Exercises_wk5 (including solutions)

Before you start, first import the NumPy library (as `np`).

```
In [1]: import numpy as np
```

Q1: Create an array ('arr_1') by manually entering the following values in NumPy's `array()` function: [10, 25, 20, 35, 30], and print this aray. Then (in seperate cells) perform the following operations: (1) check the shape (i.e., the length of the array along each dimension) of the array, (2) check the number of dimensions of the array, and (3) check the number of elements of the array.

```
In [2]: arr_1 = np.array([10, 25, 20, 35, 30])
        arr_1

Out[2]: array([10, 25, 20, 35, 30])
```

```
In [3]: arr_1.shape

Out[3]: (5,)
```

```
In [4]: arr_1.ndim

Out[4]: 1
```

```
In [5]: arr_1.size

Out[5]: 5
```

Q2: Create the following list-of-lists: [[1, 2, 3, 4, 5], [10, 25, 20, 35, 30]] and then create an array ('arr_2') by passing this list-of-lists to NumPy's `array()` function, and print this array. Then (in seperate cells) perform the following operations: (1) check the shape (i.e., the length of the array along each dimension) of the array, (2) check the number of dimensions of the array, and (3) check the number of elements of the array.

```
In [6]: lst = [[1, 2, 3, 4, 5], [10, 25, 20, 35, 30]]
        arr_2 = np.array(lst)
        arr_2

Out[6]: array([[ 1,  2,  3,  4,  5],
               [10, 25, 20, 35, 30]])
```

```
In [7]: arr_2.shape

Out[7]: (2, 5)
```

```
In [8]: arr_2.ndim

Out[8]: 2
```

```
In [9]: arr_2.size

Out[9]: 10
```

Q3: Create a two-dimensional array ('arr_3') consisting of three rows and four columns by using NumPy's `arange()` and `reshape()` functions. The array should contain the following values: [0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]. Print the array, and then (in a seperate cell) convert this array to a one-dimensional array by using NumPy's `ravel()` function.

```
In [10]: arr_3 = np.arange(12).reshape(3, 4)
         arr_3

Out[10]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])

In [11]: arr_3.ravel()

Out[11]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Q4: Create a two-dimensional array ('arr_4') consisting of two rows and four columns by using NumPy's `linspace()` and `reshape()` functions. The array should contain values between 0 and (up to and including) 6. Print the array, and then (in a seperate cell) transpose this array (i.e., invert the columns with the rows) by using NumPy's `transpose()` function.

```
In [12]: arr_4 = np.linspace(0, 6, 8).reshape(2, 4)
         arr_4

Out[12]: array([[0.        , 0.85714286, 1.71428571, 2.57142857],
                [3.42857143, 4.28571429, 5.14285714, 6.        ]])

In [13]: arr_4.transpose()

Out[13]: array([[0.        , 3.42857143],
                [0.85714286, 4.28571429],
                [1.71428571, 5.14285714],
                [2.57142857, 6.        ]])
```

Q5: Create a two-dimensional array ('arr_5') consisting of five rows and five columns by using NumPy's `arange()` and `reshape()` functions. The array should contain the following values: [10, 20, 30, 40, 50], [60, 70, 80, 90, 100], [110, 120, 130, 140, 150], [160, 170, 180, 190, 200], [210, 220, 230, 240, 250]. Print the array, and then (in seperate cells) perform the following operations: (1) extract the second row of the array, (2) extract the second column of the array, (3) extract the second and fourth row of the array, (4) extract the second and fourth column of the array, and (5) extract the value in the middle of the array (i.e., 130) using negative indexing.

```
In [14]: arr_5 = np.arange(10, 260, 10).reshape(5, 5)
         arr_5

Out[14]: array([[ 10,  20,  30,  40,  50],
                [ 60,  70,  80,  90, 100],
                [110, 120, 130, 140, 150],
                [160, 170, 180, 190, 200],
                [210, 220, 230, 240, 250]])
```

```
In [15]: arr_5[1, :]

Out[15]: array([ 60,  70,  80,  90, 100])
```

```
In [16]: arr_5[:, 1]

Out[16]: array([ 20,  70, 120, 170, 220])
```

```
In [17]: arr_5[1:4:2, :]

Out[17]: array([[ 60,  70,  80,  90, 100],
                [160, 170, 180, 190, 200]])
```

```
In [18]: arr_5[:, 1:4:2]

Out[18]: array([[ 20,  40],
                [ 70,  90],
                [120, 140],
                [170, 190],
                [220, 240]])
```

```
In [19]: arr_5[-3, -3]

Out[19]: 130
```

Q6: Create a two-dimensional array ('arr_6') consisting of five rows and five columns by using NumPy's `arange()` and `reshape()` functions. The array should contain the following values: [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]. Print the array, and then (in seperate cells) perform the following operations: (1) use Boolean indexing to extract all elements with an even value from the array, and (2) use Boolean indexing to replace all elements with an odd value with the number 1000.

```
In [20]: arr_6 = np.arange(25).reshape(5, 5)
         arr_6

Out[20]: 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]])
```

```
In [21]: arr_6[arr_6 % 2 == 0]

Out[21]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24])
```

```
In [22]: arr_6[arr_6 % 2 != 0] = 1000
         arr_6
```

```
Out[22]: array([[   0, 1000,    2, 1000,    4],
                [1000,    6, 1000,    8, 1000],
                [  10, 1000,   12, 1000,   14],
                [1000,   16, 1000,   18, 1000],
                [  20, 1000,   22, 1000,   24]])
```

Q7: Create a first array ('arr_7a') by manually entering the following values in NumPy's `array()` function: [0, 1, 2, 3], convert it to a two-dimensional (2x2) array by using NumPy's `reshape()` function, and print this array. Next, create a second array ('arr_7b') by manually entering the following values in NumPy's `array()` function: [5, 4, 3, 2], convert it to a two-dimensional (2x2) array by using NumPy's `reshape()` function, and print this array. Then (in separate cells) perform the following operations on these two arrays: (1) concatenate the two arrays vertically (i.e., along the rows) and assign the resulting array to a variable ('arr_7'), and (2) split 'arr_7' into four equally shaped arrays along the vertical axis (i.e., along the rows).

```
In [23]: arr_7a = np.array([0, 1, 2, 3]).reshape(2, 2)
         arr_7a
```

```
Out[23]: array([[0, 1],
                [2, 3]])
```

```
In [24]: arr_7b = np.array([5, 4, 3, 2]).reshape(2, 2)
         arr_7b
```

```
Out[24]: array([[5, 4],
                [3, 2]])
```

```
In [25]: arr_7 = np.concatenate((arr_7a, arr_7b), axis=0)
         arr_7
```

```
Out[25]: array([[0, 1],
                [2, 3],
                [5, 4],
                [3, 2]])
```

```
In [26]: arr_7 = np.vsplit(arr_7, 4)
         arr_7
```

```
Out[26]: [array([[0, 1]]), array([[2, 3]]), array([[5, 4]]), array([[3,
         2]])]
```

Q8: Create a first array ('arr_8a') by manually entering the following values in NumPy's `array()` function: [0, 1, 2, 3], convert it to a two-dimensional (2x2) array by using NumPy's `reshape()` function, and print this array. Next, create a second array ('arr_8b') by manually entering the following values in NumPy's `array()` function: [5, 4, 3, 2], convert it to a two-dimensional (2x2) array by using NumPy's `reshape()` function, and print this array. Then (in separate cells) perform the following operations on these two arrays: (1) subtract 1 from the value of all elements of 'arr_8b', (2) multiply the two arrays element-wise, and (3) calculate the matrix product of the two arrays.

```
In [27]: arr_8a = np.array([0, 1, 2, 3]).reshape(2, 2)
         arr_8a

Out[27]: array([[0, 1],
                [2, 3]])
```

```
In [28]: arr_8b = np.array([5, 4, 3, 2]).reshape(2, 2)
         arr_8b

Out[28]: array([[5, 4],
                [3, 2]])
```

```
In [29]: arr_8b -= 1
         arr_8b

Out[29]: array([[4, 3],
                [2, 1]])
```

```
In [30]: arr_8a * arr_8b  # or np.multiply(arr_8a, arr_8b)

Out[30]: array([[0, 3],
                [4, 3]])
```

```
In [31]: np.dot(arr_8a, arr_8b)

Out[31]: array([[ 2,  1],
                [14,  9]])
```

Q9: Create a two-dimensional array ('arr_9') consisting of five rows and five columns by using NumPy's `arange()` and `reshape()` functions. The array should contain the following values: [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]. Print the array, and then (in seperate cells) perform the following operations: (1) calculate the average of all values along the rows of the array, (2) calculate the average of all values along the columns of the array, (3) calculate the sum of all values of the second row of the array, (4) calculate the sum of all values of the second column of the array, (5) calculate the sum of all values of the second and fourth row of the array, and (6) calculate the sum of all values of the second and fourth column of the array.

```
In [32]: arr_9 = np.arange(25).reshape(5, 5)
         arr_9

Out[32]: 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]])
```

```
In [33]: arr_9.mean(axis=0)

Out[33]: array([10., 11., 12., 13., 14.])
```

```
In [34]: arr_9.mean(axis=1)

Out[34]: array([ 2.,  7., 12., 17., 22.])
```

```
In [35]: arr_9[1, :].sum()

Out[35]: 35
```

```
In [36]: arr_9[:, 1].sum()

Out[36]: 55
```

```
In [37]: arr_9[1:4:2, :].sum()

Out[37]: 120
```

```
In [38]: arr_9[:, 1:4:2].sum()

Out[38]: 120
```

Q10: Define a function ('descr_stats') that calculates and returns five basic descriptive statistics (mean, standard deviation, median, minimum and maximum) of all values in an array. Next, create an array ('arr_10') by manually entering the following values in NumPy's `array()` function: [1, 1, 1, 2, 2, 3, 3, 4, 5, 5]. Then perform the following operations: (1) call the function on the array, and (2) (in a separate cell) use f-strings to print the following sentence: "The mean and median of this array are respectively: MEAN and MEDIAN.", where you use calling the function to fill in the MEAN and MEDIAN in this sentence.

```
In [39]: def descr_stats(arr):
             return np.mean(arr), np.std(arr), np.median(arr), np.min(arr), n
         p.max(arr)

         arr_10 = np.array([1, 1, 1, 2, 2, 3, 3, 4, 5, 5])
         descr_stats(arr_10)

Out[39]: (2.7, 1.4866068747318506, 2.5, 1, 5)
```

```
In [40]: print(f"The mean and median of this array are respectively: {descr_s
         tats(arr_10)[0]} and {descr_stats(arr_10)[2]}.")

         The mean and median of this array are respectively: 2.7 and 2.5.
```