

LECTURE 17 — DICTIONARIES, PART 2

31.1 Overview

- Recap
- More IMDB examples:
 - Dictionaries of string/set pairs
 - Converting dictionaries with one key to another
 - Combining information from multiple dictionaries
- A different view of dictionaries: storing attribute/value pairs.
- Accessing APIs and getting data back as a dictionary.

31.2 Recap of dictionaries

- On the surface, dictionaries look like lists, except, you can have anything for indices (keys), not just numbers starting with 0.
- The following two store the same information:

```
>>> listoption = ['a', 'b', 'c']  
>>> dictoption = {0: 'a', 1: 'b', 2: 'c'}
```

Note that this is a new way for us to initialize a dictionary.

- You would access them in the same way:

```
>>> listoption[1]  
'b'  
>>> dictoption[1]  
'b'
```

- You would update them in the same way:

```
>>> listoption[1] = 'd'  
>>> dictoption[1] = 'd'
```

- But you can't extend them in the same way. For example:

```
>>> listoption[10] = 'e'
```

is illegal, but:

```
>>> dictoption[10] = 'e'
```

is perfectly fine. Be sure you can explain why.

- Of course the power of a dictionary is that keys can be anything, or at least anything *hashable*

```
>>> d = {'Gru':3, 'Margo':4}
>>> d['Gru']
3
```

- This dictionary has strings as keys and integers as values. The values can really be anything:

```
>>> d2 = {'Gru': set( [123,456] ), 'Margo': set( [456] ) }
>>> d2['Gru']
{456, 123}
```

- Note that since keys can be anything, we need to know how to print or iterate through the values in a dictionary. This is actually quite trivial using a dictionary:

```
>>> d2.keys()
dict_keys(['Gru', 'Margo'])
>>> list(d2.keys())
['Gru', 'Margo']
>>> for key in d2: # or, for key in d2.keys():
...     print(key, d2[key])
Gru {123, 456}
Margo {456}
```

31.3 Copying and Aliasing Dictionaries

- We'll take a few minutes in class for you to try to predict the output of the following:

```
d = dict()
d[15] = 'hi'
L = []
L.append(d)
d[20] = 'bye'
L.append(d.copy())
d[15] = 'hello'
del d[20]
print(L)
```

- The result may surprise you, but it reflects the difference between making an alias to an object and making a full copy of an object.
 - An alias is also sometimes known as a *shallow copy*
 - A full copy is also sometimes known as a *deep copy*
- Assignment between lists, between sets, and between dictionaries all involve aliasing / shallow copying!
- At this point we will take a few minutes for you to work on the first lecture exercise.

31.4 Back to IMDB: Dictionaries Whose Values Are Sets

- In our IMDB data, an individual may be listed more than once for each movie. For example, Tom Hanks is listed six times for *Polar Express*
- In order to determine who was involved in the most different movies, we need to keep a set of movies for each individual instead of a count.
- We will modify our solution to the IMDB example to find out who was involved in the most different movies?
 - The solution will be posted on the course website.

31.5 Converting to Other Dictionaries: The N Busiest Individuals

- Suppose we want to find the top 10 or 25 busiest individuals in the IMDB, based on the number of different movies they are involved in.
- Now we need a different dictionary:
 - Keys are integers representing the number of movies
 - Values are lists of actors.
 - * Why don't we need sets here?
- We will show how to extend our code to build this dictionary from our original dictionary.
- Next, we will need to access the keys from this dictionary in reverse order and print out names of the individuals, stopping when we've printed the top N busiest (allowing more in the case of ties).

31.6 More That We Can Do With the IMDB

- We now have an actors dictionary whose keys are actor names and whose values are sets of movies.
- We can also construct a different dictionary whose keys are movies and whose values are sets of actors.
- Using this we can find all sorts of information:
 - What movie involved the most people?
 - How many different people have been in movies that included Meryl Streep?
 - Solve the “degrees of Kevin Bacon” problem:
 1. Who has been in a movie with Kevin Bacon? These people are degree 1.
 2. Who is not a degree 1 individual and has been in a movie with a person who was in a movie with Kevin Bacon? These people are degree 2 individuals.
 3. Who is not a degree 1 or 2 individual, and has been in a movie with a degree 2 individual (in a movie with a person who has been in a movie with a person who was in a movie with Kevin Bacon)? These people are degree 3 individuals.
 4. Etc.

31.7 Attribute / Value Pairs

- We can use dictionaries to construct even more complicated data structures: dictionaries as values, lists of dictionaries, etc.
- Consider the problem of representing all the houses a real estate company is trying to sell.
- We could keep a list with information about each property, but a list of what?
- We will look at describing each house as a dictionary, with the keys being the “attributes”, and the values being, well, the values of the attributes.
- Examples include the listing reference number, the address, the number of bedrooms, the price, whether or not it has a pool, the style of the house, the age, etc.
 - Some properties will not be known and therefore they will not be represented in the dictionary.
- We will work through a made-up example in class, producing a list of dictionaries. This list will be called *houses*.
- As an exercise you can think about write coding that finds all houses in our house list that have at least 4 bedrooms (attribute is `bedrooms`, value is an integer), a pool (attribute is `pool`, value a string describing if the pool is above ground or below), for a price below \$300,000 (attribute is `price`, value is an int).
- Overall, this a simple Python implementation of the storage and access of information in a *database*.

31.8 Accessing APIs

- Many APIs (Application Programming Interfaces) accessible on the internet return values that are **JSON** (Java Script Object Notation) strings. These are easily loaded into Python objects, often involving dictionaries.
- The best way to understand the dictionary structure returned by an API is to seek documentation. If that fails, you can print the top level keys and values to explore.
- Public APIs, which do not not require authentication, are accessed as follows:

```
import urllib.request
import json

url = "enter your public url here"
f = urllib.request.urlopen(url)
rawcontent = f.read()
content = json.loads(rawcontent.decode("utf-8"))
```

- An example of a public API (used in our image lab):
 - **nominatim** gives you a bounding box of geolocation for a given location. Let's see this for 'Troy, NY':

```
url = "http://nominatim.openstreetmap.org/"\
      "search?q={}&format=json&polygon_geojson=1&addressdetails=0"\
      .format('Troy, NY')
```

- Many sources require authentication with an API key through the `oauth2` authentication module. But, the overall method of access remains the same after authentication.
- Once we understand the structure, we can write code to extract the information we want.

31.9 Final Example

- Given the following dictionary for hobbies for people:

```
hobby = {'Gru':set(['Hiking','Cooking']), 'Edith':set(['Hiking','Board Games'])}
```

create a new dictionary that lists people for each hobby:

```
{'Hiking': {'Gru','Edith'}, 'Cooking':{'Gru'}, 'Board Games':{'Edith'}}
```

31.10 Summary

- Dictionaries of sets.
- Dictionaries where the keys are numbers.
- A variety of examples to extract information from the IMDB data set.
- Dictionaries as database — storing attribute / value pairs.
- Accessing information from public APIs

31.11 Additional Dictionary Practice Problems

- Create a dictionary to store the favorite colors of the following individuals

- Thomas prefers red
- Ashok prefers green
- Sandy prefers red
- Alison prefers orange
- Fei prefers green
- Natasha prefers blue

Then add some others of your own. Now, write code to change Fei's preference to green and to remove Sandy's preference from the dictionary.

- Using the dictionary from the first problem, write code to find which color is most commonly preferred. Use a second dictionary, one that associates strings (representing the colors) with the counts. Output the most common color. If there are ties, output all tied colors.
- Complete the fast, list solution to the movie counting problem based on sorting, as outlined at the start of the lecture notes.
- Write a program that uses a dictionary that associates integers (the key) and sets strings (the values) to find the number of movies in each year of the IMDB. Start from any of the IMDB examples. Write additional code that uses the `years_and_movies` dictionary to find the year that has the most movies.
- Use a dictionary to determine which last names are most common in the IMDB data we have provided. Count individual people not the movies they appear in. For example, 'Hanks , Tom' counts as one instance of the name 'Hanks' despite the fact that he is in many movies. Assume that the last name ends with the first ' , ' in the actual name. Start this problem by thinking about what the dictionary keys and values should be.

6. Which two individuals have the most movies in common? To solve this you will need to start from the dictionary that associates each individual with the set of movies s/he is involved in. Then you will need double for loops. At first glance this appears that it might be very, very slow, but it can be made much faster by intelligently terminating loops. To illustrate, if you find a pair of individuals with k movies in common then you never have to even consider an individual involved in fewer than k movies!