THE UNIVERSITY OF CHICAGO

MASTERS IN COMPUTATIONAL SOCIAL SCIENCE
THE UNIVERSITY OF CHICAGO

MACS 30111

Dictionaries

Misc

- Today:
  - We will use information from TT3: https://classes.ssd.uchicago.edu/macss/macs30121/modules/tt/tt3.html
  - Materials: https://classroom.github.com/a/VZDXJhdo

Virtual environments

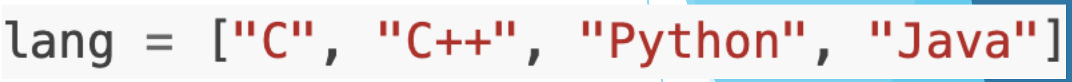
- Setting up and managing: conda vs venv
- Installs: pip vs conda

Topics:

- Introduction to Python dictionaries
- Useful dictionary operations
- Constructing dictionaries
- Iterating over dictionaries
- Accumulation
- Data structure and time complexity
- Sets

Review Lists:

- List creation and basic usage



```
lang = ["C", "C++", "Python", "Java"]
```

- List iteration
- Adding, removing elements from a list
- List slicing
- Other operations
- Tuples
- Strings
- List Comprehensions
- Lists in Memory

Topics:

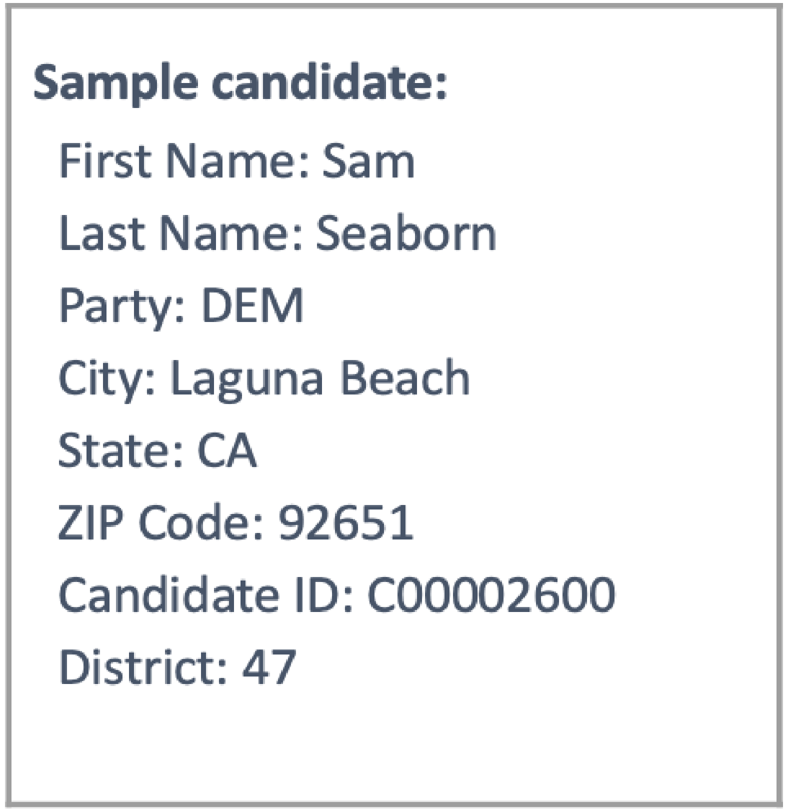
- ❑ **Introduction to Python dictionaries**
- ❑ Useful dictionary operations
- ❑ Constructing dictionaries
- ❑ Iterating over dictionaries
- ❑ Accumulation
- ❑ Dictionaries as objects
- ❑ Data structure and time complexity
- ❑ Sets

Dictionaries

- What do we know?
- What are they like?
- What are they NOT like?
- When would we use them?

Representing candidates using a *dictionary*

- Map keys to values
- Each value is associated with a **unique** key rather than a position in a sequence



Sample candidate:

First Name: Sam

Last Name: Seaborn

Party: DEM

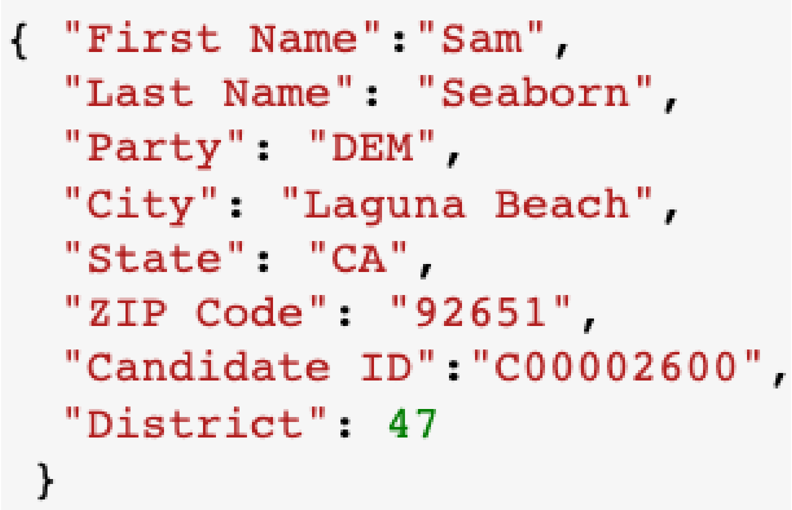
City: Laguna Beach

State: CA

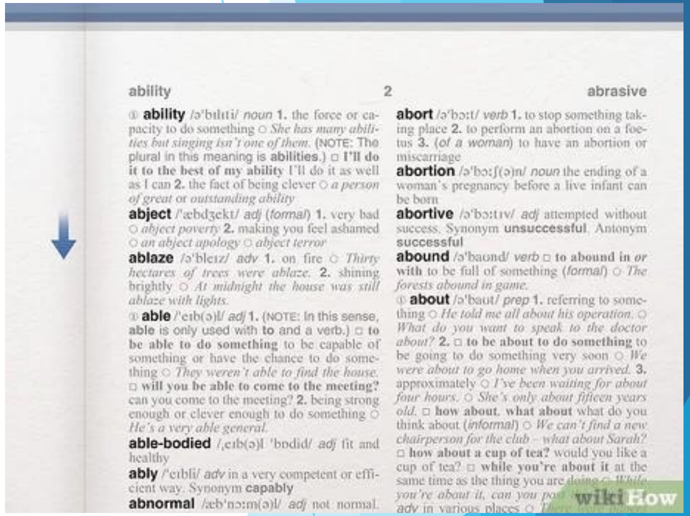
ZIP Code: 92651

Candidate ID: C00002600

District: 47



```
{ "First Name":"Sam",
  "Last Name": "Seaborn",
  "Party": "DEM",
  "City": "Laguna Beach",
  "State": "CA",
  "ZIP Code": "92651",
  "Candidate ID":"C00002600",
  "District": 47
 }
```



ability 2 abrasive

① **ability** /ə'bɪlɪti/ *noun* **1.** the force or capacity to do something ○ *She has many abilities but singing isn't one of them.* (NOTE: The plural in this meaning is **abilities**.) □ **I'll do it to the best of my ability** I'll do it as well as I can **2.** the fact of being clever ○ *a person of great or outstanding ability*

abject /'æbdʒekt/ *adj* (*formal*) **1.** very bad ○ *abject poverty* **2.** making you feel ashamed ○ *an abject apology* ○ *abject terror*

ablaze /ə'bleɪz/ *adv* **1.** on fire ○ *Thirty hectares of trees were ablaze.* **2.** shining brightly ○ *At midnight the house was still ablaze with lights.*

① **able** /'eɪb(ə)l/ *adj* **1.** (NOTE: In this sense, **able** is only used with **to** and a verb.) □ **to be able to do something** to be capable of something or have the chance to do something ○ *They weren't able to find the house.* □ **will you be able to come to the meeting?** can you come to the meeting? **2.** being strong enough or clever enough to do something ○ *He's a very able general.*

able-bodied /ˌeɪb(ə)l 'bɒdid/ *adj* fit and healthy

ably /'eɪbli/ *adv* in a very competent or efficient way. Synonym **capably**

abnormal /æb'nɔːm(ə)l/ *adj* not normal.

abort /ə'bɔːt/ *verb* **1.** to stop something taking place **2.** to perform an abortion on a foetus **3.** (*of a woman*) to have an abortion or miscarriage

abortion /ə'bɔːʃ(ə)n/ *noun* the ending of a woman's pregnancy before a live infant can be born

abortive /ə'bɔːtɪv/ *adj* attempted without success. Synonym **unsuccessful**. Antonym **successful**

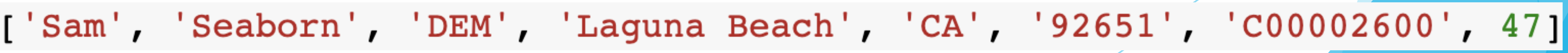
abound /ə'baʊnd/ *verb* □ **to abound in** *or* **with** to be full of something (*formal*) ○ *The forests abound in game.*

① **about** /ə'baʊt/ *prep* **1.** referring to something ○ *He told me all about his operation.* ○ *What do you want to speak to the doctor about?* **2.** □ **to be about to do something** to be going to do something very soon ○ *We were about to go home when you arrived.* **3.** approximately ○ *I've been waiting for about four hours.* ○ *She's only about fifteen years old.* □ **how about, what about** what do you think about (*informal*) ○ *We can't find a new chairperson for the club – what about Sarah?* □ **how about a cup of tea?** would you like a cup of tea? □ **while you're about it** at the same time as the thing you are doing ○ *While you're about it, can you po*

adv in various places ○ *The*

wikiHow

Coding practice: 2.2

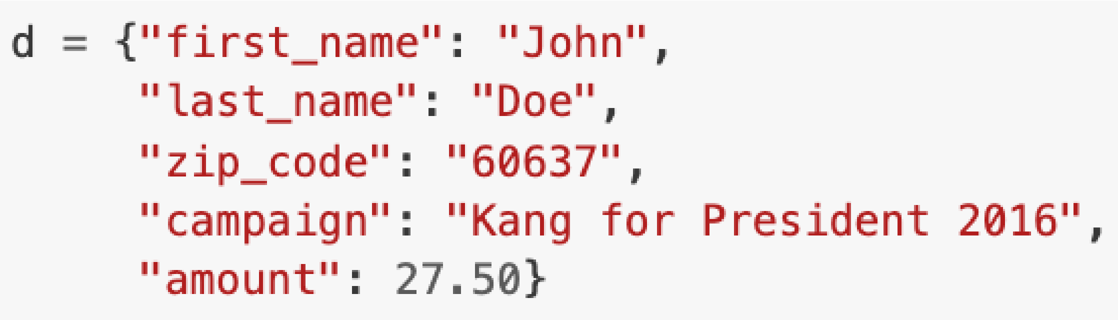


```
['Sam', 'Seaborn', 'DEM', 'Laguna Beach', 'CA', '92651', 'C00002600', 47]
```

Topics:

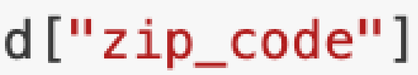
- Introduction to Python dictionaries
- **Useful dictionary operations**
- Constructing dictionaries
- Iterating over dictionaries
- Accumulation
- Dictionaries as objects
- Data structure and time complexity
- Sets

Access information in a dictionary



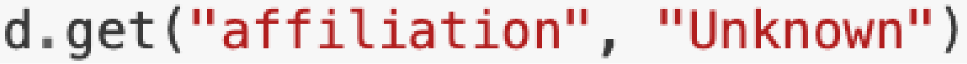
```
d = {"first_name": "John",
     "last_name": "Doe",
     "zip_code": "60637",
     "campaign": "Kang for President 2016",
     "amount": 27.50}
```

- subscript notation



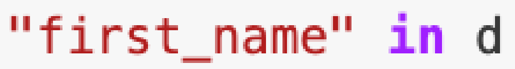
```
d["zip_code"]
```

- *get* method



```
d.get("affiliation", "Unknown")
```

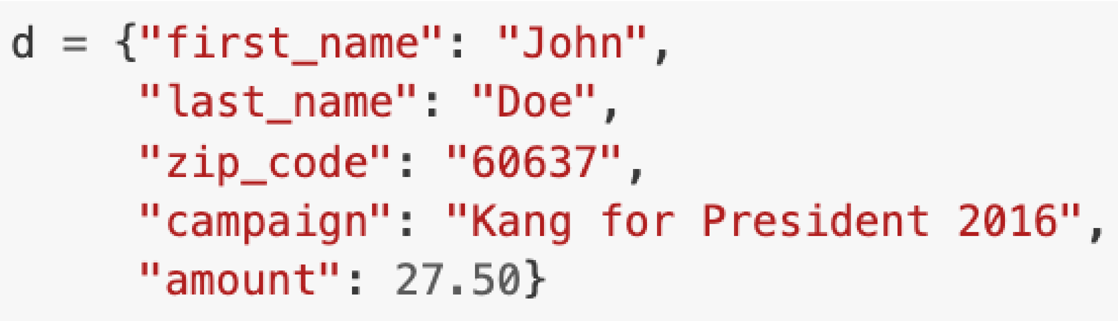
- *in* operator



```
"first_name" in d
```

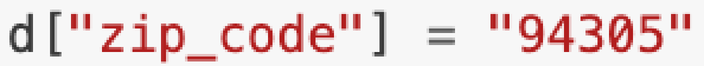
Coding practice: 2.2.1

Updating dictionaries

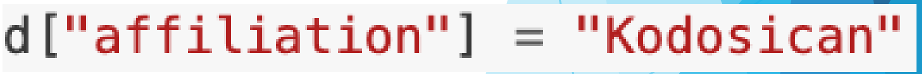


```
d = {"first_name": "John",
     "last_name": "Doe",
     "zip_code": "60637",
     "campaign": "Kang for President 2016",
     "amount": 27.50}
```

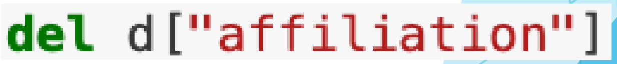
- Updating the value associated with a key
- Adding new key value pairs
- Remove an entry



```
d["zip_code"] = "94305"
```



```
d["affiliation"] = "Kodosican"
```



```
del d["affiliation"]
```

Coding practice: 2.2.1

Dictionaries

```
d = {"first_name": "John", "last_name": "Doe", "zip_code": "60637",
"campaign": "Kang for President 2016", "amount": 27.50}
```

```
d.get("first_name")
```

What is the difference?

```
  d.get("affiliation") vs d.get("affiliation", "Unknown")
```

TT3: all fun, all the time

Pull up all materials

Exercise: TT3

- Load your files:

```
import json
CFPB_16 = json.load(open("cfpb16_1000.json"))
```

(alternative: import cfpb)

- What is CFPB_16? What structure does it have?

Topics:

- Introduction to Python dictionaries
- Useful dictionary operations
- Constructing dictionaries
- **Iterating over dictionaries**
- Accumulation
- Dictionaries as objects
- Data structure and time complexity
- Sets

Iterating over dictionaries

- ***keys()*** yields a list-like object with the keys in a dictionary
- ***values( )*** yields a list-like object with the values in a dictionary
- ***items( )*** yields a list-like object with key/value tuples from a dictionary

Used in conjunction with for loops to iterate over dictionaries.

Coding practice: 2.2.1

TT3

Party continues

TT3: PARTY!

- Look over the TT3 code if you have not done so already
- Create a map of the functions.
- Draw lines to show where / how functions are called / used by others

Exercise: TT3

- Get the keys from cfpb.CFPB_16 or CFPB_16

TT3

Party continues

Prepping Task 1

- **Task 1:** In cfpb.py, write a function
  - **def** find_companies(complaints):
- that **takes a list of complaints and returns a list** (**or set** — see above) of the companies that received at least one complaint.
- Remember: we’ve included a variable called CFPB_16 that contains information from 1000 complaints in 2016. You will be using this variable when testing these functions, as shown below.

Misc

- *Today:*
  - *We will use information from TT3: https://classes.ssd.uchicago.edu/macss/macs30121/modules/tt/tt3.html*
  - *Materials: https://classroom.github.com/a/VZDXJhdo*
- *Midterm exam:*
- Notecards:
  - What questions do you have for this week’s content?
  - What part of learning Python has been your favorite so far?
  - What has been the most challenging?
  - What do you want us to focus on for Thursday?

Summary

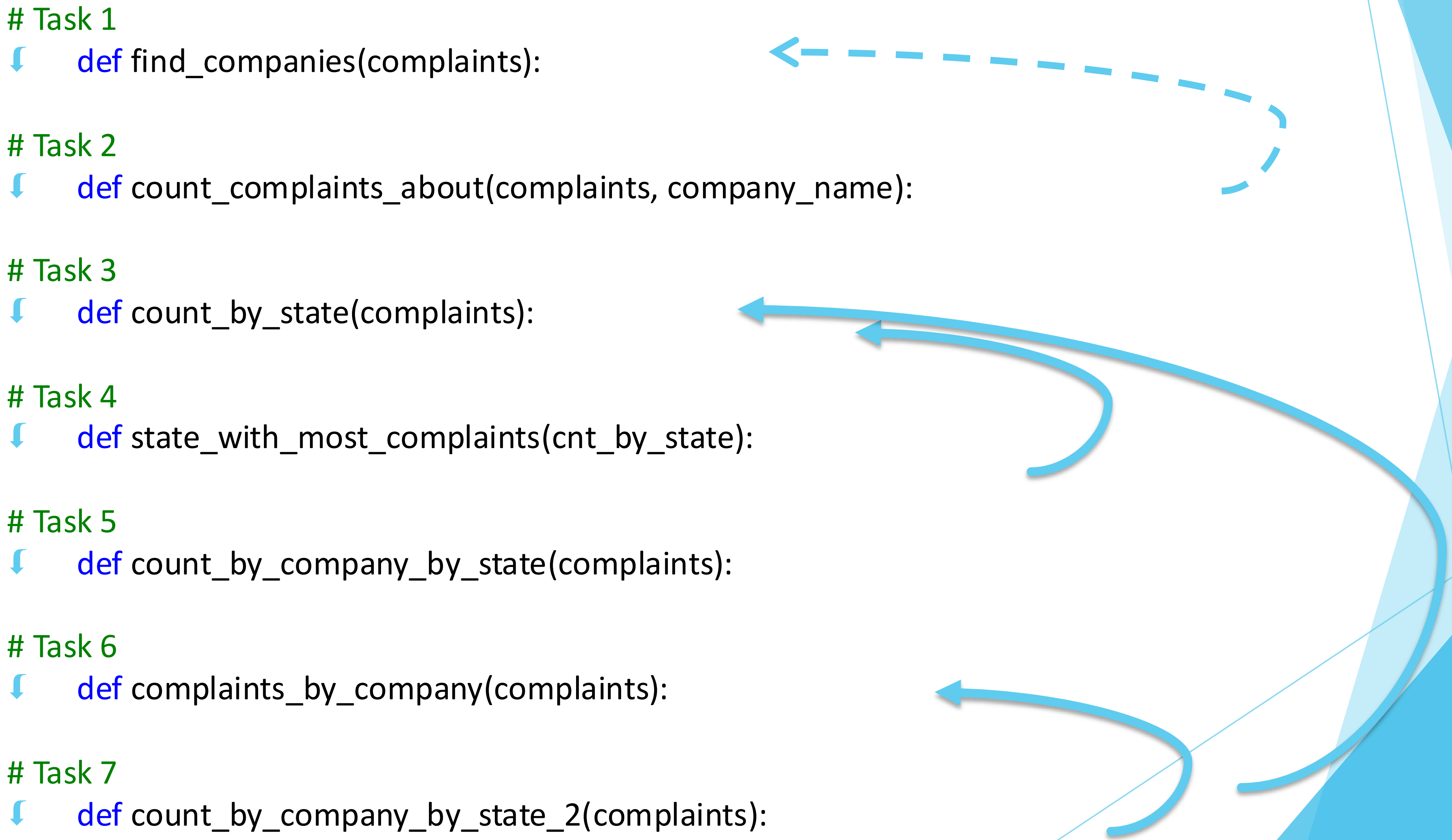
- Dictionaries are very useful for accumulating values associated with keys.

- The ***in*** operator, the ***not in*** operator, and the ***get*** method all allow us to handle previously seen and previously unseen keys cleanly.

Topics:

- Introduction to Python dictionaries
- Useful dictionary operations
- Constructing dictionaries
- Iterating over dictionaries
- Accumulation
- **Sets**
- Dictionaries as objects
- Data structure and time complexity

Anatomy of a script



```
# Task 1
    def find_companies(complaints):

# Task 2
    def count_complaints_about(complaints, company_name):

# Task 3
    def count_by_state(complaints):

# Task 4
    def state_with_most_complaints(cnt_by_state):

# Task 5
    def count_by_company_by_state(complaints):

# Task 6
    def complaints_by_company(complaints):

# Task 7
    def count_by_company_by_state_2(complaints):
```

Sets

- Making a set: set([list])
- Only count things once
- add() to add something
- update([list]) to add multiple
- remove() or discard() to remove an element
  - If you choose something that isn’t present, only ‘remove’ will let you know with an error message
- union() or `|’ will bring together sets
- intersection() or `&’ will return common elements
- difference() returns only elements unique to first set

https://www.pythoncheatsheet.org/cheatsheet/sets

Sets v Lists

- Set tasks:
  - Make a set, s1, that contains the following elements: 1,2,3,3,4,5,4,3,4
    - How many elements will it contain?
  - Add 3 to your set
  - Add 5,6,7 to your set
  - Compare s1 with a second set, s2, that contains 3, 6,9.
  - What will s1.difference(s2) produce?

Caution

- Do not add/remove keys as you iterate over a dictionary

```
# INCORRECT!
for cand_id, cand_tot in cand_to_total.items():
    if cand_tot < 50000:
        del cand_to_total[cand_id]
```

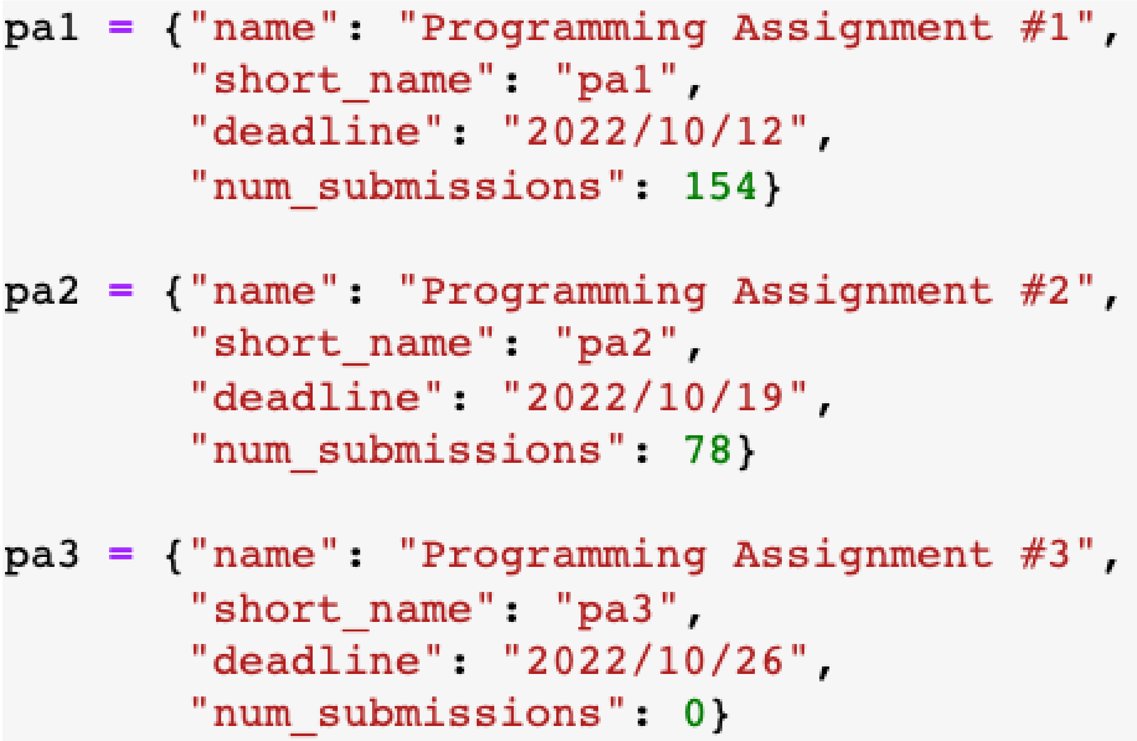
Common pitfall: 2.2.1

Topics:

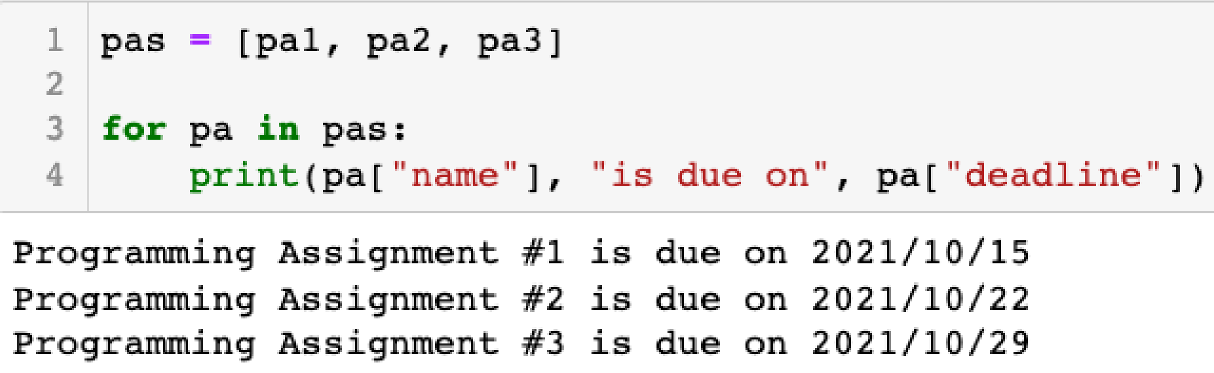
- ❑ Introduction to Python dictionaries
- ❑ Useful dictionary operations
- ❑ Constructing dictionaries
- ❑ Iterating over dictionaries
- ❑ Accumulation
- ❑ Sets
- ❑ **Dictionaries as objects**
- ❑ Data structure and time complexity

Dictionaries as objects

- Dictionaries are also commonly used to store "objects”
- E.g., keep track of programming assignments



```
pa1 = {"name": "Programming Assignment #1",
       "short_name": "pa1",
       "deadline": "2022/10/12",
       "num_submissions": 154}

pa2 = {"name": "Programming Assignment #2",
       "short_name": "pa2",
       "deadline": "2022/10/19",
       "num_submissions": 78}

pa3 = {"name": "Programming Assignment #3",
       "short_name": "pa3",
       "deadline": "2022/10/26",
       "num_submissions": 0}
```



```
pas = [pa1, pa2, pa3]

for pa in pas:
    print(pa["name"], "is due on", pa["deadline"])
```

```
Programming Assignment #1 is due on 2021/10/15
Programming Assignment #2 is due on 2021/10/22
Programming Assignment #3 is due on 2021/10/29
```

Topics:

- ❑ Introduction to Python dictionaries
- ❑ Useful dictionary operations
- ❑ Constructing dictionaries
- ❑ Iterating over dictionaries
- ❑ Accumulation
- ❑ Dictionaries as objects
- ❑ **Data structure and time complexity**
- ❑ Sets

Data structure and time complexity

- Your choice of data structure can have a considerable impact on code efficiency
- List: running time proportional to the size of the list $\mathcal{O}(n)$ because we need to iterate over the list
- Dict: implemented using a *hash table* that is optimized to access key-value mappings very quickly, in constant time $\mathcal{O}(1)$

	N = 5000	N = 20000	N = 43582
Lists	0.158 ms	0.572 ms	1.17 ms
Dictionaries	0.00471 ms	0.00465 ms	0.00498 ms

%timeit

Coding practice: 2.2.5

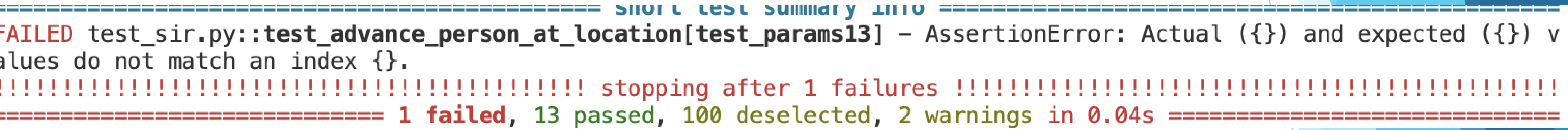
Big-O notation

- Complexity of Python operations: https://wiki.python.org/moin/TimeComplexity
- If *n* is the size of the input to a problem:
  - ***O*(*n*^2)** means the running time is roughly proportional to *n*^2 (i.e., as n gets bigger, the running time grows quadratically).
  - ***O*(*n*)** means the running time grows linearly.
  - ***O*(log *n*)** means the running time grows logarithmically.
  - ***O*(1)** means the running time is constant.

Dictionaries and data: who, what, where, when, why, how

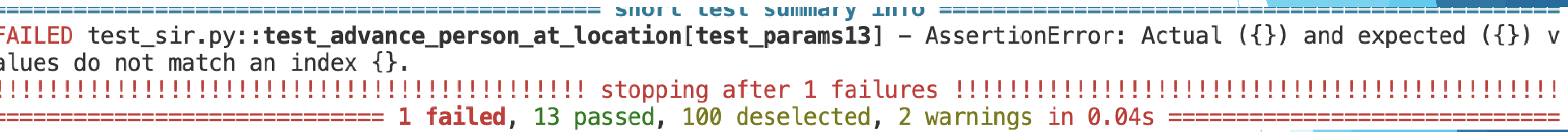
- Use to work with data
- Often when you are working with a database
- PROBABLY NOT if you’re trying to do data manipulation
- Think of it as a super structured table
- Can use to create a database (hello, R!)

Troubleshooting



```
============================================ short test summary info ==============================================
FAILED test_sir.py::test_advance_person_at_location[test_params13] - AssertionError: Actual ({}) and expected ({}) v
alues do not match an index {}.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! stopping after 1 failures !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
============================ 1 failed, 13 passed, 100 deselected, 2 warnings in 0.04s =============================
```

Troubleshooting



```
======================================== short test summary info ========================================
FAILED test_sir.py::test_advance_person_at_location[test_params13] - AssertionError: Actual ({}) and expected ({}) v
alues do not match an index {}.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! stopping after 1 failures !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
============================ 1 failed, 13 passed, 100 deselected, 2 warnings in 0.04s ============================
```

- Look at the structure:
- Test_sir.py: this is our file
- Test_advance_person_at_location: this is our function
- Test_params13: this is our set of values we are testing

Test file

Test function

Test values

- This is in our test file
- What is the structure?
- How can we pull out the trouble-case?

TT3: task 8

TT3: comparing code

Task 8: With your team, compare the implementations of count_by_company_by_state and count_by_company_by_state_2, and discuss the advantages and disadvantages of each.

Some aspects to consider:

1. Which is easier to read and understand?
2. Which uses less code?
3. Which is easier to debug?
4. Which is faster?
    - As a proxy for speed, you can ask:
        - Which requires fewer passes through the data?
        - Which implementation would you choose?

TT3: function comparison

```
def count_by_company_by_state(complaints):
'''
Computes a dict of {company: {state: count, state: count}} for all states
and companies

Inputs:
complaints (list) A list of complaints, where each complaint is a
dictionary

Returns: (dict) with count per company per state
'''

by_company_by_state = {}
for complaint in complaints:
        company = complaint["Company"]
        state = complaint["State"]
                if company not in by_company_by_state:
                        by_company_by_state[company] = {}
                if state not in by_company_by_state[company]:
                        by_company_by_state[company][state] = 0
                        by_company_by_state[company][state] += 1

return by_company_by_state
```

TT3: function comparison

```
def count_by_company_by_state_2(complaints):
'''
Computes a dict of {company: {state: count, state: count}} for all states and companies

Inputs: complaints (list) A list of complaints, where each complaint is a dictionary

Returns: (dict) with count per company per state

This implementation involves composing complaints_by_company with count_by_state
'''

by_company = complaints_by_company(complaints)
by_company_by_state = {company: count_by_state(company_complaints) for company, company_complaints in by_company.items()}
return by_company_by_state
```

TT3: function comparison

```
def count_by_company_by_state(complaints):
'''
Computes a dict of {company: {state: count, state: count}} for all states
and companies

Inputs:
complaints (list) A list of complaints, where each complaint is a
dictionary

Returns: (dict) with count per company per state
'''

by_company_by_state = {}
for complaint in complaints:
company = complaint["Company"]
state = complaint["State"]
if company not in by_company_by_state:
by_company_by_state[company] = {}
if state not in by_company_by_state[company]:
by_company_by_state[company][state] = 0
by_company_by_state[company][state] += 1

return by_company_by_state
```

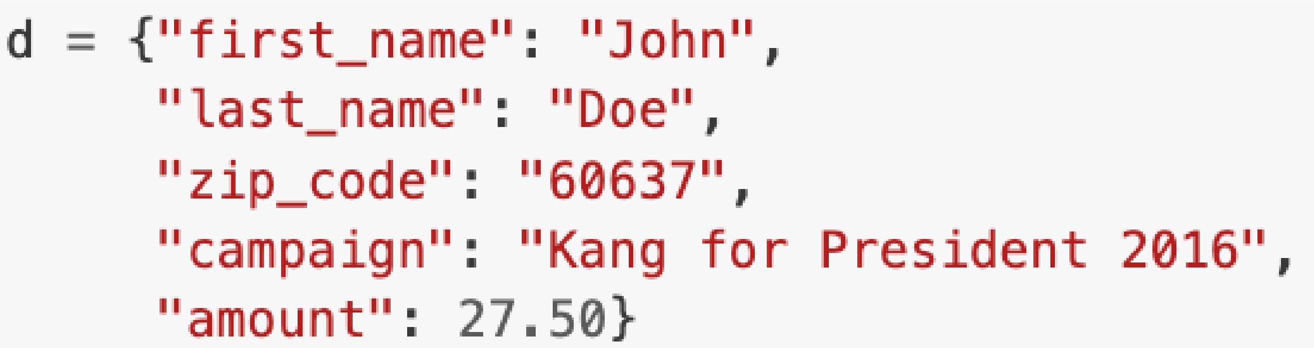
```
def count_by_company_by_state_2(complaints):
'''
Computes a dict of {company: {state: count, state: count}} for all states and
companies

Inputs: complaints (list) A list of complaints, where each complaint is a
dictionary

Returns: (dict) with count per company per state

This implementation involves composing complaints_by_company with
count_by_state
'''

by_company = complaints_by_company(complaints)
by_company_by_state = {company: count_by_state(company_complaints)
for company, company_complaints in by_company.items()}

return by_company_by_state
```

Additional practice

Topics:

- ❑ Introduction to Python dictionaries
- ❑ Useful dictionary operations
- ❑ **Constructing dictionaries**
- ❑ Iterating over dictionaries
- ❑ Accumulation
- ❑ Dictionaries as objects
- ❑ Data structure and time complexity
- ❑ Sets

Construct *dict*: dictionary literals



```
d = {"first_name": "John",
     "last_name": "Doe",
     "zip_code": "60637",
     "campaign": "Kang for President 2016",
     "amount": 27.50}
```

Coding practice: 2.2.2

Construct *dict*: empty dictionary + dictionary updates

- Start with an empty dictionary
- Add key/value pairs using the subscript notation

```
d = {}

d["first_name"] = "John"
d["last_name"] = "Doe"
d["zip_code"] = "60637"
d["campaign"] = "Kang for President 2016"
d["amount"] = 27.50
d["affiliation"] = "Kodosican”
d
```

TT3: How would we get the first element? How would we get the first element’s product?

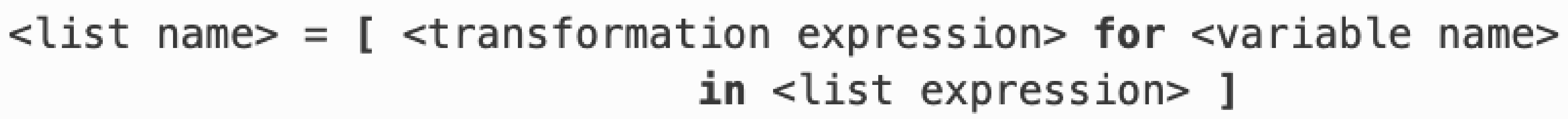
Construct *dict*: dict constructor

- Use the *dict* constructor with a list of key-value pairs or another dictionary

```
d = {"first_name": "John", "last_name": "Doe",
"zip_code": "60637", "campaign": "Kang for President
2016", "amount": 27.50}

keys_and_data = [("first_name", "John"),
("last_name", "Doe"), ("zip_code", "60637"),
("campaign", "Kang for President 2016"), ("amount",
27.50)]

d_from_list = dict(keys_and_data)

d_from_dict = dict(d)
```

Construct *dict*: dictionary comprehensions

```
d = {key: value for key, value in
keys_and_data}
```



```
<list name> = [ <transformation expression> for <variable name>
                                 in <list expression> ]
```

Coding practice: 2.2.2

PA2 (!!!)

- Review of initial assignment
- Importance of sketching
- DUE DATE 2/14 START AHEAD!!!

PA2: the “Technical Challenge”

Image source

Recipes contain helpful instructions like:

- Make cookies
- Make custard
- Bake

PA2

- Language model
- Based on cellular automata
- VERY OPEN ENDED
- For the model, specify:
  - a region,
  - the speakers in a region,
  - a speaker’s neighborhood,
  - community centers in a region,
  - a speaker’s engagement level,
  - language transmission rules,
  - a step in the simulation, and
  - the stopping conditions for the simulation.

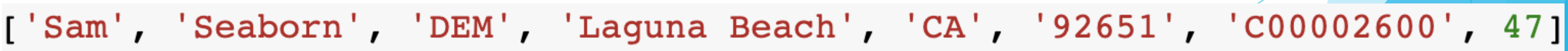
PA2

- Suggested workflow:
- Sketch out what you will need
- “ Big rocks first”:
  - README files!!
    - Main file + test folder
  - Sketch (rough draft ON PAPER)
  - Functions: (what do you need to include to reduce errors)?
  - Docstrings
  - Text
- TEST AS YOU GO!!

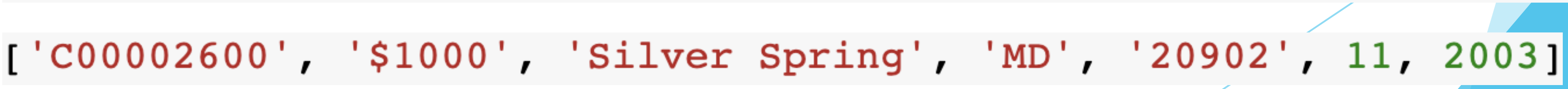
Representing campaign data

Represent political candidates and contributions.

Sample candidate: First Name: Sam Last Name: Seaborn Party: DEM City: Laguna Beach State: CA ZIP Code: 92651 **Candidate ID: C00002600** District: 47	**Sample contribution:** **Candidate ID: C00002600** Amount: $1000 City: Silver Spring State: MD ZIP Code: 20902 Month: 11 Year: 2003



```
['Sam', 'Seaborn', 'DEM', 'Laguna Beach', 'CA', '92651', 'C00002600', 47]
```

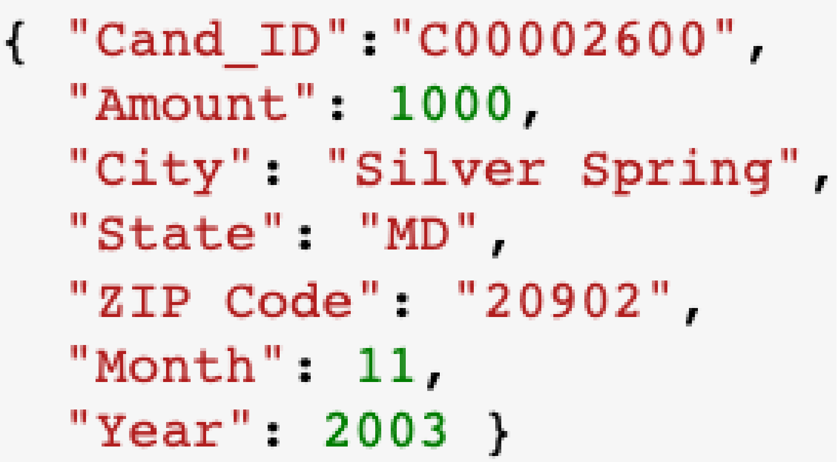


```
['C00002600', '$1000', 'Silver Spring', 'MD', '20902', 11, 2003]
```

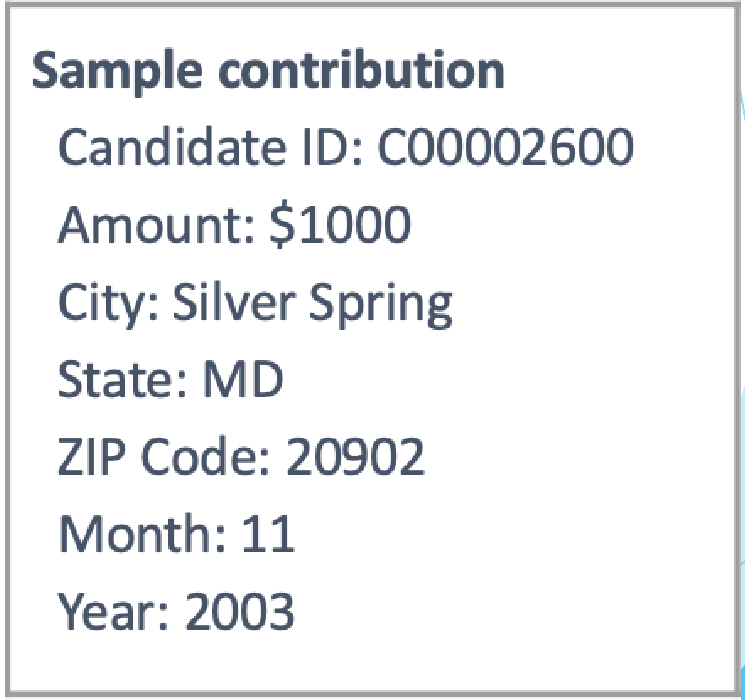
Disadvantages

Representing contributions

- Keys are **often strings**, but other immutable types (integers, booleans) can be used as well.
- Values can have any type, including different types for different values.



```
{ "Cand_ID":"C00002600",
  "Amount": 1000,
  "City": "Silver Spring",
  "State": "MD",
  "ZIP Code": "20902",
  "Month": 11,
  "Year": 2003 }
```



Sample contribution

Candidate ID: C00002600

Amount: $1000

City: Silver Spring

State: MD

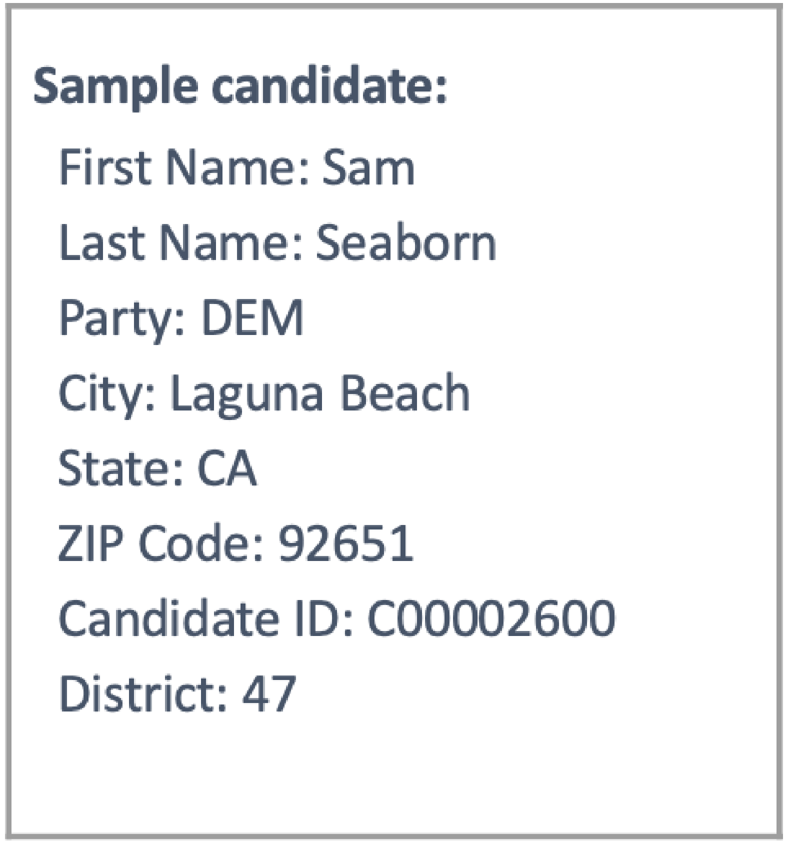
ZIP Code: 20902

Month: 11

Year: 2003

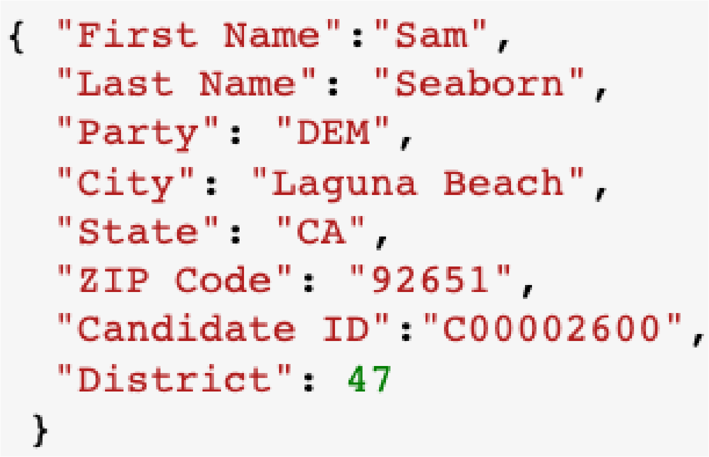
Values can have any type

- Values can have any type, including dictionaries, list of dictionaries, etc.



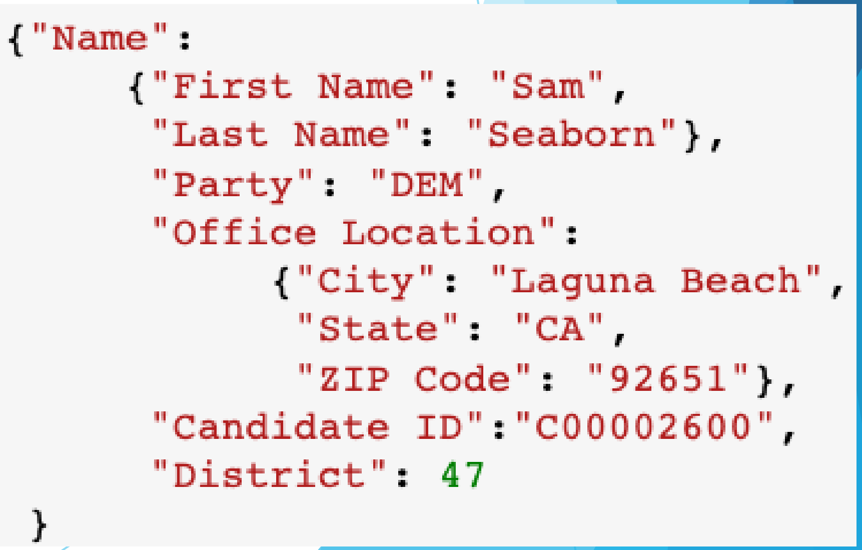
Sample candidate:

First Name: Sam
Last Name: Seaborn
Party: DEM
City: Laguna Beach
State: CA
ZIP Code: 92651
Candidate ID: C00002600
District: 47



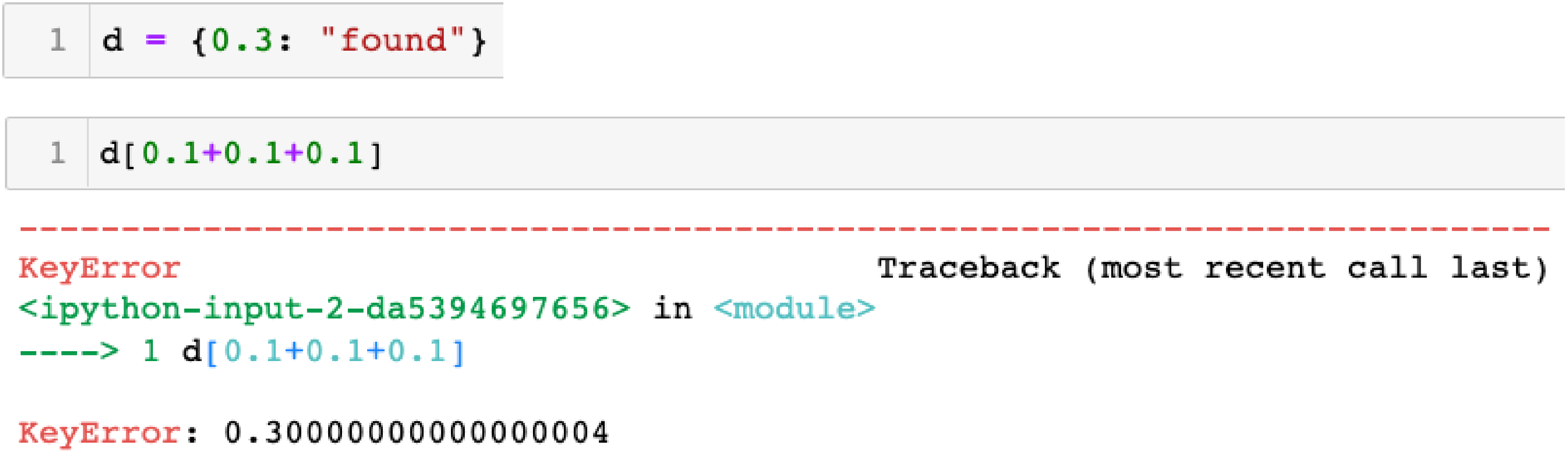
```
{ "First Name":"Sam",
  "Last Name": "Seaborn",
  "Party": "DEM",
  "City": "Laguna Beach",
  "State": "CA",
  "ZIP Code": "92651",
  "Candidate ID":"C00002600",
  "District": 47
 }
```

Nested dictionaries



```
{"Name":
     {"First Name": "Sam",
      "Last Name": "Seaborn"},
      "Party": "DEM",
      "Office Location":
            {"City": "Laguna Beach",
             "State": "CA",
             "ZIP Code": "92651"},
      "Candidate ID":"C00002600",
      "District": 47
 }
```

Bad keys



```
d = {0.3: "found"}
```

```
d[0.1+0.1+0.1]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-2-da5394697656> in <module>
----> 1 d[0.1+0.1+0.1]

KeyError: 0.30000000000000004
```

- Computers use binary system instead of a base 10 system
- 0.1 and 0.3 can’t be precisely represented in a binary system
- The binary representation of 0.3 is not equal to that of 0.1+0.1+0.1

Representing contributions

Sample contribution

Candidate ID: C00002600

Amount: $1000

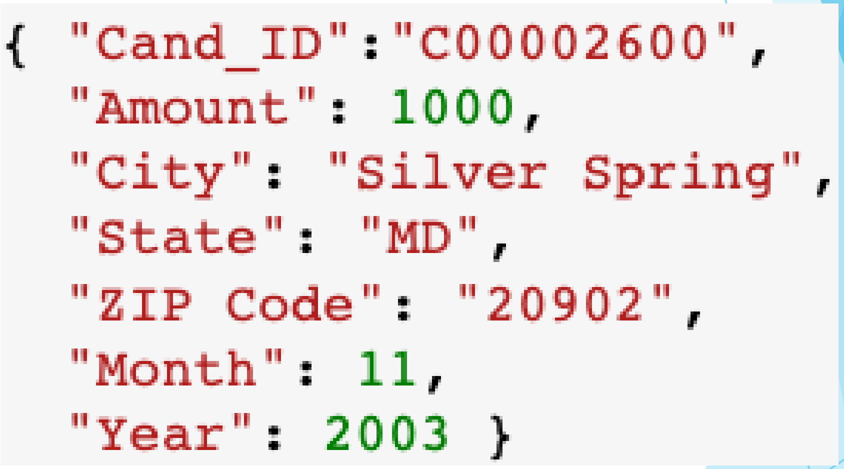
City: Silver Spring

State: MD

ZIP Code: 20902

Month: 11

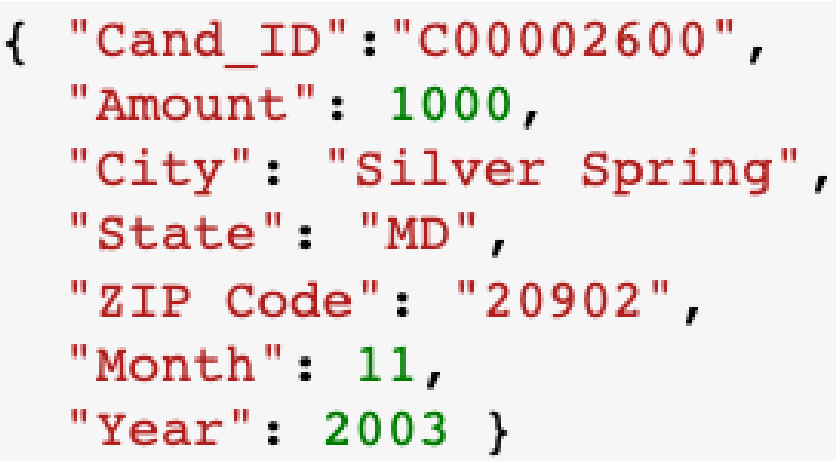
Year: 2003



```
{ "Cand_ID":"C00002600",
  "Amount": 1000,
  "City": "Silver Spring",
  "State": "MD",
  "ZIP Code": "20902",
  "Month": 11,
  "Year": 2003 }
```

Total contributions for all candidates

- Mapping from candidate IDs to total contributions received by the candidates
- Accumulate values based on keys



```
{ "Cand_ID":"C00002600",
  "Amount": 1000,
  "City": "Silver Spring",
  "State": "MD",
  "ZIP Code": "20902",
  "Month": 11,
  "Year": 2003 }
```

```
cand_to_total =
    {'C00002600': 433680,
     'C00012229': 469755,
     'C00013128': 398652,
     'C00017830': 561314,
     'C00019075': 538150,
         …
    }
```

Coding practice: 2.2.3