Python tutorial day 3

Finish going over python fundamentals then go into matrix representations of networks

Agenda:

- go over solutions to Friday's worksheet
- Finsh python fundamentals
- Start looking at how networks can be represented in python

Useful python functions

In [3]:
```
# length function

# Recall lists
L = [1, 2, "Cindy", 0.4]
print(len(L))

L2 = [1, 2, "Cindy", 0.4, 10, "m"]
print(len(L2))

L3 = []
print(len(L3))
```

```
4
6
0
```

In [5]:
```
# type function - tells you the type of data of a variable

a = 1
print(type(a))

b = "Dominic"
print(type(b))
```

```
<class 'int'>
<class 'str'>
```

Now let's look at some useful functions in the numpy library

numpy library - python library that allows you to efficiently work with matrices

any matrix is an array

In [8]:
```
#import the numpy library and call it "np" form here onwards
import numpy as np
```

In [18]:
```
# size and shape

A = np.array([[1,0, 4], [0, 1, 2]])
print(A)
print(type(A))
print("The size of matrix A is",np.size(A))
print("The shape of matrix A is", np.shape(A))
print("The number of rows in matrix A is", np.shape(A)[0])
print("The number of columns in matrix A is", np.shape(A)[1])
```

```
[[1 0 4]
 [0 1 2]]
<class 'numpy.ndarray'>
The size of matrix A is 6
The shape of matrix A is (2, 3)
The number of rows in matrix A is 2
The number of columns in matrix A is 3
```

In [26]:
```
# zeros - useful for when you need to create an array that you will put numbers into

B = np.zeros((2,3))
print(B)

A = np.array([[1,0, 4], [0, 1, 2]])
# code that will output a matrix of all zeros the same shape as A
C = np.zeros(np.shape(A))
print(C)

# use len on lists and np.shape and np.size on numpy arrays
```

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

In [35]:
```
# arange and linspace
# arange returns evenly spaced numbers within a given interval - good for creating integers

a = np.arange(5)    # input: stoping integer
print(a)

b = np.arange(1, 5, 1)     # inputs: start, stop, step
print(b)

c = np.arange(1, 5, 2)
print(c)


# linspace returns evenly spaced numbers within a given interval - good for creating decimal numbers

d = np.linspace(1, 5, 3)     # inputs: start, stop, number of samples
print(d)
```

```
[0 1 2 3 4]
[1 2 3 4]
[1 3]
[1. 3. 5.]
```

There are many other useful numpy functions and you can check them out at the numpy documentation:
https://numpy.org/doc/stable/user/absolute_beginners.html

In [42]:
```
# pseudocode - informal way of writing code in a way that a human can understand. It's a useful tool
# for solving difficult problems or coding difficult algorithms. One of the best approaches
# to start implementing an algorithm.

# For example, if you get a question like "write a function that squares each number in a list".

# function def
#       for each number in the list
#           square the number

def square_list(L):
    for i in range(len(L)):
        L[i] = L[i] * L[i]
    return L


L = [10, 9, 5, 0 , -0.3, -2]
squareL = square_list(L)
#print(squareL)


# Try writing the pseudocode for "Write a function that squares each number in a numpy array"

# pseudocode iteration 1
# function def
#     for each element in the array
#         square the element

# pseudocode iteration 2
# function def
#     for each row in the array
#         for each element in row
#             square the element

A = np.array([[1, 4, 5, -2],[0.1, 0, 0, 2]])

def mat_square(A):
    for i in range(np.shape(A)[0]):
        for j in range(np.shape(A)[1]):
            A[i][j] = A[i][j] * A[i][j]
    return A

Asquare = mat_square(A)
print(Asquare)
```

```
[[1.0e+00 1.6e+01 2.5e+01 4.0e+00]
 [1.0e-02 0.0e+00 0.0e+00 4.0e+00]]
```

In [50]:
```
# Now we can try to tackle the solution to problem 3 from this weekend's worksheet.
# Start writing the pseudocode for that problem.

# iteration 1

# create A and B
# Create empty matrix to store C
# fill in the number of the array

# iteration 2

# create A and B
# create a matrix of all zeros called C which same
#   size as A and B
# for each row in A
#     for each element in row
#         set element of C equal to the corresponding
#         numbers of A and B multiplied together

A = ([[1, -2, 0], [2, 3, -5]])

B = ([[0, -3, 4], [1, 2, 3]])

C = np.zeros((2, 3))

for i in range(np.shape(C)[0]):
    for j in range(np.shape(C)[1]):
        C[i][j] = A[i][j] * B[i][j]
print(C)
```

```
[[  0.   6.   0.]
 [  2.  -6. -15.]]
```

In [51]:
```
# before we can do problem 4, look at how to compute a dot product

a = [1, 4, 0, -1]
b = [0, -2, 3, 1]
# dot product should be: 1*0 + 4*-2 + 0*3 + -1*1 = -9

#pseudocode
# initialize sum variable to zero
# for each pair of numbers in the same position in a and b
#     add multiplication of those numbers to sum variable

s = 0
for i in range(len(a)):
    s = s + a[i]*b[i]

print(s)
```

```
-9
```

In [ ]:
```
# problem 4 pseudocode solution - Let's do the pseudocode together then you should go home and
# code it up yourself.

# pseudocode iteration 1

# function def (input1, input2)
#     determine if matrix-matrix product is possible
#     if possible, compute
#     else, print "not compatible"

# pseudocode iteration 2

# function def (input1, input2)
#     if number of columns of input1 = number of rows of input2
#         compute matrix-matrix product
#     else
#         print "not compatible"


# pseudocode iteration 3

# function def (input1, input2)
#     if number of columns of input1 = number of rows of input2
#         create matrix of zeros of correct size, call it out
#         fill in entries of out
#     else
#         print "not compatible"


# pseudocode iteration 4

# function def (input1, input2)
#     if number of columns of input1 = number of rows of input2
#         create matrix of zeros of correct size, call it out
#         for each row of out
#             for each entry in row
#                 compute the value that goes in entry
#                 put value into entry of out - dot product
#     else
#         print "not compatible"
```

In [69]:
```
# problem 5

# solution 1 - adjacency matrix
A = np.array([[0, 3, 0, 7, 0, 15], [3,0,1,0,0,9],[0,1,0,2,0,2],[7,0,2,0,4,10],
    [0,0,0,4,0,3],[15,9,2,10,3,0]])
#print(A)

flightNet = {
    0: "Seattle",
    1: "San Francisco",
    2: "Los Angeles",
    3: "Houston",
    4: "Miami",
    5: "New York City"
}

#print(flightNet[0])


# solution 2 - adjacency list
flightNet2 = {
    "Seattle": [["San Francisco", 3], ["Houston", 7], ["New York City", 15]],
    "San Francisco": [["Seattle",3], ["Los Angeles",1], ["New York City",9]],
    "Los Angeles": [["San Francisco",1], ["New York City",2], ["Houston",2]],
    "Houston": [["Los Angeles",2], ["Seattle",7], ["New York City", 10], ["Miami",4]],
    "Miami": [["Houston",4], ["New York City",3]],
    "New York City": [["Seattle",15], ["San Francisco",9], ["Los Angeles",2], ["Houston",10], ["Miami",3]]
}

# how to access all of the keys in a dictionary
#for item in flightNet2:
#    print(item)

# how to access the values for a specific key
#print(flightNet2["San Francisco"])

#try printing out all of the cities that Miami is connected to

a = flightNet2["Miami"]
print(a)
for x in a:
    print(x[0])

# Try to figure out how to loop through all of the edge and that should make problem 2 easier
```

```
[['Houston', 4], ['New York City', 3]]
Houston
New York City
```

In [ ]:
```

```