-yourself

Utilizando o comando **while**, mostre quantas vezes um número é divisível por 2. Mostre na tela se o número é divisível apenas por 2 ou se por algum outro número - pense em como capturar essas condições utilizando o **break** e a cláusula **else** na saída do laço. Dica: Não deixe o laço continuar caso a divisão seja menor ou igual a 1.

Parabéns! Você finalizou o roteiro 3!

Agora é hora de você praticar com exercícios.

Este roteiro foi inspirado em metodologias ativas de ensino.

Este conteúdo foi confeccionado pelo professor:

Marcílio Francisco de Oliveira Neto (marciliofoneto@gmail.com)