Data mining and knowledge representation in the city: a practical approach.
Foursquare hosts a rich database of venues: perhaps the largest ever available under a single service umbrella.

More than 50 million entries and the set is ever growing.

Seed database of points-of-interest was used to kick start, but data was augmented through crowdsourcing, thanks to the millions of Foursquare users around the globe.
As you have seen previously in the course, the API provides a multi-dimensional information view of a venue.

Foursquare is effectively a venue information crowdsourcing machine. There are hundreds of attributes describing a venue in Foursquare.

This is just the beginning of the mobile/physical web. It may take years, but we will get to a point where everything in the physical space is mapped digitally.
Given a set of Foursquare venues, how can we begin exploiting them in a research/application context?
our targets ...
We will learn ..

To measure the geographic distance between pairs of venues.

To infer the geographic center of a city.

To get all places within a radius given a pair of geographic coordinates.

Move across the hierarchy of venue categories in Foursquare.

Apply a grid over a city.

amongst others :)
Exercise:

Load module with bin_data method and run it on the distances.txt file

What does the output look like?
bin_data*

limits the range of the data number of data points (bins)

```python
def bin_data(data, min_value=10**-2, max_value=10**2, STEPS=50):
    """Filters the values contained in data, removing all values smaller than min_value and all values larger than max_value."
    Returns.
```

Did you see this output?

What sort of distribution is this? Not power-law? We will answer this tomorrow.

* which is probably a very bad name :(
Introduction to the files contained in the spatial retrieval directory ...
This is experimental code - not the sort of real world validated software. Use at your own risk and report problems please :)

WARNING
class locationUtility:

    def __init__(self):
        self.allLatsDict = {}
        self.allLongsDict = {}
        self.allLats = []
        self.allLongs = []
        self.locationData = {}
        self.cats = catFile.Categories()  # initialization of the 'foursquare place categories' object.

    def load_locality_location_data(self, categoryHierarchy=None):

        # key is latitude (resp. longitude, value is location identifier
        allLocalityLatsDict = {}
        allLocalityLongsDict = {}

        allLocalityLats = []
        allLocalityLongs = []
        localityLocationData = {}

        for l in fileinput.input(['./brussels_locations/brussels_locationData.txt']):
            dataSplits = l.split('*;*')
            locationLink = dataSplits[0]
            (latitude, longitude, category, totalPeople, totalCheckins, title) = eval(dataSplits[1])

            ### append values to global location variables
            self.allLatsDict = allLocalityLatsDict
            self.allLongsDict = allLocalityLongsDict
            self.allLats = allLocalityLats
            self.allLongs = allLocalityLongs
            self.locationData = localityLocationData

        print 'Number of places loaded in Brussels : %d' % (len(self.allLats))


def get_nearby_categories(self, latitude, longitude, distance, area_shape='square'):
    # first get all locations
    location_list = self.get_nearby_locations(latitude, longitude, distance, area_shape)

    nearby_category_frequency = {}
    # then for every location, get category
    for loc in location_list:
        category = self.locationData[loc][2]
        nearby_category_frequency.setdefault(category, 0)
        nearby_category_frequency[category] += 1

    return nearby_category_frequency


def get_nearby_locations(self, latitude, longitude, distance, area_shape='square'):
    # returns nearby locations given a point

    if area_shape == 'square':
        return location_list_insquare

    elif area_shape == 'circle':
        location_list_incircle = []
        for location in location_list_insquare:
            if geolocator.getDistance([location[0:2], self.locationData[location][0:2]]) <= distance:
                location_list_incircle.append(location)
        return location_list_incircle
def area_recursive_splitter_v2(self, latitude, longitude, distance, numberOfSquares):
    # split an area in squares of certain minimum size: number of squares better be power of 4
    ([lat_up, lat_down], [long_left, long_right]) = self.get_rectangle_coordinates(latitude, longitude, distance/2.0)
    if numberOfSquares >= 4:  # check
        newDist = distance/2.0
        newNumberOfSquares = numberOfSquares/4

        #0.86*
        pointsNW = self.area_recursive_splitter_v2(lat_up, long_left, newDist, newNumberOfSquares)
        pointsNE = self.area_recursive_splitter_v2(lat_up, long_right, newDist, newNumberOfSquares)
        pointsSE = self.area_recursive_splitter_v2(lat_down, long_right, newDist, newNumberOfSquares)
        pointsSW = self.area_recursive_splitter_v2(lat_down, long_left, newDist, newNumberOfSquares)

        return pointsSE + pointsSW + pointsNW + pointsNE
    else:
        return [(latitude, longitude)]

Given a central point \((x,y)\) and a radius (distance), return a grid centered at \((x,y)\).

The grid is represented as a list of tuples, where each tuple is the center of the grid’s cell represented with geographic coordinates.

The parameter numberOfSquares is the number of cells in the grid. This version is limited to numberOfSquares being a power of 2.
def get_rectangle_coordinates(self, latitude, longitude, distance):
    d = distance          # distance in km
    R = 6371.0            # earth radius in km

    lat1 = latitude
    long1 = longitude

    # convert to radians
    long1 = long1 * pi / 180.0
    lat1 = lat1 * pi / 180.0

---

Key method used by the previous method: afterall to get a grid of cell, you need to get a cell in the first place ...

The rest of the code is gibberish maths which i worked out one fine day during the first days of my phd ... it is the opposite process of the haversine formula for distance measurement..
Create a new file named main_geo.py and add the following code:

```python
import locationUtility as lu
import numpy
import random
import categories

#initialize location utility object: this object will allows to access venue
loc_utility = lu.locationUtility()

#load location data from file
loc_utility.load_locality_location_data()
```
Exercise:

Get the center of Brussels in terms of latitude and longitude coordinates

Hint: you will need to access the latitudes variable in the locationUtility object --> loc_utility.allLats
Exercise 2:

Get the list of places within 200 meter radius from the center.
Exercise:

Modify locationUtility file to load the dataset you have collected about Namur.
Combo-Exercise:

For all places in a Namur (or Brussels), get their popularity and then use the bin_data method to plot the probability density function.
class Categories(object):
    def __init__(self):
        self._categories = None  # self._resources = Resources()

        self._root = None
        self._name_category_map = {}
        self._short_name_category_map = {}

        self._load()
        self._load_colors()

    def _load(self):
        json_data = open('/Users/anastasiosnoulas/Desktop/new_age/localityAnalysis/categories.json')
        data = json.load(json_data)
        json_data.close()
        self._categories = data

        self._root = _Node(None, None)
        nodeQueue = deque([])
        nodeQueue.append(self._root)

        objectQueue = deque([])
        objectQueue.append(self._categories)

        while len(nodeQueue) > 0:
            node = nodeQueue.popleft()
            category = objectQueue.popleft()
            if not (node.data is None):
                self._name_category_map[category.data['name']] = node
Exercise:

Get top categories in the center of Brussels and Namur.
Exercise:

Get categories popularity in terms of check-in number and plot their probability density function.
Exercise:

Apply a 5km radius grid with 1024 cells over the center of Brussels. Then measure the number of cells that have at least 5 places.

How can we calculate the area of the city in km**2 under those terms?

How can we calculate the density of the city?
Information entropy in the city

```python
def get_area_entropy(self, latitude, longitude, radius, area_shape='square'):
    dictionaryLocFrequency = self.get_nearby_categories(latitude, longitude, radius, area_shape)

    entropy = 0.0
    log_base = 2  # 10.0

    totalLocations = 0.0
    for category, locationNumber in dictionaryLocFrequency.items():
        totalLocations = totalLocations + locationNumber

    sumProbability = 0
    for category, locationNumber in dictionaryLocFrequency.items():
        probability = locationNumber / (totalLocations * 1.0)
        entropy += probability * math.log(probability, log_base)

    entropy = - entropy

    return entropy
```

Check wikipedia’s article on information entropy
Exercise:
Get the information entropy of geographic areas in Brussels (Namur), in terms of place categories.

a) Plot the probability density of entropy.

a) For each area plot its entropy value and its most popular category. (pylab.text)

hint: use a grid centered at geographic center of the city (64 cells)
for l in open('./brussels_locations/brussels_trajectories.txt'):
    lineSplits = l.split(';;;*')
    checkin_list = eval(lineSplits[1])

for i in range(0,len(checkin_list)-1):
    place1 = checkin_list[i][0]
    place2 = checkin_list[i+1][0]

    time1 = checkin_list[i][1]
    time2 = checkin_list[i+1][1]