**Cortes de NumPy**

***Mário Leite***

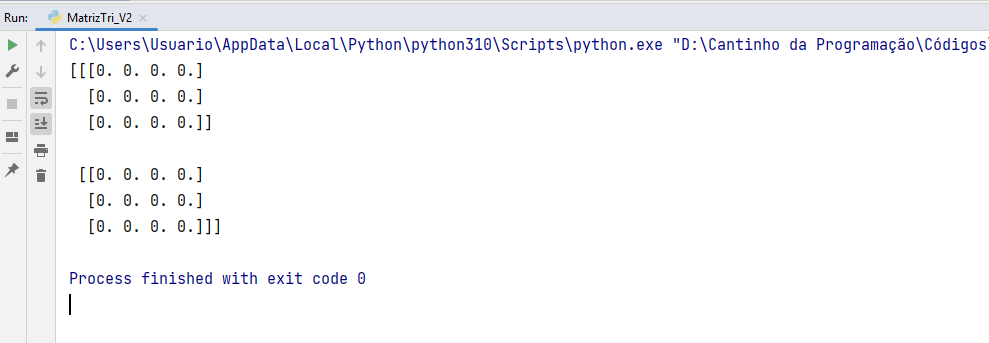
...

Uma profissão muito em alta no momento é “Ciência de Dados” (*Data Science* na tradução para o Inglês - e conhecida pela sigla **DS**); o motor principal desta aeronave de conhecimentos é a biblioteca **NumPy** do Python; embora outras linguagens e outras bibliotecas também possam ser utilizadas. Um exemplo é a biblioteca **Pandas**, que também pertence ao pacote do Python; mas NumPy é mais eficaz e mais completa para tratamento de *arrays*, por isto vamos nos ater à esta biblioteca e à linguagem Python (embora a linguagem **R** também seja uma boa opção em DS). Entre os vários recursos oferecidos por NumPy está o **NDArray** (N-dimensional *array*), que é um contêiner de estrutura de dados com várias funções e recursos de computação numérica. Essa estrutura é projetada para armazenar dados em várias dimensões (1D, 2D, 3D, etc.), permitindo a manipulação eficiente de grandes conjuntos de dados homogêneos. E além de manipular estruturas multidimensionais, NDArray também oferece vários outros recursos: *tipagens*, *operações* *vetorizadas*, *broadcasting*, *etc*, que podem ser vistos nos dois *links* abaixo...

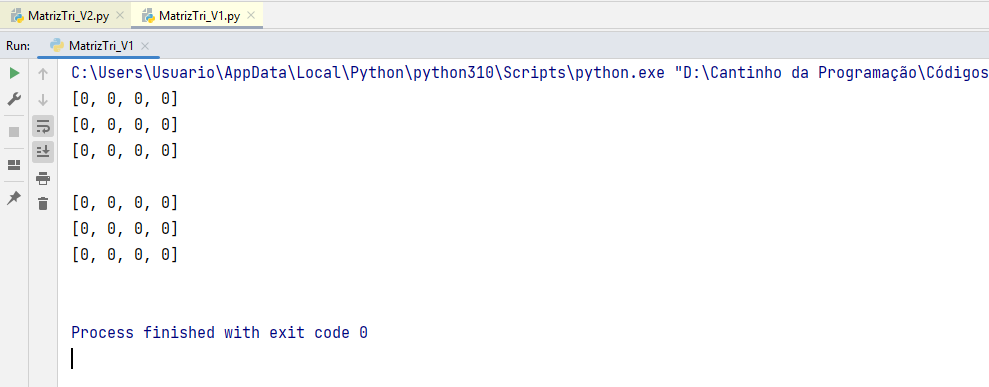
<https://numpy.org/devdocs/reference/generated/numpy.ndarray.html> <https://numpy.org/doc/stable/reference/arrays.ndarray.html>

Mas, na prática, qual é a real importância dessas bibliotecas na implementação de programas? Muito! Nos casos de *loops* para repetir um bloco de linhas de código, o uso dessas bibliotecas torna a solução muito mais eficiente! Por exemplo, vamos supor que seja necessário trabalhar com uma matriz tridimensional de inteiros: **2x3x4**. Uma solução tradicional é mostrada abaixo no programa **“MatrizTri\_V1”** com *loops* explícitos aninhados; e uma segunda solução usando recursos do NDArray é apresentada no programa **“MatrizTri\_V2”.** Note que a versão **V2** é muito mais eficiente; o código fica mais limpo e mais compacto. Na primeira versão **V1** foi necessário usar *loops* aninhados para tratar as matrizes, o que implicou escrever dezoito linhas efetivas de código (sem as linhas em banco e os comentários). Já na versão V2, usando os recursos da biblioteca Numpy, foram necessárias apenas três linhas efetivas; a função ***zeros()*** resolveu a parte mais crucial do problema. As **figuras 1** e **2** mostram saídas dos programas **“MatrizTri\_V1”** e **“MatrizTri\_V2”**, respectivamente; em ambas podemos ver a matriz-array por duas perspectivas diferentes, mas mostrando que é uma matriz **2x3x4**.

--------------------------------------------------------------------------------------------------------------------------------------



**Figura 1 - Saída do programa “MatrizTri\_V1”**



**Figura 2 - Saída do programa “MatrizTri\_V2”**

'''

**MatrizTri\_V1.py**

--------------------------------------------------------------------------------

Cria uma matriz tridimensional (2x3x4) sem usar recursos da biblioteca "Numpy".

--------------------------------------------------------------------------------

'''

*#Definindo as dimensões do array*

dim1 = 2 *#primeira dimensão*

dim2 = 3 *#segunda dimensão*

dim3 = 4 *#terceira dimensão*

*#Inicia o array em vazio*

matArray = []

*#Preenche a matriz-array com zeros usando loops aninhados*

**for** i **in** **range**(dim1):

matriz2D = [] *#cria uma nova "matriz" (2D)*

**for** j **in** **range**(dim2):

linha = [] *# cria uma nova linha*

**for** k **in** **range**(dim3):

linha.**append**(0) *#adiciona zero à linha*

matriz2D.**append**(linha) *#adiciona a linha à matriz*

matArray.**append**(matriz2D) *#adiciona a matriz ao array*

*#Inverte as linhas de cada matriz para melhor compreensão*

**for** mat **in** matArray:

mat.**reverse**()

*#Exibe a matriz-array resultante*

**for** mat **in** matArray:

**for** linha **in** mat:

**print**(linha)

**print**() *#para separar as matrizes*

**#Fim do programa MatrizTri\_V1" -------------------------------------------------**

'''

**MatrizTri\_V2.py**

----------------------------------------------------------------------------

Cria uma matriz tridimensional (2x3x4) usando recursos da biblioteca "Numpy".

----------------------------------------------------------------------------

'''

*#Importa a biblioteca NumPy usando um alias*

**import** **numpy** **as** np

*#Cria e preenche com zeros matriz-array 2x3x4*

matArray = np.**zeros**((2, 3, 4)) *#****np*** *é usado como o alias padrão da biblioteca*

*#Exibe a mariz-array resultante*

**print**(matArray)

**#Fim do programa MatrizTri\_V2" ---------------------------------------------**