Intro to NumPy, SciPy, and Matplotlib

Numpy

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

Comparison of Numpy Array and List

In [2]:
```
x = [2, 3, 4, 6] # python lsit
y = np.array(x) # numpy array
```

In [4]:
```
print(type(x), x)
print(type(y), y)
```

```
<class 'list'> [2, 3, 4, 6]
<class 'numpy.ndarray'> [2 3 4 6]
```

In [5]:
```
print(x[1:3])
```

```
[3, 4]
```

In [6]:
```
print(y[1:3])
```

```
[3 4]
```

In [7]:
```
print(x[0,2])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-39192171db81> in <module>
----> 1 print(x[0,2])

TypeError: list indices must be integers or slices, not tuple
```

In [10]:
```
print(y[[0, 2]])
```

```
[2 4]
```

In [11]:
```
print(y[y>3])
```

```
[4 6]
```

In [12]:
```
print(x * 5)
```

```
[2, 3, 4, 6, 2, 3, 4, 6, 2, 3, 4, 6, 2, 3, 4, 6, 2, 3, 4, 6]
```

In [14]:
```
print(y * 5)
```

```
[10 15 20 30]
```

In [15]:
```
print(x ** 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-2d3f09e96d2e> in <module>
----> 1 print(x ** 2)

TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

In [16]:
```
print(y ** 2)
```

```
[ 4  9 16 36]
```

In [17]:
```
matrix = [[1,2,4],[3,1,0]]
nd_array = np.array(matrix)
```

In [18]:
```
print(matrix[1][2])
```

```
0
```

In [19]:
```
print(nd_array[1,2])
```

```
0
```

In [20]:
```
print(np.random.rand())
```

```
0.265135890249639
```

In [23]:
```
print(np.random.randn())
```

```
2.1814425202615237
```

In [24]:
```
print(np.random.randn(4))
```

```
[-1.5899826   0.17418988  1.29172966  0.30406861]
```

In [25]:
```
print(np.random.randn(4, 5))
```

```
[[-0.91479045  0.25611691 -0.75125591 -1.11094856 -0.69541786]
 [-0.04597834  0.00584917  0.65712128 -0.80405922  0.34694974]
 [-0.44462258  0.39503849  1.57422095  1.77275988  0.15955268]
 [-0.82786049 -1.20678598 -0.69452704  0.77085027 -1.22260465]]
```

range() Method in Python and arange() in Numpy

Numpy is great for vectors and matrices

In [28]:
```
print(np.arange(0, 8 ,0.1)) #floating step works in numpy
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5
 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3
 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.  7.1
 7.2 7.3 7.4 7.5 7.6 7.7 7.8 7.9]
```

In [33]:
```
print(range(0, 8 ,0.1)) # floating step doesn't work in python
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-ef20aafa9f7c> in <module>
----> 1 print(range(0, 8 ,0.1)) # floating step doesn't work in python

TypeError: 'float' object cannot be interpreted as an integer
```

In [32]:
```
%timeit np.arange(0, 10000) # tests time to create function and convert
%timeit range(0, 10000) #python is usually slower, not in this case though
```

```
4.76 µs ± 110 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
198 ns ± 1.95 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

SciPy

great for scientific models

In [41]:
```
from scipy import optimize
```

In [49]:
```
def f(x):
    return (x[0] - 3.2) ** 2 + (x[1] - 0.1) **2 + 3 #can be done algebraically

print(f([3.2, 0.1]))
```

```
3.0
```

In [51]:
```
x_min = optimize.minimize(f, [5, 5]) # finds min using numerical methods
print(x_min)
```

```
      fun: 3.0000000000011435
 hess_inv: array([[ 0.94055055, -0.16183475],
       [-0.16183475,  0.55494947]])
      jac: array([-2.056360224e-06,  5.36441803e-07])
  message: 'Optimization terminated successfully.'
     nfev: 12
      nit: 3
     njev: 4
   status: 0
  success: True
        x: array([3.19999896, 0.10000026])
```

In [39]:
```
print(x_min.x)
```

```
[-6.24615879e+101  5.58845184e+050]
```

In [40]:
```
from scipy import linalg
```

In [46]:
```
a = np.array([[3 ,2 , 0], [1, -1, 0], [0 , 5, 1]])
b = np.array([2, 4, -1])

x = linalg.solve(a, b)
print(x)
```

```
[ 2. -2.  9.]
```

In [50]:
```
print(np.dot(a, x)) #dot product
```

```
[ 2.  4. -1.]
```

In [53]:
```
X = np.random.randn(4, 3) #dot product of vector
U, D, V = linalg.svd(X)
print(U.shape, D.shape, V.shape)
print(type(U), type(D), type(V))
```

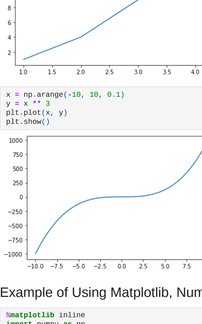
```
(4, 4) (3,) (3, 3)
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Matplotlib

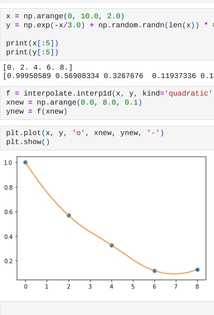
In [54]:
```
%matplotlib inline
```

In [55]:
```
from matplotlib import pylab as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



In [58]:
```
x = np.arange(-10, 10, 0.1)
y = x ** 3
plt.plot(x, y)
plt.show()
```



Example of Using Matplotlib, Numpy, and SciPy Together

In [61]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
```

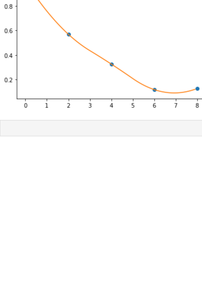
In [62]:
```
x = np.arange(0, 10.0, 2.0)
y = np.exp(-x/3.0) + np.random.randn(len(x)) * 0.05

print(x[:5])
print(y[:5])
```

```
[0. 2. 4. 6. 8.]
[0.99950589 0.56908334 0.3267676  0.11937336 0.12783699]
```

In [66]:
```
f = interpolate.interp1d(x, y, kind='quadratic')
xnew = np.arange(0.0, 8.0, 0.1)
ynew = f(xnew)
```

In [67]:
```
plt.plot(x, y, 'o', xnew, ynew, '-')
plt.show()
```



In [ ]: