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

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 [3]:

```
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 [6]:

```
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 [8]:

```
#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: 21
O número digitado foi: 21
Digite um número: 1
O número digitado foi: 1
Digite um número: 2
O número digitado foi: 2
Digite um número: 5
O número digitado foi: 5
Digite um número: 3
O número digitado foi: 3
```

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

In [9]:

```
#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: 3
O número digitado foi: 3
Digite um número: 9
O número digitado foi: 9
Digite um número: 10
O número digitado foi: 10
ACERTO
Digite um número: 2
O número digitado foi: 2
Digite um número: 0
O número digitado foi: 0
```

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 [11]:

```
#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: 2
Digite um numero qualquer - para sair, digite 1: 5
Digite um numero qualquer - para sair, digite 1: 0
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 [13]:

```
#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
```