For Loop

Como já estamos cansados de saber, **FOR** se trata de um **laco de repetição**, onde queremos que o nosso código se repita algumas vezes!

For com números

E para loops **FOR**, onde será percorrido por números, que é mais normal, o costume é utilizarmos a função **range()**, onde será retornado um objeto do tipo **range**. Dito isso, o **FOR** interage com esse objeto por debaixo dos panos chamando o método **next()** para cada interação dessa lista que o range acaba interagindo por "debaixo dos panos".

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

```
0
1
2
3
4
```

For com letras

Além de utilizar o **FOR** iterando através de número, podemos utilizar o **FOR** para iterar através de **strings**!! Até porque, como sabemos, uma **String** é uma sequência de caracteres, logo o for irá interagir a cada caractere na **String**!

```
palavra = 'Google'
for letra in palavra:
    print(f'\'{letra}\' esta na palavra: \'{palavra}\'')
```

```
'G' esta na palavra: 'Google'
'o' esta na palavra: 'Google'
'o' esta na palavra: 'Google'
'g' esta na palavra: 'Google'
'l' esta na palavra: 'Google'
'e' esta na palavra: 'Google'
```

For com IF e ELSE

Agora vamos ver como que podemos realizar IF e ELSE dentro de um laco FOR!!

```
compra_validacao = False

for tentativa in range(3):
    if compra_validacao:
        print('a compra de R$ 10,00 foi realizado com sucesso.')
        break
    else:
        print('falha na compra, tente novamente...')

if not compra_validacao:
    print('falha na compra!!')
```

```
falha na compra, tente novamente...
falha na compra, tente novamente...
falha na compra, tente novamente...
falha na compra!!
```

Veja que nesse código, estamos tentando realizar um pagamento, e essa tentativa será feito ao menos 3 vezes pelo **FOR**. E caso a **flag compra_validacao** for **True** a mensagem de sucesso será exibida, e em seguida sairemos do loop com **break**, para que não seja exibido a mensagem repetidas vezes! Caso contrário, será exibido a mensagem de "tente novamente" pela instrução do **ELSE** a cada interação do **FOR**.

E for fim é validado se após todas as tentativas de pagamento algum deu certo, senão será printado na tela que teve "falha na compra".

ELSE como fosse BREAK do loop FOR

Porém, no **Python** temos a opção de que o bloco de instrução do **ELSE** fique fora do bloco de **loop**, assim ele funciona como uma espécie de **break** no loop **FOR**.

```python
compra_validacao = False

for tentativa in range(3):
    if compra_validacao:
        print('a compra de R$ 10,00 foi realizado com sucesso.')
        break
else:
    print('falha na compra, tente novamente...')
```

```
falha na compra, tente novamente...
```

Veja que dessa vez, logo na primeira tentativa de verificar a **compra_validacao**, a execução do nosso código acaba saindo do **loop**, sem precisar definir um **break** manualmente antes de passar por todas as tentativas! Isso porque, a definição do **ELSE** ficou fora do loop **FOR**, logo a execução do código sairá do loop automaticamente!

Loops aninhados (Nested Loops)

Nested Loop, basicamente é quando executamos **loops** dentro de outro **loop**!! Como por exemplo um laco **FOR** dentro de outro laco **FOR**!!

```python
for numero1 in range(5):
    print(f'LOOP 1 - {numero1}')

    for numero2 in range(5):
        print(f'LOOP 2 - {numero2}')
```

```
LOOP 1 - 0
LOOP 2 - 0
LOOP 2 - 1
LOOP 2 - 2
LOOP 2 - 3
LOOP 2 - 4
LOOP 1 - 1
LOOP 2 - 0
LOOP 2 - 1
LOOP 2 - 2
LOOP 2 - 3
LOOP 2 - 4
LOOP 1 - 2
```

Esse recurso é muito útil geralmente quando precisamos percorrer por uma matriz, tabela, etc.