Calculating Distance Lab

Introduction

In this lab, you will write methods to calculate the distance of various neighbors from each other. Once again, let's assume that the x coordinates represent avenues of a neighbor, the y coordinates represent streets. We will also assume that the distance between each street and the distance between each avenue is the same.

We will work up to a function called `nearest_neighbors` that given a neighbor, finds the other neighbors who are closest.

Getting Started

Let's declare a variable `neighbors` and assign it to a list of dictionaries, each representing the location of a neighbor.

In [1]:
```
neighbors = [{'name': 'Fred', 'avenue': 4, 'street': 8}, {'name': 'Suzie',
             {'name': 'Bob', 'avenue': 5, 'street': 8}, {'name': 'Edgar', '
             {'name': 'Steven', 'avenue': 3, 'street': 6}, {'name': 'Natali
```

Press shift + enter to run the code in the gray boxes.

In [2]:
```
neighbors
```

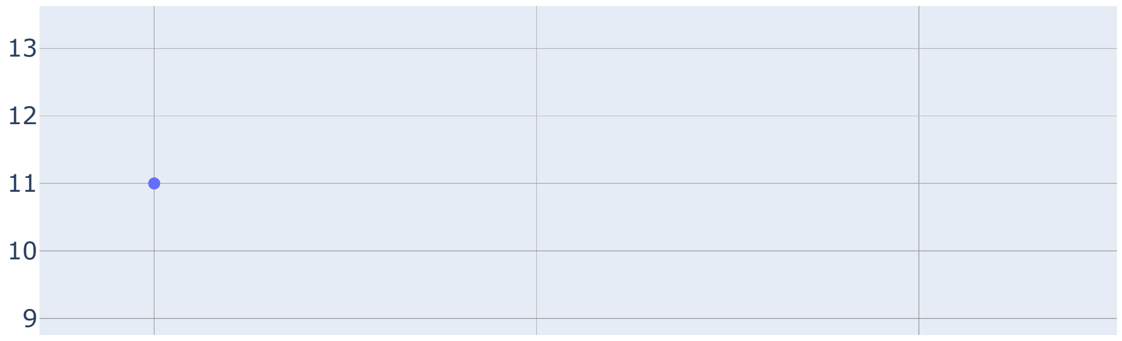
Out[2]:
```
[{'name': 'Fred', 'avenue': 4, 'street': 8},
 {'name': 'Suzie', 'avenue': 1, 'street': 11},
 {'name': 'Bob', 'avenue': 5, 'street': 8},
 {'name': 'Edgar', 'avenue': 6, 'street': 13},
 {'name': 'Steven', 'avenue': 3, 'street': 6},
 {'name': 'Natalie', 'avenue': 5, 'street': 4}]
```

In [3]:
```
fred = neighbors[0]
natalie = neighbors[5]
```

We'll also plot our neighbors, to get a sense of our data.

In [4]:
```
import plotly

plotly.offline.init_notebook_mode(connected=True)
trace0 = dict(x=list(map(lambda xpar: xpar['avenue'],neighbors)),
              y=list(map(lambda ypar: ypar['street'],neighbors)),
              text=list(map(lambda txt: txt['name'],neighbors)),
              mode='markers')
plotly.offline.iplot(dict(data=[trace0], layout={'xaxis': {'dtick': 1}, 'ya
```



We'll start by focusing on the neighbors Fred and Natalie, and points (4, 8) and (5, 4) respectively.

Calculating the sides of the triangle

Remember that to calculate the distance, we draw a diagonal line between the two points, form a right triangle around the diagonal line, and then use the Pythagorean Theorem to calculate the hypotenuse of the triangle, that is the distance. Let's start with imagining we formed a right triangle around the two points and now can move onto calculating the legs of our right triangle.

Write a function called `street_distance` that calculates how far **in streets** two neighbors are from each other. So for example, with Natalie at street 4, and Fred at street 8, our `street_distance` function should return the number 4.

In [5]:
```
def street_distance(first_neighbor, second_neighbor):
    if (first_neighbor['street'] > second_neighbor['street']):
        return first_neighbor['street'] - second_neighbor['street']
    else:
        return second_neighbor['street'] - first_neighbor['street']
    pass
```

Now execute the code below. As you can see from the comment to the right, the expected returned street distance is 4.

In [6]:
```
street_distance(fred, natalie) # 4
```

Out[6]:
```
4
```

Write a function called `avenue_distance` that calculates how far in avenues two neighbors are from each other. The distance should always be positive.

In [7]:
```
def avenue_distance(first_neighbor, second_neighbor):
    if (first_neighbor['avenue'] > second_neighbor['avenue']):
        return first_neighbor['avenue'] - second_neighbor['avenue']
    else:
        return second_neighbor['avenue'] - first_neighbor['avenue']
    pass
```

In [8]:
```
avenue_distance(fred, natalie) #  1
```

Out[8]:
```
1
```

Calculating the distance

Now let's begin writing functions involved with calculating that hypotenuse of our right triangle. Using the Pythagorean Theorem, $a^2 + b^2 = c^2$, write a function called `distance_between_neighbors_squared` that calculates c^2, the length of the hypotenuse squared.

In [9]:
```
def distance_between_neighbors_squared(first_neighbor, second_neighbor):
    return pow(street_distance(first_neighbor, second_neighbor),2)+pow(aven
    pass
```

In [10]:
```
distance_between_neighbors_squared(fred, natalie) # 17
```

Out[10]:
```
17
```

Now let's move onto the next step and write a function called `distance` , that given two neighbors returns the distance between them.

> You may have to Google some math to do this.

In [11]:
```
import math
def distance(first_neighbor, second_neighbor):
    return pow(distance_between_neighbors_squared(first_neighbor, second_ne
    pass
```

In [12]:
```
distance(fred, natalie) # 4.123105625617661
```

Out[12]:
```
4.123105625617661
```

Writing Our "Nearest Neighbors" Functions

This next section will work up to building a `nearest_neighbor` function. This is a function that given one neighbor, will tell us which neighbors are closest. How do we write something like this? Can we use our calculation of distance between two neighbors to figure out the closest neighbors to an individual?

Sure, we first need to calculate the distances between one neighbor and then all of the others. Next, we sort those neighbors by their distance from the selected_neighbor. Finally, we select a given number of the closest neighbors. Let's work through it.

Note that we already have a function that calculates the distance between two neighbors. We may think we could simply use this function to loop through our neighbors, but that would just return a list of distances.

In [13]:
```
distances = []
for neighbor in neighbors:
    distance_between = distance(fred, neighbor)
    distances.append(distance_between)

distances
```

Out[13]:
```
[0.0,
 4.242640687119285,
 1.0,
 5.385164807134504,
 2.23606797749979,
 4.123105625617661]
```

The returned list from the above procedure isn't super helpful. We need to know the person associated with each distance.

So let's accomplish this by writing a function called `distance_with_neighbor` that works like our distance function but instead of returning a float, returns a dictionary representing the `second_neighbor` , and also adds in the a key value pair indicating distance from the `first_neighbor` .

In [14]:
```
port math
f distance_with_neighbor(first_neighbor, second_neighbor):
  X = {}
  #X = dict()
  if (str(first_neighbor['name']) != str(second_neighbor['name'])):
      distance_between = distance(first_neighbor, second_neighbor)
      X.update( {'avenue':second_neighbor['avenue'],'distance':distance_betw
      #X.update([ ('avenue',second_neighbor['avenue']),('distance',distance_

  #list(filter(None,X))
  #X = { k:v for k,v in X.items() if v != None }
  #X = dict( [ (k,v) for k,v in X.items() if v != None ] )
  #empty_keys = [k for k,v in X.items() if not v]
  #for k in empty_keys:
  #del X[k]
  return X
  pass
```

In [15]:
```
distance_with_neighbor(fred, natalie)
# {'avenue': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street':
```

Out[15]:
```
{'avenue': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street':
4}
```

Now write a function called `distance_all` that returns a list representing the distances between a `first_neighbor` and the rest of the neighbors. The list should not return the `first_neighbor` in its collection of neighbors.

In [16]:
```
def distance_all(first_neighbor, neighbors):
    return list(map(lambda neighbor: distance_with_neighbor(first_neighbor,
    pass
```

In [17]:
```
X = distance_all(fred, neighbors)
X = list(filter(None,X))
print(list(X))
# [{'avenue': 1, 'distance': 4.242640687119285, 'name': 'Suzie', 'street':
#  {'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8},
#  {'avenue': 6, 'distance': 5.385164807134504, 'name': 'Edgar', 'street':
#  {'avenue': 3, 'distance': 2.23606797749979, 'name': 'Steven', 'street':
#  {'avenue': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street'
```

```
[{'avenue': 1, 'distance': 4.242640687119285, 'name': 'Suzie', 'street':
11}, {'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8}, {'avenu
e': 6, 'distance': 5.385164807134504, 'name': 'Edgar', 'street': 13}, {'a
venue': 3, 'distance': 2.23606797749979, 'name': 'Steven', 'street': 6},
{'avenue': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street':
4}]
```

Finally, write a function called `nearest_neighbors` that given a neighbor, returns a list of neighbors, ordered from closest to furthest from the neighbor. The function should take an optional third argument that specifies how many "nearest" neighbors are returned.

In [18]:
```
X = {}
Y = {}
Z = {}
def nearest_neighbors(first_neighbor, neighbors, number = None):
    X = distance_all(first_neighbor, neighbors)
    X = list(filter(None,X))
    Y = sorted(X,key = lambda i:i['distance'])
    print(list(Y))
    Z = Y[0:number]
    return Z
    pass
nearest_neighbors(fred, neighbors, 2)
```

```
[{'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8}, {'avenue':
3, 'distance': 2.23606797749979, 'name': 'Steven', 'street': 6}, {'avenu
e': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street': 4},
{'avenue': 1, 'distance': 4.242640687119285, 'name': 'Suzie', 'street': 1
1}, {'avenue': 6, 'distance': 5.385164807134504, 'name': 'Edgar', 'stree
t': 13}]
```

Out[18]:
```
[{'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8},
 {'avenue': 3, 'distance': 2.23606797749979, 'name': 'Steven', 'street':
6}]
```

In [19]:
```
nearest_neighbors(fred, neighbors, 2)
# [{'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8},
#  {'avenue': 3, 'distance': 2.23606797749979, 'name': 'Steven', 'street':
```

```
[{'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8}, {'avenue':
3, 'distance': 2.23606797749979, 'name': 'Steven', 'street': 6}, {'avenu
e': 5, 'distance': 4.123105625617661, 'name': 'Natalie', 'street': 4},
{'avenue': 1, 'distance': 4.242640687119285, 'name': 'Suzie', 'street': 1
1}, {'avenue': 6, 'distance': 5.385164807134504, 'name': 'Edgar', 'stree
t': 13}]
```

Out[19]:
```
[{'avenue': 5, 'distance': 1.0, 'name': 'Bob', 'street': 8},
 {'avenue': 3, 'distance': 2.23606797749979, 'name': 'Steven', 'street':
6}]
```

Summary

In this lab, you built out the nearest neighbors. We'll review building out these functions in the next section.