|  |
| --- |
| Lists |
|  |
| In [1]: cookies = ['chocolate chip', 'peanut butter', 'oatmeal', 'sugar']  In [2]: cookies.append('Tirggel')  In [3]: print(cookies)  ['chocolate chip', 'peanut butter', 'oatmeal', 'sugar', 'Tirggel']  In [4]: print(cookies[2])  oatmeal |
| Combining Lists |
| In [1]: cakes = ['strawberry', 'vanilla']  In [2]: desserts = cookies + cakes  In [3]: print(desserts)  ['chocolate chip', 'peanut butter', 'oatmeal', 'sugar', 'Tirggel',  'strawberry', 'vanilla']  \*.extend() method merges a list into another list at the end |
| Finding Elements in a List ---(.index() method) |
| In [1]: position = cookies.index('sugar')  In [2]: print(position)  3  In [3]: cookies[3]  'sugar' |
| Removing Elements in a List. ---(.pop() method) |
| In [1]: name = cookies.pop(position)In [2]: print(name)  sugarIn [3]: print(cookies)  ['chocolate chip', 'peanut butter', 'oatmeal', 'Tirggel', 'Biscotti', 'digestive', 'fortune'] |
| Iterating ---(for loops) |
| In [1]: for cookie in cookies:  ...: print(cookie)  chocolate chip  … |
| Sorting ---(sorted() function) |
| In [2]: sorted\_cookies = sorted(cookies)  In [3]: print(sorted\_cookies)  ['Biscotti', 'Tirggel', 'chocolate chip', 'digestive', 'fortune', 'oatmeal'] |

|  |
| --- |
| Tuples |
| Zipping |
| In [1]: top\_pairs = zip(us\_cookies, in\_cookies)  In [2]: print(top\_pairs)  [('Chocolate Chip', 'Punjabi'), ('Brownies', 'Fruit Cake Rusk'),  ('Peanut Butter', 'Marble Cookies'), ('Oreos', 'Kaju Pista Cookies'),  ('Oatmeal Raisin', 'Almond Cookies')] |
| Unpacking |
| In [1]: us\_num\_1, in\_num\_1 = top\_pairs[0]  In [2]: print(us\_num\_1)  Chocolate Chip  In [3]: print(in\_num\_1)  Punjabi |
| More Unpacking in Loops |
| In [1]: for us\_cookie, in\_cookie in top\_pairs:  ...: print(in\_cookie)  ...: print(us\_cookie)  Punjabi  Chocolate Chip  Fruit Cake Rusk  Brownies  # ..etc.. |
| Enumerating positions |
| In [1]: 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')  # ..etc..  # print('Rank {}: {} and {}'.format(idx, girl\_name, boy\_name)) |
| Use zip(), enumerate(), or () to make tuples |
| In [1]: item = ('vanilla', 'chocolate')  In [2]: print(item)  ('vanilla', 'chocolate') |

|  |
| --- |
| Sets for unordered and unique data |
| Creating Sets (from a list) |
| In [1]: cookies\_eaten\_today = ['chocolate chip', 'peanut butter',  ...: 'chocolate chip', 'oatmeal cream', 'chocolate chip']  In [2]: types\_of\_cookies\_eaten = set(cookies\_eaten\_today)  In [3]: print(types\_of\_cookies\_eaten)  set(['chocolate chip', 'oatmeal cream', 'peanut butter']) |
| \* Kalem sayisi uzerinden seti olusturuyor. |
| Modifying Sets ---( .add() adds single elements) |
| In [1]: types\_of\_cookies\_eaten.add('biscotti')  In [2]: types\_of\_cookies\_eaten.add('chocolate chip')  In [3]: print(types\_of\_cookies\_eaten)  set(['chocolate chip', 'oatmeal cream', 'peanut butter', 'biscotti']) |
| Modifying Sets ---( .update() merges in another set or list) |
| In [4]: cookies\_hugo\_ate = ['chocolate chip', 'anzac']  In [5]: types\_of\_cookies\_eaten.update(cookies\_hugo\_ate)  In [6]: print(types\_of\_cookies\_eaten)  set(['chocolate chip', 'anzac', 'oatmeal cream', 'peanut butter',  'biscotti']) |
| \* Update, sadece farkli olanlari ekliyor. |
| Removing data from sets ---(discard() and pop()) |
| In [1]: types\_of\_cookies\_eaten.discard('biscotti')  In [2]: print(types\_of\_cookies\_eaten)  set(['chocolate chip', 'anzac', 'oatmeal cream', 'peanut butter',  'biscotti'])  In [3]: types\_of\_cookies\_eaten.pop()  'chocolate chip'  In [4]:types\_of\_cookies\_eaten.pop()  'anzac' |
| Set Operations – Similarities (.union() and .intersection()) |
| In [1]: cookies\_jason\_ate = set(['chocolate chip', 'oatmeal cream',  ...: 'peanut butter'])  In [2]: cookies\_hugo\_ate = set(['chocolate chip', 'anzac'])  In [3]: cookies\_jason\_ate.union(cookies\_hugo\_ate)  set(['chocolate chip', 'anzac', 'oatmeal cream', 'peanut butter'])  In [4]: cookies\_jason\_ate.intersection(cookies\_hugo\_ate)  set(['chocolate chip']) |
| Set Operations - Differences |
| In [1]: cookies\_jason\_ate.difference(cookies\_hugo\_ate)  set(['oatmeal cream', 'peanut butter'])  In [2]: cookies\_hugo\_ate.difference(cookies\_jason\_ate)  set(['anzac']) |

|  |
| --- |
| DICTIONARIES |
| **Creating and looping through dictionaries** |
| \* Created by dict() or {}  In [1]: art\_galleries = {}  In [2]: for name, zip\_code in galleries:  ...: art\_galleries[name] = zip\_code  In [3]: for name in art\_galleries:  ...: print(name)  Zwirner David Gallery  …. |
| **Safely finding by key (cont.)** |
| In [5]: art\_galleries.get('Louvre', 'Not Found')  Out[5]: 'Not Found'  In [6]: art\_galleries.get('Zarre Andre Gallery')  Out[6]: '10011' |
| **Working with nested data** |
| In [1]: art\_galleries.keys()  Out[1]: dict\_keys(['10021', '10013', '10001', …., '10128'])  In [2]: print(art\_galleries['10027'])  {"Paige's Art Gallery": '(212) 531-1577',  ….  'Inner City Art Gallery Inc': '(212) 368-4941'}  In [3]: art\_galleries['10027']['Inner City Art Gallery Inc']  Out[3]: '(212) 368-4941' |
| **Adding and extending dictionaries --- .update() function ---** |
| In [1]: print(galleries\_10007)  {'Nyabinghi Africian Gift Shop': '(212) 566-3336'}  In [2]: art\_galleries['10007'] = galleries\_10007  In [3]: galleries\_11234 = [('A J ARTS LTD', '(718) 763-5473'),  ...: ('Doug Meyer Fine Art', '(718) 375-8006'),  ...: ('Portrait Gallery', '(718) 377-8762')]  In [4]: art\_galleries['11234'].update(galleries\_11234)  In [5]: print(art\_galleries['11234'])  {'Portrait Gallery': '(718) 377-8762', 'A J ARTS LTD': '(718) 763-5473',  'Doug Meyer Fine Art': '(718) 375-8006'} |
| **Popping and deleting --- del ---- / --- .pop() ---** |
| In [1]: del art\_galleries['11234']  In [2]: galleries\_10310 = art\_galleries.pop('10310')  In [3]: print(galleries\_10310)  {'New Dorp Village Antiques Ltd': '(718) 815-2526'} |
| **Working with dictionaries more pythonically --- .items() ---** |
| .items() method returns an object we can iterate over |
| In [1]: for gallery, phone\_num in art\_galleries.items():  ...: print(gallery)  ...: print(phone\_num)  'Miakey Art Gallery'  '(718) 686-0788'  'Morning Star Gallery Ltd'  '(212) 334-9330'}  'New York Art Expo Inc'  '(212) 363-8280' |
| Checking dictionaries for data ---.get()’ten daha kullanisli imis |
| In [1]: '11234' in art\_galleries  Out[1]: False  In [2]: if '10010' in art\_galleries:  ...: print('I found: %s' % art\_galleries['10010'])  ...: else:  ...: print('No galleries found.')  I found: {'Nyabinghi Africian Gift Shop': '(212) 566-3336'} |
| # Exercise:  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]) |
| print(names.get(7))  print(type(names.get(100)))  print(names.get(105, 'Not Found')) |
| print(boy\_names.keys())  print(boy\_names[2013].keys())  for year in boy\_names:  print(year, boy\_names.get(2, 'Unknown')) |
| boy\_names[2011] = names\_2011  boy\_names[2012].update([(1,'Casey'), (2,'Aiden')])  for year in boy\_names:  for rank in sorted(boy\_names[year], reverse=True)[:1]:  if not rank:  print(year, 'No Data Available')  print(year, boy\_names[year].get(rank, 'Not Available')) |

|  |
| --- |
| Working with CSV files |
| CSV reader --- csv.reader() --- |
| \* reads a file object and returns the lines from the file as tuples |
| In [1]: import csv  In [2]: csvfile = open('ART\_GALLERY.csv', 'r')  In [3]: for row in csv.reader(csvfile):  ...: print(row)  ['NAME', 'the\_geom', 'TEL', 'URL', 'ADDRESS1', 'ADDRESS2', 'CITY', 'ZIP']  ["O'reilly William & Co Ltd",'POINT (-73.96273074561996 40.773800871637576)',  '(212) 396-1822', '52 E 76th St', '', 'New York', '10021']  In [4]: csvfile.close() |
| Creating a dictionary from a file --- .DictReader() --- |
| In [1]: import csv  In [2]: csvfile = open('ART\_GALLERY.csv', 'r')  In [3]: for row in csv.DictReader(csvfile):  ...: print(row)  OrderedDict([('NAME', 'Odyssia Gallery'),  ('the\_geom', 'POINT (-73.96269813635554 40.7618747512849)'),  ('TEL', '(212) 486-7338'),  ('URL', 'http://www.livevillage.com/newyork/art/odyssia-gallery.html'),  ('ADDRESS1', '305 E 61st St'),  ('ADDRESS2', ''),  ('CITY', 'New York'), ('ZIP', '10021')])  In [ |
| \* If data doesn't have a header row, you can pass in the column names |
|  |

|  |
| --- |
| **Collection Module** |
| **--- Counter() ---** |
| # Special dictionary used for counting data, measuring frequency |
| In [1]: from collections import Counter  In [2]: nyc\_eatery\_count\_by\_types = Counter(nyc\_eatery\_types)  In [3]: print(nyc\_eatery\_count\_by\_type)  Counter({'Mobile Food Truck': 114, 'Food Cart': 74, 'Snack Bar': 24,  'Specialty Cart': 18, 'Restaurant': 15, 'Fruit & Vegetable Cart': 4})  In [4]: print(nyc\_eatery\_count\_by\_types['Restaurant'])  15 |
| --- .most\_common() --- |
| # method returns the counter values in descending order |
| In [1]: print(nyc\_eataery\_count\_by\_types.most\_common(3))  [('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)] |
| **Dictionaries of unknown structure - defaultdict** |
| Dictionary Handling |
| In [1]: 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 [2]: print(eateries\_by\_park['M010'])  {'MOHAMMAD MATIN','PRODUCTS CORP.', 'Loeb Boathouse Restaurant',  'Nandita Inc.', 'SALIM AHAMED', 'THE NY PICNIC COMPANY',  'THE NEW YORK PICNIC COMPANY, INC.', 'NANDITA, INC.',  'JANANI FOOD SERVICE, INC.'} |
| --- defaultdict() --- |
| # Pass it a default type that every key will have even if it doesn't  currently exist. Works exactly like a dictionary |
| In [1]: from collections import defaultdict  In [2]: eateries\_by\_park = defaultdict(list)  In [3]: for park\_id, name in nyc\_eateries\_parks:  ...: eateries\_by\_park[park\_id].append(name)  In [4]: print(eateries\_by\_park['M010'])  {'MOHAMMAD MATIN','PRODUCTS CORP.', 'Loeb Boathouse Restaurant',  'Nandita Inc.', 'SALIM AHAMED', 'THE NY PICNIC COMPANY',  'THE NEW YORK PICNIC COMPANY, INC.', 'NANDITA, INC.',  'JANANI FOOD SERVICE, INC.'} |
| In [1]: from collections import defaultdict  In [2]: eatery\_contact\_types = defaultdict(int)  In [3]: for eatery in nyc\_eateries:  ...: if eatery.get('phone'):  ...: eatery\_contact\_types['phones'] += 1  ...: if eatery.get('website'):  ...: eatery\_contact\_types['websites'] += 1  In [4]: print(eatery\_contact\_types)  defaultdict(<class 'int'>, {'phones': 28, 'websites': 31}) |
| **--- OrderedDict() ---** |
| In [1]: from collections import OrderedDict  In [2]: nyc\_eatery\_permits = OrderedDict()  In [3]: for eatery in nyc\_eateries:  ...: nyc\_eatery\_permits[eatery['end\_date']] = eatery  In [4]: print(list(nyc\_eatery\_permits.items()[:3])  ('2029-04-28', {'name': 'Union Square Seasonal Cafe',  'location': 'Union Square Park', 'park\_id': 'M089',  'start\_date': '2014-04-29', 'end\_date': '2029-04-28',  'description': None, 'permit\_number': 'M89-SB-R', 'phone': '212-677-7818',  'website': 'http://www.thepavilionnyc.com/', 'type\_name': 'Restaurant'}) |
| **--- .popitem() ---** |
| # method returns items in reverse insertion order |
| In [1]: print(nyc\_eatery\_permits.popitem()) |
| # You can use the last=False keyword argument to return the items in insertion order |
| In [3]: print(nyc\_eatery\_permits.popitem(last=False)) |
| **namedtuple** |
| Creating a namedtuple |
| In [1]: from collections import namedtuple  In [2]: Eatery = namedtuple('Eatery', ['name', 'location', 'park\_id',  ...: 'type\_name'])  In [3]: eateries = []  In [4]: for eatery in nyc\_eateries:  ...: details = Eatery(eatery['name'],  ...: eatery['location'],  ...: eatery['park\_id'],  ...: eatery['type\_name'])  ...: eateries.append(details)  In [5]: print(eateries[0])  Eatery(name='Mapes Avenue Ballfields Mobile Food Truck',  location='Prospect Avenue, E. 181st Street',  park\_id='X289', type\_name='Mobile Food Truck') |
| Leveraging namedtuples |
| In [1]: for eatery in eateries[:3]:  ...: print(eatery.name)  ...: print(eatery.park\_id)  ...: print(eatery.location) |
| ############3 |
| # 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] |

|  |
| --- |
| DateTime |
| .strptime(). ----- # From string to datetime |
| In [1]: from datetime import datetime  In [2]: print(parking\_violations\_date)  06/11/2016  In [3]: date\_dt = datetime.strptime(parking\_violations\_date, '%m/%d/%Y')  In [4]: 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, ..., 2013, 2014, ..., 9998, 9999) |
| .strftime() ----- Datetime to String |
| In [1]: date\_dt.strftime('%m/%d/%Y')  Out[1]: '06/11/2016' |
| # isoformat() method outputs a datetime as an ISO standard string |
| In [1]: date\_dt.isoformat()  Out[1]: '2016-06-11T00:00:00' |
| Datetime Components |
| In [1]: daily\_violations = defaultdict(int)  In [2]: for violation in parking\_violations:  ...: violation\_date = datetime.strptime(violation[4], '%m/%d/%Y')  ...: daily\_violations[violation\_date.day] += 1  In [3]: print(sorted(daily\_violations.items()))  [(1, 80986), (2, 79831), (3, 74610), (4, 69555), (5, 68729), (6, 76232), …… (31, 44125)] |
| .now() / .utcnow() |
| In [1]: from datetime import datetime  In [2]: local\_dt = datetime.now() # pendulum.now(‘Asia/Tokyo’)  In [3]: print(local\_dt)  2017-05-05 12:30:00.740415  In [4]: utc\_dt = datetime.utcnow()  In [5]: print(utc\_dt)  2017-05-05 17:30:05.467221 |
| .astimezone() |
| In [1]: from pytz import timezone  In [2]: record\_dt = datetime.strptime('07/12/2016 04:39PM',  ...: '%m/%d/%Y %H:%M%p')  In [3]: ny\_tz = timezone('US/Eastern')  In [4]: la\_tz = timezone('US/Pacific')  In [5]: ny\_dt = record\_dt.replace(tzinfo=ny\_tz)  In [6]: la\_dt = ny\_dt.astimezone(la\_tz)  In [7]: print(ny\_dt)  2016-07-12 04:39:00-04:00  In [8]: print(la\_dt)  2016-07-12 01:39:00-07:00 |
| timedelta() |
| # Used to add or subtract a set amount of time from a datetime object |
| In [1]: from datetime import timedelta  In [2]: flashback = timedelta(days=90)  In [3]: print(record\_dt)  2016-07-12 04:39:00  In [4]: print(record\_dt - flashback)  2016-04-13 04:39:00  In [5]: print(record\_dt + flashback)  2016-10-10 04:39:00 |
| Datetime differences |
| # Use the - operator to calculate the difference |
| In [1]: time\_diff = record\_dt - record2\_dt  In [2]: type(time\_diff)  Out[2]: datetime.timedelta  In [3]: print(time\_diff)  0:00:04 |
| Parsing time with pendulum ---- .parse() |
| # .parse() will attempt to convert a string to a pendulum datetime  object without the need of the format string |
| In [1]: import pendulum  In [2]: occurred = violation[4] + ' ' + violation[5] +'M'  In [3]: occurred\_dt = pendulum.parse(occurred, tz='US/Eastern')  In [4]: print(occured\_dt)  '2016-06-11T14:38:00-04:00' |
| .in\_timezone() |
| # .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 [1]: print(violation\_dts)  [<Pendulum [2016-06-11T14:38:00-04:00]>,  <Pendulum [2016-04-25T14:09:00-04:00]>,  <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 [2]: 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 [3]: print(pendulum.now('Asia/Tokyo'))  <Pendulum [2017-05-06T08:20:40.104160+09:00]> |
| # .in\_XXX() methods provide the difference in a chosen metric  # .in\_words() provides the difference in a nice expresive form |
| In [1]: diff = violation\_dts[3] - violation\_dts[2]  In [2]: diff  Out[2]: <Period [2016-04-26T07:09:00-04:00 -> 2016-04-23T07:49:00-04:00]>  In [3]: print(diff.in\_words())  '2 days 23 hours 20 minutes'  In [4]: print(diff.in\_days())  2  In [5]: print(diff.in\_hours())  71 |
| ############ |
| import pendulum  tokyo\_dt = pendulum.now('Asia/Tokyo')  la\_dt = tokyo\_dt.in\_timezone('America/Los\_Angeles')  print(la\_dt.to\_iso8601\_string()) |
| for start\_date, end\_date in date\_ranges:  start\_dt = pendulum.parse(start\_date, strict = False)  end\_dt = pendulum.parse(end\_date, strict = False)  print(end\_dt, start\_dt)  diff\_period = end\_dt - start\_dt  print(diff\_period.in\_days()) |

|  |
| --- |
| Case Study |
| import csv  csvfile = open('crime\_sampler.csv', 'r')  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) |
| #################### |
| # 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 item in crime\_data:    # Convert the first element of each item into a Python Datetime Object: date  date = datetime.strptime(item[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)) |
| ############### |
| # 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 values list  locations\_by\_month[date.month].append(row[4])    # Print the dictionary  print(locations\_by\_month) |
| ############# |
| # Import Counter from collections  from collections import Counter  # Loop over the items from locations\_by\_month using tuple expansion of the month and locations  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)) |
| ################### |
| # Create the CSV file: csvfile  csvfile = open('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) |
| #################### |
| # 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) |
| # Create a unique list of crimes for the first block: n\_state\_st\_crimes  n\_state\_st\_crimes = set('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('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) |
| ############### |
| # 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) |
|  |
|  |
|  |
|  |
|  |