NumPy practice

In [ ]:
```
# Import library
import numpy as np
```

In [ ]:
```
# Reading the example code
a = np.arange(8)
a2 = a[np.newaxis, :]
a2.shape
```

Out[ ]:
```
(1, 8)
```

In [ ]:
```
# Make 1D array from python lists
python_list = [1, 2, 3, 4, 5, 6]
arr = np.array(python_list)
arr
```

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

In [ ]:
```
# Make 2D array from python nested lists
python_list2 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
arr2 = np.array(python_list2)
arr2
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Access first element from the 2D array above
arr2[0]
```

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

In [ ]:
```
# Create an array of zeros
np.zeros(8)
```

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

In [ ]:
```
# Create an array of ones
np.ones(5)
```

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

In [ ]:
```
# Create an empty array with 4 elements, this empty function
# initializes random values depending on the state of the memory
np.empty(4)
```

Out[ ]:
```
array([2.12199579e-314, 6.36598737e-314, 1.06099790e-313, 1.48539705e-313])
```

In [ ]:
```
# Create a range of an array
np.arange(7)
```

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

In [ ]:
```
# Create a range of an array with steps
np.arange(1, 10, 3)
```

Out[ ]:
```
array([1, 4, 7])
```

In [ ]:
```
# Create an array with values equally seperated in a specified interval
np.linspace(1, 10, num=3)
```

Out[ ]:
```
array([ 1. ,  5.5, 10. ])
```

In [ ]:
```
# Create an array with specific data type
x = np.empty(4, dtype=np.int32)
x
```

Out[ ]:
```
array([-1193301859,   435819932,   535202197,  2102086784])
```

In [ ]:
```
# Sort elements of an array
arr3 = np.array([2, 1, 58, 31, 7, 4, 6, 80])
np.sort(arr3)
```

Out[ ]:
```
array([ 1,  2,  4,  6,  7, 31, 58, 80])
```

In [ ]:
```
# Concatenate two arrays
arr4 = np.array([1, 2, 3, 4])
arr5 = np.array([15, 16, 17, 18])
np.concatenate((arr4, arr5))
```

Out[ ]:
```
array([ 1,  2,  3,  4, 15, 16, 17, 18])
```

In [ ]:
```
# Concatenate 2d arrays along first axis
arr6 = np.array([[1, 12],
                 [3, 34]])

arr7 = np.array([[5, 56]])

np.concatenate((arr6, arr7), axis=0)
```

Out[ ]:
```
array([[ 1, 12],
       [ 3, 34],
       [ 5, 56]])
```

In [ ]:
```
#Create an example array
arr8 = np.array([[[0, 1, 2, 3],
                  [4, 5, 6, 7]],
                  [[0, 1, 2, 3],
                   [4, 5, 6, 7]],
                  [[0 ,1 ,2, 3],
                   [4, 5, 6, 7]]])
```

In [ ]:
```
# Find dimension of the array
arr8.ndim
```

Out[ ]:
```
3
```

In [ ]:
```
# Find the size of the array
arr8.size
```

Out[ ]:
```
24
```

In [ ]:
```
# Find the shape of the array
arr8.shape
```

Out[ ]:
```
(3, 2, 4)
```

In [ ]:
```
# Reshape an array
arr
```

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

In [ ]:
```
arr.reshape(2, 3)
```

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

In [ ]:
```
# Reshape with Fortran format
np.reshape(arr, newshape=(2,3), order='F')
```

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

In [ ]:
```
# Convert 1d into 2d array
arr.shape
```

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

In [ ]:
```
arr_2d = arr[np.newaxis, :]
arr_2d.shape
```

Out[ ]:
```
(1, 6)
```

In [ ]:
```
# Another way to convert 1d into 2d
another_arr = np.expand_dims(arr, axis=1)
another_arr.shape
```

Out[ ]:
```
(6, 1)
```

In [ ]:
```
# Indexing and slicing of an array
data = np.array([1, 2, 3])

# Index second element of the array
data[1]
```

Out[ ]:
```
2
```

In [ ]:
```
# Slice first 2 elements of the array
data[0:2]
```

Out[ ]:
```
array([1, 2])
```

In [ ]:
```
new_arr = np.array([[1 , 2, 3, 4],
                    [5, 6, 7, 8],
                    [9, 10, 11, 12]])
new_arr
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Find elements less than 7
new_arr[new_arr < 7]
```

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

In [ ]:
```
# Find elements greater or equal to 7
seven_up = (new_arr >= 7) # mask
new_arr[seven_up]
```

Out[ ]:
```
array([ 7,  8,  9, 10, 11, 12])
```

In [ ]:
```
# Select elements that are divisible by 3
divisible_by_3 = a[a%3==0]
divisible_by_3
```

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

In [ ]:
```
# Select elements greater than 3 and less than 10
c = a[(a > 3) & (a < 10)]
c
```

Out[ ]:
```
array([4, 5, 6, 7])
```

In [ ]:
```
# Create new array
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
a
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Find the indices of elements greater than 5
b = np.nonzero(a > 5)
b
```

Out[ ]:
```
(array([1, 1, 1, 2, 2, 2, 2], dtype=int64),
 array([1, 2, 3, 0, 1, 2, 3], dtype=int64))
```

In [ ]:
```
# Find the coordinates of elements greater than 5
corrdinates = list(zip(b[0], b[1]))
for corrd in corrdinates:
    print(corrd)
```

```
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
```

In [ ]:
```
# Make vertical stacking two arrays
a1 = np.array([[1, 1],
               [2, 2]])

a2 = np.array([[3, 3],
               [4, 4]])
```

In [ ]:
```
np.vstack((a1, a2))
```

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

In [ ]:
```
# Horizontal stacking
np.hstack((a1, a2))
```

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

In [ ]:
```
# Create an array and reshape it
x = np.arange(1, 25).reshape(2, 12)
x
```

Out[ ]:
```
array([[ 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 [ ]:
```
# Split an array into three equally shaped arrays
np.hsplit(x, 3)
```

Out[ ]:
```
[array([[ 1,  2,  3,  4],
        [13, 14, 15, 16]]),
 array([[ 5,  6,  7,  8],
        [17, 18, 19, 20]]),
 array([[ 9, 10, 11, 12],
        [21, 22, 23, 24]])]
```

In [ ]:
```
# Split an array after third and fourth column
np.hsplit(x, (3, 4))
```

Out[ ]:
```
[array([[ 1,  2,  3],
        [13, 14, 15]]),
 array([[ 4],
        [16]]),
 array([[ 5,  6,  7,  8,  9, 10, 11, 12],
        [17, 18, 19, 20, 21, 22, 23, 24]])]
```

In [ ]:
```
# Create an array
a = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8],
              [9, 10, 11, 12]])
a
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Create an array by slicing existing array
b1 = a[0, :]
b1
```

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

In [ ]:
```
# Replace first element of b1
b1[0] = 99
b1
```

Out[ ]:
```
array([99,  2,  3,  4])
```

In [ ]:
```
# View array "a" again
# notice the first element of the corresponding array
# is also modified
a
```

Out[ ]:
```
array([[99,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Make copy of an array
b2 = a.copy()
b2
```

Out[ ]:
```
array([[99,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Add arrays together
data = np.array([5, 2])
ones = np.ones(2, dtype=int)
data + ones
```

Out[ ]:
```
array([6, 3])
```

In [ ]:
```
# Find sum of the array
a = np.array([1, 2, 3, 4])
a.sum()
```

Out[ ]:
```
10
```

In [ ]:
```
# Find sum along first axis (rows)
b = np.array([[1, 1], [2, 2]])
b.sum(axis=0)
```

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

In [ ]:
```
# Find sum along second axis (columns)
b.sum(axis=1)
```

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

In [ ]:
```
# Broadcasting
data = np.array([1.0, 2.0])
data * 1.6
```

Out[ ]:
```
array([1.6, 3.2])
```

In [ ]:
```
data.max()
```

Out[ ]:
```
2.0
```

In [ ]:
```
data.min()
```

Out[ ]:
```
1.0
```

In [ ]:
```
data.sum()
```

Out[ ]:
```
3.0
```

In [ ]:
```
# Create new array
a = np.array([[0.45053314, 0.17296777, 0.34376245, 0.5510652],
              [0.54627315, 0.05093587, 0.40067661, 0.55645993],
              [0.12697628, 0.82485143, 0.26590556, 0.56917101]])
a
```

Out[ ]:
```
array([[0.45053314, 0.17296777, 0.34376245, 0.5510652 ],
       [0.54627315, 0.05093587, 0.40067661, 0.55645993],
       [0.12697628, 0.82485143, 0.26590556, 0.56917101]])
```

In [ ]:
```
a.sum()
```

Out[ ]:
```
4.8595784
```

In [ ]:
```
a.min()
```

Out[ ]:
```
0.05093587
```

In [ ]:
```
# Find mininmum along first axis (rows)
a.min(axis=0)
```

Out[ ]:
```
array([0.12697628, 0.05093587, 0.26590556, 0.5510652 ])
```

In [ ]:
```
# Generate a 2x4 array of random integer
np.random.randint(5, size=(2, 4))
```

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

In [ ]:
```
# Create an array
a = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])
a
```

Out[ ]:
```
array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])
```

In [ ]:
```
# Find unique values of "a"
np.unique(a)
```

Out[ ]:
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

In [ ]:
```
# Find unique values and their indices in array "a"
unique_values, indices_list = np.unique(a, return_index=True)
unique_values, indices_list
```

Out[ ]:
```
(array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20]),
 array([ 0,  2,  3,  4,  5,  6,  7, 12, 13, 14], dtype=int64))
```

In [ ]:
```
# Find the frequencey count of unique values
unique_values, occurrence_count = np.unique(a, return_counts=True)
unique_values, occurrence_count
```

Out[ ]:
```
(array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20]),
 array([3, 2, 2, 2, 1, 1, 1, 1, 1, 1], dtype=int64))
```

In [ ]:
```
# Create new array
data = np.array([1, 2, 3, 4, 5, 6])
data
```

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

In [ ]:
```
# Reshape array to 2x3
data.reshape(2, 3)
```

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

In [ ]:
```
# Reshape array to 3x2
data.reshape(3, 2)
```

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

In [ ]:
```
# Create array
arr = np.arange(6).reshape((2, 3))
arr
```

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

In [ ]:
```
# Transpose array
arr.transpose()
```

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

In [ ]:
```
# Reverse 1d array
reversed_arr = np.flip(data)
print('Reversed Array: ', reversed_arr)
```

```
Reversed Array:  [6 5 4 3 2 1]
```

In [ ]:
```
# Create 2d array
arr_2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
arr_2d
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Reverse 2d array along the rows axis
reversed_arr_rows = np.flip(arr_2d, axis=0)
print(reversed_arr_rows)
```

```
[[ 9 10 11 12]
 [ 5  6  7  8]
 [ 1  2  3  4]]
```

In [ ]:
```
# Reverse 2d array along the columns axis
reversed_arr_columns = np.flip(arr_2d, axis=1)
print(reversed_arr_columns)
```

```
[[ 4  3  2  1]
 [ 8  7  6  5]
 [12 11 10  9]]
```

In [ ]:
```
# Reverse elements on a specific row
arr_2d[1] = np.flip(arr_2d[1]) # reverse second row
arr_2d
```

```
[[ 1  2  3  4]
 [ 8  7  6  5]
 [ 9 10 11 12]]
```

In [ ]:
```
# Reverse elements on a specific column
arr_2d[:,2] = np.flip(arr_2d[:,2]) # reverse third column
arr_2d
```

Out[ ]:
```
array([[ 1,  2, 11,  4],
       [ 8,  7,  6,  5],
       [ 9, 10,  3, 12]])
```

In [ ]:
```
# Create new 2d array
x = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
x
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [ ]:
```
# Flatten array to 1d
x.flatten()
```

Out[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [ ]:
```
# Ravel array to 1d (for permanent change)
x.ravel()
```

Out[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [ ]:
```
# Access docstring using help() function
help(min)
```

```
Help on built-in function min in module builtins:

min(...)
    min(iterable, *[, default=obj, key=func]) -> value
    min(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its smallest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the smallest argument.
```

In [ ]:
```
# Another way to access docstring using "?"
min?
```

In [ ]:
```
# Create an array
a = np.array([1, 2, 3, 4, 5, 6])
a
```

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

In [ ]:
```
# Save single array using np.save
np.save('my_arr', a)
```

In [ ]:
```
# Load the array (.npy extension)
b = np.load('my_arr.npy')
b
```

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

In [ ]:
```
# Create two 1d array
arr1 = np.array([5, 6, 7, 8])
arr2 = np.array([9, 10, 11, 12])
```

In [ ]:
```
# Save more to one arrays together
np.savez('arrays_together', arr1, arr2)
```

In [ ]:
```
# Load the above saved file (.npz extension)
arr_combo = np.load('arrays_together.npz')

# Query the list of arr_combo
arr_combo.files
```

Out[ ]:
```
['arr_0', 'arr_1']
```

In [ ]:
```
# Access first array from arr_combo
arr_combo['arr_0']
```

Out[ ]:
```
array([5, 6, 7, 8])
```

In [ ]:
```
# Access second array
arr_combo['arr_1']
```

Out[ ]:
```
array([ 9, 10, 11, 12])
```

In [ ]:
```
# Create new array
csv_arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
csv_arr
```

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

In [ ]:
```
# Save array as csv
np.savetxt('new_file.csv', csv_arr)
```

In [ ]:
```
# Load csv array
np.loadtxt('new_file.csv')
```

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

In [ ]:
```
# Create new array
a = np.array([[-2.58289208,  0.43014843, -1.24082018, 1.59572603],
              [ 0.99027828, 1.17150989,  0.94125714, -0.14692469],
              [ 0.76989341,  0.81299683, -0.95068423, 0.11769564],
              [ 0.20484034,  0.34784527,  1.96979195, 0.51992837]])
a
```

Out[ ]:
```
array([[-2.58289208,  0.43014843, -1.24082018,  1.59572603],
       [ 0.99027828,  1.17150989,  0.94125714, -0.14692469],
       [ 0.76989341,  0.81299683, -0.95068423,  0.11769564],
       [ 0.20484034,  0.34784527,  1.96979195,  0.51992837]])
```

In [ ]:
```
# Create dataframe from array
import pandas as pd
df = pd.DataFrame(a)
df
```

Out[ ]:

	0	1	2	3
0	-2.582892	0.430148	-1.240820	1.595726
1	0.990278	1.171510	0.941257	-0.146925
2	0.769893	0.812997	-0.950684	0.117696
3	0.204840	0.347845	1.969792	0.519928

In [ ]:
```
# Save dataframe
df.to_csv('pd.csv')
```

In [ ]:
```
# Load csv file
data = pd.read_csv('pd.csv')
data
```

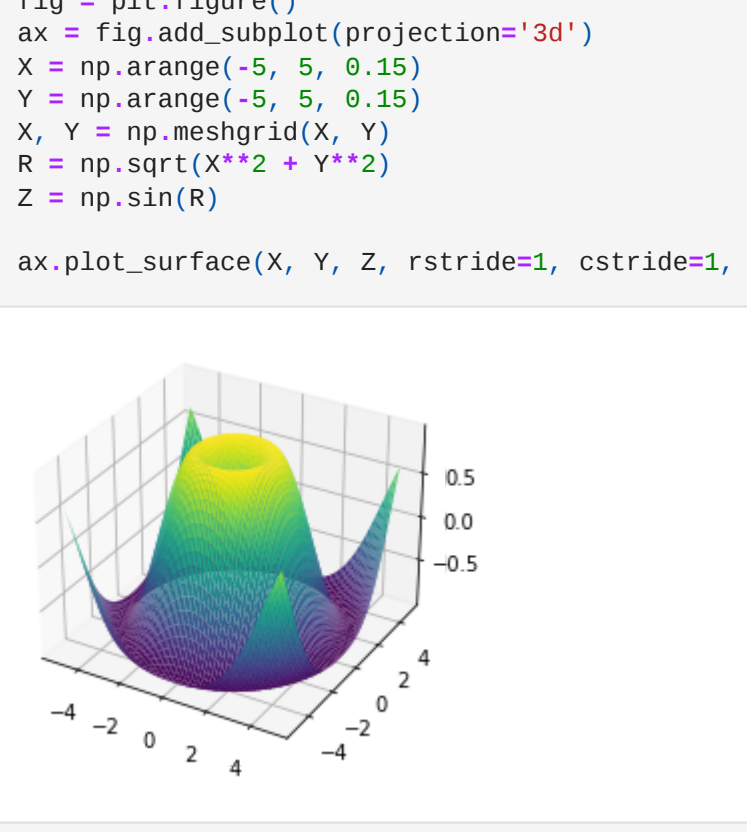
Out[ ]:

	Unnamed: 0	0	1	2	3
0	0	-2.582892	0.430148	-1.240820	1.595726
1	1	0.990278	1.171510	0.941257	-0.146925
2	2	0.769893	0.812997	-0.950684	0.117696
3	3	0.204840	0.347845	1.969792	0.519928

In [ ]:
```
# Create new array
a = np.array([2, 1, 5, 7, 4, 6, 8, 14, 10, 9, 18, 20, 22])
a
```

Out[ ]:
```
array([ 2,  1,  5,  7,  4,  6,  8, 14, 10,  9, 18, 20, 22])
```

In [ ]:
```
# Create plot from array
import matplotlib.pyplot as plt
plt.plot(a);
```



In [ ]:
```
# Create two 1d arrays and plot them in a graph
x = np.linspace(0, 5, 20)
y = np.linspace(0, 10, 20)
plt.plot(x, y, 'purple') # line
plt.plot(x, y, 'o');     # dots
```


In [ ]:
```
# Create 3d plot with three arrays
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
X = np.arange(-5, 5, 0.15)
Y = np.arange(-5, 5, 0.15)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='viridis');
```


In [ ]: