# Counting made easy

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

## Collections Module

- Part of Standard Library
- Advanced data containers

## Counter

- Special dictionary used for counting data, measuring frequency

```
from collections import Counter
nyc_eatery_count_by_types = Counter(nyc_eatery_types)
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}
```

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

```
15
```

## Counter to find the most common

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

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

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Dictionaries of unknown structure - defaultdict

DATA TYPES FOR DATA SCIENCE IN PYTHON

**Jason Myers**
Instructor

## Dictionary Handling

```python
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)
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

```
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'])
```

```
{'MOHAMMAD MATIN','PRODUCTS CORP.', 'Loeb Boathouse Restaurant',
'Nandita Inc.', 'SALIM AHAMED', 'THE NY PICNIC COMPANY',
'THE NEW YORK PICNIC COMPANY, INC.', 'NANDITA, INC.', ...}
```

## defaultdict (cont.)

```python
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': 28, 'websites': 31})
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Maintaining Dictionary Order with OrderedDict

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

## Order in Python dictionaries

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

## Getting started with OrderedDict

```python
from collections import OrderedDict

nyc_eatery_permits = OrderedDict()

for eatery in nyc_eateries:
    nyc_eatery_permits[eatery['end_date']] = eatery

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', ...})
```

# OrderedDict power feature

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

```
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', ...})
```

```
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', ...})
```

# OrderedDict power feature (2)

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

```
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'})
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# namedtuple

DATA TYPES FOR DATA SCIENCE IN PYTHON

**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

```
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)
```

## Print the first element

```python
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

```
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 ...
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# There and Back Again a DateTime Journey

DATA TYPES FOR DATA SCIENCE IN PYTHON

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

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

```
06/11/2016
```

## Parsing strings into datetimes

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

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

```
date_dt.strftime('%m/%d/%Y')
```

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

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

```
date_dt.isoformat()
```

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Working with Datetime Components and current time

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

## Datetime Components

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

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

    daily_violations[violation_date.day] += 1
```

## Datetime Components - Results

```
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

```
from datetime import datetime

local_dt = datetime.now()

print(local_dt)
```

```
2017-05-05 12:30:00.740415
```

## What is the deal with utcnow

```
utc_dt = datetime.utcnow()

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

```python
from pytz import timezone

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

ny_tz = timezone('US/Eastern')

a_tz = timezone('US/Pacific')

ny_dt = record_dt.replace(tzinfo=ny_tz)

la_dt = ny_dt.astimezone(la_tz)
```

## Timezones in action - results

```
print(ny_dt)
```

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

```
print(la_dt)
```

```
2016-07-12 01:39:00-07:00
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Time Travel (Adding and Subtracting Time)

DATA TYPES FOR DATA SCIENCE IN PYTHON

**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

```python
from datetime import timedelta

flashback = timedelta(days=90)

print(record_dt)
```

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

## Adding and subtracting timedeltas

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

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

```
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

```
time_diff = record_dt - record2_dt

type(time_diff)
```

```
datetime.timedelta
```

```
print(time_diff)
```

```
0:00:04
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# HELP! Libraries to make it easier

DATA TYPES FOR DATA SCIENCE IN PYTHON

**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

```
import pendulum

occurred = violation[4] + ' ' + violation[5] +'M'

occurred_dt = pendulum.parse(occurred, tz='US/Eastern')

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

```
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]>]
```

## More timezone hopping

```python
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
```

```python
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

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

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

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

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

## More human than human

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

```
2
```

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

```
71
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Case Study - Counting Crimes

DATA TYPES FOR DATA SCIENCE IN PYTHON

**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

```
import csv

csvfile = open('ART_GALLERY.csv', 'r')

for row in csv.reader(csvfile):
    print(row)
```

# Part 1 - Step 2

- Create and use a Counter with a slight twist

```
from collections import Counter

nyc_eatery_count_by_types = Counter(nyc_eatery_types)
```

- Use date parts for Grouping like in Chapter 4

```
daily_violations = defaultdict(int)

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.

```
from collections import defaultdict

eateries_by_park = defaultdict(list)

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.

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

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Case Study - Crimes by District and Differences by Block

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

# Part 2 - Step 1

- Read in the CSV data as a dictionary

```
import csv

csvfile = open('ART_GALLERY.csv', 'r')

for row in csv.DictReader(csvfile):
    print(row)
```

- Pop out the key and store the remaining dict

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

# Part 2 - Step 2

- Pythonically iterate over the Dictionary

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

## Wrapping Up

- Use sets for uniqueness

```
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)
```

```
set(['chocolate chip', 'oatmeal cream', 'peanut butter'])
```

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

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Final thoughts

DATA TYPES FOR DATA SCIENCE IN PYTHON

**Jason Myers**
Instructor

## Congratulations

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Introduction and lists

DATA TYPES FOR DATA SCIENCE IN PYTHON

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

## Accessing single items in list

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

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

```
print(cookies)
```

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

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

```
sugar
```

# Combining Lists

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

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

desserts = cookies + cakes

print(desserts)
```

```
['chocolate chip', 'peanut butter', 'sugar', 'Tirggel',
'strawberry', 'vanilla']
```

- `.extend()` method merges a list into another list at the end

# Finding Elements in a List

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

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


print(position)
```

```
3
```

```
cookies[3]
```

```
'sugar'
```

## Removing Elements in a List

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

```
name = cookies.pop(position)


print(name)
```

```
sugar
```

```
print(cookies)
```

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

## Iterating over lists

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

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

```
chocolate chip
peanut butter
Tirggel
```

## Sorting lists

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

```
print(cookies)
```

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

```
sorted_cookies = sorted(cookies)

print(sorted_cookies)
```

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Meet the Tuples

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

## Tuple, Tuple

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

# Zipping tuples

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

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

```
[('Chocolate Chip', 'Punjabi'), ('Brownies', 'Fruit Cake Rusk'),
('Peanut Butter', 'Marble Cookies'), ('Oreos', 'Kaju Pista Cookies')
('Oatmeal Raisin', 'Almond Cookies')]
```

# Unpacking tuples

- Unpacking tuples is a very expressive way for working with data

```
us_num_1, in_num_1 = top_pairs[0]

print(us_num_1)
```

```
Chocolate Chip
```

```
print(in_num_1)
```

```
Punjabi
```

## More Unpacking in Loops

- Unpacking is especially powerful in loops

```
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

```
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

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

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

- Beware of tailing commas!

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

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

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Sets for unordered and unique data

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

## Set

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

## Creating Sets

- Sets are created from a list

```
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)
```

```
set(['chocolate chip', 'oatmeal cream', 'peanut butter'])
```

## Modifying Sets

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

```
types_of_cookies_eaten.add('biscotti')

types_of_cookies_eaten.add('chocolate chip')

print(types_of_cookies_eaten)
```

```
set(['chocolate chip', 'oatmeal cream', 'peanut butter', 'biscotti']
```

## Updating Sets

```python
cookies_hugo_ate = ['chocolate chip', 'anzac']

types_of_cookies_eaten.update(cookies_hugo_ate)

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)

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

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

```
types_of_cookies_eaten.pop()
types_of_cookies_eaten.pop()
```

```
'chocolate chip'
'anzac'
```

## Set Operations - Similarities

- `.union()` set method returns a set of all the names ( `or` )
- `.intersection()` method identifies overlapping data ( `and` )

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

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

```
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!

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

```
set(['oatmeal cream', 'peanut butter'])
```

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

```
set(['anzac'])
```

# Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Using dictionaries

DATA TYPES FOR DATA SCIENCE IN PYTHON

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 {}

```
art_galleries = {}


for name, zip_code in galleries:
    art_galleries[name] = zip_code
```

## Printing in the loop

```python
for name in art_galleries:
    print(name)
```

```
Zwirner David Gallery
Zwirner & Wirth
Zito Studio Gallery
Zetterquist Galleries
Zarre Andre Gallery
```

# Safely finding by key

```
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

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

```
'Not Found'
```

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

```
'10011'
```

# Working with nested dictionaries

```
art_galleries.keys()
```

```
dict_keys(['10021', '10013', '10001', '10009', '10011',
   ...: '10022', '10027', '10019', '11106', '10128'])
```

```
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'}
```

- The `.keys()` method shows the keys for a given dictionary

# Accessing nested data

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

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

- Common way to deal with repeating data structures
- Can be accessed using multiple indices or the `.get()` method

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Altering dictionaries

DATA TYPES FOR DATA SCIENCE IN PYTHON

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

```
print(galleries_10007)
```

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

```
art_galleries['10007'] = galleries_10007
```

## Updating a dictionary

```python
galleries_11234 = [
  ('A J ARTS LTD', '(718) 763-5473'),
  ('Doug Meyer Fine Art', '(718) 375-8006'),
  ('Portrait Gallery', '(718) 377-8762')]
art_galleries['11234'].update(galleries_11234)
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.

```
del art_galleries['11234']

galleries_10310 = art_galleries.pop('10310')

print(galleries_10310)
```

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

# Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Pythonically using dictionaries

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

# Working with dictionaries more pythonically

- `.items()` method returns an object we can iterate over

```
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

```
'11234' in art_galleries
```

```
False
```

```
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'}
```

## Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

# Working with CSV files

DATA TYPES FOR DATA SCIENCE IN PYTHON

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,1002
```

# 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

```
import csv

csvfile = open('ART_GALLERY.csv', 'r')

for row in csv.reader(csvfile):
    print(row)
```

# Reading from a CSV - Results

```
['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']
```

```
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

```
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'), ...
```

# Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON