THE UNIVERSITY OF CHICAGO

MASTERS IN COMPUTATIONAL SOCIAL SCIENCE
THE UNIVERSITY OF CHICAGO

MACS 30111

Dictionaries

Misc

- Canvas updated!
- Today:
  - We will use information from TT3: https://classes.ssd.uchicago.edu/macss/macs30121/modules/tt/tt3.html
  - Materials: https://classroom.github.com/a/3utna2Nx
- Ed!!

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

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

Topics:

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

Representing campaign data

Represent political candidates and contributions.

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

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

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

Disadvantages

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
 }
```

Coding practice: 2.2

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

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

Bad keys

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

```
1  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

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
 }
```

Exercise: TT3

- Load your files:

```
import cfpb
cfpb.CFPB_16
```

- What is this?

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")
```

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

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

Exercise: TT3

- Get the keys from `cfpb.CFPB_16`

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**
- Dictionaries as objects
- Data structure and time complexity
- Sets

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

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.

Coding practice

- "M3_contributions.csv"
- Map candidate ids to total contribution amounts
- Map states to lists of contributions from that state
- Computing contributions by state

```
cand_to_total =
    {'C00002600': 433680,
     'C00012229': 469755,
     'C00013128': 398652,
     'C00017830': 561314,
     'C00019075': 538150,
     'C00020974': 27499,
     'C00025973': 1047469,
     'C00026757': 311653,
     'C00028316': 12000,
     'C00037606': 422452,
     'C00051227': 235500,
        …
    }
```

```
state_to_conts =
   {'MD': [ … ],
    'VA': [ … ],
    'TN': [ … ],
    'CT': [ … ],
    'OR': [ … ],
      …
    'IA': [ … ]}
```

```
[{'Cand_ID': 'C00013128',
  'Amount': 250,
  'City': 'Waterloo',
  'State': 'IA',
  'Zipcode': '50701',
  'Month': 5,
  'Year': 2009},

 {'Cand_ID': 'C00104752',
  'Amount': 500,
  'City': 'ANKENY',
  'State': 'IA',
  'Zipcode': '50021',
  'Month': 3,
  'Year': 2010}]
```

```
{'MD': 410566,
 'VA': 934381,
 'TN': 25950,
 'DC': 661180,
 'TX': 265798,
 'MI': 259300,
 'IL': 133748,
 'OH': 79350,
 'NJ': 193593,
 'MA': 142000,
 'WI': 178659,
 'NY': 417644,
 'CA': 1447936,
```

Topics:

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

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 $O(n)$ 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 $O(1)$

	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.

Topics:

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

Sets

- Store **unordered** collection of **distinct** elements
- Define a set:

```
zipcodes = {"60637", "07974"}

zipcodes = set(["60637", "07974"])

empty_set = set()
```

- Typical operations
  - $e \in S$ • membership: whether e appears in set S
  - $S \subseteq T$ • subset: whether every element in S appears in T
  - $S \cup T$ • union: whether $e \in S$ and/or $e \in T$
  - $S \cap T$ • intersection: whether $e \in S$ and $e \in T$
  - $S - T$ • difference: whether $e \in S$ and $e \notin T$

Coding practice: 2.2.6