Personal Notes:

- Note: My personal notes are included in this notebook. The in-class portion is after that below.

In [2]:
```
import numpy as np
```

Creating ndarrays:

In [3]:
```
# From a list:
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

Out[3]:
```
array([6. , 7.5, 8. , 0. , 1. ])
```

In [4]:
```
# From a list of lists:
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
```

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

In [5]:
```
arr2.ndim
```

Out[5]:
```
2
```

In [7]:
```
arr2.shape
```

Out[7]:
```
(2, 4)
```

In [10]:
```
# Initializing with 0's using a convenience function:
np.zeros(10)
```

Out[10]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [11]:
```
np.zeros((3, 6))
```

Out[11]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

In [16]:
```
data1 = ([1, 2, 3], [4, 5, 6])
```

In [17]:
```
data = np.array(data1)
```

In [18]:
```
data
```

Out[18]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [19]:
```
data + data
```

Out[19]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

Can see that Numpy Arrays are lists, but do not behave like lists.

In [21]:
```
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list1 + list2
```

Out[21]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [22]:
```
data * data
```

Out[22]:
```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

In [23]:
```
data ** 2
```

Out[23]:
```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

In [24]:
```
data2 = [6, 7.5, 8, 0, 1]
arr2 = np.array(data2)
arr2
```

Out[24]:
```
array([6. , 7.5, 8. , 0. , 1. ])
```

Can see that it made decision to make all of the values floats, because there was one float already.

Indexing and Slicing:

If I make a slice of an array, it is not a independent of the previous array. It changes the original.

In [29]:
```
arr = np.arange(10)
arr
```

Out[29]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [31]:
```
arr[5:8] = 12
arr
```

Out[31]:
```
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

In [32]:
```
arr_slice = arr[5:8]
arr_slice
```

Out[32]:
```
array([12, 12, 12])
```

In [33]:
```
arr_slice[1] = 12345
arr
```

Out[33]:
```
array([    0,     1,     2,     3,     4,    12, 12345,    12,     8,
           9])
```

In [34]:
```
arr_slice[:] = 64
arr
```

Out[34]:
```
array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

NumPy defaults to views rather than copies because copies are expensive and NumPy is designed with large data use cases in mind.

If you want a copy of a slice of an ndarray instead of a view, use .copy().

In [35]:
```
arr_slice_copy = arr[5:8].copy()
arr_slice_copy
```

Out[35]:
```
array([64, 64, 64])
```

In [38]:
```
arr_slice_copy[:] = 99
arr_slice_copy
```

Out[38]:
```
array([99, 99, 99])
```

In [39]:
```
arr
```

Out[39]:
```
array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

See, now it doesn't change the original!

Boolean Indexing:

You can pass a boolean representation of an array to the array indexer (i.e. the [] suffix) and it will return only those cells that are True.

In [40]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
names
```

Out[40]:
```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')
```

In [41]:
```
data = np.random.randn(7, 4)
data
```

Out[41]:
```
array([[-1.17502117e+00,  1.05041321e-01,  9.05461541e-02,
         4.11095722e-02],
       [-1.42536488e-02, -4.76217125e-01,  2.12561808e+00,
         1.06726985e+00],
       [-4.39853303e-01, -6.71648934e-01,  4.40549020e-01,
        -1.81138787e-03],
       [ 1.14804620e+00,  8.58055154e-01, -4.66216525e-01,
         1.22870771e-03],
       [ 7.65595085e-01,  1.05322825e+00, -5.85484627e-02,
        -9.66680476e-01],
       [ 3.25662243e-01, -7.24176710e-01,  3.93943504e-01,
        -7.57894256e-01],
       [ 4.08135246e-01, -1.09713769e+00, -8.63586511e-02,
         1.20344803e+00]])
```

In [42]:
```
names == "Bob"
```

Out[42]:
```
array([ True, False, False,  True, False, False, False])
```

In [43]:
```
data[names == 'Bob']
```

Out[43]:
```
array([[-1.17502117,  0.10504132,  0.09054615,  0.04110957],
       [ 1.1480462 ,  0.85805515, -0.46621652,  0.00122871]])
```

In [44]:
```
bix = names != 'Bob'
bix
```

Out[44]:
```
array([False,  True,  True, False,  True,  True,  True])
```

In [45]:
```
data[~bix] # Back to Bob
```

Out[45]:
```
array([[-1.17502117,  0.10504132,  0.09054615,  0.04110957],
       [ 1.1480462 ,  0.85805515, -0.46621652,  0.00122871]])
```

In [46]:
```
arr = np.arange(32).reshape((8, 4))
arr
```

Out[46]:
```
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, 27],
       [28, 29, 30, 31]])
```

In [47]:
```
arr[[1, 5, 7, 2], [0, 3, 1, 2]]  # Grab rows, then select columns from each row
```

Out[47]:
```
array([ 4, 23, 29, 10])
```

In [48]:
```
arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]] # Grab rows, then reorder columns
```

Out[48]:
```
array([[ 4,  7,  5,  6],
       [20, 23, 21, 22],
       [28, 31, 29, 30],
       [ 8, 11,  9, 10]])
```

In-class Activities:

5.1.

In [53]:
```
import time
t0 = time.time()
vals = []
for i in range(1,100001):
    if i % 2 == 1:
        i *= -1
    vals.append(i)
t1 = time.time()
delta_time = t1 - t0
# print time
print("runtime:", delta_time)
```

```
runtime: 0.04683709144592285
```

5.2.

In [56]:
```
t0 = time.time()
vals = [i * -1 if i % 2 == 1 else i for i in range(1, 100001)]
print("runtime:", time.time() - t0) # faster!
```

```
runtime: 0.025129079818725586
```

5.3.

In [58]:
```
randos = np.random. randint(1, 7, 10)
print(",".join(randos.astype(str)))
```

```
3,4,5,4,6,4,3,3,5,1
```

In [59]:
```
type(randos)
```

Out[59]:
```
numpy.ndarray
```

5.4.

In [66]:
```
randos1 = np.random.randint(10)
randos1
type(randos1)
```

Out[66]:
```
int
```

In [74]:
```
randos2 = np.random.randint(1, 21, 5)
randos2
type(randos2) # now puts it in an array.
```

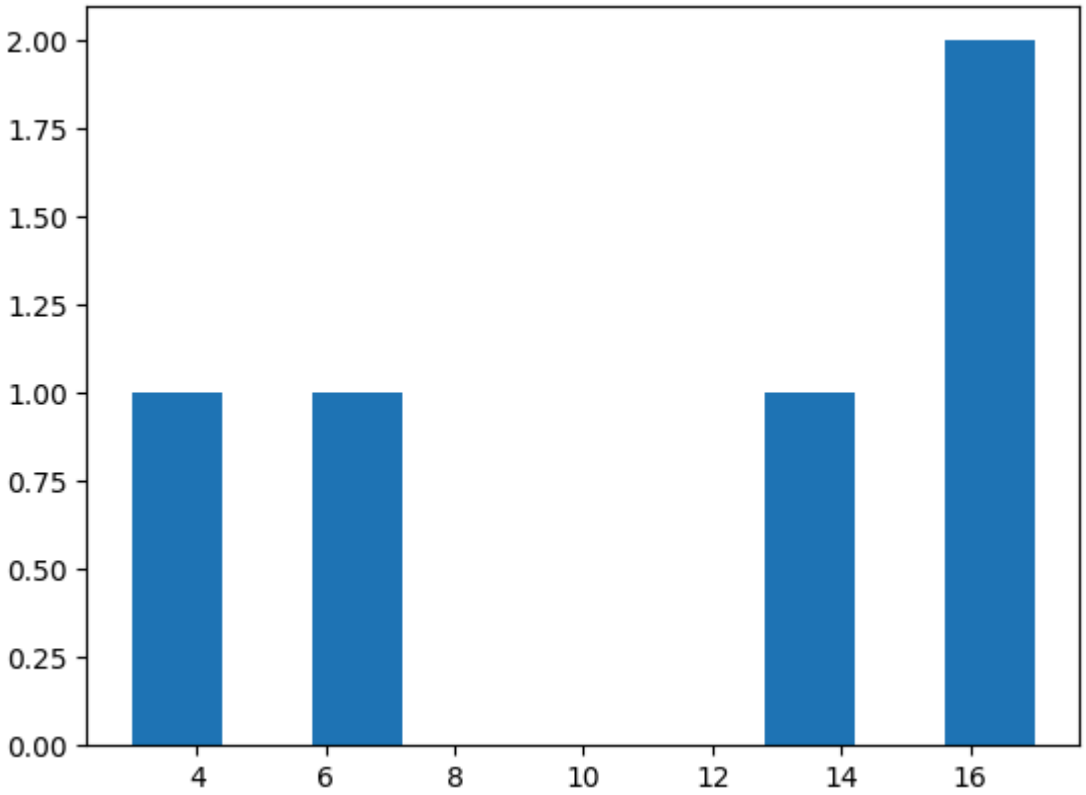
Out[74]:
```
numpy.ndarray
```

5.5.

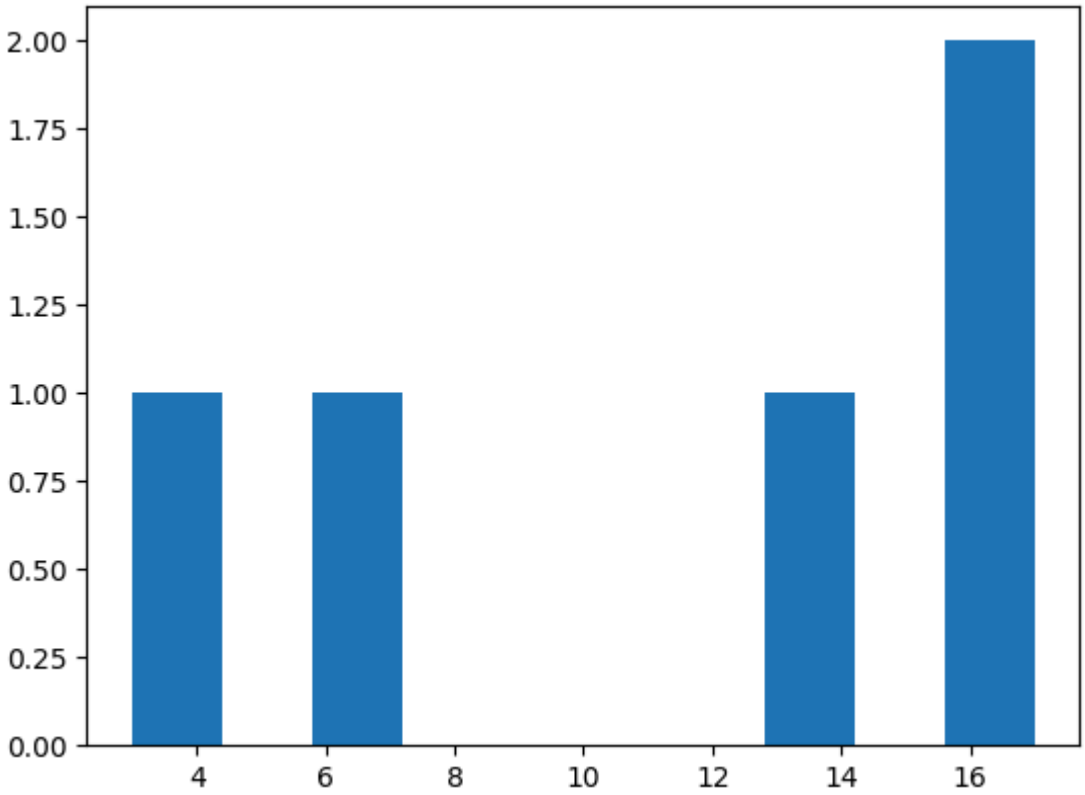
In [75]:
```
import matplotlib
from matplotlib.pyplot import hist

hist(randos2)
```

Out[75]:
```
(array([1., 0., 1., 0., 0., 0., 0., 1., 0., 2.]),
 array([ 3. ,  4.4,  5.8,  7.2,  8.6, 10. , 11.4, 12.8, 14.2, 15.6, 17. ]),
 <BarContainer object of 10 artists>)
```



In [76]:
```
# can use a semicolon to eliminate the text:
hist(randos2);
```



5.6.

In [77]:
```
x = np.random.randn(2, 3)
x
```

Out[77]:
```
array([[ 1.14826305, -0.31213746, -1.14672703],
       [ 0.30370801, -2.70580545, -0.30731313]])
```

5.7.

In [78]:
```
x * 2
```

Out[78]:
```
array([[ 2.2965261 , -0.62427492, -2.29345406],
       [ 0.60741601, -5.41161089, -0.61462626]])
```

5.8.

In [79]:
```
x + x
```

Out[79]:
```
array([[ 2.2965261 , -0.62427492, -2.29345406],
       [ 0.60741601, -5.41161089, -0.61462626]])
```

5.9.

In [82]:
```
# get the reciprocal of the matrix:
1 / x
```

Out[82]:
```
array([[ 0.87088059, -3.20371673, -0.87204712],
       [ 3.29263628, -0.36957572, -3.25400998]])
```

5.10.

In [84]:
```
my_shape = (2, 4)
array1 = np.zeros(my_shape)
array2 = np.ones(my_shape)
```

5.11.

In [90]:
```
# idnetity matrix (4 x 4, so only put one number in):
new_shape = (4)
identity_matrix = np.identity(new_shape)
identity_matrix
```

Out[90]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

5.12.

In [92]:
```
r_vector = np.random.randn(5)
r_vector[1:4]
```

Out[92]:
```
array([ 0.13417681, -1.00204304, -0.66969925])
```

In [93]:
```
r_vector[1:-1]
```

Out[93]:
```
array([ 0.13417681, -1.00204304, -0.66969925])
```

5.13.

In [94]:
```
r_vector[r_vector > 0.15]
```

Out[94]:
```
array([1.11676574])
```

5.14.

In [ ]:
```

```