# NumPy Practice

This notebook offers a set of exercises for different tasks with NumPy.

It should be noted there may be more than one different way to answer a question or complete an exercise.

Exercises are based off (and directly taken from) the quick introduction to NumPy notebook.

Different tasks will be detailed by comments or text.

For further reference and resources, it's advised to check out the NumPy documentation.

And if you get stuck, try searching for a question in the following format: "how to do XYZ with numpy", where XYZ is the function you want to leverage from NumPy.

In [ ]:
```
import numpy as np
import pandas as pd
np.random.seed(1)
```

In [ ]:
```
# Create a 1-dimensional NumPy array using np.array()
one_d = np.array([1,2,3,4,5])

# Create a 2-dimensional NumPy array using np.array()
two_d = np.array([[1,2,3,4,5],[6,7,8,9,10]])

# Create a 3-dimensional Numpy array using np.array()
three_d = np.random.randint(0,30,(5,3,2))
```

Now we've you've created 3 different arrays, let's find details about them.

Find the shape, number of dimensions, data type, size and type of each array.

In [ ]:
```
# Attributes of 1-dimensional array (shape,
# number of dimensions, data type, size and type)

print("For one-dimensional array:")
print("Shape = {}".format(one_d.shape))
print("Dimensions = {}".format(one_d.ndim))
print("Data type = {}".format(one_d.dtype))
print("Size = {}".format(one_d.size))
print("Type = {}".format(type(one_d)))
```

```
For one-dimensional array:
Shape = (5,)
Dimensions = 1
Data type = int32
Size = 5
Type = <class 'numpy.ndarray'>
```

In [ ]:
```
# Attributes of 2-dimensional array
print("For two-dimensional array:")
```

```
print("Shape = {}".format(two_d.shape))
print("Dimensions = {}".format(two_d.ndim))
print("Data type = {}".format(two_d.dtype))
print("Size = {}".format(two_d.size))
print("Type = {}".format(type(two_d)))
```

```
For two-dimensional array:
Shape = (2, 5)
Dimensions = 2
Data type = int32
Size = 10
Type = <class 'numpy.ndarray'>
```

In [ ]:
```
# Attributes of 3-dimensional array
# Attributes of 2-dimensional array
print("For three-dimensional array:")
print("Shape = {}".format(three_d.shape))
print("Dimensions = {}".format(three_d.ndim))
print("Data type = {}".format(three_d.dtype))
print("Size = {}".format(three_d.size))
print("Type = {}".format(type(three_d)))
```

```
For three-dimensional array:
Shape = (5, 3, 2)
Dimensions = 3
Data type = int32
Size = 30
Type = <class 'numpy.ndarray'>
```

In [ ]:
```
# Import pandas and create a DataFrame out of one
# of the arrays you've created
df = pd.DataFrame(two_d)
df
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 4 | 5 |
| **1** | 6 | 7 | 8 | 9 | 10 |

In [ ]:
```
# Create an array of shape (10, 2) with only ones
np.ones((10,2))
```

Out[ ]:
```
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]])
```

In [ ]:
```
# Create an array of shape (7, 2, 3) of only zeros
np.zeros((7,2,3))
```

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

       [[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.]],

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

In [ ]:
```
# Create an array within a range of 0 and 100 with step 3
np.arange(0,101,3)
```

Out[ ]:
```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48,
       51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99])
```

In [ ]:
```
# Create a random array with numbers between 0 and 10 of size (7, 2)
np.random.randint(0,10,(7,2))
```

Out[ ]:
```
array([[7, 7],
       [9, 1],
       [7, 0],
       [6, 9],
       [9, 7],
       [6, 9],
       [1, 0]])
```

In [ ]:
```
# Create a random array of floats between 0 & 1 of shape (3, 5)
np.random.random((3,5))
```

Out[ ]:
```
array([[0.89460666, 0.08504421, 0.03905478, 0.16983042, 0.8781425 ],
       [0.09834683, 0.42110763, 0.95788953, 0.53316528, 0.69187711],
       [0.31551563, 0.68650093, 0.83462567, 0.01828828, 0.75014431]])
```

In [ ]:
```
# Set the random seed to 42
np.random.seed(42)

# Create a random array of numbers between 0 & 10 of size (4, 6)
np.random.randint(0,10,(4,6))
```

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

Run the cell above again, what happens?

Are the numbers in the array different or the same? Why do think this is?

## The numbers are the same when we run the cell again because we have set a definite seed for the random generator.

In [ ]:
```
# Create an array of random numbers between 1 & 10 of size (3, 7)
# and save it to a variable
var = np.random.randint(0,10,(3,7))
s = set()
# Find the unique numbers in the array you just created
for i in var:
    for j in i:
        s.add(j)
print("All unique elements are: {}".format(s))
```

```
All unique elements are: {0, 1, 2, 3, 4, 6, 8, 9}
```

In [ ]:
```
# Find the 0'th index of the latest array you created
var[0]
```

Out[ ]:
```
array([8, 0, 9, 2, 6, 3, 8])
```

In [ ]:
```
# Get the first 2 rows of latest array you created
var[0:2,:]
```

Out[ ]:
```
array([[8, 0, 9, 2, 6, 3, 8],
       [2, 4, 2, 6, 4, 8, 6]])
```

In [ ]:
```
# Get the first 2 values of the first 2 rows of the latest array
var[0:2,0:2]
```

Out[ ]:
```
array([[8, 0],
       [2, 4]])
```

In [ ]:
```
# Create a random array of numbers between 0 & 10 and an array of ones
# both of size (3, 5), save them both to variables
array1 = np.random.randint(0,10,(3,5))
array2 = np.random.randint(0,10,(3,5))
array1,array2
```

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

In [ ]:
```
# Add the two arrays together
array_sum = array2 + array1
array_sum
```

Out[ ]:
```
array([[ 7,  6,  4, 15,  8],
       [11,  3, 10,  7, 15],
       [10,  5,  6,  9, 16]])
```

In [ ]:
```
# Create another array of ones of shape (5, 3)
one = np.ones((5,3))
```

In [ ]:
```
# Try add the array of ones and the other most recent array together
array_sum + one
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[41], line 2
      1 # Try add the array of ones and the other most recent array together
----> 2 array_sum + one

ValueError: operands could not be broadcast together with shapes (3,5) (5,3)
```

When you try the last cell, it produces an error. Why do think this is?

How would you fix it?

The error occurs because we are trying to add two arrays of different dimensions. To fix this we must reshape all array to the same shape

In [ ]:
```
# Create another array of ones of shape (3, 5)
one = np.ones((3,5))
```

In [ ]:
```
# Subtract the new array of ones from the other most recent array
array_sum - one
```

Out[ ]:
```
array([[ 6.,  5.,  3., 14.,  7.],
       [10.,  2.,  9.,  6., 14.],
       [ 9.,  4.,  5.,  8., 15.]])
```

In [ ]:
```
# Multiply the ones array with the latest array
array_sum * one
```

Out[ ]:
```
array([[ 7.,  6.,  4., 15.,  8.],
       [11.,  3., 10.,  7., 15.],
       [10.,  5.,  6.,  9., 16.]])
```

In [ ]:
```
# Take the latest array to the power of 2 using '**'
array_sum ** 2
```

Out[ ]:
```
array([[ 49,  36,  16, 225,  64],
       [121,   9, 100,  49, 225],
       [100,  25,  36,  81, 256]])
```

In [ ]:
```
# Do the same thing with np.square()
np.square(array_sum)
```

Out[ ]:
```
array([[ 49,  36,  16, 225,  64],
       [121,   9, 100,  49, 225],
       [100,  25,  36,  81, 256]])
```

In [ ]:
```
# Find the mean of the latest array using np.mean()
np.mean(array_sum)
```

Out[ ]:
```
8.8
```

In [ ]:
```
# Find the maximum of the latest array using np.max()
np.max(array_sum)
```

Out[ ]: 16

In [ ]:
```
# Find the minimum of the latest array using np.min()
np.min(array_sum)
```

Out[ ]: 3

In [ ]:
```
# Find the standard deviation of the latest array
np.std(array_sum)
```

Out[ ]: 3.919183588453085

In [ ]:
```
# Find the variance of the latest array
np.var(array_sum)
```

Out[ ]: 15.359999999999998

In [ ]:
```
# Reshape the latest array to (3, 5, 1)
array_sum.reshape((3,5,1))
```

Out[ ]:
```
array([[[ 7],
        [ 6],
        [ 4],
        [15],
        [ 8]],

       [[11],
        [ 3],
        [10],
        [ 7],
        [15]],

       [[10],
        [ 5],
        [ 6],
        [ 9],
        [16]]])
```

In [ ]:
```
# Transpose the latest array
array_sum.T
```

Out[ ]:
```
array([[ 7, 11, 10],
       [ 6,  3,  5],
       [ 4, 10,  6],
       [15,  7,  9],
       [ 8, 15, 16]])
```

What does the transpose do?

In [ ]:
```
# Create two arrays of random integers between 0 to 10
# one of size (3, 3) the other of size (3, 2)
array1 = np.random.randint(0,10,(3,3))
array2 = np.random.randint(0,10,(3,2))
```

In [ ]:
```
# Perform a dot product on the two newest arrays you created
np.dot(array1,array2)
```

Out[ ]:
```
array([[135,  79],
       [ 58,  12],
       [ 70,  56]])
```

In [ ]:
```
# Create two arrays of random integers between 0 to 10
# both of size (4, 3)
array1 = np.random.randint(0,10,(4,3))
array2 = np.random.randint(0,10,(4,3))
```

In [ ]:
```
# Perform a dot product on the two newest arrays you created
np.dot(array1,array2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[57], line 2
      1 # Perform a dot product on the two newest arrays you created
----> 2 np.dot(array1,array2)

ValueError: shapes (4,3) and (4,3) not aligned: 3 (dim 1) != 4 (dim 0)
```

It doesn't work. How would you fix it?

I will fix this by making sure the number of columns of the first array is the same as the number of rows of the second array

In [ ]:
```
# Take the latest two arrays, perform a transpose on one of them and then perfor
# a dot product on them both
np.dot(array1,array2.T)
```

Out[ ]:
```
array([[ 40,  36,  10,  12],
       [153, 111,  57,  60],
       [140, 100,  52,  56],
       [ 48,  30,  37,  18]])
```

Notice how performing a transpose allows the dot product to happen.

Why is this?

Checking out the documentation on `np.dot()` may help, as well as reading Math is Fun's guide on the dot product.

Let's now compare arrays.

In [ ]:
```
# Create two arrays of random integers between 0 & 10 of the same shape
# and save them to variables
array1 = np.random.randint(0,10,(4,3))
array2 = np.random.randint(0,10,(4,3))
```

In [ ]:
```
# Compare the two arrays with '>'
array1 > array2
```

Out[ ]:
```
array([[False, False,  True],
       [False,  True,  True],
       [ True,  True,  True],
       [False, False, False]])
```

What happens when you compare the arrays with `>`?

We get an array of boolean values

In [ ]:
```
# Compare the two arrays with '>='
array1 >= array2
```

Out[ ]:
```
array([[False,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [False, False, False]])
```

In [ ]:
```
# Find which elements of the first array are greater than 7
array2[array2 > 7]
```

Out[ ]:
```
array([9, 8])
```

In [ ]:
```
# Which parts of each array are equal? (try using '==')
array1[array1 == array2]
```

Out[ ]:
```
array([4, 6])
```

In [ ]:
```
# Sort one of the arrays you just created in ascending order
array1.sort()
```

In [ ]:
```
# Sort the indexes of one of the arrays you just created
array1.argsort()
```

Out[ ]:
```
array([[1, 0, 2],
       [0, 2, 1],
       [0, 1, 2],
       [1, 0, 2]], dtype=int64)
```

In [ ]:
```
# Find the index with the maximum value in one of the arrays you've created
array1.argmax()
```

Out[ ]:
```
2
```

In [ ]:
```
# Find the index with the minimum value in one of the arrays you've created
array1.argmin()
```

Out[ ]:
```
6
```

In [ ]:
```
# Find the indexes with the maximum values down the 1st axis (axis=1)
# of one of the arrays you created
array1.argmin(axis=1)
```

Out[ ]:
```
array([1, 0, 0, 1], dtype=int64)
```

In [ ]:
```
# Find the indexes with the minimum values across the 0th axis (axis=0)
# of one of the arrays you created
array1.argmin(axis=0)
```

Out[ ]:
```
array([2, 2, 1], dtype=int64)
```

In [ ]:
```
# Create an array of normally distributed random number
np.random.normal(size=(2,5))
```

Out[ ]:
```
array([[ 1.40395874, -0.0185508 , -1.67350462, -1.07253183, -0.99258618],
       [ 0.10234768, -0.43260928, -0.6591823 ,  0.0039373 ,  0.4777541 ]])
```

In [ ]:
```
# Create an array with 10 evenly spaced numbers between 1 and 100
np.linspace(0,100,10)
```

Out[ ]:
```
array([  0.        ,  11.11111111,  22.22222222,  33.33333333,
        44.44444444,  55.55555556,  66.66666667,  77.77777778,
        88.88888889, 100.        ])
```

# Extensions

For more exercises, check out the NumPy quickstart tutorial. A good practice would be to read through it and for the parts you find interesting, add them into the end of this notebook.

Pay particular attention to the section on broadcasting. And most importantly, get hands-on with the code as much as possible. If in dobut, run the code, see what it does.

The next place you could go is the Stack Overflow page for the top questions and answers for NumPy. Often, you'll find some of the most common and useful NumPy functions here. Don't forget to play around with the filters! You'll likely find something helpful here.

Finally, as always, remember, the best way to learn something new is to try it. And try it relentlessly. If you get interested in some kind of NumPy function, asking yourself, "I wonder if NumPy could do that?", go and find out.