DATA TYPES FOR DATA SCIENCE

Introduction and lists

Jason Myers
Instructor

Data types

- Data type system sets the stage for the capabilities of the language
- Understanding data types empowers you as a data scientist

Container sequences

- Hold other types of data
- Used for aggregation, sorting, and more
- Can be mutable (list, set) or immutable (tuple)
- Iterable

Lists

- Hold data in order it was added
- Mutable
- Index

```
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

- Using operators, you can combine two lists into a new one

```
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 and Removing Elements in a List

- .index() method locates the position of a data element in a list

```
In [1]: position = cookies.index('sugar')

In [2]: print(position)
3

In [3]: cookies[3]
'sugar'
```

- .pop() method removes an item from a list and allows you to save it

```
In [1]: name = cookies.pop(position)

In [2]: print(name)
sugar

In [3]: print(cookies)
['chocolate chip', 'peanut butter', 'oatmeal', 'Tirggel',
'Biscotti', 'digestive', 'fortune']
```

Iterating and Sorting

- for loops are the most common way of interating over a list

```
In [1]: for cookie in cookies:
   ...:     print(cookie)
chocolate chip
peanut butter
oatmeal
Tirggel
Biscotti
digestive
fortune
```

- sorted() function sorts data in numerical or alphabetical order and returns a new list

```
In [1]: print(cookies)
['chocolate chip', 'oatmeal', 'Tirggel', 'Biscotti', 'digestive', 'fortune']

In [2]: sorted_cookies = sorted(cookies)

In [3]: print(sorted_cookies)
['Biscotti', 'Tirggel', 'chocolate chip', 'digestive', 'fortune', 'oatmeal']
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Meet the Tuples

Jason Myers
Instructor

Tuple, Tuple

- Hold data in order
- Index
- *Immutable*
- Pairing
- Unpackable

Zipping and Unpacking

- Tuples are commonly created by zipping lists together with zip()
- Two lists: us_cookies, in_cookies

```
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 tuples is a very expressive way for working with data

```
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

- Unpacking is especially powerful 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

- Another useful tuple creation method is the enumerate() function
- Enumeration is used in loops to return the position and the data in that position while looping

```
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..
```

Be careful when making tuples

- Use zip(), enumerate(), or () to make tuples

```
In [1]: item = ('vanilla', 'chocolate')

In [2]: print(item)
('vanilla', 'chocolate')
```

- Beware of tailing commas!

```
In [1]: item2 = 'butter',

In [2]: print(item2)
('butter',)
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Sets for unordered and unique data

Jason Myers
Instructor

Set

- Unique
- Unordered
- Mutable
- Python's implementation of Set Theory from Mathematics

Creating Sets

- Sets are created 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'])
```

Modifying Sets

- .add() adds single elements
- .update() merges in another set or list

```
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'])

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'])
```

Removing data from sets

- .discard() safely removes an element from the set by value
- .pop() removes and returns an arbitrary element from the set

  (KeyError when empty)

```
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() set method returns a set of all the names (|)
- .intersection() method identifies overlapping data (&)

```
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

- .difference() method identifies data present in the set on which the method was used that is not in the arguments (-)
- Target is important!

```
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'])
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Using dictionaries

Jason Myers
Instructor

Creating and looping through dictionaries

- Hold data in key/value pairs
- Nestable (use a dictionary as the value of a key within a dictionary)
- Iterable
- 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
Zwirner & Wirth
Zito Studio Gallery
Zetterquist Galleries
Zarre Andre Gallery
```

Safely finding by key

```
In [4]: art_galleries['Louvre']
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-1-4f51c265f287> in <module>()
----> 1 art_galleries['Louvre']

KeyError: 'Louvre'
```

- Getting a value from a dictionary is done using the key as an index
- If you ask for a key that does not exist that will stop your program from running in a KeyError

Safely finding by key (cont.)

- .get() method allows you to safely access a key without error or exception handling
- If a key is not in the dictionary, .get() returns None by default or you can supply a value to return

```
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', '10009', '10011', '10022',
   ...: '10027', '10019', '11106', '10128'])

In [2]: print(art_galleries['10027'])
{"Paige's Art Gallery": '(212) 531-1577',
'Triple Candie': '(212) 865-0783',
'Africart Motherland Inc': '(212) 368-6802',
'Inner City Art Gallery Inc': '(212) 368-4941'}

In [3]: art_galleries['10027']['Inner City Art Gallery Inc']
Out[3]: '(212) 368-4941'
```

- The .keys() method shows the keys for a given dictionary
- Common way to deal with repeating data structures
- Can be accessed using multiple indices or the .get() method

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Altering dictionaries

Jason Myers
Instructor

Adding and extending dictionaries

- Assignment to add a new key/value to a dictionary
- .update() method to update a dictionary from another dictionary, tuples or keywords

```
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 from dictionaries

- del instruction deletes a key/value
- .pop() method safely removes a key/value from a dictionary.

```
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'}
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Pythonically using dictionaries

Jason Myers
Instructor

Working with dictionaries more pythonically

- .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() does a lot of work to check for a key
- in operator is much more efficient and clearer

```
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'}
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Working with CSV files

Jason Myers
Instructor

CSV Files

```
NAME,TEL,ADDRESS1,ADDRESS2,CITY,ZIP
O'reilly William & Co Ltd,(212) 396-1822,52 E 76th St,,New York,10021
```

Reading from a file using CSV reader

- Python csv module
- open() function provides a variable that represents a file, takes a path and a mode
- csv.reader() reads a file object and returns the lines from the file as tuples
- .close() method closes file objects

```
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

- Often we want to go from CSV file to dictionary
- DictReader does just that
- If data doesn't have a header row, you can pass in the column names

```
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 [4]: csvfile.close()
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Counting made easy

Jason Myers
Instructor

Collections Module

- Part of Standard Library
- Advanced data containers

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
```

Counter to find the most common

- .most_common() method returns the counter values in descending order

```
In [1]: print(nyc_eatery_count_by_types.most_common(3))
[('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)]
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Dictionaries of unknown structure - defaultdict

Jason Myers
Instructor

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.'}
```

Using 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.'}
```

defaultdict (cont.)

```
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})
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Maintaining Dictionary Order with OrderedDict

Jason Myers
Instructor

Order in Python dictionaries

- Python version < 3.6 NOT ordered
- Python version > 3.6 ordered

Getting started with 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'})
```

OrderedDict power feature

- .popitem() method returns items in reverse insertion order

```
In [1]: print(nyc_eatery_permits.popitem())
('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'})

In [2]: print(nyc_eatery_permits.popitem())
('2027-03-31', {'name': 'Dyckman Marina Restaurant',
'location': 'Dyckman Marina Restaurant', 'park_id': 'M028',
'start_date': '2012-04-01', 'end_date': '2027-03-31',
'description': None, 'permit_number': 'M28-R', 'phone': None,
'website': None, 'type_name': 'Restaurant'})
```

OrderedDict power feature (2)

- You can use the last=False keyword argument to return the items in insertion order

```
In [3]: print(nyc_eatery_permits.popitem(last=False))
('2012-12-07', {'name': 'Mapes Avenue Ballfields Mobile Food Truck',
'location': 'Prospect Avenue, E. 181st Street', 'park_id': 'X289',
'start_date': '2009-07-01', 'end_date': '2012-12-07',
'description': None, 'permit_number': 'X289-MT', 'phone': None,
'website': None, 'type_name': 'Mobile Food Truck'})
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

namedtuple

Jason Myers
Instructor

What is a namedtuple?

- A tuple where each position (column) has a name
- Ensure each one has the same properties
- Alternative to a pandas DataFrame row

Creating a namedtuple

- Pass a name and a list of fields

```
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

- Each field is available as an attribute of the namedtuple

```
In [1]: for eatery in eateries[:3]:
   ...:     print(eatery.name)
   ...:     print(eatery.park_id)
   ...:     print(eatery.location)

Mapes Avenue Ballfields Mobile Food Truck
X289
Prospect Avenue, E. 181st Street

Claremont Park Mobile Food Truck
X008
East 172 Street between Teller & Morris avenues

Slattery Playground Mobile Food Truck
X085
North corner of Valenti Avenue & East 183 Street
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

There and Back Again a DateTime Journey

Jason Myers
Instructor

From string to datetime

- 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 [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
```

Time Format Strings

Directive	Meaning	Example
%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

Full list available in the Python documentation

Datetime to String

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

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Working with Datetime Components and current time

Jason Myers
Instructor

Datetime Components

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

```
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),
(7, 82477), (8, 72472), (9, 80415), (10, 75387), (11, 73287), (12, 74614),
(13, 75278), (14, 81803), (15, 79122), (16, 80692), (17, 73677), (18, 75927),
(19, 80813), (20, 80992), (21, 78138), (22, 81872), (23, 78104), (24, 63490),
(25, 78898), (26, 78830), (27, 80164), (28, 81954), (29, 80585), (30, 65864),
(31, 44125)]
```

What is the deal with now

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

```
In [1]: from datetime import datetime

In [2]: local_dt = datetime.now()

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
```

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

Timezones in action

```
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
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Time Travel (Adding and Subtracting Time)

Jason Myers
Instructor

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 [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
- Returns a timedelta with 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
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

HELP! Libraries to make it easier

Jason Myers
Instructor

Parsing time with pendulum

- .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'
```

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 [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]>
```

Humanizing differences

- .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
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Case Study - Counting Crimes

Jason Myers
Instructor

Data Set Overview

```
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
```

- Chicago Open Data Portal https://data.cityofchicago.org/

Part 1 - Step 1

- Read data from CSV

```
In [1]: import csv

In [2]: csvfile = open('ART_GALLERY.csv', 'r')

In [3]: for row in csv.reader(csvfile):
   ...:     print(row)
```

Part 1 - Step 2

- Create and use a Counter with a slight twist

```
In [1]: from collections import Counter

In [2]: nyc_eatery_count_by_types = Counter(nyc_eatery_types)
```

- Use date parts for Grouping like in Chapter 4

```
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
```

Part 1 - Step 3

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

```
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)
```

Part 1 - Final

- Find 5 most common locations for crime each month.

```
In [1]: print(nyc_eatery_count_by_types.most_common(3))
[('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)]
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Case Study - Crimes by District and Differences by Block

Jason Myers
Instructor

Part 2 - Step 1

- Read in the CSV data as a dictionary

```
In [1]: import csv

In [2]: csvfile = open('ART_GALLERY.csv', 'r')

In [3]: for row in csv.DictReader(csvfile):
   ...:     print(row)
```

- Pop out the key and store the remaining dict

```
In [1]: galleries_10310 = art_galleries.pop('10310')
```

Part 2 - Step 2

- Pythonically iterate over the Dictionary

```
In [1]: for zip_code, galleries in art_galleries.items():
   ...:     print(zip_code)
   ...:     print(galleries)
```

Wrapping Up

- Use sets for uniqueness

```
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'])
```

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

```
In [1]: cookies_jason_ate.difference(cookies_hugo_ate)
set(['oatmeal cream', 'peanut butter'])
```

DATA TYPES FOR DATA SCIENCE

Let's practice!

DATA TYPES FOR DATA SCIENCE

Final thoughts

Jason Myers
Instructor