Numpy Practice. Intelligent Systems

Jose Torres Postigo

18 de abril de 2024

To be able to solve the next excercises, it is mandatory to import the next libraries:

```
import numpy as np
import math
```

Excercise 1. *Write a program that creates a unidimensional array with the positive integers smaller than 100 that are multiples of 4.*

Solution:

```
def f1():
    e1 = np.arange(1, 101)
    return e1[e1 % 4 == 0]
```

Output generated:

```
[  4   8  12  16  20  24  28  32  36  40  44  48  52  56  60
64  68  72 76  80  84  88  92  96 100]
```

Excercise 2. *Write a program that creates a bidimensional array of 5 rows and 4 columns with the integers from 0 to 19.*

Solution:

```
def f2():
    return np.arange(20).reshape(5, 4)
```

Output generated:

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

Excercise 3. *Write a program that inverts the order of the elements of a unidimensional array.*

Solution:

```
def f3(a: np.ndarray):
    return np.flip(a)
```

The next output is generated using the unidimensional array that goes from 0 to 100:

```
[100  99  98  97  96  95  94  93  92  91  90  89  88  87  86
85  84  83  82  81  80  79  78  77  76  75  74  73  72  71  70
69  68  67  66  65  64  63  62  61  60  59  58  57  56  55  54
53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38
37  36  35  34  33  32  31  30  29  28  27  26  25  24  23  22
21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6
5  4  3  2  1]
```

Excercise 4. *Write a program that inverts the order of the rows of a bidimensional array.*

Solution:

```
def f4(a: np.ndarray):
    return np.flip(a, axis=0)
```

For generating the output, the bidimensional array returned by f2() is used:

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

Excercise 5. *Write a program that computes the mean of the elements of each column of a bidimensional array.*

Solution:

```
def f5(a: np.ndarray):
    return np.mean(a, axis=0)
```

For generating the output, the bidimensional array returned by f2() is used:

```
[ 8.  9. 10. 11.]
```

Excercise 6. *Write a program that reshapes a unidimensional array into a bidimensional array with 4 rows and 3 columns.*

Solution:

```
def f6(a: np.ndarray):
    return a.reshape(4, 3)
```

The output will be generated with an unidimensional array that goes from 0 to 11:

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Excercise 7. *Write a program that, given a unidimensional array, checks whether it can be reshaped to a bidimensional array with the same number of rows and columns, i.e., a square matrix. In case that it is possible, it must print the square matrix. Otherwise, it must print an informative message.*

Solution:

```
def f7(a: np.ndarray):
    n = math.sqrt(a.size)
    if not n.is_integer():
        return (''[*] ERROR: The array passed cannot be reshaped'' +
                ''into a square matrix.'')

    return a.reshape(int(n), int(n))
```

If a unidimensional array who can be reshaped is passed, e.g. one that goes from 0 to 4, the output is:

```
[[0 1]
 [2 3]]
```

If the array cannot be reshaped, prints the next error:

```
[*] ERROR: The array passed cannot be reshaped into a square matrix.
```

Excercise 8. *Write a program that, given a bidimensional array, finds the maximum of each row.*

Solution:

```
def f8(a: np.ndarray):
    return a.max(axis=1)
```

The output will be generated by the f2() array:

```
[ 3  7 11 15 19]
```

Excercise 9. *Write a program that, given a unidimensional array, finds the number of occurrences of each of its unique values.*

Solution:

```
def f9(a: np.ndarray):
    keys, values = np.unique(a, return_counts=True)

    return dict(zip(keys, values))
```

For this output, the next array will be used:

```
[ 1, 2, 4, 2, 9, 3, 10, 9, 1, 1]
```

The output is:

```
{1: 3, 2: 2, 3: 1, 4: 1, 9: 2, 10: 1}
```

Excercise 10. *Normalize a bidimensional array with 4 rows and 3 columns by subtracting the mean and dividing by the standard deviation on each column.*

Solution:

```
def f10(a: np.ndarray):
    means = np.reshape(np.mean(a, axis=0), (1,3))
    std = np.reshape(np.std(a, axis=0), (1,3))

    return ((a - means) / std)
```

The output will be generated by the next array:

```
[[ 0.0,  0.0,  0.0], [10.0, 10.0, 10.0],
[20.0, 20.0, 20.0], [30.0, 30.0, 30.0]]
```

Thus, the output is:

```
[[-1.34164079 -1.34164079 -1.34164079]
 [-0.4472136  -0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079  1.34164079]]
```

Excercise 11. *Normalize a bidimensional array with 4 rows and 3 columns by subtracting the mean and dividing by the standard deviation on each row.*

Solution:

```
def f11(a):
    means = np.reshape(np.mean(a, axis=1), (1,3))
    std = np.reshape(np.std(a, axis=1), (1,3))

    return ((a - means) / std)
```

For the output, the next array, randomly generated for this example, is used:

```
[[0.77395605, 0.43887844, 0.85859792],
 [0.69736803, 0.09417735, 0.97562235],
 [0.7611397,  0.78606431, 0.12811363],
 [0.45038594, 0.37079802, 0.92676499]]
```

Thus, the output is:

```
[[ 0.46061695 -1.38826962  0.92765268]
 [ 0.29439964 -1.34511327  1.05071363]
 [ 0.66577875  0.74764474 -1.4134235 ]
 [-0.53878305 -0.86298855  1.40177159]]
```

Excercise 12. *Write a program that, given a bidimensional array, finds the indices (rows and columns) of the minimum and maximum elements of the array.*

Solution:

```
def f12(a: np.ndarray):
    min_indices = np.unravel_index(np.argmin(a), a.shape)
    max_indices = np.unravel_index(np.argmax(a), a.shape)

    return min_indices, max_indices
```

For the output, the input used is the array generated by f2():

```
((0, 0), (4, 3))
```

Excercise 13. *Write a program that sorts the rows of a bidimensional array according to the values of the first column.*

Solution:

```
def f13(a: np.ndarray):
    sorted_indices = np.argsort(a[..., 0])
    # sorted_indices = np.argsort(a[:, 0]) is also valid

    return a[sorted_indices]
```

The bidimensional array for the next output is:

```
[[   4   67]
 [   1 4532]
 [   3    1]
 [  10   11]]
```

The output generated by this bidimensional array is:

```
[[   1 4532]
 [   3    1]
 [   4   67]
 [  10   11]]
```

Excercise 14. *Write a program that generates a bidimensional array with 7 rows and 5 columns randomly according to the normal distribution, and then sets to zero all negative elements.*

Solution:

```
def f14():
    arr = np.random.normal(size=(7,5))
    arr[arr<0] = 0

    return arr
```

The next output is just an example, as the bidimensional array is randomly generated according to the standard normal distribution ($\mathcal{N}(\mu,\sigma^2)$), where $\mu = 0,\ \sigma = 1$.

```
[[2.34227685 0.         0.13723387 0.         0.76107772]
 [1.16906686 0.         1.35425979 0.         0.83649639]
 [0.         0.         0.         0.         0.        ]
 [1.87695855 0.         0.         0.         0.        ]
 [0.         1.43702759 2.26730391 0.         1.57711027]
 [0.         1.01056171 0.         0.         0.        ]
 [0.101735   0.         0.         0.         0.        ]]
```

Excercise 15. *Write a program that, given a unidimensional array and a positive integer k, finds the indices of the k largest values of the array.*

Solution:

```
def f15(a: np.ndarray, k: int):
    return np.argsort(a)[-k:]
```

For the sake of obtain this output, I generated a unidimensional array according to the standard normal distribution:

```
[-0.63651784 -0.7445809   0.24883129  1.12457502  0.20856384 -0.16084675
  1.17244896 -1.23206661  0.20771277  1.66158979]
```

The output generated by this array, when $k = 5$ is:

```
[4 2 3 6 9]
```

Excercise 16. *Write a program that generates a bidimensional array with 6 rows and 7 columns randomly according to the uniform distribution between 0 and 1, and then sets to zero the two first columns and sets to one the three last columns.*

Solution:

```
def f16():
    a = np.random.uniform(size=(6, 7))
    a[:, :2] = 0
    a[:, -3:] = 1

    return a
```

Output:

```
[[0.   0.   0.54599679 0.70451806 1.   1.   1.  ]
 [0.   0.   0.76914826 0.41651008 1.   1.   1.  ]
 [0.   0.   0.95342398 0.2983864  1.   1.   1.  ]
 [0.   0.   0.08806097 0.53866138 1.   1.   1.  ]
 [0.   0.   0.71054197 0.75436051 1.   1.   1.  ]
 [0.   0.   0.76881925 0.81687892 1.   1.   1.  ]]
```