Numpy Basics

In [1]:

```
import numpy as np
```

In [2]:

```
import matplotlib.pyplot as plt
```

In [3]:

```
np.set_printoptions(precision=4)     # default is 8
np.set_printoptions(suppress=True)   # default is False (scientific notation)
```

In [4]:

```
def myprint(x):
    print(x)
    print("\nData Type:", x.dtype.name, end=', ')
    print("# Dims:", x.ndim, end = ", ")
    print("Size:",   x.size, end = ", ")
    print("Shape:",  x.shape)
```

numpy.ndarray

- multidimensional, homogeneous array of fixed-size items

In [5]:

```
# 1-D using list

a = np.array([10,20,30])
type(a)
```

Out[5]:

```
numpy.ndarray
```

In [6]:

```
myprint(a)
```

```
[10 20 30]

Data Type: int64, # Dims: 1, Size: 3, Shape: (3,)
```

In [7]:

```
# 2-D using list

b = np.array([[2.5, 3.5, 4.5], [10, 20, 30]])
myprint(b)
```

```
[[ 2.5  3.5  4.5]
 [10.  20.  30. ]]

Data Type: float64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [8]:

```
# 2-D using tuples

c = np.array(((1, 2, 3), (4, 5, 6)))
myprint(c)
```

```
[[1 2 3]
 [4 5 6]]

Data Type: int64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [9]:

```
# 2-D using lists and tuples

d = np.array([(1, 2), [3, 4], (5, 6)])
myprint(d)
```

```
[[1 2]
 [3 4]
 [5 6]]

Data Type: int64, # Dims: 2, Size: 6, Shape: (3, 2)
```

In [10]:

```
# 2-D with the desired data-type (unsigned int)

e = np.array([(1, 2), [3, 4], (5, 6)], dtype = np.uint8)
myprint(e)
```

```
[[1 2]
 [3 4]
 [5 6]]

Data Type: uint8, # Dims: 2, Size: 6, Shape: (3, 2)
```

In [11]:

```
e[0, 0] = -1
e[0, 1] = -2
myprint(e)
```

```
[[255 254]
 [  3   4]
 [  5   6]]

Data Type: uint8, # Dims: 2, Size: 6, Shape: (3, 2)
```

In [12]:

```
# 1-D using strings  (224 = 7 * 32)

names1 = np.array(['Alice', 'Bob', 'Charlie'])
myprint(names1)
```

```
['Alice' 'Bob' 'Charlie']

Data Type: str224, # Dims: 1, Size: 3, Shape: (3,)
```

numpy.arange

- Return evenly spaced values within a given interval
- numpy.arange([start,]stop, [step,] dtype=None)

In [13]:

```
np.arange(10)
```

Out[13]:

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

In [14]:

```
a = np.arange(10, 20)
myprint(a)
```

```
[10 11 12 13 14 15 16 17 18 19]

Data Type: int64, # Dims: 1, Size: 10, Shape: (10,)
```

In [15]:

```
b = np.arange(10, 20, 2, dtype = np.int16)
myprint(b)
```

```
[10 12 14 16 18]

Data Type: int16, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [16]: c = np.arange(0, 1, 0.1)
         myprint(c)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]

Data Type: float64, # Dims: 1, Size: 10, Shape: (10,)
```

```
In [17]: d = np.arange(0, 1, 0.01)
         myprint(d)
```

```
[0.   0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1  0.11 0.12 0.13
 0.14 0.15 0.16 0.17 0.18 0.19 0.2  0.21 0.22 0.23 0.24 0.25 0.26 0.27
 0.28 0.29 0.3  0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.4  0.41
 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.5  0.51 0.52 0.53 0.54 0.55
 0.56 0.57 0.58 0.59 0.6  0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69
 0.7  0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.8  0.81 0.82 0.83
 0.84 0.85 0.86 0.87 0.88 0.89 0.9  0.91 0.92 0.93 0.94 0.95 0.96 0.97
 0.98 0.99]

Data Type: float64, # Dims: 1, Size: 100, Shape: (100,)
```

numpy.linspace

- Returns *num* evenly spaced samples, calculated over the interval [start, stop]
- numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

```
In [18]: a = np.linspace(10, 100, 10)
         myprint(a)
```

```
[ 10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]

Data Type: float64, # Dims: 1, Size: 10, Shape: (10,)
```

```
In [19]: np.linspace(10, 90, 5, retstep = True)
```

```
Out[19]: (array([10., 30., 50., 70., 90.]), 20.0)
```

In [20]:
```
# default number of samples is 50

np.linspace(1, 99)
```

Out[20]:
```
array([ 1.,  3.,  5.,  7.,  9., 11., 13., 15., 17., 19., 21., 23., 25.,
       27., 29., 31., 33., 35., 37., 39., 41., 43., 45., 47., 49., 51.,
       53., 55., 57., 59., 61., 63., 65., 67., 69., 71., 73., 75., 77.,
       79., 81., 83., 85., 87., 89., 91., 93., 95., 97., 99.])
```

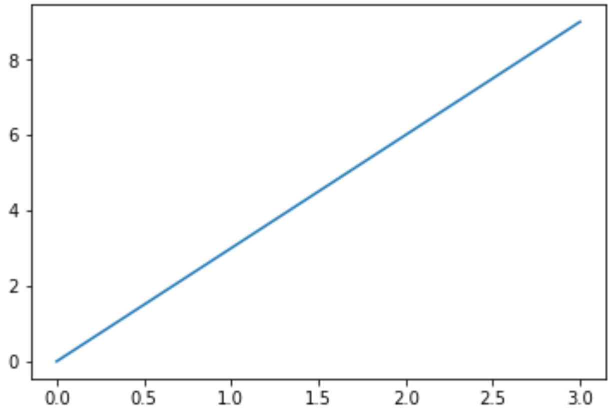
In [21]:
```
c = np.linspace(1, 5, 5)
myprint(c)
```

```
[1. 2. 3. 4. 5.]

Data Type: float64, # Dims: 1, Size: 5, Shape: (5,)
```

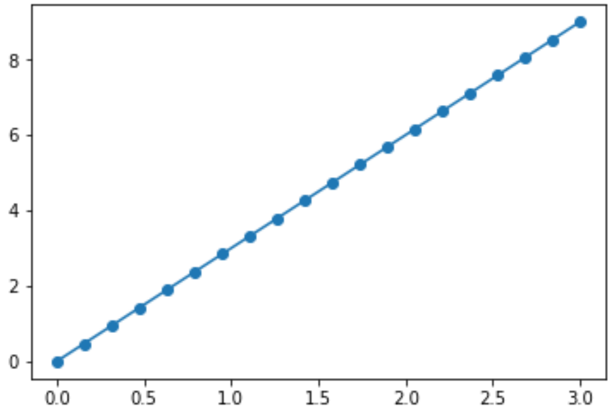
In [22]:
```
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
x[:5], y[:5]
```

Out[22]:
```
(array([0.    , 0.1579, 0.3158, 0.4737, 0.6316]),
 array([0.    , 0.4737, 0.9474, 1.4211, 1.8947]))
```

In [23]:
```
plt.plot(x, y);
```



```
In [24]: plt.plot(x, y, marker='o');
```



numpy.logspace

- Return numbers spaced evenly on a log scale
- numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
- the sequence starts at *base^start* and ends with *base^stop*

```
In [25]: d = np.logspace(1, 5, 5)
         myprint(d)
```

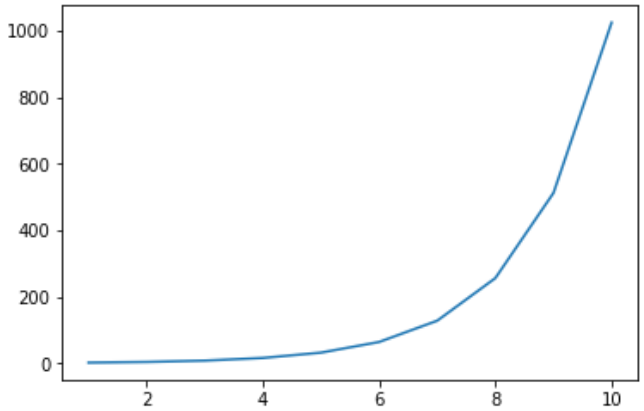
```
[    10.     100.    1000.   10000. 100000.]

Data Type: float64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [26]: np.logspace(1, 5, 5, base = 2)
```

```
Out[26]: array([ 2.,   4.,   8., 16., 32.])
```

In [27]:

```
x  = np.arange(1, 11)
y = np.logspace(1, 10, 10, base = 2)

plt.plot(x, y);
```



numpy.zeros

- Return a new array of given shape and type, filled with zeros.
- numpy.zeros(shape, dtype=float, order='C')
- C-style (store in row-major), F-style (store in column-major)

In [28]:

```
a = np.zeros(10)
myprint(a)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Data Type: float64, # Dims: 1, Size: 10, Shape: (10,)
```

In [29]:

```
b = np.zeros(10, dtype = np.int32)
myprint(b)
```

```
[0 0 0 0 0 0 0 0 0 0]

Data Type: int32, # Dims: 1, Size: 10, Shape: (10,)
```

In [30]:

```
c = np.zeros((2, 3))
myprint(c)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]

Data Type: float64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [31]:

```
# zeros_like

x = np.linspace(10, 100, 10)
y = np.zeros_like(x)
print(x)
print(y)
```

```
[ 10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

numpy.ones

- Return a new array of given shape and type, filled with ones.
- numpy.ones(shape, dtype=None, order='C')
- C-style (store in row-major), F-style (store in column-major)

In [32]:

```
a = np.ones(10)
myprint(a)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Data Type: float64, # Dims: 1, Size: 10, Shape: (10,)
```

In [33]:

```
b = np.ones(10, dtype = np.int32)
myprint(b)
```

```
[1 1 1 1 1 1 1 1 1 1]

Data Type: int32, # Dims: 1, Size: 10, Shape: (10,)
```

In [34]:
```
c = np.ones((2, 3))
myprint(c)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]

Data Type: float64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [35]:
```
# ones_like

x = np.linspace(10, 100, 10)
y = np.ones_like(x)
print(x)
print(y)
```

```
[ 10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

numpy.eye

- numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')
- Return a 2-D array with ones on the diagonal and zeros elsewhere
- N (number of rows), M (optional - number of columns, defaults to N)
- k (optional - index of diagonal, 0 - main diagonal)

In [36]:
```
a = np.eye(3)
myprint(a)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Data Type: float64, # Dims: 2, Size: 9, Shape: (3, 3)
```

In [37]:

```
b = np.eye(5, dtype= int)
myprint(b)
```

```
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]

Data Type: int64, # Dims: 2, Size: 25, Shape: (5, 5)
```

In [38]:

```
c = np.eye(5, k = 1, dtype= int)
myprint(c)
```

```
[[0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 0]]

Data Type: int64, # Dims: 2, Size: 25, Shape: (5, 5)
```

In [39]:

```
d = np.eye(5, k = -2, dtype= int)
myprint(d)
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]]

Data Type: int64, # Dims: 2, Size: 25, Shape: (5, 5)
```

In [ ]:

```

```

numpy.full

- numpy.full(shape, fill_value, dtype=None, order='C')

In [40]:

```
a = np.full(5, 10)
myprint(a)
```

```
[10 10 10 10 10]

Data Type: int64, # Dims: 1, Size: 5, Shape: (5,)
```

In [41]:

```
b = np.full((2, 3), 2.5)
myprint(b)
```

```
[[2.5 2.5 2.5]
 [2.5 2.5 2.5]]

Data Type: float64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [42]:

```
x = np.linspace(10, 100, 10)
y = np.full_like(x, 99)
print(x)
print(y)
```

```
[ 10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]
[99. 99. 99. 99. 99. 99. 99. 99. 99. 99.]
```

numpy.empty

- numpy.empty(shape, dtype=float, order='C')
- Return a new array of given shape and type, without initializing entries
- Use with caution

In [43]:

```
a = np.empty(5)
myprint(a)
```

```
[0. 0. 0. 0. 0.]

Data Type: float64, # Dims: 1, Size: 5, Shape: (5,)
```

In [44]:

```
b = np.empty((2, 2))
myprint(b)
```

```
[[  0.      0.    ]
 [384.825 252.34  ]]

Data Type: float64, # Dims: 2, Size: 4, Shape: (2, 2)
```

In [45]:
```
x = np.linspace(10, 100, 10)
y = np.empty_like(x)
print(x)
print(y)
```

```
[ 10.   20.   30.   40.   50.   60.   70.   80.   90. 100.]
[ 10.   20.   30.   40.   50.   60.   70.   80.   90. 100.]
```

Random Numbers (np.random module)

randint

- randint(low, high=None, size=None)
- return random integers from low (inclusive) to high (exclusive)
- Uses **discrete uniform** distribution

In [46]:
```
np.random.randint(10) # one random integer in the range [0, low)
```

Out[46]:
```
5
```

In [47]:
```
np.random.randint(10, 20)
```

Out[47]:
```
12
```

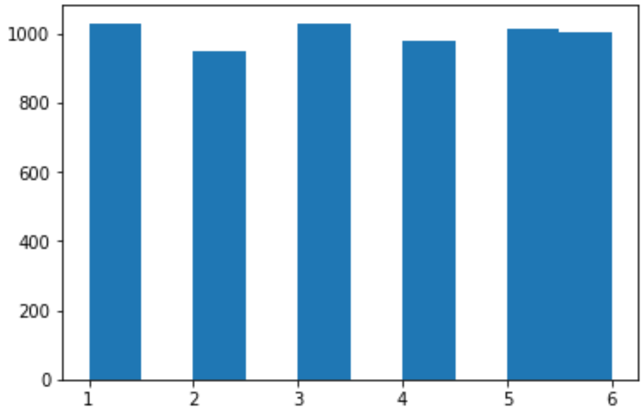
In [48]:
```
a = np.random.randint(1, 7, 20)
myprint(a)
```

```
[3 5 1 1 3 6 2 6 1 3 4 1 1 6 6 2 6 2 3 3]

Data Type: int64, # Dims: 1, Size: 20, Shape: (20,)
```

In [49]:
```
a = np.random.randint(1, 7, (4, 5))
myprint(a)
```

```
[[4 2 2 6 3]
 [1 2 3 6 4]
 [3 4 3 4 2]
 [5 3 2 1 1]]

Data Type: int64, # Dims: 2, Size: 20, Shape: (4, 5)
```

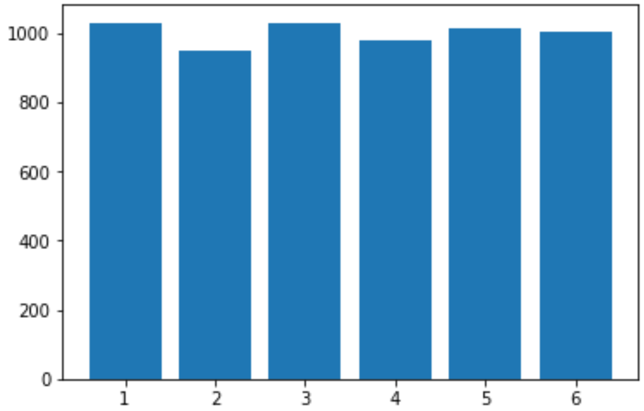
```
In [50]: a = np.random.randint(1, 7, 6000)
         plt.hist(a);
```



```
In [51]: np.unique(a)

Out[51]: array([1, 2, 3, 4, 5, 6])
```

```
In [52]: x, y = np.unique(a, return_counts = True)
         x, y

Out[52]: (array([1, 2, 3, 4, 5, 6]), array([1028,  948, 1030,  979, 1013, 1002]))
```

```
In [53]: plt.bar(x, y);
```



random or random_sample

- random(size=None)
- random_sample(size=None)
- return random floats in the interval [0.0, 1.0)
- Uses **continuous uniform** distribution

```
In [54]: a = np.random.random()
         a
```

```
Out[54]: 0.8017184857042494
```

```
In [55]: b = np.random.random(3)
         myprint(b)
```

```
[0.4315 0.0992 0.7911]

Data Type: float64, # Dims: 1, Size: 3, Shape: (3,)
```

```
In [56]: c = np.random.random((3, 3))
         myprint(c)
```

```
[[0.0375 0.5808 0.0197]
 [0.1749 0.6756 0.0247]
 [0.9885 0.9291 0.5415]]

Data Type: float64, # Dims: 2, Size: 9, Shape: (3, 3)
```

seed

```
In [57]: # for reproducible results, use a seed

         np.random.seed(123)
         np.random.random((3, 3))
```

```
Out[57]: array([[0.6965, 0.2861, 0.2269],
                [0.5513, 0.7195, 0.4231],
                [0.9808, 0.6848, 0.4809]])
```

In [58]:
```
np.random.seed(123)
np.random.random_sample((3, 3))
```

Out[58]:
```
array([[0.6965, 0.2861, 0.2269],
       [0.5513, 0.7195, 0.4231],
       [0.9808, 0.6848, 0.4809]])
```

In [59]:
```
d = np.random.random((2, 3, 4))
myprint(d)
```

```
[[[0.3921 0.3432 0.729  0.4386]
  [0.0597 0.398  0.738  0.1825]
  [0.1755 0.5316 0.5318 0.6344]]

 [[0.8494 0.7245 0.611  0.7224]
  [0.323  0.3618 0.2283 0.2937]
  [0.631  0.0921 0.4337 0.4309]]]

Data Type: float64, # Dims: 3, Size: 24, Shape: (2, 3, 4)
```

For random uniform sampling over [a, b)

- (b - a) * random_sample() + a

In [60]:
```
# 3 random samples over [10, 15)
5 * np.random.random_sample(3) + 10
```

Out[60]:
```
array([12.4684, 12.1292, 11.5613])
```

rand(d0, d1, ..., dn)

- convenience function - continuous uniform distribution [0, 1)
- for shape as tuple, use random or random_sample

In [61]:
```
np.random.rand()
```

Out[61]:
```
0.4263513069628082
```

In [62]:
```
np.random.rand(3)
```

Out[62]:
```
array([0.8934, 0.9442, 0.5018])
```

```
In [63]: np.random.seed(123)
         np.random.rand(3, 3)
```

```
Out[63]: array([[0.6965, 0.2861, 0.2269],
                [0.5513, 0.7195, 0.4231],
                [0.9808, 0.6848, 0.4809]])
```

```
In [64]: np.random.rand(2, 3, 4)
```

```
Out[64]: array([[[0.3921, 0.3432, 0.729 , 0.4386],
                 [0.0597, 0.398 , 0.738 , 0.1825],
                 [0.1755, 0.5316, 0.5318, 0.6344]],

                [[0.8494, 0.7245, 0.611 , 0.7224],
                 [0.323 , 0.3618, 0.2283, 0.2937],
                 [0.631 , 0.0921, 0.4337, 0.4309]]])
```

randn

- randn(d0, d1, ..., dn)
- convenience function
- for shape as tuple, use standard_normal
- returns random floats from the **standard normal (Gaussian)** distribution
- mean 0 and sd 1

```
In [65]: np.random.seed(36789)
```

```
In [66]: np.random.randn()
```

```
Out[66]: 0.5176406974584342
```

```
In [67]: a = np.random.randn(6)
         myprint(a)
```

```
[-0.4418 -1.6478  1.3586 -0.2121  0.2946  0.0329]

Data Type: float64, # Dims: 1, Size: 6, Shape: (6,)
```

In [68]:
```
np.random.seed(123)
b = np.random.randn(2, 3)
myprint(b)
```

```
[[-1.0856  0.9973  0.283 ]
 [-1.5063 -0.5786  1.6514]]

Data Type: float64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [69]:
```
c = np.random.randn(2, 3, 4)
myprint(c)
```

```
[[[-2.4267 -0.4289  1.2659 -0.8667]
  [-0.6789 -0.0947  1.4914 -0.6389]
  [-0.444  -0.4344  2.2059  2.1868]]

 [[ 1.0041  0.3862  0.7374  1.4907]
  [-0.9358  1.1758 -1.2539 -0.6378]
  [ 0.9071 -1.4287 -0.1401 -0.8618]]]

Data Type: float64, # Dims: 3, Size: 24, Shape: (2, 3, 4)
```

In [70]:
```
# for data from normal distribution with mean 60 and sd 5

np.random.seed(123)
x = 5 * np.random.randn(10000) + 60
x[:5]
```

Out[70]:
```
array([54.5718, 64.9867, 61.4149, 52.4685, 57.107 ])
```

In [71]:
```
np.mean(x), np.std(x)
```

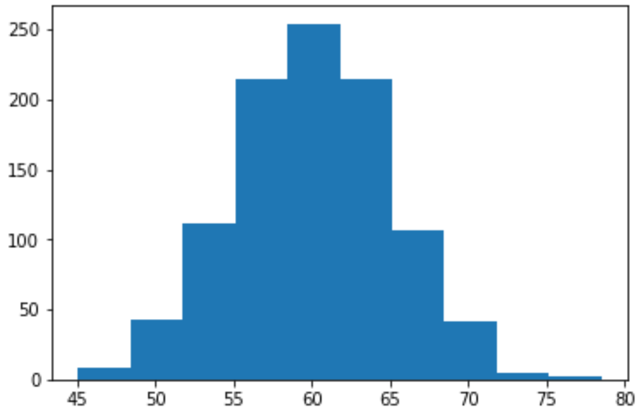
Out[71]:
```
(60.048559461457984, 4.9905692729507365)
```

numpy.random.normal(mean, sd, size)

In [72]:
```
np.random.seed(123)
np.random.normal(60, 5, 10)
```

Out[72]:
```
array([54.5718, 64.9867, 61.4149, 52.4685, 57.107 , 68.2572, 47.8666,
       57.8554, 66.3297, 55.6663])
```

```
In [73]: np.random.seed(3)
         scores = np.random.normal(60, 5, 1000)
         plt.hist(scores, bins=10);
```



standard_normal(size = None)

- similar to randn, but takes tuple as argument

```
In [74]: np.random.seed(123)
         np.random.standard_normal((2, 3))

Out[74]: array([[-1.0856,  0.9973,  0.283 ],
                [-1.5063, -0.5786,  1.6514]])
```

```
In [75]: np.random.standard_normal((2, 3, 4))

Out[75]: array([[[-2.4267, -0.4289,  1.2659, -0.8667],
                 [-0.6789, -0.0947,  1.4914, -0.6389],
                 [-0.444 , -0.4344,  2.2059,  2.1868]],

                [[ 1.0041,  0.3862,  0.7374,  1.4907],
                 [-0.9358,  1.1758, -1.2539, -0.6378],
                 [ 0.9071, -1.4287, -0.1401, -0.8618]]])
```

choice

- choice(a, size=None, replace=True, p=None)
- a : 1-D array-like or int
- Generates a random sample from a given 1-D array
- If an int, the random sample is generated as if a were np.arange(a)

In [76]:
```
np.random.seed(456789)
outcomes = np.random.choice(['H', 'T'], 200)
outcomes
```

Out[76]:
```
array(['T', 'H', 'T', 'H', 'H', 'H', 'H', 'T', 'T', 'T', 'T', 'T', 'H',
       'H', 'T', 'T', 'H', 'T', 'T', 'H', 'T', 'T', 'H', 'T', 'H', 'H',
       'T', 'T', 'H', 'H', 'H', 'T', 'H', 'T', 'H', 'T', 'H', 'H', 'T',
       'T', 'H', 'H', 'H', 'T', 'H', 'T', 'T', 'T', 'T', 'T', 'T', 'T',
       'H', 'H', 'T', 'H', 'T', 'T', 'T', 'T', 'H', 'T', 'T', 'H', 'H',
       'H', 'T', 'H', 'H', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'T', 'T',
       'H', 'T', 'H', 'T', 'T', 'T', 'T', 'H', 'T', 'H', 'T', 'H', 'T',
       'T', 'T', 'H', 'T', 'H', 'H', 'T', 'H', 'T', 'T', 'H', 'H', 'H',
       'H', 'T', 'H', 'H', 'H', 'T', 'T', 'T', 'T', 'T', 'H', 'T', 'T',
       'T', 'H', 'T', 'H', 'T', 'T', 'H', 'T', 'H', 'T', 'H', 'H', 'T',
       'H', 'H', 'T', 'T', 'T', 'H', 'H', 'H', 'T', 'H', 'H', 'T', 'T',
       'H', 'T', 'H', 'T', 'T', 'H', 'H', 'T', 'H', 'T', 'T', 'H', 'T',
       'H', 'T', 'H', 'H', 'H', 'H', 'H', 'H', 'T', 'T', 'T', 'H', 'H',
       'H', 'H', 'H', 'T', 'H', 'T', 'T', 'T', 'H', 'T', 'H', 'H', 'H',
       'T', 'H', 'H', 'T', 'H', 'H', 'H', 'T', 'T', 'H', 'T', 'T', 'T',
       'T', 'H', 'H', 'H', 'H'], dtype='<U1')
```

In [77]:
```
np.unique(outcomes, return_counts = True)
```

Out[77]:
```
(array(['H', 'T'], dtype='<U1'), array([101,  99]))
```

In [78]:
```
np.random.seed(456789)
outcomes = np.random.choice(['H', 'T'], 20, p = [0.8, 0.2])
outcomes
```

Out[78]:
```
array(['T', 'H', 'H', 'H', 'T', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H',
       'T', 'H', 'H', 'H', 'H', 'T', 'H'], dtype='<U1')
```

In [79]:
```
np.unique(outcomes, return_counts = True)
```

Out[79]:
```
(array(['H', 'T'], dtype='<U1'), array([16,  4]))
```

```
In [80]: np.random.seed(123)
         np.random.choice(6, 100)

Out[80]: array([5, 2, 4, 2, 1, 3, 2, 3, 1, 1, 0, 1, 1, 0, 0, 1, 3, 5, 4, 0, 0, 4,
                1, 3, 2, 4, 2, 4, 0, 5, 0, 1, 3, 4, 4, 4, 1, 5, 3, 2, 1, 4, 0, 3,
                2, 5, 0, 3, 2, 2, 2, 5, 2, 4, 3, 3, 5, 4, 4, 5, 3, 2, 0, 4, 3, 1,
                3, 2, 5, 1, 2, 4, 0, 1, 4, 2, 1, 1, 3, 4, 5, 1, 0, 0, 3, 1, 3, 3,
                3, 5, 1, 1, 2, 3, 3, 3, 3, 0, 1, 3])
```

```
In [81]: np.random.seed(123)
         np.random.choice(5, 5, replace = False)

Out[81]: array([1, 3, 4, 0, 2])
```

```
In [82]: # same as
         np.random.seed(123)
         np.random.permutation(np.arange(5))

Out[82]: array([1, 3, 4, 0, 2])
```

shuffle

- Modify a sequence in-place by shuffling its contents

```
In [83]: a = np.arange(10)
         np.random.shuffle(a)
         a

Out[83]: array([7, 6, 4, 0, 5, 9, 8, 2, 3, 1])
```

```
In [84]: np.random.shuffle(a)
         a

Out[84]: array([5, 2, 9, 4, 0, 6, 3, 8, 7, 1])
```

permutation

- numpy.random.permutation(x)
- If x is an integer, randomly permute np.arange(x)
- If x is an array, make a copy and shuffle the elements randomly

```
In [85]: np.random.permutation(10)
```

```
Out[85]: array([6, 5, 7, 8, 2, 3, 1, 4, 9, 0])
```

```
In [86]: a = np.arange(10)
         print(np.random.permutation(10))
         print(a)
```

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

Unique values

- Find the unique elements of an array
- unique(ar, return_index=False, return_inverse=False, return_counts=False, axis=None)

```
In [87]: a = np.array(['a', 'b', 'c', 'a', 'c', 'a', 'b', 'd'])
         myprint(a)
```

```
['a' 'b' 'c' 'a' 'c' 'a' 'b' 'd']

Data Type: str32, # Dims: 1, Size: 8, Shape: (8,)
```

```
In [88]: np.unique(a)
```

```
Out[88]: array(['a', 'b', 'c', 'd'], dtype='<U1')
```

```
In [89]: np.unique(a, return_index = True)
```

```
Out[89]: (array(['a', 'b', 'c', 'd'], dtype='<U1'), array([0, 1, 2, 7]))
```

```
In [90]: _, indices = np.unique(a, return_index = True)
         a[indices]
```

```
Out[90]: array(['a', 'b', 'c', 'd'], dtype='<U1')
```

```
In [91]: np.unique(a, return_inverse = True)
```

```
Out[91]: (array(['a', 'b', 'c', 'd'], dtype='<U1'), array([0, 1, 2, 0, 2, 0, 1, 3]))
```

In [92]:
```
# Reconstruct

values, indices = np.unique(a, return_inverse = True)
values[indices]
```

Out[92]:
```
array(['a', 'b', 'c', 'a', 'c', 'a', 'b', 'd'], dtype='<U1')
```

In [93]:
```
# counts

np.unique(a, return_counts = True)
```

Out[93]:
```
(array(['a', 'b', 'c', 'd'], dtype='<U1'), array([3, 2, 2, 1]))
```

Count Non-zero values

In [94]:
```
a = np.eye(3)
a
```

Out[94]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [95]:
```
np.count_nonzero(a)
```

Out[95]:
```
3
```

In [96]:
```
a = np.array([[0, 5, 6, 0], [1, 0, 0, 0], [2, 1, 0, 0]])
a
```

Out[96]:
```
array([[0, 5, 6, 0],
       [1, 0, 0, 0],
       [2, 1, 0, 0]])
```

In [97]:
```
np.count_nonzero(a)
```

Out[97]:
```
5
```

In [98]:
```
# along columns

np.count_nonzero(a, axis = 0)
```

Out[98]:
```
array([2, 2, 1, 0])
```

In [99]:
```
# along rows

np.count_nonzero(a, axis = 1)
```

Out[99]:
```
array([2, 1, 2])
```

In [100]:
```
a = np.array([True, False, False, True, False])
print(a)

np.count_nonzero(a)
```

```
[ True False False  True False]
```

Out[100]:
```
2
```

Reshape

- Give a new shape to an array without changing its data
- numpy.reshape(a, newshape, order='C')

In [101]:
```
a = np.arange(0, 20)
myprint(a)
```

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

Data Type: int64, # Dims: 1, Size: 20, Shape: (20,)
```

In [102]:
```
b = np.arange(0, 20).reshape(4, 5)
myprint(b)
```

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

Data Type: int64, # Dims: 2, Size: 20, Shape: (4, 5)
```

```
In [103]: c = np.arange(0, 20).reshape(5, 4)
          myprint(c)
```

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

Data Type: int64, # Dims: 2, Size: 20, Shape: (5, 4)
```

```
In [104]: d = np.arange(0, 20).reshape(2, 5, 2)
          myprint(d)
```

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

 [[10 11]
  [12 13]
  [14 15]
  [16 17]
  [18 19]]]

Data Type: int64, # Dims: 3, Size: 20, Shape: (2, 5, 2)
```

Type Conversions

```
In [105]: a = np.linspace(10, 20, 5)
          myprint(a)
```

```
[10.  12.5 15.  17.5 20. ]

Data Type: float64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [106]: b = a.astype(int)
          myprint(b)
```

```
[10 12 15 17 20]

Data Type: int64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [ ]:
```

ndarray Operations

```
In [107]: a = np.arange(5)
          myprint(a)

          [0 1 2 3 4]

          Data Type: int64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [108]: b = a + 5
          myprint(b)

          [5 6 7 8 9]

          Data Type: int64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [109]: a + a

Out[109]: array([0, 2, 4, 6, 8])
```

```
In [110]: c = a * 5
          myprint(c)

          [ 0  5 10 15 20]

          Data Type: int64, # Dims: 1, Size: 5, Shape: (5,)
```

```
In [111]: a * a

Out[111]: array([ 0,  1,  4,  9, 16])
```

```
In [112]: a.dot(a)

Out[112]: 30
```

```
In [113]: np.sum(a * a)

Out[113]: 30
```

2-D Operations

In [114]:
```
a = np.arange(1, 7).reshape(2,3)
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

In [115]:
```
b = a * a
print(b)
```

```
[[ 1  4  9]
 [16 25 36]]
```

In [116]:
```
c = a ** 3
print(c)
```

```
[[  1   8  27]
 [ 64 125 216]]
```

In [117]:
```
d = 1 / a
print(d)
```

```
[[1.     0.5    0.3333]
 [0.25   0.2    0.1667]]
```

In [118]:
```
## Matrix multiplication
```

In [119]:
```
a = np.arange(1, 7).reshape(2,3)
myprint(a)
```

```
[[1 2 3]
 [4 5 6]]

Data Type: int64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [120]:
```
b = np.arange(1, 7).reshape(3,2)
myprint(b)
```

```
[[1 2]
 [3 4]
 [5 6]]

Data Type: int64, # Dims: 2, Size: 6, Shape: (3, 2)
```

```
In [121]: c = a.dot(b)
          myprint(c)

          [[22 28]
           [49 64]]

          Data Type: int64, # Dims: 2, Size: 4, Shape: (2, 2)
```

```
In [122]: a @ b

Out[122]: array([[22, 28],
                 [49, 64]])
```

```
In [123]: d = b.dot(a)
          myprint(d)

          [[ 9 12 15]
           [19 26 33]
           [29 40 51]]

          Data Type: int64, # Dims: 2, Size: 9, Shape: (3, 3)
```

```
In [124]: b @ a

Out[124]: array([[ 9, 12, 15],
                 [19, 26, 33],
                 [29, 40, 51]])
```

Unary operators

```
In [125]: a = np.arange(1, 7).reshape(2,3)
          myprint(a)

          [[1 2 3]
           [4 5 6]]

          Data Type: int64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [126]:

```
a += 1
myprint(a)
```

```
[[2 3 4]
 [5 6 7]]

Data Type: int64, # Dims: 2, Size: 6, Shape: (2, 3)
```

In [127]:

```
a -= 5
myprint(a)
```

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

Data Type: int64, # Dims: 2, Size: 6, Shape: (2, 3)
```

Transposing arrays

In [128]:

```
a = np.arange(10).reshape(2, 5)
a
```

Out[128]:

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

In [129]:

```
a.T
```

Out[129]:

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

In [130]:

```
np.dot(a, a.T)
```

Out[130]:

```
array([[ 30,  80],
       [ 80, 255]])
```

In [131]:

```
a
```

Out[131]:

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

Comparison Operators

In [132]:
```
a = np.array([10, 20, 40])
a
```

Out[132]:
```
array([10, 20, 40])
```

In [133]:
```
b = np.array([5, 25, 30])
b
```

Out[133]:
```
array([ 5, 25, 30])
```

In [134]:
```
c = a > b
c
```

Out[134]:
```
array([ True, False,  True])
```

In [135]:
```
a[c]
```

Out[135]:
```
array([10, 40])
```

In [136]:
```
a[a > 20]
```

Out[136]:
```
array([40])
```

where

- numpy.where(condition, x, y)
- Return elements, either from x or y, depending on condition

In [137]:
```
a = [10, 20, 30, 40]
b = [60, 70, 80, 90]
c = [True, False, False, True]

result = [(a_ if c_ else b_ ) for a_, b_, c_ in zip(a, b, c)]
result
```

Out[137]:
```
[10, 70, 80, 40]
```

In [138]:

```
# same as

np.where(c, a, b)
```

Out[138]:

```
array([10, 70, 80, 40])
```

In [139]:

```
# 2-D

np.random.seed(123)
a = np.random.randn(4,4)
a
```

Out[139]:

```
array([[-1.0856,  0.9973,  0.283 , -1.5063],
       [-0.5786,  1.6514, -2.4267, -0.4289],
       [ 1.2659, -0.8667, -0.6789, -0.0947],
       [ 1.4914, -0.6389, -0.444 , -0.4344]])
```

In [140]:

```
np.where(a > 0, a, 0)
```

Out[140]:

```
array([[0.    , 0.9973, 0.283 , 0.    ],
       [0.    , 1.6514, 0.    , 0.    ],
       [1.2659, 0.    , 0.    , 0.    ],
       [1.4914, 0.    , 0.    , 0.    ]])
```

In [141]:

```
np.where(a > 0, 1, -1)
```

Out[141]:

```
array([[-1,  1,  1, -1],
       [-1,  1, -1, -1],
       [ 1, -1, -1, -1],
       [ 1, -1, -1, -1]])
```

Boolean Array operations

In [142]:
```
np.random.seed(123)
a = np.random.randn(100)
a
```

Out[142]:
```
array([-1.0856,  0.9973,  0.283 , -1.5063, -0.5786,  1.6514, -2.4267,
       -0.4289,  1.2659, -0.8667, -0.6789, -0.0947,  1.4914, -0.6389,
       -0.444 , -0.4344,  2.2059,  2.1868,  1.0041,  0.3862,  0.7374,
        1.4907, -0.9358,  1.1758, -1.2539, -0.6378,  0.9071, -1.4287,
       -0.1401, -0.8618, -0.2556, -2.7986, -1.7715, -0.6999,  0.9275,
       -0.1736,  0.0028,  0.6882, -0.8795,  0.2836, -0.8054, -1.7277,
       -0.3909,  0.5738,  0.3386, -0.0118,  2.3924,  0.4129,  0.9787,
        2.2381, -1.2941, -1.0388,  1.7437, -0.7981,  0.0297,  1.0693,
        0.8907,  1.7549,  1.4956,  1.0694, -0.7727,  0.7949,  0.3143,
       -1.3263,  1.4173,  0.8072,  0.0455, -0.2331, -1.1983,  0.1995,
        0.4684, -0.8312,  1.1622, -1.0972, -2.1231,  1.0397, -0.4034,
       -0.126 , -0.8375, -1.606 ,  1.2552, -0.6889,  1.661 ,  0.8073,
       -0.3148, -1.0859, -0.7325, -1.2125,  2.0871,  0.1644,  1.1502,
       -1.2674,  0.181 ,  1.1779, -0.335 ,  1.0311, -1.0846, -1.3635,
        0.3794, -0.3792])
```

In [143]:
```
# Number of positive values

a > 0
```

Out[143]:
```
array([False,  True,  True, False, False,  True, False, False,  True,
       False, False, False,  True, False, False, False,  True,  True,
        True,  True,  True,  True, False,  True, False, False,  True,
       False, False, False, False, False, False, False,  True, False,
        True,  True, False,  True, False, False, False,  True,  True,
       False,  True,  True,  True,  True, False, False,  True, False,
        True,  True,  True,  True,  True,  True, False,  True,  True,
       False,  True,  True,  True, False, False,  True,  True, False,
        True, False, False,  True, False, False, False, False,  True,
       False,  True,  True, False, False, False, False,  True,  True,
        True, False,  True,  True, False,  True, False, False,  True,
       False])
```

In [144]:
```
(a > 0).sum()
```

Out[144]:
```
49
```

In [145]:
```
# or

np.count_nonzero(a > 0)
```

Out[145]:
```
49
```

In [146]:
```
# any of the values true?

(a > 0).any()
```

Out[146]:
```
True
```

In [147]:
```
# all the values true?

(a > 0).all()
```

Out[147]:
```
False
```

In [148]:
```
(a < 3).all()
```

Out[148]:
```
True
```

repeat

- Repeat elements of an array
- numpy.repeat(a, repeats, axis=None)

In [149]:
```
np.repeat(10, 4)
```

Out[149]:
```
array([10, 10, 10, 10])
```

In [150]:
```
a = np.arange(0, 5)
a
```

Out[150]:
```
array([0, 1, 2, 3, 4])
```

In [151]:
```
np.repeat(a, 2)
```

Out[151]:
```
array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4])
```

In [152]:
```
np.repeat(a, a)
```

Out[152]:
```
array([1, 2, 2, 3, 3, 3, 4, 4, 4, 4])
```

In [153]:
```
np.repeat(a, [2, 3, 4, 5, 6])
```

Out[153]:
```
array([0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4])
```

```
In [154]: # 2-D

          a = np.arange(0,6).reshape(2, 3)
          a
```

```
Out[154]: array([[0, 1, 2],
                 [3, 4, 5]])
```

```
In [155]: np.repeat(a, 2)
```

```
Out[155]: array([0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5])
```

```
In [156]: # along columns

          np.repeat(a, 2, axis = 0)
```

```
Out[156]: array([[0, 1, 2],
                 [0, 1, 2],
                 [3, 4, 5],
                 [3, 4, 5]])
```

```
In [157]: # along rows

          np.repeat(a, 2, axis = 1)
```

```
Out[157]: array([[0, 0, 1, 1, 2, 2],
                 [3, 3, 4, 4, 5, 5]])
```

```
In [158]: a
```

```
Out[158]: array([[0, 1, 2],
                 [3, 4, 5]])
```

```
In [159]: np.repeat(a, [2, 4], axis = 0)
```

```
Out[159]: array([[0, 1, 2],
                 [0, 1, 2],
                 [3, 4, 5],
                 [3, 4, 5],
                 [3, 4, 5],
                 [3, 4, 5]])
```

```
In [160]: a
```

```
Out[160]: array([[0, 1, 2],
                 [3, 4, 5]])
```

```
In [161]: np.repeat(a, [2, 4, 6], axis = 1)
```

```
Out[161]: array([[0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2],
                 [3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5]])
```

tile

- numpy.tile(A, reps)
- construct an array by repeating it the specified number of times

```
In [162]: a = np.array([10, 11, 12])
          a
```

```
Out[162]: array([10, 11, 12])
```

```
In [163]: np.tile(a, 2)
```

```
Out[163]: array([10, 11, 12, 10, 11, 12])
```

```
In [164]: a
```

```
Out[164]: array([10, 11, 12])
```

```
In [165]: np.tile(a, (2,3))
```

```
Out[165]: array([[10, 11, 12, 10, 11, 12, 10, 11, 12],
                 [10, 11, 12, 10, 11, 12, 10, 11, 12]])
```

```
In [166]: a
```

```
Out[166]: array([10, 11, 12])
```

```
In [167]: np.tile(a, (4,1))

Out[167]: array([[10, 11, 12],
                 [10, 11, 12],
                 [10, 11, 12],
                 [10, 11, 12]])

In [168]: a

Out[168]: array([10, 11, 12])

In [169]: np.tile(a, (1,4))

Out[169]: array([[10, 11, 12, 10, 11, 12, 10, 11, 12, 10, 11, 12]])

In [170]: a

Out[170]: array([10, 11, 12])

In [171]: np.tile(a, (2,4,3))

Out[171]: array([[[10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12]],

                 [[10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12],
                  [10, 11, 12, 10, 11, 12, 10, 11, 12]]])

In [ ]: 
```

ravel

- numpy.ravel(a, order='C')
- Return a contiguous flattened array.

```
In [172]: a = np.arange(10).reshape(2, 5)
          a
Out[172]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [173]: np.ravel(a)   # default C-Style (row-major)
Out[173]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [174]: a
Out[174]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [175]: np.ravel(a, order = 'F')
          # column-major (Fortran-style) - default order 'C'
Out[175]: array([0, 5, 1, 6, 2, 7, 3, 8, 4, 9])
```

```
In [176]: a = np.arange(12).reshape(2,3,2)
          a
Out[176]: array([[[ 0,  1],
                  [ 2,  3],
                  [ 4,  5]],

                 [[ 6,  7],
                  [ 8,  9],
                  [10, 11]]])
```

```
In [177]: np.ravel(a)
Out[177]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
In [178]: np.ravel(a, order = 'F')
Out[178]: array([ 0,  6,  2,  8,  4, 10,  1,  7,  3,  9,  5, 11])
```

ndarray.flatten

- ndarray.flatten(order='C')

```
In [179]: a = np.arange(10).reshape(2, 5)
          a

Out[179]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [180]: a.flatten()  # default order is 'C'

Out[180]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [181]: a

Out[181]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [182]: a.flatten(order = 'F')

Out[182]: array([0, 5, 1, 6, 2, 7, 3, 8, 4, 9])
```

flatten vs ravel

- ravel() returns a flattened view of Numpy array.
- flatten() returns a flattened copy of Numpy array.

```
In [183]: a = np.arange(10).reshape(2, 5)
          a

Out[183]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [184]: b = a.ravel()
          b

Out[184]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [185]: b[5] = 500
          b

Out[185]: array([  0,   1,   2,   3,   4, 500,   6,   7,   8,   9])
```

```
In [186]: a

Out[186]: array([[  0,   1,   2,   3,   4],
                 [500,   6,   7,   8,   9]])
```

```
In [187]: a = np.arange(10).reshape(2, 5)
          a

Out[187]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [188]: b = a.flatten()
          b

Out[188]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [189]: b[5] = 500
          b

Out[189]: array([  0,   1,   2,   3,   4, 500,   6,   7,   8,   9])
```

```
In [190]: a

Out[190]: array([[0, 1, 2, 3, 4],
                 [5, 6, 7, 8, 9]])
```

```
In [ ]:
```

resize

- numpy.resize(a, new_shape)
- if size is larger, filled with repeated copies of the array

```
In [191]: a = np.arange(4)
          a

Out[191]: array([0, 1, 2, 3])
```

```
In [192]: np.resize(a, 10)

Out[192]: array([0, 1, 2, 3, 0, 1, 2, 3, 0, 1])
```

```
In [193]: np.resize(a, 2)

Out[193]: array([0, 1])
```

```
In [194]: np.resize(a, (3,3))

Out[194]: array([[0, 1, 2],
                 [3, 0, 1],
                 [2, 3, 0]])
```

```
In [195]: a

Out[195]: array([0, 1, 2, 3])
```

Copies and Views

```
In [196]: a = np.arange(10, 19)
          a

Out[196]: array([10, 11, 12, 13, 14, 15, 16, 17, 18])
```

```
In [197]: id(a)

Out[197]: 4575540848
```

```
In [198]: b = a    # No copy, same object
```

```
In [199]: id(b)

Out[199]: 4575540848
```

View - new array object looking at same data

```
In [200]: c = a.view()
          c

Out[200]: array([10, 11, 12, 13, 14, 15, 16, 17, 18])
```

```
In [201]: id(c)

Out[201]: 4575540608
```

```
In [202]: c[-1] = 99
          c
Out[202]: array([10, 11, 12, 13, 14, 15, 16, 17, 99])
```

```
In [203]: a
Out[203]: array([10, 11, 12, 13, 14, 15, 16, 17, 99])
```

Slicing an array returns a view

```
In [204]: s = a[1:4]
          s
Out[204]: array([11, 12, 13])
```

```
In [205]: s[:] = 123
          s
Out[205]: array([123, 123, 123])
```

```
In [206]: a
Out[206]: array([ 10, 123, 123, 123,  14,  15,  16,  17,  99])
```

Deep copy - complete copy of array and its data

```
In [207]: a = np.arange(10, 19)
          a
Out[207]: array([10, 11, 12, 13, 14, 15, 16, 17, 18])
```

```
In [208]: d = a.copy()
          d[:] = -1
          d
Out[208]: array([-1, -1, -1, -1, -1, -1, -1, -1, -1])
```

```
In [209]: a
Out[209]: array([10, 11, 12, 13, 14, 15, 16, 17, 18])
```

Case Study - Random Walks

In [210]:

```
np.random.seed(58317)

position = 0
walk = [position]
steps = 1000
for i in range(steps):
    step = 1 if (np.random.rand() < 0.5) else -1
    position += step
    walk.append(position)
```

In [211]:

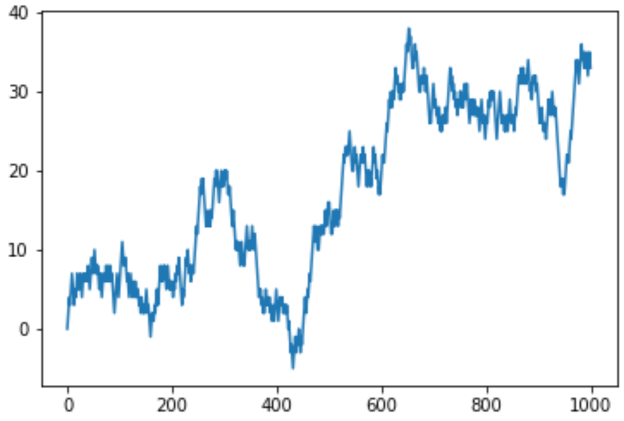
```
walk[:10]
```

Out[211]:

```
[0, 1, 2, 3, 4, 3, 4, 5, 6, 7]
```

In [212]:

```
plt.plot(walk[:1000]);
```



Case Study - Images

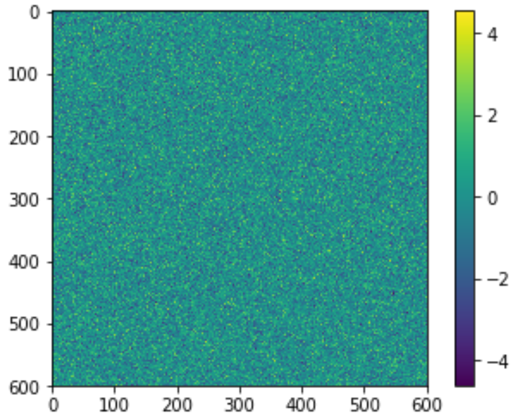
In [213]:

```
np.random.seed(63421)
image = np.random.randn(600, 600)
```

In [214]:
```
image[0,0]
```

Out[214]:
```
0.05145325936491756
```

In [215]:
```
plt.imshow(image)
plt.colorbar()
plt.show()
```



In [ ]:
```

```

In [216]:
```
np.array([np.random.randint(0,256), np.random.randint(0,256), np.random.randint(0,256)]).reshape((1, 1, 3))
```

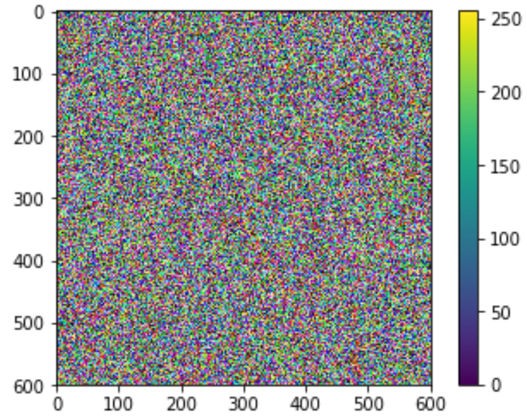
Out[216]:
```
array([[[54, 22, 75]]])
```

In [217]:
```
image = np.random.randint(0, 256, (600, 600, 3))
image[0,0]
```

Out[217]:
```
array([186,  54, 226])
```

In [218]:

```
plt.imshow(image)
plt.colorbar()
plt.show()
```



In [ ]:

```
```

In [219]:

```
import requests
```

In [220]:

```
url = 'http://www.bu.edu/csmet/files/2015/04/temkin2.jpg'
temkin = plt.imread(requests.get(url, stream=True).raw, format='jpeg')
```

In [221]:

```
temkin.shape
```

Out[221]:

```
(267, 200, 3)
```

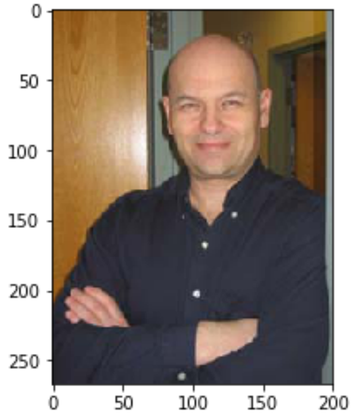
In [222]:

```
temkin[0,0]
```

Out[222]:

```
array([145,  90,  26], dtype=uint8)
```

```
In [223]: plt.imshow(temkin);
```

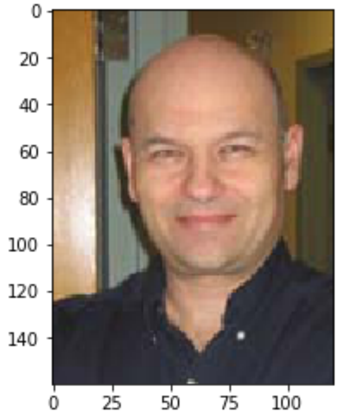


```
In [224]: # crop
```

```
In [225]: crop = temkin[7:-100, 50:-30].copy()
          crop.shape
```

```
Out[225]: (160, 120, 3)
```

```
In [226]: plt.imshow(crop);
```



```
In [227]: centerX, centerY = 75, 65
```

In [228]:
```
y, x = np.ogrid[0:160, 0:120]
```

In [229]:
```
y.shape, x.shape
```

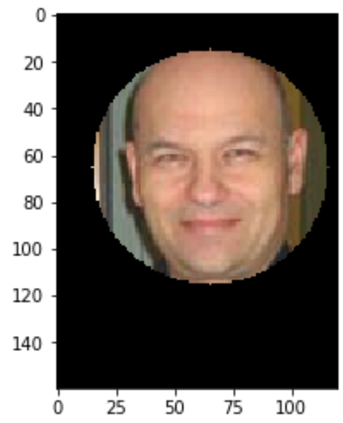
Out[229]:
```
((160, 1), (1, 120))
```

In [230]:
```
mask = ((y - centerY)**2 + (x - centerY)**2) > 2500
mask.shape
```

Out[230]:
```
(160, 120)
```

In [231]:
```
crop2 = crop.copy()
crop2[mask] = 0
```

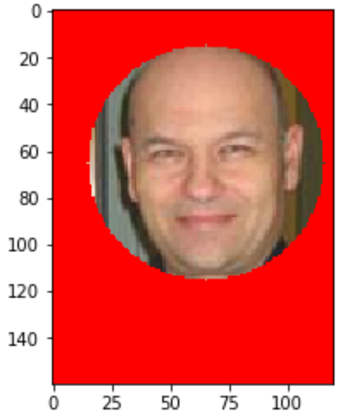
In [232]:
```
plt.imshow(crop2);
```



In [233]:
```
crop2[mask] = [255, 0, 0]
```

In [234]:

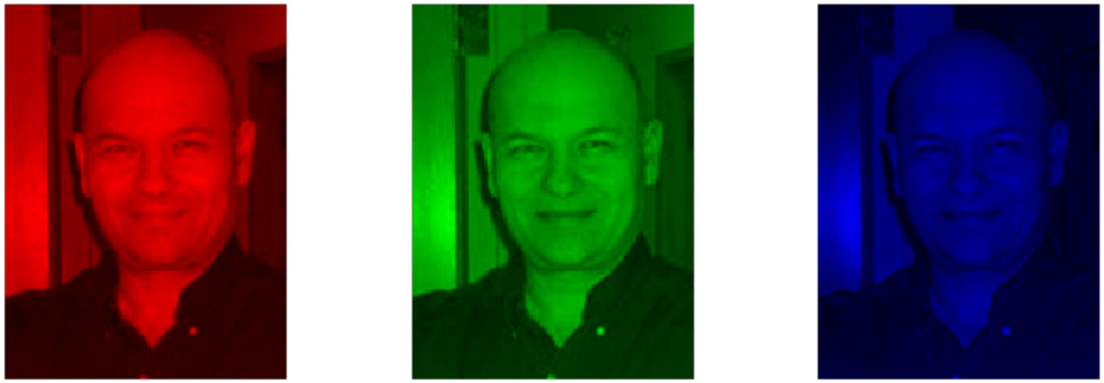
```
plt.imshow(crop2);
```



R, G, B separated images

In [235]:

```
fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(12,4))

for c, ax in zip(range(3), axs):
    new_image = np.zeros(crop.shape, dtype="uint8")
    new_image[:,:,c] = crop[:,:,c]
    ax.imshow(new_image)
    ax.set_axis_off()
```



Grayscale

```
In [236]: weights = np.array([0.299, 0.587, 0.114])
          weights
Out[236]: array([0.299, 0.587, 0.114])
```

```
In [237]: weights = np.array([0.299, 0.587, 0.114]).reshape(1,3)
          weights
Out[237]: array([[0.299, 0.587, 0.114]])
```

```
In [238]: crop.shape
Out[238]: (160, 120, 3)
```

```
In [239]: crop[0,0]
Out[239]: array([151, 100,  35], dtype=uint8)
```

```
In [240]: tile = np.tile(weights, reps=(crop.shape[0],crop.shape[1],1))
          tile.shape
Out[240]: (160, 120, 3)
```

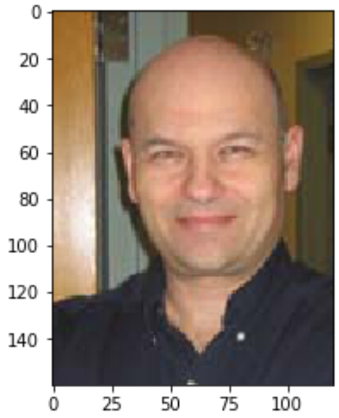
```
In [241]: tile[0,0]
Out[241]: array([0.299, 0.587, 0.114])
```

```
In [242]: tile[-1,-1]
Out[242]: array([0.299, 0.587, 0.114])
```

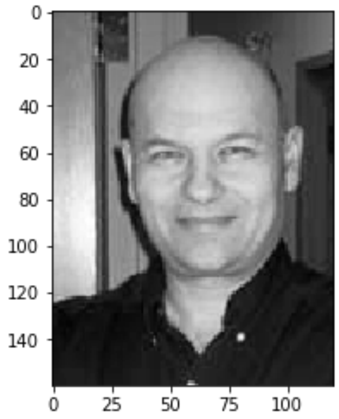
```
In [243]: gray_image = np.sum(tile * crop, axis=2)
          gray_image = gray_image.astype(int)
          gray_image.shape
Out[243]: (160, 120)
```

```
In [244]: gray_image[0,0]
Out[244]: 107
```

```
In [245]: plt.imshow(crop);
```



```
In [246]: plt.imshow(gray_image, cmap="Greys_r");
```



```
In [ ]:

In [ ]:

In [ ]:
```

Broadcasting

```
In [247]: x = np.array([1,2,3])
          x

Out[247]: array([1, 2, 3])
```

```
In [248]: y = x[:, np.newaxis]
          y

Out[248]: array([[1],
                 [2],
                 [3]])
```

```
In [249]: x+y

Out[249]: array([[2, 3, 4],
                 [3, 4, 5],
                 [4, 5, 6]])
```

```
In [ ]:
```

```
In [ ]:
```

Case Study - Distances

```
In [250]: # Distances from origin to points on a n x n grid
```

```
In [251]: x, y = np.arange(6), np.arange(6)
          x, y

Out[251]: (array([0, 1, 2, 3, 4, 5]), array([0, 1, 2, 3, 4, 5]))
```

```
In [252]: x**2

Out[252]: array([ 0,  1,  4,  9, 16, 25])
```

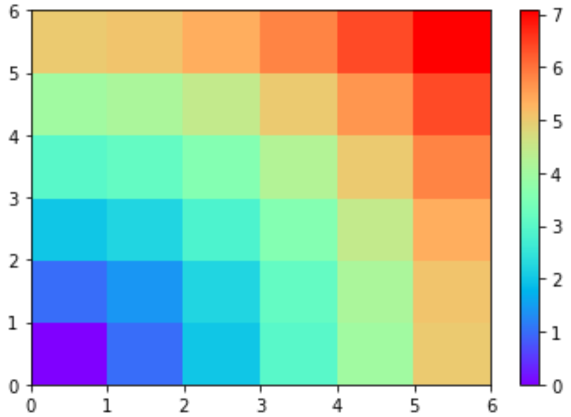
```
In [253]: y[:, np.newaxis]

Out[253]: array([[0],
                 [1],
                 [2],
                 [3],
                 [4],
                 [5]])
```

```
In [254]: y[:, np.newaxis]**2

Out[254]: array([[ 0],
                 [ 1],
                 [ 4],
                 [ 9],
                 [16],
                 [25]])
```

```
In [255]: square_distances = x**2 + y[:, np.newaxis]**2
          square_distances

Out[255]: array([[ 0,  1,  4,  9, 16, 25],
                 [ 1,  2,  5, 10, 17, 26],
                 [ 4,  5,  8, 13, 20, 29],
                 [ 9, 10, 13, 18, 25, 34],
                 [16, 17, 20, 25, 32, 41],
                 [25, 26, 29, 34, 41, 50]])
```

```
In [256]: distances = np.sqrt(square_distances)
          distances

Out[256]: array([[0.    , 1.    , 2.    , 3.    , 4.    , 5.    ],
                 [1.    , 1.4142, 2.2361, 3.1623, 4.1231, 5.099 ],
                 [2.    , 2.2361, 2.8284, 3.6056, 4.4721, 5.3852],
                 [3.    , 3.1623, 3.6056, 4.2426, 5.    , 5.831 ],
                 [4.    , 4.1231, 4.4721, 5.    , 5.6569, 6.4031],
                 [5.    , 5.099 , 5.3852, 5.831 , 6.4031, 7.0711]])
```

In [257]:
```
plt.pcolor(distances, cmap='rainbow')
plt.colorbar();
```



In [258]:
```
# Alternatively using np.ogrid
```

In [259]:
```
x, y = np.ogrid[0:6, 0:6]
```

In [260]:
```
x
```

Out[260]:
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5]])
```

In [261]:
```
y
```

Out[261]:
```
array([[0, 1, 2, 3, 4, 5]])
```

In [262]:
```
x.shape, y.shape
```

Out[262]:
```
((6, 1), (1, 6))
```

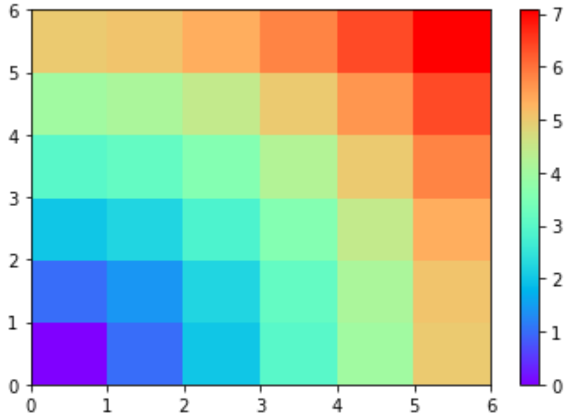
```
In [263]: x**2 + y**2
```

```
Out[263]: array([[ 0,  1,  4,  9, 16, 25],
                 [ 1,  2,  5, 10, 17, 26],
                 [ 4,  5,  8, 13, 20, 29],
                 [ 9, 10, 13, 18, 25, 34],
                 [16, 17, 20, 25, 32, 41],
                 [25, 26, 29, 34, 41, 50]])
```

```
In [264]: distances = np.sqrt(x**2 + y**2)
          distances
```

```
Out[264]: array([[0.    , 1.    , 2.    , 3.    , 4.    , 5.    ],
                 [1.    , 1.4142, 2.2361, 3.1623, 4.1231, 5.099 ],
                 [2.    , 2.2361, 2.8284, 3.6056, 4.4721, 5.3852],
                 [3.    , 3.1623, 3.6056, 4.2426, 5.    , 5.831 ],
                 [4.    , 4.1231, 4.4721, 5.    , 5.6569, 6.4031],
                 [5.    , 5.099 , 5.3852, 5.831 , 6.4031, 7.0711]])
```

```
In [265]: plt.pcolor(distances, cmap='rainbow')
          plt.colorbar();
```



```
In [ ]:
```

Convolution

```
In [266]: x = [1,2,3,4,5]
          y = [5,6,7]

          np.convolve(x, y)
Out[266]: array([ 5, 16, 34, 52, 70, 58, 35])
```

```
In [267]: x = np.array([1,2,3,4,5])
          y = np.array([5,6,7])

          np.convolve(x, y)
Out[267]: array([ 5, 16, 34, 52, 70, 58, 35])
```

```
In [268]: x1 = np.array([[0,0,1],[0,1,2],[1,2,3],[2,3,4],[3,4,5],[4,5,0],[5,0,0]])
          x1
Out[268]: array([[0, 0, 1],
                 [0, 1, 2],
                 [1, 2, 3],
                 [2, 3, 4],
                 [3, 4, 5],
                 [4, 5, 0],
                 [5, 0, 0]])
```

```
In [269]: y1 = y[::-1]
          y1
Out[269]: array([7, 6, 5])
```

```
In [270]: x1.dot(y1)
Out[270]: array([ 5, 16, 34, 52, 70, 58, 35])
```

```
In [271]: y = [5,6,7,8]
          x = [1,2,3,4,5]
```

```
In [272]: x = [0]*(len(y) - 1) + [1,2,3,4,5] + [0]* (len(y) - 1)
          x
Out[272]: [0, 0, 0, 1, 2, 3, 4, 5, 0, 0, 0]
```

```
In [273]: x1 = [x[i:i+len(y)] for i in range(len(x) - len(y) + 1)]
          x1
```

```
Out[273]: [[0, 0, 0, 1],
           [0, 0, 1, 2],
           [0, 1, 2, 3],
           [1, 2, 3, 4],
           [2, 3, 4, 5],
           [3, 4, 5, 0],
           [4, 5, 0, 0],
           [5, 0, 0, 0]]
```

```
In [274]: x1 = np.array(x1)
          x1
```

```
Out[274]: array([[0, 0, 0, 1],
                 [0, 0, 1, 2],
                 [0, 1, 2, 3],
                 [1, 2, 3, 4],
                 [2, 3, 4, 5],
                 [3, 4, 5, 0],
                 [4, 5, 0, 0],
                 [5, 0, 0, 0]])
```

```
In [275]: y1 = np.array(y[::-1])
          y1
```

```
Out[275]: array([8, 7, 6, 5])
```

```
In [276]: x1.dot(y1)
```

```
Out[276]: array([ 5, 16, 34, 60, 86, 82, 67, 40])
```

```
In [ ]:
```