Python Tutorial 1 Exercises with Solutions

January 18, 2022

This is the solution of "Python Tutorial 1 Exercises" for Prof. Xin Tong's DSO 530 class at the University of Southern California in spring 2022.

1. Write a Python function to find the Max of three numbers.

hint: You could write a Python function to find the Max of two numbers, then write a new function using the first function to find the Max of three numbers.

Test Code:

print(max_of_three(3, 6, -5))

# the created Python function by you named *max_of_three*

Answer:

[1]:
```python
def max_of_two( x, y ):
    if x > y:
        return x
    else:
        return y
def max_of_three( x, y, z ):
    return max_of_two( x, max_of_two( y, z ) )

print(max_of_three(3, 6, -5))
```

```
6
```

2. What's the difference between *np.zeros* and *np.empty*?

Answer:

empty, unlike *zeros*, does not set the array values to zero, and may therefore be marginally faster. On the other hand, it requires the user to manually set all the values in the array, and should be used with caution.

np.zeros

Return a new array setting values to zero.

```
[2]: import numpy as np
     np.zeros((2, 3, 3))
```

```
[2]: array([[[0., 0., 0.],
             [0., 0., 0.],
             [0., 0., 0.]],

            [[0., 0., 0.],
             [0., 0., 0.],
             [0., 0., 0.]]])
```

np.empty

Return a new uninitialized array. (the elements may be zeros or not)

```
[3]: np.empty((2, 3, 3))
```

```
[3]: array([[[1.89146896e-307, 1.37961302e-306, 6.23053614e-307],
             [6.23053954e-307, 9.34609790e-307, 8.45593934e-307],
             [9.34600963e-307, 1.86921143e-306, 6.23061763e-307]],

            [[8.90104239e-307, 6.89804132e-307, 1.22387805e-307],
             [1.42418172e-306, 2.04712906e-306, 7.56589622e-307],
             [1.11258277e-307, 8.90111708e-307, 1.44635573e-307]]])
```

From official documentation:

https://numpy.org/devdocs/reference/generated/numpy.empty.html?highlight=empty#numpy.empty

3. Write a NumPy program to multiply a 5x3 matrix by a 3x2 matrix and create a real matrix product.

Requirement:

Use np.random.randn to generate the 5x3 matrix and the 3x2 matrix.

Set the random seed to 15 before generating the matrices: *np.random.seed(15)*.

Answer:

```
[4]: import numpy as np
     np.random.seed(15)
     x = np.random.randn(5,3)
     print("First array:")
     print(x)
     y = np.random.randn(3,2)
     print("Second array:")
print(y)
z = np.dot(x, y)
print("Dot product of two arrays:")
print(z)
```

```
First array:
[[-0.31232848  0.33928471 -0.15590853]
 [-0.50178967  0.23556889 -1.76360526]
 [-1.09586204 -1.08776574 -0.30517005]
 [-0.47374837 -0.20059454  0.35519677]
 [ 0.68951772  0.41058968 -0.56497844]]
Second array:
[[ 0.59939069 -0.16293631]
 [ 1.6002145   0.6816272 ]
 [ 0.0148801  -0.08777963]]
Dot product of two arrays:
[[ 0.35340159  0.29584093]
 [ 0.04995007  0.39713854]
 [-2.40204898 -0.53610729]
 [-0.59966929 -0.09071893]
 [ 1.06191512  0.21711522]]
```

4. Read the code and answer the following questions:

[5]:
```
import numpy as np
arr1 = np.arange(27).reshape(3,3,3)
arr2 = arr1[1].copy()
arr2[2] = 50
```

What's the output of *arr1[2,0]* and *arr2[2,0]*?

Answer:

[6]:
```
arr1[2,0]
```

[6]:
```
array([18, 19, 20])
```

[7]:
```
arr2[2,0]
```

[7]:
```
50
```

To understand the above , you can see what is *arr1* and *arr2*:

[8]:
```
arr1
```

```
[8]: array([[[ 0,  1,  2],
         [ 3,  4,  5],
         [ 6,  7,  8]],

        [[ 9, 10, 11],
         [12, 13, 14],
         [15, 16, 17]],

        [[18, 19, 20],
         [21, 22, 23],
         [24, 25, 26]]])
```

```
[9]: np.shape(arr1)
```

```
[9]: (3, 3, 3)
```

```
[10]: arr2
```

```
[10]: array([[ 9, 10, 11],
        [12, 13, 14],
        [50, 50, 50]])
```

```
[11]: np.shape(arr2)
```

```
[11]: (3, 3)
```

5. Read the code and answer the following questions:

```
[12]: import numpy as np
      arr1 = np.array([[1,21,3,22,5],[23,7,24,9,25],[11,26,27,28,15]])
      arr1
```

```
[12]: array([[ 1, 21,  3, 22,  5],
        [23,  7, 24,  9, 25],
        [11, 26, 27, 28, 15]])
```

(1) Get array *arr2* transformed from *arr1* with only one line of code:

```
[13]: arr2 = np.array([[21,22,23,24],[25,26,27,28,]])
      arr2
```

```
[13]: array([[21, 22, 23, 24],
        [25, 26, 27, 28]])
```

Answer:

[14]:
```python
arr2 = arr1[arr1>20].reshape(2,4)
arr2
```

[14]:
```
array([[21, 22, 23, 24],
       [25, 26, 27, 28]])
```

(2) What's the output of *arr2[-1,-2]*?

Answer:

[15]:
```python
arr2[-1,-2]
```

[15]:
```
27
```

6. Write a NumPy program to complete the following functions:

(1)Create an array 0-19 of 4,5 shape and print it

(2)Swap axis0 with axis1 and print it

(2)Swap swap column1 with column4 and print it

Answer:

(1)

[16]:
```python
import numpy as np
arr = np.arange(20).reshape(4,5)
print("Original array:")
print(arr)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

(2)

[17]:
```python
print("After swapping axis0 with axis1 using 'swapaxes':")
arr1 = arr.swapaxes(0,1)
print(arr1)
```

```
After swapping axis0 with axis1 using 'swapaxes':
[[ 0  5 10 15]
 [ 1  6 11 16]
 [ 2  7 12 17]
 [ 3  8 13 18]
 [ 4  9 14 19]]
```

or

[18]:
```python
print("After swapping axis0 with axis1 using 'transpose':")
arr2 = arr.transpose(1,0)
print(arr2)
```

```
After swapping axis0 with axis1 using 'transpose':
[[ 0  5 10 15]
 [ 1  6 11 16]
 [ 2  7 12 17]
 [ 3  8 13 18]
 [ 4  9 14 19]]
```

(3)

[19]:
```python
print("Original array:")
print(arr)
print("\nAfter swapping column1 with column4:")
arr3 = arr
arr3[:,[0,3]] = arr3[:,[3,0]]
print(arr3)
```

```
Original array:
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

After swapping column1 with column4:
[[ 3  1  2  0  4]
 [ 8  6  7  5  9]
 [13 11 12 10 14]
 [18 16 17 15 19]]
```

7. Write a NumPy program generate 50 random numbers from N(2,3)(the normal distribution with mean 2 and variance 3) and compute their mean and variance.

Requirements:

Write the program two times:

for the first time, set seed to 1 and for the second time, set seed to 2.

See the difference between the two results.

Answer:

[20]:
```python
import numpy as np
print("For the first time: seed = 1:")
```

```
np.random.seed(1)
x = np.random.randn(50) * np.sqrt(3) + 2
print("Mean:")
print(np.mean(x))
print("Variance:")
print(np.var(x))
```

```
For the first time: seed = 1:
Mean:
1.9558069869033519
Variance:
2.8203101328818163
```

[21]:
```
print("For the second time: seed = 2:")
np.random.seed(2)
x = np.random.randn(50) * np.sqrt(3) + 2
print("Mean:")
print(np.mean(x))
print("Variance:")
print(np.var(x))
```

```
For the second time: seed = 2:
Mean:
1.7579563395376352
Variance:
3.248931171309009
```

More about np.random:

https://docs.scipy.org/doc/numpy-1.14.0/reference/routines.random.html