CHAPTER

TWENTYNINE

LECTURE 16 — DICTIONARIES, PART 1

29.1 Overview

- More on IMDB
- Dictionaries and dictionary operations
- Solutions to the problem of counting how many movies are associated with each individual
- Other applications

29.2 How Many Movies is Each Person Involved In?

- Goals:
  - Count movies for each person.
  - Who is the busiest?
  - What movies do two people have in common?
- Best solved with the notion of a dictionary, but we'll at least consider how to use a list.

29.3 List-Based Solution — Straightforward Version

- Core data structure is a list of two-item lists, each giving a person's name and the count of movies.
- For example, after reading the first seven lines of our shortened `hanks.txt` file, we would have the list

```
[ ["Hanks, Jim", 3], ["Hanks, Colin", 1],
  ["Hanks, Bethan", 1], ["Hanks, Tom", 2] ]
```

- Just like our solution from the sets lectures, we can start from the following code:

```
imdb_file = input("Enter the name of the IMDB file ==> ").strip()
count_list = []
for line in open(imdb_file, encoding = "ISO-8859-1"):
    words = line.strip().split('|')
    name = words[0].strip()
```

- Like our list solution for finding all IMDB people, this solution is VERY slow — once again $O(N^2)$ ("order of N squared").

29.4 List-Based Solution — Faster Version Based on Sorting

- There is an alternate solution that would work for the number of unique names solution from lecture 15 as well. It is based on sorting.
- Append each name to the end of the list **without** checking if it is already there.
- After reading all of the movies, sort the entire resulting list
  - As a result, all instances of each name will now be next to each other.
- Go back through the list, counting the occurrence of each name
- This solution will be **much** faster than the first, but it is also more involved to write than the one we are about to write using dictionaries

29.5 Introduction to Dictionaries

- Association between "keys" (like words in an English dictionary) and "values" (like definitions in an English dictionary). The values can be **anything**.
- Examples:

```
>>> heights = dict()
>>> heights = {}    # either of these works
>>> heights['belgian horse'] = 162.6
>>> heights['indian elephant'] = 280.0
>>> heights['tiger'] = 91.0
>>> heights['lion'] = 97.0
>>> heights
{'belgian horse': 162.6, 'tiger': 91.0, 'lion': 97.0, 'indian elephant': 280.0}
>>> 'tiger' in heights
True
>>> 'giraffe' in heights
False
>>> 91.0 in heights
False
>>> list(heights.keys())
['belgian horse', 'tiger', 'lion', 'indian elephant']
>>> sorted(heights.keys())
['belgian horse', 'indian elephant', 'lion', 'tiger']
>>> heights.values()
dict_values([162.6, 91.0, 97.0, 280.0])
>>> list(heights.values())
[97.0, 162.6, 91.0, 280.0]
```

- Details:
  - Two initializations; either would work.
  - Syntax is very much like the subscripting syntax for lists, except dictionary subscripting/indexing uses keys instead of integers!
  - The keys, in this example, are animal species (or subspecies) names; the values are floats.
  - The `in` method tests only for the presence of the key, like looking up a word in the dictionary without checking its definition.
  - The keys are NOT ordered.

- Just as in sets, the implementation uses *hashing* of keys.
  - Conceptually, sets are dictionaries without values.

29.6 Lecture Exercise 1

You will have five minutes to work on the first lecture exercise.

29.7 Back to Our IMDB Problem

- Even though our coverage of dictionaries has been brief, we already have enough tools to solve our problem of counting movies.
- Once again we'll use the following as a starting point

```
imdb_file = input("Enter the name of the IMDB file ==> ").strip()
counts = dict()
for line in open(imdb_file, encoding = "ISO-8859-1"):
    words = line.strip().split('|')
    name = words[0].strip()
```

- The solution we give in class will output the counts for the first 100 individuals in alphabetical order. It will be up to you as an exercise to find the most frequently occuring individual.
- We will impose an ordering on the output by sorting the keys.
- We'll test first on our smaller data set and then again later on our larger ones.

29.8 Key Types

- Thus far, the *keys* in our dictionary have been strings.
- Keys can be any "hashable" type — string, int, float, booleans.
  - Lists, sets and other dictionaries can not be keys.
- Strings are by far the most common key type
- We will see an example of integers as the key type by the end of the Lecture 17 (next set of) notes.
- Float and boolean are general poor choices. Can you think why?

29.9 Value Types

- So far, the *values* in our dictionaries have been integers and floats.
- But, any type can be the values
  - boolean
  - int
  - float
  - string

  - list
  - tuple
  - set
  - other dictionaries

- Here is an example using our IMDB code and a set:

```
>>> people = dict()
>>> people['Hanks, Tom'] = set()
>>> people['Hanks, Tom'].add('Big')
>>> people['Hanks, Tom'].add('Splash')
>>> people['Hanks, Tom'].add('Forest Gump')
>>> print(people['Hanks, Tom'])
{'Forest Gump', 'Big', 'Splash'}
```

- Here is another example where we store the continent and the population for a country instead of just the population:

```
countries = dict()
countries.clear()
countries['Algeria'] =  (37100000, 'Africa')
countries['Canada'] = (34945200, 'North America' )
countries['Uganda'] = (32939800, 'Africa')
countries['Morocco'] = (32696600, 'Africa')
countries['Sudan'] = (30894000, 'Africa')
```

- We access the values in the entries using *two consecutive subscripts*. For example,

```
name = "Canada"
print("The population of {} is {}".format(name, countries[name][0]))
print("It is in the continent of", countries[name][1])
```

29.10 Removing Values: Sets and Dictionaries

- For a set:
  - `discard` removes the specified element, and does nothing if it is not there
  - `remove` removes the specified element, but fails (throwing an exception) if it is not there
- For a dictionary, it is the `del` function.
- For both sets and dictionaries, the `clear` method empties the container.
- We will look at toy examples in class

29.11 Other Dictionary Methods

- The following dictionary methods are useful, but not so much as the ones we've discussed.
  - `get`
  - `pop`
  - `popitem`

  - `update`

- Use the `help` function in Python to figure out how to use them and to find other dictionary methods.

29.12 Summary of Dictionaries

- Associate "keys" with "values"
- Feels like indexing, except we are using keys instead of integer indices.
- Makes counting and a number of other operations simple and fast.
- Keys can be any "hashable" value, usually strings, sometimes integers.
- Values can any type whatsoever.

29.13 Additional Practice

1. Write a function that takes the IMDB dictionary — which associates strings representing names with integers representing the count of movies — and an integer representing a `min_count`, and removes all individuals from the dictionary involved in fewer than `min_count` movies.