DataCamp Data Science Courses

Data Types for Data Science

Chap 1: Fundamental data types

In [1]:

```
# Import libraries
import pandas as pd
import numpy as np
```

Introduction and lists

In [2]:

```
cookies = ['chocolate chip', 'peanut butter', 'oatmeal', 'sugar']
```

In [3]:

```
cookies.append('Tirggel')
```

In [4]:

```
print(cookies)
```

```
['chocolate chip', 'peanut butter', 'oatmeal', 'sugar', 'Tirggel']
```

In [5]:

```
print(cookies[2])
```

```
oatmeal
```

In [6]:

```
cakes = ['strawberry', 'vanilla']
```

In [7]:

```
desserts = cookies + cakes
print(desserts)
```

```
['chocolate chip', 'peanut butter', 'oatmeal', 'sugar', 'Tirggel', 'strawber
ry', 'vanilla']
```

In [8]:

```
position = cookies.index('sugar')
```

In [9]:

```
print(position)
```

```
3
```

In [10]:

```
cookies[3]
```

Out[10]:

```
'sugar'
```

In [11]:

```
name = cookies.pop(position)
```

In [12]:

```
print(name)
```

```
sugar
```

In [13]:

```
print(cookies)
```

```
['chocolate chip', 'peanut butter', 'oatmeal', 'Tirggel']
```

In [14]:

```
for cookie in cookies:
    print(cookie)
```

```
chocolate chip
peanut butter
oatmeal
Tirggel
```

In [15]:

```
print(cookies)
sorted_cookies = sorted(cookies)
print(sorted_cookies)
```

```
['chocolate chip', 'peanut butter', 'oatmeal', 'Tirggel']
['Tirggel', 'chocolate chip', 'oatmeal', 'peanut butter']
```

In [16]:

```
# EXERCISES
```

In [17]:

```
# Manipulating lists for fun and profit
########################################

# Create a list containing the names: baby_names
baby_names = ['Ximena', 'Aliza', 'Ayden', 'Calvin']

# Extend baby_names with 'Rowen' and 'Sandeep'
baby_names.extend(['Rowen','Sandeep'])

# Print baby_names
print(baby_names)

# Find the position of 'Aliza': position
position = baby_names.index('Aliza')

# Remove 'Aliza' from baby_names
baby_names.pop(position)

# Print baby_names
print(baby_names)
```

```
['Ximena', 'Aliza', 'Ayden', 'Calvin', 'Rowen', 'Sandeep']
['Ximena', 'Ayden', 'Calvin', 'Rowen', 'Sandeep']
```

In [18]:

```
# Looping over lists
####################

records = pd.read_csv('datasets/baby_names.csv').values
```

In [19]:

```
# Create the empty list: baby_names
baby_names = []

# Loop over records
for row in records:
    # Add the name to the list
    baby_names.append(row[3])

# Sort the names in alphabetical order
for name in sorted(baby_names):
    # Print each name
    print(name)
```

```
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AALIYAH
AARAV
AARAV
AARAV
AARAV
AARAV
AARON
AARON
AARON
AARON
```

Meet the Tuples

In [20]:

```
us_cookies = ['Chocolate Chip','Brownies','Peanut Butter','Oreos','Oatmeal']
in_cookies = ['Punjabi','Fruit Cake Rusk','Marble Cookies',
              'Kaju Pista Cookies','Almond Cookies']
```

In [21]:

```
top_pairs = list(zip(us_cookies, in_cookies))
print(top_pairs)
```

```
[('Chocolate Chip', 'Punjabi'), ('Brownies', 'Fruit Cake Rusk'), ('Peanut Bu
tter', 'Marble Cookies'), ('Oreos', 'Kaju Pista Cookies'), ('Oatmeal', 'Almo
nd Cookies')]
```

In [22]:

```
us_num_1, in_num_1 = top_pairs[0]
print(us_num_1)
print(in_num_1)
```

```
Chocolate Chip
Punjabi
```

In [23]:

```
for us_cookie, in_cookie in top_pairs:
    print(in_cookie)
    print(us_cookie)
```

```
Punjabi
Chocolate Chip
Fruit Cake Rusk
Brownies
Marble Cookies
Peanut Butter
Kaju Pista Cookies
Oreos
Almond Cookies
Oatmeal
```

In [24]:

```
for idx, item in enumerate(top_pairs):
    us_cookie, in_cookie = item
    print(idx, us_cookie, in_cookie)
```

```
0 Chocolate Chip Punjabi
1 Brownies Fruit Cake Rusk
2 Peanut Butter Marble Cookies
3 Oreos Kaju Pista Cookies
4 Oatmeal Almond Cookies
```

In [25]:

```
item = ('vanilla', 'chocolate')
print(item)
```

```
('vanilla', 'chocolate')
```

In [26]:

```
item2 = 'butter',
print(item2)
```

```
('butter',)
```

In [27]:

```
# EXERCISES
```

In [28]:

```
# Using and unpacking tuples
############################

girl_names = set([x for x in records[records[:,1] == 'FEMALE',3]])
boy_names = set([x for x in records[records[:,1] == 'MALE',3]])
```

In [29]:

```
# Pair up the boy and girl names: pairs
pairs = zip(girl_names,boy_names)

# Iterate over pairs
for idx, pair in enumerate(pairs):
    # Unpack pair: girl_name, boy_name
    girl_name, boy_name = pair
    # Print the rank and names associated with each rank
    print('Rank {}: {} and {}'.format(idx, girl_name, boy_name))
```

```
Rank 0: ALEXA and ARMANI
Rank 1: Rebecca and Ben
Rank 2: Molly and SHULEM
Rank 3: DALIA and BERISH
Rank 4: ISABELA and Mauricio
Rank 5: Helena and Jaiden
Rank 6: MILENA and JAMES
Rank 7: Liba and JESUS
Rank 8: JOANNA and Esteban
Rank 9: Alexia and CHASE
Rank 10: Logan and Austin
Rank 11: Alexandra and Raphael
Rank 12: KAYLEEN and Aaron
Rank 13: LIBBY and Kameron
Rank 14: Gia and EFRAIM
Rank 15: RIVKA and BOUBACAR
Rank 16: Cecelia and Devin
Rank 17: MILA and Hillel
Rank 18: ADDISON and MYLES
Rank 19: GRACE and ABRAHAM
```

In [30]:

```
# Making tuples by accident
############################

# Create the normal variable: normal
normal = 'simple'

# Create the mistaken variable: error
error = 'trailing comma',

# Print the types of the variables
print(type(normal))
print(type(error))
```

```
<class 'str'>
<class 'tuple'>
```

Sets for unordered and unique data

In [31]:

```
cookies_eaten_today = ['chocolate chip', 'peanut butter',
                       'chocolate chip', 'oatmeal cream', 'chocolate chip']
types_of_cookies_eaten = set(cookies_eaten_today)
print(types_of_cookies_eaten)
```

```
{'oatmeal cream', 'chocolate chip', 'peanut butter'}
```

In [32]:

```
types_of_cookies_eaten.add('biscotti')
types_of_cookies_eaten.add('chocolate chip')
print(types_of_cookies_eaten)

cookies_hugo_ate = ['chocolate chip', 'anzac']
types_of_cookies_eaten.update(cookies_hugo_ate)
print(types_of_cookies_eaten)
```

```
{'oatmeal cream', 'chocolate chip', 'biscotti', 'peanut butter'}
{'oatmeal cream', 'biscotti', 'chocolate chip', 'anzac', 'peanut butter'}
```

In [33]:

```
types_of_cookies_eaten.discard('biscotti')
print(types_of_cookies_eaten)

types_of_cookies_eaten.pop()
```

```
{'oatmeal cream', 'chocolate chip', 'anzac', 'peanut butter'}
```

Out[33]:

```
'oatmeal cream'
```

In [34]:

```
types_of_cookies_eaten.pop()
```

Out[34]:

```
'chocolate chip'
```

In [35]:

```
cookies_jason_ate = set(['chocolate chip', 'oatmeal cream','peanut butter'])
cookies_hugo_ate = set(['chocolate chip', 'anzac'])

cookies_jason_ate.union(cookies_hugo_ate)
```

Out[35]:

```
{'anzac', 'chocolate chip', 'oatmeal cream', 'peanut butter'}
```

In [36]:

```
cookies_jason_ate.intersection(cookies_hugo_ate)
```

Out[36]:

```
{'chocolate chip'}
```

In [37]:

```
cookies_jason_ate.difference(cookies_hugo_ate)
```

Out[37]:

```
{'oatmeal cream', 'peanut butter'}
```

In [38]:

```
cookies_hugo_ate.difference(cookies_jason_ate)
```

Out[38]:

```
{'anzac'}
```

In [39]:

```
# EXERCISES
```

In [40]:

```
# Finding all the data and the overlapping data between sets
################################################################

baby_names_2011 = set([x.capitalize() for x in records[records[:,0] == 2011,3]])
baby_names_2014 = set([x for x in records[records[:,0] == 2014,3]])
```

In [41]:

```
# Find the union: all_names
all_names = baby_names_2011.union(baby_names_2014)

# Print the count of names in all_names
print(len(all_names))

# Find the intersection: overlapping_names
overlapping_names = baby_names_2011.intersection(baby_names_2014)

# Print the count of names in overlapping_names
print(len(overlapping_names))
```

```
1461
986
```

In [42]:

```
# Determining set differences
##############################

# Find the difference between 2011 and 2014: differences
differences = baby_names_2011.difference(baby_names_2014)

# Print the differences
print(len(differences))
```

```
220
```

Chap 2: Dictionaries - the root of Python

Using dictionaries

In [43]:

```
data = pd.read_csv('datasets/new_york_art_galleries.csv',
                        usecols=['NAME','ZIP'])
```

In [44]:

```
galleries = list(zip(data.NAME,data.ZIP))
```

In [45]:

```
art_galleries = {}
for name, zip_code in galleries:
    art_galleries[name] = zip_code
for name in art_galleries:
    print(name)
```

```
Randel Gallery Inc
Randolph & Tate Assocs
Raphael Fodde Editions
Rare art Gallery
Raydon Gallery
Red Dot
Reece Galleries Inc The
Reeves Contemporary Inc
Regina's Art Center
Rehs Galleries Inc
Reinhold Brown Gallery
Renaissance Gallery
Resnick Ira Inc
Ricco Maresca Gallery
Richard Gray Gallery
Richard Solomon
Richard York Gallery
Rienzo Gallery
Ritter-Antik Inc
Rivington Bar & Art Gallery
```

In [46]:

```
art_galleries['Louvre']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-46-124888f6e125> in <module>()
----> 1 art_galleries['Louvre']

KeyError: 'Louvre'
```

In [47]:

```
art_galleries.get('Louvre', 'Not Found')
```

Out[47]:

```
'Not Found'
```

In [48]:

```
art_galleries.get('Zarre Andre Gallery')
```

Out[48]:

```
10011
```

In [49]:

```
# Working with nested data

import csv
art_galleries = {}
with open("datasets/new_york_art_galleries.csv", 'r') as data_file:
    data = csv.DictReader(data_file, delimiter=",")
    for row in data:
        item = art_galleries.get(row["ZIP"], dict())
        item[row["NAME"]] = str(row["TEL"])
        art_galleries[row["ZIP"]] = item
```

In [50]:

```
art_galleries.keys()
```

Out[50]:

```
dict_keys(['10021', '10013', '10001', '10009', '10011', '10022', '10027', '1
0019', '11106', '10128', '10016', '11211', '10075', '11361', '10002', '1169
1', '11231', '10003', '10014', '11204', '11209', '10012', '11234', '10038',
'10029', '10028', '11201', '11214', '10039', '10065', '10010', '10044', '112
28', '11369', '11205', '10017', '11101', '11225', '11217', '11229', '11432',
'11222', '10024', '10031', '10472', '10018', '10006', '11215', '11102', '100
23', '10304', '10040', '10032', '11221', '11385', '10026', '11218', '11375',
'10036', '11109', '11223', '10025', '10033', '11208', '11206', '11433', '104
60', '11237', '10314', '11238', '10069', '11230', '10005', '10468', '10151',
'10464', '11354', '11377', '11412', '11379', '10463', '11220', '11374', '103
10', '10004', '10007'])
```

In [51]:

```
art_galleries['10027']
```

Out[51]:

```
{'Africart Motherland Inc': '(212) 368-6802',
 'Inner City Art Gallery Inc': '(212) 368-4941',
 "Paige's Art Gallery": '(212) 531-1577',
 'Triple Candie': '(212) 865-0783'}
```

In [52]:

```
art_galleries['10027']['Inner City Art Gallery Inc']
```

Out[52]:

```
'(212) 368-4941'
```

In [53]:

```
# EXERCISES
```

In [54]:

```
# Creating and looping through dictionaries
##############################################

# y = count, z = rank
female_baby_names_2012 = {(x,z) for x,y,z in records[
    (records[:,0]==2012)&(records[:,1]=='FEMALE'),3:6]}
```

In [55]:

```
# Create an empty dictionary: names
names = {}

# Loop over the girl names
for name, rank in female_baby_names_2012:
    # Add each name to the names dictionary using rank as the key
    names[rank] = name

# Sort the names list by rank in descending order and slice the first 10 items
for rank in sorted(names,reverse=True)[:10]:
    # Print each item
    print(names[rank])
```

```
CASSANDRA
AYA
ESTY
LILAH
MIRIAM
ERIKA
SHIRA
PAOLA
JULIANA
ROSELYN
```

In [56]:

```
# Safely finding by key
#######################

# Safely print rank 7 from the names dictionary
print(names.get(7))

# Safely print the type of rank 100 from the names dictionary
print(type(names.get(100)))

# Safely print rank 105 from the names dictionary or 'Not Found'
print(names.get(105,'Not Found'))
```

```
ANGELA
<class 'NoneType'>
Not Found
```

In [57]:

```
import csv
boy_names = {}
with open("datasets/baby_names.csv", 'r') as data_file:
    data = csv.DictReader(data_file, delimiter=",")
    for num,row in enumerate(data):
        if row["GENDER"] == "MALE":
            item = boy_names.get(row["BIRTH_YEAR"], dict())
            item[int(num+1)] = row["NAME"]
            boy_names[row["BIRTH_YEAR"]] = item
boy_names = {int(k):v for k,v in boy_names.items()}
```

In [58]:

```
# Dealing with nested data
##########################

# Print a list of keys from the boy_names dictionary
print(boy_names.keys())

# Print a list of keys from the boy_names dictionary for the year 2013
print(boy_names[2013].keys())

# Loop over the dictionary
for year in boy_names:
    # Safely print the year and the third ranked name or 'Unknown'
    print(year, boy_names[year].get(3, 'Unknown'))
```

```
dict_keys([2011, 2012, 2013, 2014])
dict_keys([10888, 10889, 10890, 10891, 10892, 10893, 10894, 10895, 10896,
10897, 10898, 10899, 10900, 10901, 10902, 10903, 10904, 10905, 10906, 1090
7, 10908, 10909, 10910, 10911, 10912, 10913, 10914, 10915, 10916, 10917, 1
0918, 10919, 10920, 10921, 10922, 10923, 10924, 10925, 10926, 10927, 1092
8, 10929, 10930, 10931, 10932, 10933, 10934, 10935, 10936, 10937, 10938, 1
0939, 10940, 10941, 10942, 10943, 10944, 10945, 10946, 10947, 10948, 1094
9, 10950, 10951, 10952, 10953, 10954, 10955, 10956, 10957, 10958, 10959, 1
0960, 10961, 10962, 10963, 10964, 10965, 10966, 10967, 10968, 10969, 1097
0, 10971, 10972, 10973, 10974, 10975, 10976, 10977, 10978, 10979, 10980, 1
0981, 10982, 10983, 10984, 10985, 10986, 10987, 10988, 10989, 10990, 1099
1, 10992, 10993, 10994, 10995, 10996, 10997, 10998, 10999, 11000, 11001, 1
1002, 11003, 11004, 11005, 11006, 11007, 11008, 11009, 11010, 11011, 1101
2, 11013, 11014, 11015, 11016, 11017, 11018, 11019, 11020, 11021, 11022, 1
1023, 11024, 11025, 11026, 11027, 11028, 11029, 11030, 11031, 11032, 1103
3, 11034, 11035, 11036, 11037, 11038, 11039, 11040, 11041, 11042, 11043, 1
1044, 11045, 11046, 11047, 11048, 11049, 11050, 11051, 11052, 11053, 1105
4, 11055, 11056, 11057, 11058, 11059, 11060, 11061, 11062, 11063, 11064, 1
1065, 11066, 11067, 11068, 11069, 11070, 11071, 11072, 11073, 11074, 1107
```

Altering dictionaries

In [59]:

```
# Adding and extending dictionaries
art_galleries['10007']
```

Out[59]:

```
{'Nyabinghi Africian Gift Shop': '(212) 566-3336'}
```

In [60]:

```
art_galleries['11234']
```

Out[60]:

```
{'A J ARTS LTD': '(718) 763-5473',
 'Doug Meyer Fine Art': '(718) 375-8006',
 'Portrait Gallery': '(718) 377-8762'}
```

In [61]:

```
galleries_11234 = [('A J ARTS LTD', '(718) 763-5473'),
                   ('Doug Meyer Fine Art', '(718) 375-8006'),
                   ('Portrait Gallery', '(718) 377-8777')]
```

In [62]:

```
art_galleries['11234'].update(galleries_11234)
```

In [63]:

```
print(art_galleries['11234'])
```

```
{'Portrait Gallery': '(718) 377-8777', 'A J ARTS LTD': '(718) 763-5473', 'Do
ug Meyer Fine Art': '(718) 375-8006'}
```

In [64]:

```
# Popping and deleting from dictionaries
del art_galleries['11234']
```

In [65]:

```
galleries_10310 = art_galleries.pop('10310')
print(galleries_10310)
```

```
{'New Dorp Village Antiques Ltd': '(718) 815-2526'}
```

In [66]:

```
# EXERCISES
```

In [67]:

```
# Adding and extending dictionaries
####################################

names_2011 = {1:'AAMIR',2:'JIA',3:'GIANNA',4:'GISELLE'}
```

In [68]:

```
# Assign the names_2011 dictionary as the value to the 2011 key of boy_names
boy_names[2011] = names_2011

# Update the 2012 key in the boy_names dictionary
boy_names[2012].update([(1,'Casey'),(2,'Aiden')])

# Loop over the boy_names dictionary
for year in boy_names:
    # Loop over and sort the data for each year by descending rank
    for rank in sorted(boy_names[year], reverse=True)[:1]:
        # Check that you have a rank
        if not rank:
            print(year, 'No Data Available')
        # Safely print the year and the least popular name or 'Not Available'
        print(year, boy_names[year].get(rank,'Not Available'))
```

```
2011 GISELLE
2012 ZEV
2013 Zev
2014 Zev
```

In [69]:

```
# Popping and deleting from dictionaries
########################################

female_names = \
{2011: {1: 'Olivia', 2: 'Esther', 3: 'Rachel', 4: 'Leah', 5: 'Emma',
        6: 'Chaya', 7: 'Sarah', 8: 'Sophia', 9: 'Ava', 10: 'Miriam'},
 2012: {},
 2013: {1: 'Olivia', 2: 'Emma', 3: 'Esther', 4: 'Sophia', 5: 'Sarah',
        6: 'Leah', 7: 'Rachel', 8: 'Chaya', 9: 'Miriam', 10: 'Chana'},
 2014: {1: 'Olivia', 2: 'Esther', 3: 'Rachel', 4: 'Leah', 5: 'Emma',
        6: 'Chaya', 7: 'Sarah', 8: 'Sophia', 9: 'Ava', 10: 'Miriam'}}
```

In [70]:

```
# Remove 2011 and store it: female_names_2011
female_names_2011 = female_names.pop(2011)

# Safely remove 2015 with an empty dictionary as the default: female_names_2015
female_names_2015 = female_names.pop(2015,{})

# Delete 2012
del(female_names[2012])

# Print female_names
print(female_names)
```

```
{2013: {1: 'Olivia', 2: 'Emma', 3: 'Esther', 4: 'Sophia', 5: 'Sarah', 6: 'Le
ah', 7: 'Rachel', 8: 'Chaya', 9: 'Miriam', 10: 'Chana'}, 2014: {1: 'Olivia',
2: 'Esther', 3: 'Rachel', 4: 'Leah', 5: 'Emma', 6: 'Chaya', 7: 'Sarah', 8:
'Sophia', 9: 'Ava', 10: 'Miriam'}}
```

Pythonically using dictionaries

In [71]:

```
for gallery, phone_num in art_galleries.items():
    print(gallery)
    print(phone_num)
```

```
10021
{"O'reilly William & Co Ltd": '(212) 396-1822', 'Owen Gallery': '(212) 879
-2415', 'Pall William Inc': '(212) 758-3340', 'Pandora Old Masters': '(21
2) 628-1974', 'Peck Ian Fine Paintings Ltd': '(212) 396-2442', 'Praxis Int
ernational Art': '(212) 772-9478', 'Questroyal Fine Art': '(212) 744-358
6', 'Rabenou Yris Ancient Arts': '(212) 486-0661', 'Randel Gallery Inc':
'(212) 861-6650', 'Richard Gray Gallery': '(212) 472-8787', 'Rienzo Galler
y': '(212) 288-2226', 'Rogallerycom': '(212) 732-6887', 'Rosenberg Paul &
Co': '(212) 472-1134', 'Roth': '(212) 717-9067', 'Royal Fine Art': '(212)
452-4297', 'Sabarsky Serge Gallery': '(212) 628-6281', 'Safani Gallery In
c': '(212) 570-6360', 'Salz Trager Janet': '(212) 744-6080', "Sam's Souk":
'(212) 535-7210', 'Sands & Company Inc Fine Art At Madison Av': '(212) 988
-3900', 'Sayn-Wittgenstein': '(212) 288-1493', 'Schillay Fine Art': '(212)
861-8353', 'Schiller & Bodo Fine Art': '(212) 772-8627', 'Schlesinger Gall
ery': '(212) 734-3600', 'Scholten Japanese Art': '(212) 585-0474', 'Serbin
Inna': '(212) 249-5252', 'Shapolsky Anita Gallery': '(212) 452-1094', 'She
pherd & Derom Galleries': '(212) 861-4050', 'Shickman H Gallery': '(212) 2
49-3800', 'Sigrid Freundorfer Fine Art': '(212) 517-9700', 'Society For Re
newal of Christian Art': '(212) 288-8186', 'Soloman & Co Fine Art Galleri
```

In [72]:

```
'11234' in art_galleries
```

Out[72]:

```
False
```

In [73]:

```
if '10010' in art_galleries:
    print('I found: %s' % art_galleries['10010'])
else:
    print('No galleries found.')
```

```
I found: {'Robertson African Arts': '(212) 206-0912', 'Sabbatique Ltd': '(21
2) 213-9344', 'Senior & Shopmaker': '(212) 213-6767', 'Shawn Dulaney Studi
o': '(212) 255-7986', 'Studio Tamar': '(212) 529-4247', 'Arts Du Monde': '(2
12) 243-4477', 'Arts India': '(212) 725-6092', 'Atelier A-E Ent': '(212) 620
-8103', 'Back To Africa Imports': '(212) 462-4848', 'JUNE BATEMAN FINE  GALL
ERY': '(212) 925-7951', 'Decor Art Gallery Inc': '(212) 481-3728', 'Equity A
rt Brokers': '(212) 620-7604', 'M K J Art Gallery': '(212) 481-7447', 'Morga
n Art Consulting': '(212) 447-0490'}
```

In [74]:

```
# EXERCISES
```

In [75]:

```
# Working with dictionaries more pythonically
##############################################

import csv
baby_names = {}
with open("datasets/baby_names.csv", 'r') as data_file:
    data = csv.DictReader(data_file, delimiter=",")
    for num,row in enumerate(data):
        item = baby_names.get(row["BIRTH_YEAR"], dict())
        item[int(num+1)] = row["NAME"]
        baby_names[row["BIRTH_YEAR"]] = item
baby_names = {int(k):v for k,v in boy_names.items()}
```

In [76]:

```
# Iterate over the 2014 nested dictionary
for rank, name in baby_names[2014].items():
    # Print rank and name
    print(rank, name)

# Iterate over the 2012 nested dictionary
for rank, name in baby_names[2012].items():
    # Print rank and name
    print(rank, name)
```

```
12957 Aahil
12958 Aaron
12959 Aayan
12960 Abdullah
12961 Adam
12962 Adrian
12963 Adyan
12964 Ahmed
12965 Ahnaf
12966 Aidan
12967 Aiden
12968 Alan
12969 Albert
12970 Alex
12971 Alexander
12972 Alfred
12973 Ali
12974 Allen
12975 Alston
```

In [77]:

```
# Checking dictionaries for data
################################

# Check to see if 2011 is in baby_names
if 2011 in baby_names:
    # Print 'Found 2011'
    print('Found 2011')

# Check to see if rank 1 is in 2012
if 1 in baby_names[2012]:
    # Print 'Found Rank 1 in 2012' if found
    print('Found Rank 1 in 2012')
else:
    # Print 'Rank 1 missing from 2012' if not found
    print('Rank 1 missing from 2012')

# Check to see if Rank 5 is in 2013
if 5 in baby_names[2013]:
   # Print 'Found Rank 5'
   print('Found Rank 5')
```

```
Found 2011
Found Rank 1 in 2012
```

Working with CSV files

In [78]:

```
# Reading from a file using CSV reader

import csv
csvfile = open('datasets/new_york_art_galleries.csv', 'r')
for row in csv.reader(csvfile):
    print(row)
csvfile.close()
```

```
['Exotiqa II', 'POINT (-73.78597884804125 40.84660712240781)', '(212) 721-
4394', 'http://www.speendex.com/New-York/New-York-City/Shopping/Arts/Exoti
qa-II-l497278.html', '280 City Island Ave', '', 'Bronx', '10464']
['Eye Jammie Fine Arts Gallery', 'POINT (-74.00430835963606 40.74926699052
0084)', '(212) 645-0061', 'http://www.eyejammie.com/', '516 W 25th St',
 '', 'New York', '10001']
['Eyebeam Atelier', 'POINT (-74.00705639760449 40.74698648235827)', '(212)
 937-6580', 'http://eyebeam.org/', '540 W 21st St', '', 'New York', '1001
1']
['Ezair Gallery', 'POINT (-73.96488957373968 40.77214609107672)', '(212) 6
28-2224', 'http://www.ezairgallery.com/', '905 Madison Ave', '', 'New Yor
k', '10021']
['Faith Art Gallery Corp', 'POINT (-73.98502183713724 40.69145797289582)',
 '(718) 596-4659', 'http://ny.citysquares.com/brooklyn/downtown-brooklyn/a
rts-entertainment/artists-art-dealers/faith-art-gallery-corp', '393 Bridge
 St', '', 'Brooklyn', '11201']
['Feature Inc', 'POINT (-74.00465742633867 40.74938527551454)', '(212) 675
-7772', 'http://featureinc.com/', '530 W 25th St', '', 'New York', '1000
1']
['Fields-Walton', 'POINT (-73.9799941617584340.724344525211 03)', '(212) 4
```

In [79]:

```
# Creating a dictionary from a file

import csv
csvfile = open('datasets/new_york_art_galleries.csv', 'r')
for row in csv.DictReader(csvfile):
    print(row)
csvfile.close()
```

```
p://www.nyc.com/arts__attractions/indians_on_columbus.947/map_location.asp
x'), ('ADDRESS1', '452 Columbus Ave'), ('ADDRESS2', ''), ('CITY', 'New Yor
k'), ('ZIP', '10024')])
OrderedDict([('NAME', 'Infinity Fine Arts Inc'), ('the_geom', 'POINT (-74.
00371024848562 40.749516496539414)'), ('TEL', '(212) 645-6880'), ('URL',
 'http://www.speendex.com/New-York/New-York-City/Shopping/Arts/Infinity-Fi
ne-Arts-Inc-1496199.html'), ('ADDRESS1', '511 W 25th St'), ('ADDRESS2',
 ''), ('CITY', 'New York'), ('ZIP', '10001')])
OrderedDict([('NAME', 'Inner City Art Gallery Inc'), ('the_geom', 'POINT
 (-73.94456809199143 40.81352929524498)'), ('TEL', '(212) 368-4941'), ('UR
L', 'http://outside.in/harlem-manhattan-ny/inner-city-art-gallery-inc'),
 ('ADDRESS1', '2251 Adam Clayton Powell Jr Blvd'), ('ADDRESS2', ''), ('CIT
Y', 'New York'), ('ZIP', '10027')])
OrderedDict([('NAME', 'Innovative Art Gallery Inc'), ('the_geom', 'POINT
 (-73.99954513707648 40.73816930578245)'), ('TEL', '(212) 367-8844'), ('UR
L', 'http://www.nyc.com/arts__attractions/innovative_art_gallery_inc.804/e
ditorial_review.aspx'), ('ADDRESS1', '154 W 14th St'), ('ADDRESS2', ''),
 ('CITY', 'New York'), ('ZIP', '10011')])
OrderedDict([('NAME', 'Inter Art Gallery'), ('the_geom', 'POINT (-74.00418
890924199 40.74805554041248)'), ('TEL', '(212) 647-1811'), ('URL', 'htt
```

In [80]:

```
# EXERCISES
```

In [81]:

```
# Reading from a file using CSV reader
#######################################

# Import the python CSV module
import csv

# Create a python file object in read mode for the baby_names.csv file: csvfile
csvfile = open('datasets/baby_names.csv','r')

# Loop over a csv reader on the file object
for row in csv.reader(csvfile):
    # Print each row
    print(row)
    # Add the rank and name to the dictionary
    baby_names[row[5]] = row[3]

# Print the dictionary keys
print(baby_names.keys())
```

```
['BIRTH_YEAR', 'GENDER', 'ETHNICTY', 'NAME', 'COUNT', 'RANK']
['2011', 'FEMALE', 'HISPANIC', 'GERALDINE', '13', '75']
['2011', 'FEMALE', 'HISPANIC', 'GIA', '21', '67']
['2011', 'FEMALE', 'HISPANIC', 'GIANNA', '49', '42']
['2011', 'FEMALE', 'HISPANIC', 'GISELLE', '38', '51']
['2011', 'FEMALE', 'HISPANIC', 'GRACE', '36', '53']
['2011', 'FEMALE', 'HISPANIC', 'GUADALUPE', '26', '62']
['2011', 'FEMALE', 'HISPANIC', 'HAILEY', '126', '8']
['2011', 'FEMALE', 'HISPANIC', 'HALEY', '14', '74']
['2011', 'FEMALE', 'HISPANIC', 'HANNAH', '17', '71']
['2011', 'FEMALE', 'HISPANIC', 'HAYLEE', '17', '71']
['2011', 'FEMALE', 'HISPANIC', 'HAYLEY', '13', '75']
['2011', 'FEMALE', 'HISPANIC', 'HAZEL', '10', '78']
['2011', 'FEMALE', 'HISPANIC', 'HEAVEN', '15', '73']
['2011', 'FEMALE', 'HISPANIC', 'HEIDI', '15', '73']
['2011', 'FEMALE', 'HISPANIC', 'HEIDY', '16', '72']
['2011', 'FEMALE', 'HISPANIC', 'HELEN', '13', '75']
['2011', 'FEMALE', 'HISPANIC', 'IMANI', '11', '77']
['2011', 'FEMALE', 'HISPANIC', 'INGRID', '11', '77']
```

In [82]:

```
# Creating a dictionary from a file
###################################

# Import the python CSV module
import csv

# Create a python file object in read mode for the `baby_names.csv` file: csvfile
csvfile = open('datasets/baby_names.csv','r')

# Loop over a DictReader on the file
for row in csv.DictReader(csvfile):
    # Print each row
    print(row)
    # Add the rank and name to the dictionary: baby_names
    baby_names[row['RANK']] = row['NAME']

# Print the dictionary
print(baby_names.keys())
```

```
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GERALDINE'), ('COUNT', '13'), ('RANK', '75')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GIA'), ('COUNT', '21'), ('RANK', '67')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GIANNA'), ('COUNT', '49'), ('RANK', '42')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GISELLE'), ('COUNT', '38'), ('RANK', '51')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GRACE'), ('COUNT', '36'), ('RANK', '53')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'GUADALUPE'), ('COUNT', '26'), ('RANK', '62')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'HAILEY'), ('COUNT', '126'), ('RANK', '8')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'HALEY'), ('COUNT', '14'), ('RANK', '74')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
ISPANIC'), ('NAME', 'HANNAH'), ('COUNT', '17'), ('RANK', '71')])
OrderedDict([('BIRTH_YEAR', '2011'), ('GENDER', 'FEMALE'), ('ETHNICTY', 'H
```

Chap 3: Meet the collections module

Counting made easy

In [83]:

```
# Counter

nyc_eatery_types = pd.read_csv('datasets/NYC_Eateries.csv',\
    usecols=['type_name'])['type_name'].tolist()
```

In [84]:

```
from collections import Counter

nyc_eatery_count_by_types = Counter(nyc_eatery_types)
print(nyc_eatery_count_by_types)
```

```
Counter({'Mobile Food Truck': 114, 'Food Cart': 74, 'Snack Bar': 24, 'Specia
lty Cart': 18, 'Restaurant': 15, 'Fruit & Vegetable Cart': 4})
```

In [85]:

```
print(nyc_eatery_count_by_types['Restaurant'])
```

```
15
```

In [86]:

```
print(nyc_eatery_count_by_types.most_common(3))
```

```
[('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)]
```

In [87]:

```
# EXERCISES
```

In [88]:

```
# Using Counter on lists
########################

stations=pd.read_csv('datasets/cta_daily_station_totals.csv',
                     header=None,usecols=[1])[1].tolist()
```

In [89]:

```
# Import the Counter object
from collections import Counter

# Print the first ten items from the stations list
print(stations[:10])

# Create a Counter of the stations list: station_count
station_count = Counter(stations)

# Print the station_count
print(station_count)
```

```
['stationname', 'Austin-Forest Park', 'Austin-Forest Park', 'Austin-Forest P
ark', 'Austin-Forest Park', 'Austin-Forest Park', 'Austin-Forest Park', 'Aus
tin-Forest Park', 'Austin-Forest Park', 'Austin-Forest Park']
Counter({'Austin-Forest Park': 700, 'Harlem-Lake': 700, 'Pulaski-Lake': 700,
'Quincy/Wells': 700, 'Davis': 700, "Belmont-O'Hare": 700, 'Jackson/Dearbor
n': 700, 'Sheridan': 700, 'Damen-Brown': 700, 'Morse': 700, '35th/Archer': 7
00, '51st': 700, 'Dempster-Skokie': 700, 'Pulaski-Cermak': 700, 'LaSalle/Van
Buren': 700, 'Ashland-Lake': 700, 'Oak Park-Forest Park': 700, 'Sox-35th-Dan
Ryan': 700, 'Randolph/Wabash': 700, 'Damen-Cermak': 700, 'Western-Forest Par
k': 700, 'Cumberland': 700, '79th': 700, 'Kedzie-Homan-Forest Park': 700, 'S
tate/Lake': 700, 'Main': 700, 'Central-Lake': 700, 'Ashland/63rd': 700, 'Ind
iana': 700, 'Western-Orange': 700, 'Division/Milwaukee': 700, 'Grand/State':
700, 'Berwyn': 700, 'UIC-Halsted': 700, 'Southport': 700, 'Washington/Dearbo
rn': 700, 'Clark/Lake': 700, 'Forest Park': 700, 'Noyes': 700, 'Cicero-Cerma
k': 700, 'Clinton-Forest Park': 700, 'California-Cermak': 700, '95th/Dan Rya
n': 700, 'Merchandise Mart': 700, 'Racine': 700, 'Cicero-Lake': 700, 'Grand/
Milwaukee': 700, 'Garfield-South Elevated': 700, 'Foster': 700, 'Diversey':
700, 'Wilson': 700, "Irving Park-O'Hare": 700, 'Jackson/State': 700, 'Califo
rnia/Milwaukee': 700, '54th/Cermak': 700, 'Damen/Milwaukee': 700, 'Kostner':
700, 'Ridgeland': 700, 'Clark/Division': 700, 'Madison/Wabash': 700, 'North/
Clybourn': 700, 'Armitage': 700, 'Western/Milwaukee': 700, 'Adams/Wabash': 7
00, 'Dempster': 700, 'Laramie': 700, 'Chicago/Franklin': 700, 'East 63rd-Cot
tage Grove': 700, 'Washington/Wells': 700, 'Western-Cermak': 700, "Harlem-
O'Hare": 700, 'Granville': 700, 'Lawrence': 700, 'Central Park': 700, 'Monro
e/Dearborn': 700, 'Sedgwick': 700, 'Medical Center': 700, 'Rosemont': 700,
'18th': 700, 'South Boulevard': 700, 'Library': 700, 'Francisco': 700, 'Thor
ndale': 700, "O'Hare Airport": 700, 'Howard': 700, '63rd-Dan Ryan': 700, 'Pu
laski-Forest Park': 700, 'Midway Airport': 700, 'Halsted/63rd': 700, 'Pulask
i-Orange': 700, 'Cicero-Forest Park': 700, 'Harlem-Forest Park': 700, '69t
h': 700, 'Cermak-Chinatown': 700, 'Rockwell': 700, 'Logan Square': 700, 'Pol
k': 700, 'Kedzie-Cermak': 700, 'Linden': 700, 'Ashland-Orange': 700, 'Kedzie
-Lake': 700, '47th-South Elevated': 700, 'Monroe/State': 700, '35-Bronzevill
e-IIT': 700, 'Halsted-Orange': 700, 'King Drive': 700, 'Kedzie-Midway': 700,
'Clinton-Lake': 700, 'Garfield-Dan Ryan': 700, 'Kedzie-Brown': 700, 'Jarvi
s': 700, 'Argyle': 700, 'Wellington': 700, 'Fullerton': 700, '47th-Dan Rya
n': 700, "Addison-O'Hare": 700, 'Central-Evanston': 700, 'Austin-Lake': 700,
'43rd': 700, 'Jefferson Park': 700, 'Kimball': 700, 'Loyola': 700, 'Paulin
a': 700, 'Belmont-North Main': 700, "Montrose-O'Hare": 700, 'LaSalle': 700,
'Oak Park-Lake': 700, 'California-Lake': 700, 'Bryn Mawr': 700, 'Roosevelt':
700, 'Chicago/Milwaukee': 700, 'Addison-North Main': 700, '87th': 700, 'Addi
son-Brown': 700, 'Chicago/State': 700, 'Irving Park-Brown': 700, 'Western-Br
own': 700, 'Harrison': 700, 'Montrose-Brown': 700, 'Morgan-Lake': 700, 'Lak
e/State': 700, 'Conservatory': 700, 'Oakton-Skokie': 700, 'Cermak-McCormick
Place': 700, 'stationname': 1})
```

In [90]:

```
# Finding most common elements
##############################

# Import the Counter object
from collections import Counter

# Create a Counter of the stations list: station_count
station_count = Counter(stations)

# Find the 5 most common elements
print(station_count.most_common(5))
```

```
[('Austin-Forest Park', 700), ('Harlem-Lake', 700), ('Pulaski-Lake', 700),
('Quincy/Wells', 700), ('Davis', 700)]
```

Dictionaries of unknown structure - Defaultdict

In [91]:

```
# Dictionary Handling

nyc_eateries_parks = \
    pd.read_csv('datasets/NYC_Eateries.csv',usecols=['park_id','name']) \
                [['park_id','name']].to_records(index=False)
```

In [92]:

```
eateries_by_park = {}
for park_id, name in nyc_eateries_parks:
    if park_id not in eateries_by_park:
        eateries_by_park[park_id] = []
    eateries_by_park[park_id].append(name)
```

In [93]:

```
print(eateries_by_park['M010'])
```

```
['Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food Car
t', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food C
art', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food
Cart', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Foo
d Cart', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park F
ood Cart', 'THE NEW YORK PICNIC COMPANY, INC.', 'SALIM AHAMED', 'NANDITA, IN
C.', 'HOSSAIN ALI', 'THE NY PICNIC COMPANY', 'JANANI FOOD SERVICES, INC.',
'UNLIMITED NUTS, INC.', 'UNLIMITED NUTS, INC.', 'MUN TRADING COMPANY', 'SALI
M AHAMED', 'JANANI FOOD SERVICE, INC.', 'MOHAMMAD MATIN', 'MUHAMMAD T. ISLA
M', 'JANANI FOOD SERVICES, INC.', 'MUN TRADING COMPANY', 'INDULGE BISTRO CAF
E, INC.', 'JANANI FOOD SERVICES, INC.', 'Nandita Inc.', 'SHARMIN, INC.', 'Lo
eb Boathouse Restaurant', 'Tavern on the Green', 'Conservatory Water Snack B
ar', 'Mineral Springs Cafe', 'Ballfields Cafe', 'Arsenal Snack Bar', 'Wafels
& Dinges', 'GROM', 'NANDITA, INC.', 'CAFÉ PRODUCTS CORP.']
```

In [94]:

```
# Using defaultdict

from collections import defaultdict

eateries_by_park = defaultdict(list)
for park_id, name in nyc_eateries_parks:
    eateries_by_park[park_id].append(name)

print(eateries_by_park['M010'])
```

```
['Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food Car
t', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food C
art', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Food
Cart', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park Foo
d Cart', 'Central Park Food Cart', 'Central Park Food Cart', 'Central Park F
ood Cart', 'THE NEW YORK PICNIC COMPANY, INC.', 'SALIM AHAMED', 'NANDITA, IN
C.', 'HOSSAIN ALI', 'THE NY PICNIC COMPANY', 'JANANI FOOD SERVICES, INC.',
'UNLIMITED NUTS, INC.', 'UNLIMITED NUTS, INC.', 'MUN TRADING COMPANY', 'SALI
M AHAMED', 'JANANI FOOD SERVICE, INC.', 'MOHAMMAD MATIN', 'MUHAMMAD T. ISLA
M', 'JANANI FOOD SERVICES, INC.', 'MUN TRADING COMPANY', 'INDULGE BISTRO CAF
E, INC.', 'JANANI FOOD SERVICES, INC.', 'Nandita Inc.', 'SHARMIN, INC.', 'Lo
eb Boathouse Restaurant', 'Tavern on the Green', 'Conservatory Water Snack B
ar', 'Mineral Springs Cafe', 'Ballfields Cafe', 'Arsenal Snack Bar', 'Wafels
& Dinges', 'GROM', 'NANDITA, INC.', 'CAFÉ PRODUCTS CORP.']
```

In [95]:

```
# Using defaultdict for counting

nyc_eateries = pd.read_csv('datasets/NYC_Eateries.csv')\
                            .to_dict(orient='records')
```

In [96]:

```
from collections import defaultdict

eatery_contact_types = defaultdict(int)
for eatery in nyc_eateries:
    if eatery.get('phone'):
        eatery_contact_types['phones'] += 1
    if eatery.get('website'):
        eatery_contact_types['websites'] += 1

print(eatery_contact_types)
```

```
defaultdict(<class 'int'>, {'phones': 249, 'websites': 249})
```

In [97]:

```
# EXERCISES
```

In [98]:

```
# Creating dictionaries of an unknown structure
###############################################

entries = pd.read_csv('datasets/cta_daily_station_totals.csv',
                      usecols=['date','stationname','rides'])\
                      [['date','stationname','rides']].to_records(index=False)
```

In [99]:

```
# Create an empty dictionary: ridership
ridership = {}

# Iterate over the entries
for date, stop, riders in entries:
    # Check to see if date is already in the dictionary
    if date not in ridership:
        # Create an empty list for any missing date
        ridership[date] = []
    # Append the stop and riders as a tuple to the date keys list
    ridership[date].append((stop,riders))

# Print the ridership for '03/09/2016'
print(ridership['03/09/2016'][:10])
```

```
[('Austin-Forest Park', 2128), ('Harlem-Lake', 3769), ('Pulaski-Lake', 150
2), ('Quincy/Wells', 8139), ('Davis', 3656), ("Belmont-O'Hare", 5294), ('Jac
kson/Dearborn', 8369), ('Sheridan', 5823), ('Damen-Brown', 3048), ('Morse',
4826)]
```

In [100]:

```
# Safely appending to a key's value list
########################################

# Import defaultdict
from collections import defaultdict

# Create a defaultdict with a default type of list: ridership
ridership = defaultdict(list)

# Iterate over the entries
for date, stop, riders in entries:
    # Use the stop as the key of ridership and append the riders to its value
    ridership[stop].append(riders)

# Print the first 10 items of the ridership dictionary
print(list(ridership.items())[:1])
```

```
[('Austin-Forest Park', [587, 1386, 785, 625, 1752, 1777, 1269, 1435, 1631,
771, 588, 2065, 2108, 2012, 2069, 2003, 953, 706, 1216, 2115, 2132, 2185, 20
72, 854, 585, 2095, 2251, 2133, 2083, 2074, 953, 596, 1583, 2263, 2179, 210
5, 2076, 1049, 612, 2095, 2191, 2117, 1931, 1943, 800, 584, 1434, 2078, 186
9, 1455, 1830, 841, 621, 1884, 2100, 2046, 2066, 2016, 875, 615, 1975, 2391,
2058, 2035, 2008, 989, 635, 2105, 2148, 2152, 2155, 2182, 1340, 718, 2191, 2
220, 2154, 2248, 2183, 1073, 664, 1924, 2060, 2049, 2138, 1930, 972, 693, 20
59, 2060, 2120, 2062, 1751, 928, 664, 2047, 2032, 2030, 1899, 2096, 1012, 68
8, 2090, 2160, 2182, 2184, 2235, 1060, 732, 2090, 2161, 2115, 2203, 2180, 88
5, 738, 2152, 2175, 2230, 2218, 2320, 1207, 773, 2171, 2090, 2225, 2333, 209
8, 1042, 678, 2048, 2097, 2118, 2198, 2273, 1095, 779, 2103, 2119, 2090, 220
6, 2081, 1095, 767, 795, 2025, 2171, 2271, 2175, 910, 668, 2148, 2110, 2198,
2152, 2138, 1129, 773, 2041, 2156, 2172, 2093, 2010, 1225, 843, 2006, 2126,
2062, 2341, 2022, 1134, 832, 1938, 2142, 2117, 2076, 1932, 1155, 1172, 2022,
2097, 2152, 2093, 1445, 1205, 884, 1946, 2044, 2146, 2247, 2226, 1162, 1039,
1983, 2094, 2175, 2037, 2106, 1086, 877, 2031, 2141, 2125, 2190, 2192, 1207,
757, 2038, 2164, 2177, 2066, 2181, 1247, 974, 1997, 2118, 2102, 2119, 2072,
1082, 752, 1926, 2152, 2075, 2086, 2086, 1129, 850, 2026, 2021, 2055, 2120,
2099, 1162, 813, 2195, 2333, 2267, 2253, 2148, 976, 836, 2221, 2401, 2177, 2
356, 2160, 1043, 897, 787, 2197, 2366, 2335, 2215, 1055, 787, 2296, 2391, 23
59, 2367, 2187, 1103, 837, 2286, 2424, 2321, 2333, 2231, 1164, 773, 2393, 23
91, 2435, 2489, 2305, 1031, 757, 2366, 2457, 2435, 2400, 2458, 1083, 890, 20
38, 2438, 2386, 2467, 2245, 1078, 752, 2237, 2437, 2461, 2345, 2336, 1035, 7
35, 2299, 2351, 2349, 2344, 2264, 937, 788, 2284, 2256, 2426, 2383, 2245, 10
86, 799, 2227, 2399, 2038, 2363, 2156, 1005, 709, 2229, 2233, 2319, 2207, 21
03, 947, 631, 2142, 2229, 1887, 642, 971, 852, 666, 2098, 2192, 2237, 2200,
2168, 1059, 820, 1992, 2130, 2172, 2192, 2213, 1261, 715, 1904, 2049, 2056,
2082, 1944, 834, 715, 1664, 1820, 1585, 1022, 523, 774, 650, 1300, 1512, 153
4, 1418, 586, 799, 599, 1835, 1891, 1974, 1983, 1938, 823, 502, 1865, 1939,
1907, 2095, 1979, 855, 493, 883, 1968, 2044, 2078, 2021, 804, 609, 2053, 206
5, 2145, 2118, 1993, 879, 664, 2128, 2013, 2158, 2167, 1970, 950, 643, 1988,
2023, 2038, 2041, 1766, 782, 601, 1384, 1995, 2080, 2098, 2172, 1051, 671, 1
995, 2111, 2084, 2107, 2002, 940, 749, 1941, 2056, 2061, 2048, 1950, 895, 64
9, 2004, 2148, 2128, 2079, 2097, 1129, 634, 2056, 2001, 2104, 2193, 2104, 92
9, 707, 1869, 2000, 1927, 1826, 1586, 881, 590, 1899, 2037, 2071, 2061, 199
2, 1014, 884, 2005, 2126, 2055, 2114, 1990, 927, 669, 2072, 2131, 2235, 218
4, 2126, 1193, 918, 2063, 2088, 2105, 2126, 1978, 1054, 861, 2066, 2152, 205
6, 2154, 2173, 956, 748, 2017, 2129, 2066, 2276, 2151, 1142, 797, 1897, 191
2, 2054, 2000, 1993, 1035, 847, 2020, 2099, 2113, 2110, 2048, 1234, 881, 203
8, 2122, 2078, 2172, 1977, 1069, 895, 878, 2030, 2136, 2211, 2222, 1041, 92
5, 2036, 2189, 2190, 2065, 2239, 1130, 837, 2063, 2153, 2119, 2184, 2050, 10
97, 774, 2043, 2162, 1946, 2074, 2127, 1101, 903, 1955, 2153, 2116, 2190, 20
```

```
05, 1013, 810, 894, 1910, 2038, 2171, 2047, 1180, 887, 2009, 2084, 2090, 210
1, 2137, 1101, 835, 2015, 2256, 2152, 2102, 2061, 1054, 775, 1913, 2104, 216
0, 2159, 2223, 1080, 847, 1969, 2070, 2084, 2148, 2076, 1093, 803, 1919, 210
8, 2107, 2094, 1836, 1099, 846, 1916, 2134, 2107, 2002, 2091, 1055, 909, 206
7, 2243, 2102, 2081, 2128, 1000, 743, 2217, 2225, 2261, 2322, 2245, 1079, 87
2, 833, 2313, 2307, 2318, 2267, 1036, 778, 2294, 2356, 2387, 2411, 2279, 107
7, 774, 2299, 2420, 2266, 2359, 2302, 1086, 815, 2278, 2398, 2209, 2218, 218
6, 943, 715, 2222, 2415, 2402, 2219, 2310, 1068, 930, 1860, 2309, 2346, 229
5, 2321, 1101, 770, 2285, 2353, 2389, 2238, 2143, 1101, 704, 2215, 2267, 222
7, 2315, 2218, 1033, 733, 2136, 2326, 2338, 2286, 3226, 1040, 729, 2200, 223
1, 2311, 2230, 1896, 991, 705, 2257, 2330, 2353, 2324, 2152, 1031, 640, 217
9, 2195, 1763, 564, 1002, 847, 630, 2015, 2224, 2197])]
```

Maintaining Dictionary Order with OrderedDict

In [101]:

```
from collections import OrderedDict

nyc_eatery_permits = OrderedDict()
for eatery in nyc_eateries:
    nyc_eatery_permits[eatery['end_date']] = eatery
```

In [102]:

```
print(list(nyc_eatery_permits.items())[:1])
```

```
[('31/12/2018', {'name': 'Van Cortlandt Park Mobile Food Truck', 'location':
'Broadway Between West 240 & West 263rd Street', 'park_id': 'X092', 'start_d
ate': '28/8/2014', 'end_date': '31/12/2018', 'description': nan, 'permit_num
ber': 'X92-1-MT', 'phone': nan, 'website': nan, 'type_name': 'Mobile Food Tr
uck'})]
```

In [103]:

```
print(nyc_eatery_permits.popitem())
```

```
('14/8/2021', {'name': 'CAFÉ PRODUCTS CORP.', 'location': 'COLUMBUS CIR. BEH
IND THE USS MAINE MONUMENT', 'park_id': 'M010', 'start_date': '15/8/2016',
'end_date': '14/8/2021', 'description': nan, 'permit_number': 'M10-W59-CG',
'phone': nan, 'website': nan, 'type_name': 'Specialty Cart'})
```

In [104]:

```
print(nyc_eatery_permits.popitem())
```

```
('24/11/2020', {'name': 'NANDITA, INC.', 'location': 'WEST SIDE OF EAST DRIV
E AND 63RD ST.', 'park_id': 'M010', 'start_date': '25/11/2015', 'end_date':
'24/11/2020', 'description': nan, 'permit_number': 'M10-63-ED-CG', 'phone':
nan, 'website': nan, 'type_name': 'Specialty Cart'})
```

In [105]:

```
print(nyc_eatery_permits.popitem(last=False))
```

```
('31/12/2018', {'name': 'Van Cortlandt Park Mobile Food Truck', 'location':
'Broadway Between West 240 & West 263rd Street', 'park_id': 'X092', 'start_d
ate': '28/8/2014', 'end_date': '31/12/2018', 'description': nan, 'permit_num
ber': 'X92-1-MT', 'phone': nan, 'website': nan, 'type_name': 'Mobile Food Tr
uck'})
```

In [106]:

```
# EXERCISES
```

In [107]:

```
# Working with OrderedDictionaries
###################################

entries = pd.read_csv('datasets/cta_daily_station_totals.csv',
                      usecols=['date','rides'])\
                      [['date','rides']].to_records(index=False)
```

In [108]:

```
# Import OrderedDict from collections
from collections import OrderedDict

# Create an OrderedDict called: ridership_date
ridership_date = OrderedDict()

# Iterate over the entries
for date, riders in entries:
    # If a key does not exist in ridership_date, set it to 0
    if not date in ridership_date:
        ridership_date[date] = 0

    # Add riders to the date key in ridership_date
    ridership_date[date] += riders

# Print the first 31 records
print(list(ridership_date.items())[:31])
```

```
[('01/01/2015', 233956), ('01/02/2015', 432144), ('01/03/2015', 273207), ('0
1/04/2015', 217632), ('01/05/2015', 538868), ('01/06/2015', 556918), ('01/0
7/2015', 416984), ('01/08/2015', 475074), ('01/09/2015', 524144), ('01/10/20
15', 282850), ('01/11/2015', 227240), ('01/12/2015', 605068), ('01/13/2015',
609226), ('01/14/2015', 608109), ('01/15/2015', 622792), ('01/16/2015', 6128
33), ('01/17/2015', 335555), ('01/18/2015', 244490), ('01/19/2015', 411497),
('01/20/2015', 618377), ('01/21/2015', 619945), ('01/22/2015', 623914), ('0
1/23/2015', 612177), ('01/24/2015', 333440), ('01/25/2015', 226964), ('01/2
6/2015', 605287), ('01/27/2015', 626168), ('01/28/2015', 625531), ('01/29/20
15', 622695), ('01/30/2015', 618395), ('01/31/2015', 337018)]
```

In [109]:

```
# Powerful Ordered popping
##########################

# Print the first key in ridership_date
print(list(ridership_date.keys())[0])

# Pop the first item from ridership_date and print it
print(ridership_date.popitem(last=False))

# Print the last key in ridership_date
print(list(ridership_date.keys())[-1])

# Pop the last item from ridership_date and print it
print(ridership_date.popitem())
```

```
01/01/2015
('01/01/2015', 233956)
11/30/2016
('11/30/2016', 631904)
```

What do you mean I don't have any class? Namedtuple

In [110]:

```
# Creating a namedtuple

from collections import namedtuple

Eatery = namedtuple('Eatery', ['name', 'location', 'park_id','type_name'])
eateries = []
for eatery in nyc_eateries:
    details = Eatery(eatery['name'],
                     eatery['location'],
                     eatery['park_id'],
                     eatery['type_name'])
    eateries.append(details)
```

In [111]:

```
print(eateries[0])
```

```
Eatery(name='Central Park Food Cart', location='At the entrance to the path
leading to the Pond, East Drive & E 61st Street', park_id='M010', type_name
='Food Cart')
```

In [112]:

```
# Leveraging namedtuples

for eatery in eateries[:3]:
    print(eatery.name)
    print(eatery.park_id)
    print(eatery.location)
```

```
Central Park Food Cart
M010
At the entrance to the path leading to the Pond, East Drive & E 61st Street
Central Park Food Cart
M010
At the intersection of paths, east side of East Drive, at approximately East
70 Street
Central Park Food Cart
M010
Central Park Mall Area, northwest side of the Bandshell
```

In [113]:

```
# EXERCISES
```

In [114]:

```
# Creating namedtuples for storing data
########################################

entries = pd.read_csv('datasets/cta_daily_station_totals.csv',
                      usecols=['date','stationname','rides'])\
                      [['date','stationname','rides']].to_records(index=False)
```

In [115]:

```
# Import namedtuple from collections
from collections import namedtuple

# Create the namedtuple: DateDetails
DateDetails = namedtuple('DateDetails', ['date', 'stop', 'riders'])

# Create the empty list: labeled_entries
labeled_entries = []

# Iterate over the entries
for date, stop, riders in entries:
    # Append a new DateDetails namedtuple instance for each entry to labeled_entries
    labeled_entries.append(DateDetails(date,stop,riders))

# Print the first 5 items in labeled_entries
print(labeled_entries[:5])
```

```
[DateDetails(date='01/01/2015', stop='Austin-Forest Park', riders=587), Date
Details(date='01/02/2015', stop='Austin-Forest Park', riders=1386), DateDeta
ils(date='01/03/2015', stop='Austin-Forest Park', riders=785), DateDetails(d
ate='01/04/2015', stop='Austin-Forest Park', riders=625), DateDetails(date
='01/05/2015', stop='Austin-Forest Park', riders=1752)]
```

In [116]:

```
# Leveraging attributes on namedtuples
######################################

# Iterate over the first twenty items in labeled_entries
for item in labeled_entries[:5]:
    # Print each item's stop
    print(item.stop)

    # Print each item's date
    print(item.date)

    # Print each item's riders
    print(item.riders)
```

```
Austin-Forest Park
01/01/2015
587
Austin-Forest Park
01/02/2015
1386
Austin-Forest Park
01/03/2015
785
Austin-Forest Park
01/04/2015
625
Austin-Forest Park
01/05/2015
1752
```

Chap 4: Handling Dates and Times

There and Back Again a DateTime Journey

The datetime module is part of the Python standard library
Use the datetime type from inside the datetime module
.strptime() method converts from a string to a datetime object

In [117]:

```
# From string to datetime
parking_violations_date = "06/11/2016"
```

In [118]:

```
from datetime import datetime
print(parking_violations_date)
```

```
06/11/2016
```

In [119]:

```
date_dt = datetime.strptime(parking_violations_date, '%m/%d/%Y')
print(date_dt)
```

```
2016-06-11 00:00:00
```

%d - Day of the month as a zero-padded decimal number. (01, 02, ..., 31)
%m - Month as a zero-padded decimal number. (01, 02, ..., 12)
%Y - Year with century as a decimal number. (0001, 0002, ..., 9998, 9999)

.strftime() method uses a format string to convert a datetime object to a string

In [120]:

```
# Datetime to String
date_dt.strftime('%m/%d/%Y')
```

Out[120]:

```
'06/11/2016'
```

isoformat() method outputs a datetime as an ISO standard string

In [121]:

```
date_dt.isoformat()
```

Out[121]:

```
'2016-06-11T00:00:00'
```

In [122]:

```
# EXERCISES
```

In [123]:

```
# Strings to DateTimes
######################

dates_list = pd.Series.tolist(pd.read_csv('datasets/cta_daily_summary_totals.csv',
                          usecols=['service_date'])['service_date'])
```

In [124]:

```
# Import the datetime object from datetime
from datetime import datetime

# Iterate over the dates_list
for date_str in dates_list:
    # Convert each date to a datetime object: date_dt
    date_dt = datetime.strptime(date_str,'%m/%d/%Y')

    # Print each date_dt
    print(date_dt)
```

```
2001-01-30 00:00:00
2001-01-31 00:00:00
2001-02-01 00:00:00
2001-02-02 00:00:00
2001-02-03 00:00:00
2001-02-04 00:00:00
2001-02-05 00:00:00
2001-02-06 00:00:00
2001-02-07 00:00:00
2001-02-08 00:00:00
2001-02-09 00:00:00
2001-02-10 00:00:00
2001-02-11 00:00:00
2001-02-12 00:00:00
2001-02-13 00:00:00
2001-02-14 00:00:00
2001-02-15 00:00:00
2001-02-16 00:00:00
2001-02-17 00:00:00
2001-02-18 00:00:00
```

In [125]:

```
# Converting to a String
#########################

datetimes_list = pd.Series.tolist(pd.to_datetime(pd.read_csv('datasets/cta_daily_summary_to
                          usecols=['service_date'])['service_date']))
```

In [126]:

```
# Loop over the first 10 items of the datetimes_list
for item in datetimes_list[:10]:
    # Print out the record as a string in the format of 'MM/DD/YYYY'
    print(item.strftime('%m/%d/%Y'))

    # Print out the record as an ISO standard string
    print(item.isoformat())
```

```
01/01/2001
2001-01-01T00:00:00
01/02/2001
2001-01-02T00:00:00
01/03/2001
2001-01-03T00:00:00
01/04/2001
2001-01-04T00:00:00
01/05/2001
2001-01-05T00:00:00
01/06/2001
2001-01-06T00:00:00
01/07/2001
2001-01-07T00:00:00
01/08/2001
2001-01-08T00:00:00
01/09/2001
2001-01-09T00:00:00
01/10/2001
2001-01-10T00:00:00
```

Working with Datetime Components and current time

Datetime Components

- day, month, year, hour, minute, second, and more are available from a datetime instance
- Great for grouping data

In [145]:

```
parking_violations = pd.read_csv('datasets/nyc_parking_violations_2017.csv').values
```

In [146]:

```
daily_violations = defaultdict(int)
for violation in parking_violations:
    violation_date = datetime.strptime(violation[4], '%m/%d/%Y')
    daily_violations[violation_date.day] += 1
```

In [148]:

```
print(sorted(daily_violations.items()))
```

```
[(1, 211), (2, 151), (3, 140), (4, 133), (5, 153), (6, 150), (7, 193), (8, 1
55), (9, 169), (10, 145), (11, 134), (12, 174), (13, 166), (14, 203), (15, 1
82), (16, 176), (17, 137), (18, 143), (19, 191), (20, 196), (21, 159), (22,
202), (23, 172), (24, 93), (25, 148), (26, 171), (27, 178), (28, 187), (29,
176), (30, 133), (31, 79)]
```

.NOW Method

- .now() method returns the current local datetime
- .utcnow() method returns the current UTC datetime

In [149]:

```
from datetime import datetime
local_dt = datetime.now()
print(local_dt)
```

```
2018-05-13 18:23:08.140739
```

In [150]:

```
utc_dt = datetime.utcnow()
print(utc_dt)
```

```
2018-05-13 14:23:09.598511
```

Timezones

- Naive datetime objects have no timezone data
- Aware datetime objects have a timezone
- Timezone data is available via the pytz module via the timezone object
- Aware objects have .astimezone() so you can get the time in another timezone

In [151]:

```
from pytz import timezone
record_dt = datetime.strptime('07/12/2016 04:39PM','%m/%d/%Y %H:%M%p')

ny_tz = timezone('US/Eastern')
la_tz = timezone('US/Pacific')

ny_dt = record_dt.replace(tzinfo=ny_tz)
la_dt = ny_dt.astimezone(la_tz)

print(ny_dt)
```

```
2016-07-12 04:39:00-04:56
```

In [152]:

```
print(la_dt)
```

```
2016-07-12 02:35:00-07:00
```

In [153]:

```
# EXERCISES
```

In [154]:

```
# Pieces of Time
################

import csv
csvfile = open('datasets/cta_daily_summary_totals.csv', 'r')
daily_summaries = []
next(csv.reader(csvfile), None)
for row in csv.reader(csvfile):
    daily_summaries.append(tuple(r for r in row))
```

In [155]:

```
# Create a defaultdict of an integer: monthly_total_rides
monthly_total_rides = defaultdict(int)

# Loop over the list daily_summaries
for daily_summary in daily_summaries:
    # Convert the service_date to a datetime object
    service_datetime = datetime.strptime(daily_summary[0],'%m/%d/%Y')

    # Add the total rides to the current amount for the month
    monthly_total_rides[service_datetime.month] += int(daily_summary[4])

# Print monthly_total_rides
print(monthly_total_rides)
```

```
defaultdict(<class 'int'>, {1: 515062454, 2: 500276873, 3: 557894281, 4: 544
878980, 5: 564403630, 6: 553707053, 7: 552970459, 8: 558434623, 9: 57477089
8, 10: 652199892, 11: 538491629, 12: 500560093})
```

In [156]:

```
# Creating DateTime Objects... Now
##################################

# Import datetime from the datetime module
from datetime import datetime

# Compute the local datetime: local_dt
local_dt = datetime.now()

# Print the local datetime
print(local_dt)

# Compute the UTC datetime: utc_dt
utc_dt = datetime.utcnow()

# Print the UTC datetime
print(utc_dt)
```

```
2018-05-13 18:25:20.238095
2018-05-13 14:25:20.238095
```

In [157]:

```
# Timezones
############

import csv
csvfile = open('datasets/cta_daily_summary_totals.csv', 'r')
daily_summaries = []
next(csv.reader(csvfile), None)
for row in csv.reader(csvfile):
    daily_summaries.append(tuple((datetime.strptime(row[0],'%m/%d/%Y'),row[3])))
```

In [158]:

```
# Create a Timezone object for Chicago
chicago_usa_tz = timezone('US/Central')

# Create a Timezone object for New York
ny_usa_tz = timezone('US/Eastern')

# Iterate over the daily_summaries list
for orig_dt, ridership in daily_summaries:

    # Make the orig_dt timezone "aware" for Chicago
    chicago_dt = orig_dt.replace(tzinfo=chicago_usa_tz)

    # Convert chicago_dt to the New York Timezone
    ny_dt = chicago_dt.astimezone(ny_usa_tz)

    # Print the chicago_dt, ny_dt, and ridership
    print('Chicago: %s, NY: %s, Ridership: %s' % (chicago_dt, ny_dt, ridership))
```

```
Chicago: 2001-01-01 00:00:00-05:51, NY: 2001-01-01 00:51:00-05:00, Ridersh
ip: 126455
Chicago: 2001-01-02 00:00:00-05:51, NY: 2001-01-02 00:51:00-05:00, Ridersh
ip: 501952
Chicago: 2001-01-03 00:00:00-05:51, NY: 2001-01-03 00:51:00-05:00, Ridersh
ip: 536432
Chicago: 2001-01-04 00:00:00-05:51, NY: 2001-01-04 00:51:00-05:00, Ridersh
ip: 550011
Chicago: 2001-01-05 00:00:00-05:51, NY: 2001-01-05 00:51:00-05:00, Ridersh
ip: 557917
Chicago: 2001-01-06 00:00:00-05:51, NY: 2001-01-06 00:51:00-05:00, Ridersh
ip: 255356
Chicago: 2001-01-07 00:00:00-05:51, NY: 2001-01-07 00:51:00-05:00, Ridersh
ip: 169825
Chicago: 2001-01-08 00:00:00-05:51, NY: 2001-01-08 00:51:00-05:00, Ridersh
ip: 590706
Chicago: 2001-01-09 00:00:00-05:51, NY: 2001-01-09 00:51:00-05:00, Ridersh
ip: 599905
Chicago: 2001-01-10 00:00:00-05:51, NY: 2001-01-10 00:51:00-05:00, Ridersh
```

Time Travel (Adding and Subtracting Time)

Incrementing through time

- timedelta is used to represent an amount of change in time
- Used to add or subtract a set amount of time from a datetime object

In [159]:

```
from datetime import timedelta
flashback = timedelta(days=90)
print(record_dt)
```

```
2016-07-12 04:39:00
```

In [160]:

```
print(record_dt - flashback)
```

```
2016-04-13 04:39:00
```

In [161]:

```
print(record_dt + flashback)
```

```
2016-10-10 04:39:00
```

Datetime differences

- Use the - operator to calculate the difference
- Returns a timedelta with the difference

In [162]:

```
record2_dt = datetime.strptime('06/9/2016 02:57PM','%m/%d/%Y %H:%M%p')
```

In [163]:

```
time_diff = record_dt - record2_dt
type(time_diff)
```

Out[163]:

```
datetime.timedelta
```

In [164]:

```
print(time_diff)
```

```
33 days, 1:42:00
```

In [165]:

```
# EXERCISES
```

In [166]:

```
# Finding a time in the future and from the past
################################################

import csv
csvfile = open('datasets/cta_daily_summary_totals.csv', 'r')
daily_summaries = {}
review_dates = []
next(csv.reader(csvfile), None)
for row in csv.reader(csvfile):
    if(datetime.strptime(row[0],'%m/%d/%Y') > datetime.strptime('12/21/2013','%m/%d/%Y') ):
        review_dates.append(datetime.strptime(row[0],'%m/%d/%Y'))
    daily_summaries[datetime.strptime(row[0],'%m/%d/%Y')] = {'day_type':row[1], 'total_ride
review_dates
```

Out[166]:

```
[datetime.datetime(2013, 12, 22, 0, 0),
 datetime.datetime(2013, 12, 23, 0, 0),
 datetime.datetime(2013, 12, 24, 0, 0),
 datetime.datetime(2013, 12, 25, 0, 0),
 datetime.datetime(2013, 12, 26, 0, 0),
 datetime.datetime(2013, 12, 27, 0, 0),
 datetime.datetime(2013, 12, 28, 0, 0),
 datetime.datetime(2013, 12, 29, 0, 0),
 datetime.datetime(2013, 12, 30, 0, 0),
 datetime.datetime(2013, 12, 31, 0, 0)]
```

In [167]:

```
# Import timedelta from the datetime module
from datetime import timedelta

# Build a timedelta of 30 days: glanceback
glanceback = timedelta(days=30)

# Iterate over the review_dates as date
for date in review_dates:
    # Calculate the date 30 days back: prior_period_dt
    prior_period_dt = date - glanceback

    # Print the review_date, day_type and total_ridership
    print('Date: %s, Type: %s, Total Ridership: %s' %
          (date,
           daily_summaries[date]['day_type'],
           daily_summaries[date]['total_ridership']))

    # Print the prior_period_dt, day_type and total_ridership
    print('Date: %s, Type: %s, Total Ridership: %s' %
          (prior_period_dt,
           daily_summaries[prior_period_dt]['day_type'],
           daily_summaries[prior_period_dt]['total_ridership']))
```

```
Date: 2013-12-22 00:00:00, Type: U, Total Ridership: 685457
Date: 2013-11-22 00:00:00, Type: W, Total Ridership: 1752614
Date: 2013-12-23 00:00:00, Type: W, Total Ridership: 1236510
Date: 2013-11-23 00:00:00, Type: A, Total Ridership: 1048943
Date: 2013-12-24 00:00:00, Type: W, Total Ridership: 815873
Date: 2013-11-24 00:00:00, Type: U, Total Ridership: 674817
Date: 2013-12-25 00:00:00, Type: U, Total Ridership: 363078
Date: 2013-11-25 00:00:00, Type: W, Total Ridership: 1641025
Date: 2013-12-26 00:00:00, Type: W, Total Ridership: 995622
Date: 2013-11-26 00:00:00, Type: W, Total Ridership: 1681213
Date: 2013-12-27 00:00:00, Type: W, Total Ridership: 1191650
Date: 2013-11-27 00:00:00, Type: W, Total Ridership: 1441786
Date: 2013-12-28 00:00:00, Type: A, Total Ridership: 911223
Date: 2013-11-28 00:00:00, Type: U, Total Ridership: 554312
Date: 2013-12-29 00:00:00, Type: U, Total Ridership: 627779
Date: 2013-11-29 00:00:00, Type: W, Total Ridership: 1074544
Date: 2013-12-30 00:00:00, Type: W, Total Ridership: 1142767
Date: 2013-11-30 00:00:00, Type: A, Total Ridership: 1013178
Date: 2013-12-31 00:00:00, Type: W, Total Ridership: 116130
Date: 2013-12-01 00:00:00, Type: U, Total Ridership: 704442
```

In [168]:

```
# Finding differences in DateTimes
###################################

date_ranges = [('1/30/2001', '3/1/2001'),
               ('3/31/2001', '4/30/2001'),
               ('5/30/2001', '6/29/2001'),
               ('7/29/2001', '8/28/2001'),
               ('9/27/2001', '10/27/2001')]
for num in range(len(date_ranges)):
    date_ranges[num] = tuple((datetime.strptime(date_ranges[num][0],'%m/%d/%Y'),\
                              datetime.strptime(date_ranges[num][1],'%m/%d/%Y')))
```

In [169]:

```
# Iterate over the date_ranges
for start_date, end_date in date_ranges:
    # Print the End and Start Date
    print(end_date, start_date)
    # Print the difference between each end and start date
    print(end_date - start_date)
```

```
2001-03-01 00:00:00 2001-01-30 00:00:00
30 days, 0:00:00
2001-04-30 00:00:00 2001-03-31 00:00:00
30 days, 0:00:00
2001-06-29 00:00:00 2001-05-30 00:00:00
30 days, 0:00:00
2001-08-28 00:00:00 2001-07-29 00:00:00
30 days, 0:00:00
2001-10-27 00:00:00 2001-09-27 00:00:00
30 days, 0:00:00
```

HELP! Libraries to make it easier

Parsing time with pendulum

- .parse() will attempt to convert a string to a pendulum datetime object without the need of the format string

In [176]:

```
import pendulum
occurred = str(violation[4]) + ' ' + str(violation[5]) +'M'
occurred_dt = pendulum.parse(occurred, tz='US/Eastern')
print(occurred_dt)
```

```
2016-09-13T00:14:00-04:00
```

Timezone hopping with pendulum

- in_timezone() method converts a pendulum time object to a desired timezone.
- now() method accepts a timezone you want to get the current time in

In [177]:

```
violation_dts = [
    "2016-06-11 14:38:00-04:00",
    "2016-04-25 14:09:00-04:00",
    "2016-04-23 07:49:00-04:00",
    "2016-04-26 07:09:00-04:00",
    "2016-01-04 09:52:00-05:00"]
violation_dts = [pendulum.parse(x) for x in violation_dts]
```

In [178]:

```
print(violation_dts)
```

```
[<Pendulum [2016-06-11T14:38:00-04:00]>, <Pendulum [2016-04-25T14:09:00-04:0
0]>, <Pendulum [2016-04-23T07:49:00-04:00]>, <Pendulum [2016-04-26T07:09:00-
04:00]>, <Pendulum [2016-01-04T09:52:00-05:00]>]
```

In [179]:

```
for violation_dt in violation_dts:
    print(violation_dt.in_timezone('Asia/Tokyo'))
```

```
2016-06-12T03:38:00+09:00
2016-04-26T03:09:00+09:00
2016-04-23T20:49:00+09:00
2016-04-26T20:09:00+09:00
2016-01-04T23:52:00+09:00
```

In [180]:

```
print(pendulum.now('Asia/Tokyo'))
```

```
2018-05-14T01:14:57.066511+09:00
```

Humanizing differences

- in_XXX() methods provide the difference in a chosen metric
- in_words() provides the difference in a nice expresive form

In [181]:

```
diff = violation_dts[3] - violation_dts[2]
diff
```

Out[181]:

```
<Period [2016-04-23T07:49:00-04:00 -> 2016-04-26T07:09:00-04:00]>
```

In [182]:

```
print(diff.in_words())
```

```
2 days 23 hours 20 minutes
```

In [183]:

```
print(diff.in_days())
```

```
3
```

In [184]:

```
print(diff.in_hours())
```

```
71
```

In [185]:

```
# EXERCISES
```

In [188]:

```
# Localizing time with pendulum
################################

# Import the pendulum module
import pendulum

# Create a now datetime for Tokyo: tokyo_dt
tokyo_dt = pendulum.now('Asia/Tokyo')

# Covert the tokyo_dt to Los Angeles: la_dt
la_dt = tokyo_dt.in_timezone('America/Los_Angeles')

# Print the ISO 8601 string of la_dt
print(la_dt.to_iso8601_string())
```

```
2018-05-14T03:50:53-07:00
```

In [191]:

```
# Humanizing Differences with Pendulum
#######################################

date_ranges = [('1/30/2001', '3/1/2001'),
               ('3/31/2001', '4/30/2001'),
               ('5/30/2001', '6/29/2001'),
               ('7/29/2001', '8/28/2001'),
               ('9/27/2001', '10/27/2001')]
```

In [192]:

```
# Iterate over date_ranges
for start_date, end_date in date_ranges:

    # Convert the start_date string to a pendulum date: start_dt
    start_dt = pendulum.parse(start_date)

    # Convert the end_date string to a pendulum date: end_dt
    end_dt = pendulum.parse(end_date)

    # Print the End and Start Date
    print(end_dt, start_dt)

    # Calculate the difference between end_dt and start_dt: diff_period
    diff_period = end_dt - start_dt

    # Print the difference in days
    print(diff_period.in_days())
```

```
2001-03-01T00:00:00+00:00 2001-01-30T00:00:00+00:00
30
2001-04-30T00:00:00+00:00 2001-03-31T00:00:00+00:00
30
2001-06-29T00:00:00+00:00 2001-05-30T00:00:00+00:00
30
2001-08-28T00:00:00+00:00 2001-07-29T00:00:00+00:00
30
2001-10-27T00:00:00+00:00 2001-09-27T00:00:00+00:00
30
```

Chap 5: Answering Data Science Questions

Counting within Date Ranges

Data Set Overview

Chicago Crime Data Date,Block,Primary Type,Description, Location Description,Arrest,Domestic, District
05/23/2016 05:35:00 PM,024XX W DIVISION ST,ASSAULT,SIMPLE, STREET,false,true,14

03/26/2016 08:20:00 PM,019XX W HOWARD ST,BURGLARY,FORCIBLE ENTRY, SMALL RETAIL STORE,false,false,24

Part 1 - Step 1

- Read data from CSV

In [195]:

```
import csv
csvfile = open('datasets/nyc_eateries.csv', 'r', encoding="utf8")
for row in csv.reader(csvfile):
    print(row)
```

```
['name', 'location', 'park_id', 'start_date', 'end_date', 'description',
'permit_number', 'phone', 'website', 'type_name']
['Central Park Food Cart', 'At the entrance to the path leading to the Pon
d, East Drive & E 61st Street', 'M010', '1/1/2014', '31/12/2018', '', 'M10
-61-ED-C', '', '', 'Food Cart']
['Central Park Food Cart', 'At the intersection of paths, east side of Eas
t Drive, at approximately East 70 Street', 'M010', '8/3/2013', '31/12/201
6', '', 'M10-70-ED-C', '', '', 'Food Cart']
['Central Park Food Cart', 'Central Park Mall Area, northwest side of the
Bandshell', 'M010', '23/2/2010', '31/12/2014', '', 'M10-72-1A-C', '', '',
'Food Cart']
['Central Park Food Cart', 'West 72 Street path, west of Daniel Webster',
'M010', '25/3/2010', '31/12/2014', '', 'M10-72-3-C', '', '', 'Food Cart']
['Central Park Food Cart', 'East Drive and 72 Street', 'M010', '27/6/201
5', '31/12/2019', '', 'M10-72-ED-C', '', '', 'Food Cart']
['Central Park Food Cart', 'West Drive, between West 74 and West 75 street
s', 'M010', '30/4/2015', '31/12/2019', '', 'M10-74-WD-C', '', '', 'Food Ca
rt']
['Central Park Food Cart', 'East of West Drive, West 81 Street path to Del
```

Part 1 - Step 2

- Create and use a Counter with a slight twist

In [199]:

```
from collections import Counter

nyc_eatery_count_by_types = Counter(nyc_eatery_types)

nyc_eatery_count_by_types
```

Out[199]:

```
Counter({'Food Cart': 74,
         'Fruit & Vegetable Cart': 4,
         'Mobile Food Truck': 114,
         'Restaurant': 15,
         'Snack Bar': 24,
         'Specialty Cart': 18})
```

- Use date parts for Grouping like in Chapter 4

In [200]:

```
daily_violations = defaultdict(int)

for violation in parking_violations:
    violation_date = datetime.strptime(violation[4], '%m/%d/%Y')
    daily_violations[violation_date.day] += 1

daily_violations
```

Out[200]:

```
defaultdict(int,
            {1: 211,
             2: 151,
             3: 140,
             4: 133,
             5: 153,
             6: 150,
             7: 193,
             8: 155,
             9: 169,
             10: 145,
             11: 134,
             12: 174,
             13: 166,
             14: 203,
             15: 182,
             16: 176,
             17: 137,
```

Part 1 - Step 3

- Group data by Month
- The date components we learned about earlier.

In [202]:

```
from collections import defaultdict

eateries_by_park = defaultdict(list)

for park_id, name in nyc_eateries_parks:
    eateries_by_park[park_id].append(name)

eateries_by_park
```

Out[202]:

```
defaultdict(list,
            {'B007': ['YAKUBOV, GAVRIEL'],
             'B016': ['Maria Hernandez Pushcart'],
             'B049': ['Soho Square Gourmet Carts'],
             'B057': ['Marine Park Mobile Truck'],
             'B058': ['Mccarren Park Mobile Trucks'],
             'B060': ['McKinley Plaza Fruit & Vegetable Cart'],
             'B066': ['Owls Head Park Mobile Food Truck'],
             'B073': ['Prospect Park Parade Ground Cart',
              'Prospect Park Mount Food Cart'],
             'B087': ['Sunset Park Food Cart'],
             'B098': ['Martin Luther Playground Mobile Truck'],
             'B100': ['Seth Low Park Mobile Food Truck'],
             'B111': ['J.J. Byrne Memorial Park Mobile Food Truck'],
             'B113C': ['Crossroads To Go'],
             'B126': ['Red Hook Recreation Center Mobile Food Truck',
              'El Olomega Red Hook Salvadoran Pupusa',
              'El Olomega Red Hook Salvadoran Pupusa'],
```

Part 1 - Final

- Find 5 most common locations for crime each month.

In [203]:

```
print(nyc_eatery_count_by_types.most_common(3))
```

```
[('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)]
```

In [204]:

```
# EXERCISES
```

In [206]:

```
# Reading your data with CSV Reader and Establishing your Data Containers
###############################################################################

# Import the csv module
import csv

# Create the file object: csvfile
csvfile = open('datasets/crime_sampler.csv', 'r')

# Create an empty list: crime_data
crime_data = []

# Loop over a csv reader on the file object
for row in csv.reader(csvfile):

    # Append the date, type of crime, location description, and arrest
    crime_data.append((row[0], row[2], row[4], row[5]))

# Remove the first element from crime_data
crime_data.pop(0)

# Print the first 10 records
print(crime_data[:10])
```

```
[('05/23/2016 05:35:00 PM', 'ASSAULT', 'STREET', 'false'), ('03/26/2016 08:2
0:00 PM', 'BURGLARY', 'SMALL RETAIL STORE', 'false'), ('04/25/2016 03:05:00
PM', 'THEFT', 'DEPARTMENT STORE', 'true'), ('04/26/2016 05:30:00 PM', 'BATTE
RY', 'SIDEWALK', 'false'), ('06/19/2016 01:15:00 AM', 'BATTERY', 'SIDEWALK',
'false'), ('05/28/2016 08:00:00 PM', 'BATTERY', 'GAS STATION', 'false'), ('0
7/03/2016 03:43:00 PM', 'THEFT', 'OTHER', 'false'), ('06/11/2016 06:55:00 P
M', 'PUBLIC PEACE VIOLATION', 'STREET', 'true'), ('10/04/2016 10:20:00 AM',
'BATTERY', 'STREET', 'true'), ('02/14/2017 09:00:00 PM', 'CRIMINAL DAMAGE',
'PARK PROPERTY', 'false')]
```

In [207]:

```
# Find the Months with the Highest Number of Crimes
#####################################################

# Import necessary modules
from collections import Counter
from datetime import datetime

# Create a Counter Object: crimes_by_month
crimes_by_month = Counter()

# Loop over the crime_data list
for crime in crime_data:

    # Convert the first element of each item into a Python Datetime Object: date
    date = datetime.strptime(crime[0], '%m/%d/%Y %I:%M:%S %p')

    # Increment the counter for the month of the row by one
    crimes_by_month[date.month] += 1

# Print the 3 most common months for crime
print(crimes_by_month.most_common(3))
```

```
[(1, 1948), (2, 1862), (7, 1257)]
```

In [210]:

```
# Transforming your Data Containers to Month and Location
############################################################

# Import necessary modules
from collections import defaultdict
from datetime import datetime

# Create a dictionary that defaults to a list: locations_by_month
locations_by_month = defaultdict(list)

# Loop over the crime_data list
for row in crime_data:
    # Convert the first element to a date object
    date = datetime.strptime(row[0],'%m/%d/%Y %I:%M:%S %p')

    # If the year is 2016
    if date.year == 2016:
        # Set the dictionary key to the month and add the location (fifth element) to the v
        locations_by_month[date.month].append(row[2])

# Print the dictionary
print(locations_by_month)
```

```
T', 'SIDEWALK', 'SCHOOL, PRIVATE, BUILDING', 'RESIDENCE', 'STREET', 'ALLE
Y', 'RESIDENCE', 'HOTEL/MOTEL', 'SIDEWALK', 'RESIDENCE PORCH/HALLWAY', 'SI
DEWALK', 'ALLEY', 'SIDEWALK', 'ALLEY', 'ALLEY', 'RESIDENCE', 'ALLEY', 'CTA
BUS', 'DEPARTMENT STORE', 'RESTAURANT', 'RESIDENCE', 'STREET', 'SIDEWALK',
'DEPARTMENT STORE', 'OTHER', 'APARTMENT', 'RESIDENTIAL YARD (FRONT/BACK)',
'HOTEL/MOTEL', 'BAR OR TAVERN', 'APARTMENT', 'STREET', 'STREET', 'APARTMEN
T', 'OTHER', 'SCHOOL, PRIVATE, BUILDING', 'ABANDONED BUILDING', 'GAS STATI
ON', 'APARTMENT', 'BANK', 'STREET', 'RESIDENCE', 'SMALL RETAIL STORE', 'AP
ARTMENT', 'STREET', 'RESIDENCE', 'STREET', 'STREET', 'RESIDENCE', 'APARTME
NT', 'RESTAURANT', 'APARTMENT', 'STREET', 'STREET', 'RESIDENCE', 'ABANDONE
D BUILDING', 'RESIDENCE', 'AIRPORT TERMINAL UPPER LEVEL - SECURE AREA', 'P
ARKING LOT/GARAGE(NON.RESID.)', 'SIDEWALK', 'RESIDENCE', 'RESIDENCE', 'RES
IDENCE', 'STREET', 'STREET', 'SCHOOL, PUBLIC, BUILDING', 'RESIDENCE', 'STR
EET', 'SIDEWALK', 'RESIDENTIAL YARD (FRONT/BACK)', 'SMALL RETAIL STORE',
'RESIDENCE', 'PARKING LOT/GARAGE(NON.RESID.)', 'BANK', 'RESIDENCE', 'ALLE
Y', 'RESTAURANT', 'APARTMENT', 'STREET', 'RESIDENCE PORCH/HALLWAY', 'SIDEW
ALK', 'DEPARTMENT STORE', 'GAS STATION', 'PARKING LOT/GARAGE(NON.RESID.)',
'PARK PROPERTY', 'RESIDENTIAL YARD (FRONT/BACK)', 'VEHICLE NON-COMMERCIA
L', 'STREET', 'STREET', 'SIDEWALK', 'STREET', 'RESIDENCE', 'STREET', 'APAR
TMENT', 'ATM (AUTOMATIC TELLER MACHINE)', 'STREET', 'SCHOOL, PUBLIC, BUILD
```

Well done! It is difficult to draw quick insights from this output - the .most_common() method would be useful here!

In [216]:

```
# Find the Most Common Crimes by Location Type by Month in 2016
################################################################

# Import Counter from collections
from collections import Counter

# Loop over the items from locations_by_month using tuple expansion of the month and locati
for month, locations in locations_by_month.items():
    # Make a Counter of the locations
    location_count = Counter(locations)
    # Print the month
    print(month)
    # Print the most common location
    print(location_count.most_common(5))
```

```
5
[('STREET', 241), ('RESIDENCE', 175), ('APARTMENT', 128), ('SIDEWALK', 11
1), ('OTHER', 41)]
3
[('STREET', 240), ('RESIDENCE', 190), ('APARTMENT', 139), ('SIDEWALK', 9
9), ('OTHER', 52)]
4
[('STREET', 213), ('RESIDENCE', 171), ('APARTMENT', 152), ('SIDEWALK', 9
6), ('OTHER', 40)]
6
[('STREET', 245), ('RESIDENCE', 164), ('APARTMENT', 159), ('SIDEWALK', 12
3), ('PARKING LOT/GARAGE(NON.RESID.)', 44)]
7
[('STREET', 309), ('RESIDENCE', 177), ('APARTMENT', 166), ('SIDEWALK', 12
5), ('OTHER', 47)]
10
[('STREET', 248), ('RESIDENCE', 206), ('APARTMENT', 122), ('SIDEWALK', 9
2), ('OTHER', 62)]
12
[('STREET', 207), ('RESIDENCE', 158), ('APARTMENT', 136), ('OTHER', 47)
```

Fantastic work. It looks like most crimes in Chicago in 2016 took place on the street.

Dictionaries with Time Windows for Keys

Part 2 - Step 1

- Read in the CSV data as a dictionary

In [217]:

```
import csv
csvfile = open('datasets/new_york_art_galleries.csv', 'r')
for row in csv.DictReader(csvfile):
    print(row)
```

```
7288487 40.76396460949374)'), ('TEL', '(212) 977-6190'), ('URL', 'http://j
adite.com/'), ('ADDRESS1', '413 W 50th St'), ('ADDRESS2', ''), ('CITY', 'N
ew York'), ('ZIP', '10019')])
OrderedDict([('NAME', 'Jain Marunouchi Gallery'), ('the_geom', 'POINT (-7
3.97565215202877 40.763430134805255)'), ('TEL', '(212) 969-9660'), ('URL',
 'http://www.artin2000.com/'), ('ADDRESS1', '24 W 57th St'), ('ADDRESS2',
 ''), ('CITY', 'New York'), ('ZIP', '10019')])
OrderedDict([('NAME', 'James Cohan Gallery'), ('the_geom', 'POINT (-74.004
09870159304 40.75039640107759)'), ('TEL', '(212) 714-9500'), ('URL', 'htt
p://www.jamescohan.com/'), ('ADDRESS1', '533 W 26th St'), ('ADDRESS2',
 ''), ('CITY', 'New York'), ('ZIP', '10001')])
OrderedDict([('NAME', 'James Graham & Sons Gallery'), ('the_geom', 'POINT
 (-73.96267506646566 40.77588115815901)'), ('TEL', '(212) 535-5767'), ('UR
L', 'http://www.jamesgrahamandsons.com/'), ('ADDRESS1', '1014 Madison Av
e'), ('ADDRESS2', ''), ('CITY', 'New York'), ('ZIP', '10021')])
OrderedDict([('NAME', 'Jane St Lifer Fine Art Inc'), ('the_geom', 'POINT
 (-73.98862441738362 40.77701410325828)'), ('TEL', '(212) 825-2059'), ('UR
L', 'http://www.linkedin.com/in/stliferart'), ('ADDRESS1', '140 Riverside
 Blvd'), ('ADDRESS2', ''), ('CITY', 'New York'), ('ZIP', '10069')])
```

- Pop out the key and store the remaining dict

In [232]:

```
art_galleries = {}
with open("datasets/new_york_art_galleries.csv", 'r') as data_file:
    data = csv.DictReader(data_file, delimiter=",")
    for row in data:
        item = art_galleries.get(row["ZIP"], dict())
        item[row["NAME"]] = str(row["TEL"])
        art_galleries[row["ZIP"]] = item
```

In [233]:

```
galleries_10310 = art_galleries.pop('10310')
galleries_10310
```

Out[233]:

```
{'New Dorp Village Antiques Ltd': '(718) 815-2526'}
```

Part 2 - Step 2

- Pythonically iterate over the Dictionary

In [235]:

```
for zip_code, galleries in art_galleries.items():
    print(zip_code)
    print(galleries)
```

```
uchifritos': '(212) 598-4124', 'Damp Frog Productions Inc': '(212) 226-636
1', 'Harlan & Weaver Intaglio': '(212) 925-5421', 'Landy Fine Art': '(212)
505-3702', 'Downtown Music Gallery': '(212) 473-0043', 'Fowa Enterprises C
orp': '(212) 571-3838', 'Ludlow 38': '(212) 228-6848', 'One Twenty Eight':
'(212) 674-0244'}
11691
{'Photo & Art Gallery': '(718) 471-7840'}
11231
{'Picture This Corporate Art': '(718) 625-2722', 'Kentler International Dr
awing Space': '(718) 875-2098'}
10003
{'Place Des Vosges': '(212) 995-2899', 'Ritter-Antik Inc': '(212) 673-221
3', 'Savacou Galleries': '(212) 473-6904', 'Talwar Gallery': '(212) 673-30
96', 'Three East Third St Corp': '(212) 533-7749', 'Viewpoint Gallery':
'(212) 242-5478', 'Vincent Fremont Enterprises': '(212) 414-1881', 'Washin
gton Sq Outdoor Art Exhibit Inc': '(212) 982-6255', 'Whitfield James Fine
Art': '(212) 982-8050', '12 Below': '(212) 777-6777', 'Art Gallery & Custo
m Framing': '(212) 473-6802', 'Chinoh Art Gallery': '(212) 255-0377', 'Cur
atorial Art Advisory Service': '(212) 463-0586', 'Decor Art Gallery VII+':
'(212) 673-1366', 'Erotics Gallery': '(212) 633-2241', 'Goethe Institut Wa
```

Wrapping Up

- Use sets for uniqueness

In [236]:

```
cookies_eaten_today = ['chocolate chip', 'peanut butter',
                       'chocolate chip', 'oatmeal cream', 'chocolate chip']

types_of_cookies_eaten = set(cookies_eaten_today)

print(types_of_cookies_eaten)
```

```
{'oatmeal cream', 'chocolate chip', 'peanut butter'}
```

- difference() set method as at the end of Chapter 1

In [237]:

```
cookies_jason_ate.difference(cookies_hugo_ate)
```

Out[237]:

```
{'oatmeal cream', 'peanut butter'}
```

In [238]:

```
# EXERCISES
```

In [239]:

```
# Reading your Data with DictReader and Establishing your Data Containers
###############################################################################

# Create the CSV file: csvfile
csvfile = open("datasets/crime_sampler.csv","r")

# Create a dictionary that defaults to a list: crimes_by_district
crimes_by_district = defaultdict(list)

# Loop over a DictReader of the CSV file
for row in csv.DictReader(csvfile):
    # Pop the district from each row: district
    district = row.pop('District')
    # Append the rest of the data to the list for proper district in crimes_by_district
    crimes_by_district[district].append(row)
```

Brilliant work. You're now ready to analyze crime by district.

In [240]:

```
# Determine the Arrests by District by Year
############################################

# Loop over the crimes_by_district using expansion as district and crimes
for district, crimes in crimes_by_district.items():
    # Print the district
    print(district)

    # Create an empty Counter object: year_count
    year_count = Counter()

    # Loop over the crimes:
    for crime in crimes:
        # If there was an arrest
        if crime['Arrest'] == 'true':
            # Convert the Date to a datetime and get the year
            year = datetime.strptime(crime['Date'], '%m/%d/%Y %I:%M:%S %p').year
            # Increment the Counter for the year
            year_count[year] += 1

    # Print the counter
    print(year_count)
```

```
14
Counter({2016: 59, 2017: 8})
24
Counter({2016: 51, 2017: 10})
6
Counter({2016: 157, 2017: 32})
15
Counter({2016: 154, 2017: 16})
12
Counter({2016: 72, 2017: 9})
7
Counter({2016: 181, 2017: 27})
1
Counter({2016: 124, 2017: 15})
11
Counter({2016: 275, 2017: 53})
18
Counter({2016: 92, 2017: 17})
22
```

Interesting. It looked like most arrests took place in the 11th District.

In [259]:

```
# Unique Crimes by City Block
#############################

csvfile = open("datasets/crime_sampler.csv","r")
crimes_by_block = defaultdict(list)
for row in csv.DictReader(csvfile):
    block = row.pop('Block')
    # Append the rest of the data to the list for proper district in crimes_by_district
    crimes_by_block[block].append(row['Primary Type'])
```

In [262]:

```
# Create a unique list of crimes for the first block: n_state_st_crimes
n_state_st_crimes = set(crimes_by_block['001XX N STATE ST'])

# Print the list
print(n_state_st_crimes)

# Create a unique list of crimes for the second block: w_terminal_st_crimes
w_terminal_st_crimes = set(crimes_by_block['0000X W TERMINAL ST'])

# Print the list
print(w_terminal_st_crimes)

# Find the differences between the two blocks: crime_differences
crime_differences = n_state_st_crimes.difference(w_terminal_st_crimes)

# Print the differences
print(crime_differences)
```

```
{'DECEPTIVE PRACTICE', 'CRIMINAL DAMAGE', 'OTHER OFFENSE', 'BATTERY', 'CRIMI
NAL TRESPASS', 'THEFT', 'ASSAULT', 'ROBBERY'}
{'DECEPTIVE PRACTICE', 'CRIMINAL DAMAGE', 'OTHER OFFENSE', 'NARCOTICS', 'CRI
MINAL TRESPASS', 'PUBLIC PEACE VIOLATION', 'THEFT', 'ASSAULT'}
{'ROBBERY', 'BATTERY'}
```

Well done! There are some curious differences in crime between these two city blocks.

Final thoughts