Python intermediário

Olá, bem-vind@ à segunda parte do tutorial de Python, aqui vamos nos aprofundas em algumas técnicas mais avançadas.

Neste texto vamos passar por funções, loops, condicionais e afins, então...

Condicionais

No fluxo do nosso código podemos definir caminhos distintos, ou bifurcações, utilizando o “if”. Entenda este condicional, como se fosse um comando que diz: se tal condição for verdadeira então faça xyz.

Para escrever algo utilizando “if” basta seguir a seguinte sintaxe:

```
if <--condição-->:
<--espaço--> o que fazer se a condição for verdadeira.
```

Preste atenção com o espaço ou indentação, sem isso a sua função não vai funcionar corretamente e o código vai retornar um bug chamado IndentationError: unexpected indent. Este espaço determina que tudo o que vier depois dele estaria “dentro” desta condicional, ou seja tudo que estiver indentado logo depois da sua primeira linha do if, estará dentro daquele bloco de código e será processado caso a condição for verdadeira. Exemplo:

```
if x > y:
    print("x é maior que y")
```

Uma alternativa (que eu não recomendo muito) seria escrever tudo na mesma linha:

```
if x > y: print("x é maior que y")
```

O que não funciona:

```
if x > y:
print("x é maior que y")
```

Se ainda estiver com dúvidas a respeito da indentação, veja se este vídeo consegue te trazer maior clareza.

Vamos ver agora alguns exemplos mais complexos.

- Podemos encadear mais de um if:

```
if x > y:
print("x é maior que y")
if x > z:
print("x também é maior que z")
```

- Podemos podemos encadear um outro if caso a condição anterior não seja verdadeira (elif) :

```
if x > y:
print("x é maior que y")
if x > z:
print("x também é maior que z")
elif y > z:
print("y é maior que z")
```

Note que neste caso, a condicional elif y > z só será processada caso a condicional if x > y não for verdadeira, pois a condicional if x > z está contida na anterior.

- Podemos podemos determinar o que acontece se nenhuma das condições forem verdadeiras utilizando o else:

```
if x > y:
print("x é maior que y")
if x > z:
print("x também é maior que z")
elif y > z:
print("y é maior que z")
else:
print("nenhuma das condições anteriores é verdadeira")
```

- Podemos usar alguns operadores lógicos para construir as nossas condições, exemplo:

Utilizando o operador and — considere o and como um operador que consegue juntar o resultado de diferentes condicionais, por exemplo.

```
x = 1
y = 2
z = 0

condição - x < y and x > z
valores  -  True and True
```

O resultado desta condição seria True pois o operador and depende que todas as comparações que ele conecta sejam verdadeiras para que o resultado seja True. Agora, se tivéssemos o caso abaixo:

```
x = 1
y = 2
z = 0

condição - x < y and x < z
valores  -  True and False
```

O resultado desta operação seria False pois uma das comparações não é verdadeira.

Vale reforçar que o operador pode ser utilizado para conectar quantas comparações quisermos dentro de uma mesma condicional.

Utilizando o operador or — Utilizamos este operador quando quisermos que ao menos uma das comparações seja verdadeira, exemplo:

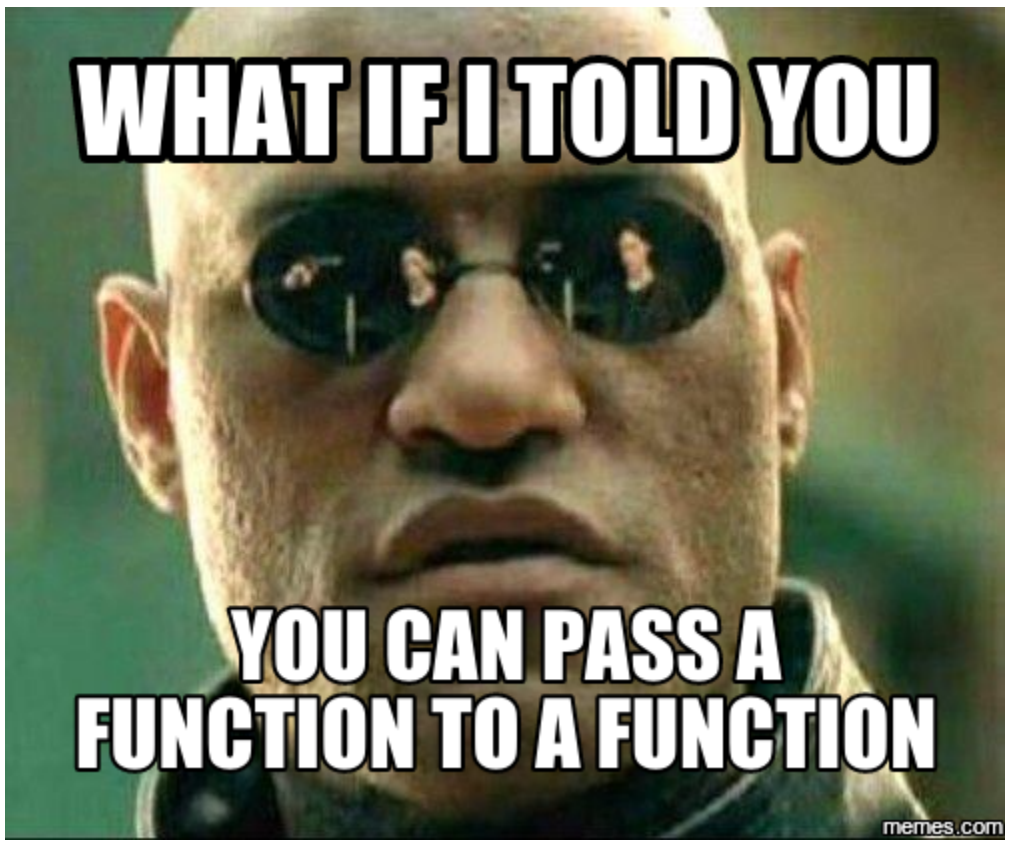
```
x = 1
y = 2
z = 0

condição - x < y **or** x < z
valores  -  True and False
```

Neste caso, o resultado seria True, pois a primeira comparação x < y é verdadeira.

Para que você se habitue mais sobre operadores lógicos em Python, separei esta página da internet.

Boa! parabéns por chegar até aqui!

Vamos seguir para o nosso próximo tópico: Funções!



Funções

Lembra de algumas funções nativas do Python, tais como print(), max() e len()? Agora você vai aprender a escrever as suas próprias funções. Uma vez que essa função for definida no seu notebook, você poderá chamá-la quantas vezes precisar, o que acaba fazendo com que o seu código fique mais curto, limpo e fácil de entender.

para escrever uma função, use a seguinte sintaxe:

```
def nome_da_função (argumento_1, argumento_2, argumento_n):
    cálculo
    return resultado
```

Exemplo:

```
def soma(x, y):
    resultado = x + y
    return resultado
```

Note que as os argumentos x e y são referenciados posteriormente no bloco da função quando definimos uma variável chamada resultado e então, definimos seu valor como a soma dos argumentos x e y e logo depois, escrevemos return para dizer à função o que ela deve retornar. Vale mencionar que sem o return a função não retornará nada. Por exemplo, se definirmos a função da seguinte forma:

```
def soma(x, y):
    resultado = x + y
```

quando escrevêssemos soma (1, 2) a função não retornaria nada. Agora se a função fosse definida como mostrado anteriormente:

```
def soma(x, y):
    resultado = x + y
    return resultado
```

O resultado de soma (1, 2) seria 3.

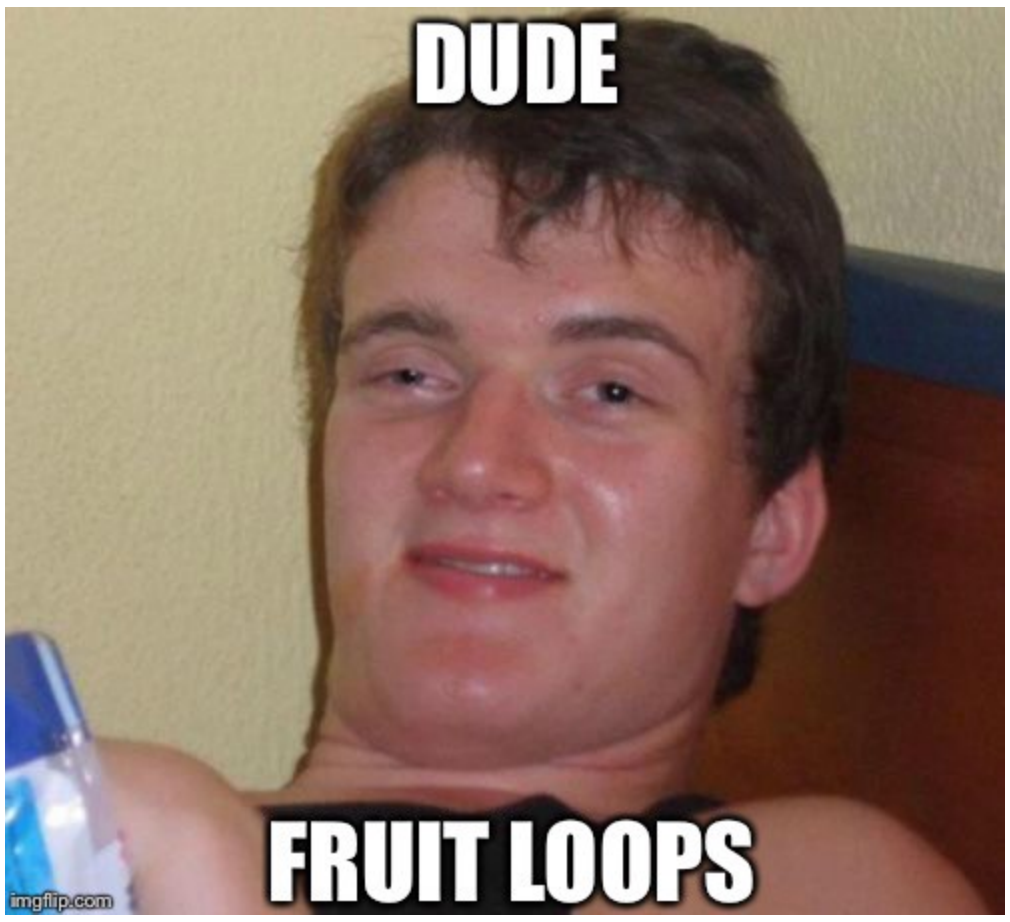
Uma função pode conter qualquer tipo de calculo, condicional, loops (ainda por vir), até mesmo outras funções. Abaixo, um exemplo de função um pouco mais complexa:

```
def valor_absoluto(num):

    if num >= 0:
        return num
    else:
        return -num
```

Se quiser aprender mais sobre as funções, indico esta página.

Sugiro que você escreva algumas e faça alguns testes no seu notebook.

Loops e iterações



- **For loops**

Os loops que começam em for são utilizados principalmente para iterar, ou em outras palavras, realizar operações repetidas em cima de um objeto. No caso vamos utilizar a lista. A sintaxe segue abaixo:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8]

for item in lista:
    print(item+1)
```

O que este código quer dizer: podemos lê-lo da seguinte forma: para cara item (item) da lista, imprima o valor do item + 1. Como a lista possui diversos integers, o Python entende que precisa passar por cada um deles e realizar a operação de imprimir o número em questão acrescido de 1.

No exemplo acima, eu utilizei a palavra item como referência para o loop, mas podemos usar qualquer outra sem problemas.

Seguem alguns outros exemplos (lembre-se de testá-los em seu notebook):

```
lista = [1, 2, 3, 4, 5, 6, 7, 8]

for n in lista:
    calculo = n*10
    print("{} vezes 10 é igual a {}".format(n,calculo))
```

```
lista = [0, 1, 2, 3, 4, 5, 6]
lista2 = [120, 34, 45, 71, 34, 19, 90]
lista3 = [21, 34, 51, 63, 23, 65, 76]

for n in lista:
    calculo = lista2[n] * lista3[n]
    print("{} vezes {} é igual a {}".format(lista2[n],lista3[n], calculo))
```

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]
numeros_impares = 0
numeros_pares = 0

for x in numeros:
        if x % 2 == 0:
             numeros_impares+=1
        else:
             numeros_pares+=1
print("Quantidade de numeros pares :",numeros_pares)
print("Quantidade de numeros impares :",numeros_impares)
```

No exemplo acima eu realizo duas novas operações:

- x += 1 → O que esta operação quer dizer? Ela não apenas soma o valor de x com 1, mas também redefine o valor de x como o resultado desta soma, exemplo:

  x = 5

  x += 1 resulta em 6
  x += 1 resulta em 7

Outros operadores similares que seguem a mesma lógica são: -=, *=, /=, **=.

- número % número → divide o número da esquerda pelo número da direita e retorna o valor remanescente. por exemplo:

  6 % 3 resulta em 0
  10 % 5 resulta em 0
  5 % 3 resulta em 2
  5 % 2 resulta em 1
  3 % 2 resulta em 1
  2 % 3 resulta em 2

- **While loops**

Os loops que começam em while continuarão a serem processados até que uma condição preestabelecida seja atingida ou que você coloque o comando break em algum lugar. Exemplos:

```
x = 10
while x != 0:
    print(x)
    x -= 1
```

Portanto, o que acontece no exemplo acima é que primeiro definimos uma variável x com o valor 10, e então escrevemos uma função que diz: enquanto x for diferente de 0, imprima o valor de x e subtraia 1.

Então, neste caso, o código continuará rodando enquanto o valor de x não atingir 0.Lembre-se de testar esses exemplos no seu notebook.

Podemos também usar o comando break em algum momento do loop para pará-lo, segue exemplo:

```
x = 10
```

```
while x != 0:
    print(x)
    x -= 1
    if x == 2:
        break
```

Esse é o básico do while, se quiser saber mais sobre esse loop, confira esta página.

Muito bem!! Parabéns por ter estudado até aqui!

Agora, vamos aos exercícios! Eles vão seguir a mesma lógica dos exercícios do texto anterior, ou seja, você vai encontrar uma série de inputs e outputs, pense nos inputs como os dados iniciais e os outputs como o resultado do seu código, sua tarefa é encontrar o caminho para transformar o input em output.

E lembre-se, aqui já espero que você recorra à comunidade (slack ou internet) para chegar nas soluções. Vale comentar também que muitas vezes existem vários caminhos possíveis para se chegar à resposta, o que está no gabarito é apenas um destes possíveis caminhos.

Arquivo: https://drive.google.com/drive/folders/1TIN2yvT5elCneRDcvsntpwYvO-pWFaLz?usp=sharing

Antes de sair, não deixe de assistir este vídeo :) ele funciona como um ótimo complemento.

Além de consolidar mais o seu conhecimento em Python e se deparar com alguns desafios, neste tutorial você passou por condicionais, funções e loops. Se você quiser continuar o seu aprendizado, existem dois cursos gratuitos que eu recomendo. Em ambos, o escopo é bem parecido com esse tutorial, mas eles possuem mais exercícios, então talvez seja interessante para você praticar.

Seguem os links.

https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-1-python-basics?ex=1

https://www.codecademy.com/learn/learn-python

Grande abraço e até logo!