Introduction to:
Computers & Programming:
Dictionaries and Sets

Adam Meyers

New York University

Outline

- Unordered Collections of Objects
  - Sets (Brief discussion)
  - Dictionaries (Main focus)
- Details
  - Definitions
  - Syntax
  - Methods and Functions
  - Usage
- Problems using dictionaries

What is a Set?

- A set is an unordered collection of objects
  - Long history in math with definitions for various operations and relations: subset, superset, intersection, union, etc.
- Python's implementation of sets
  - No indices (sets are unordered)
  - No duplicate objects (there is no way to distinguish a 2nd instance of the same item)
  - Mutable (like lists)
    - Immutable version of sets: ***frozenset***
- Sets vs Lists
  - Sets are better models of some types of problems than lists
  - Lists are more versatile and can usually handle set-oriented problems, with a few modifications, e.g., if duplicate elements are eliminated.
- I will not require sets in the homework or in tests (though if you choose to use them to solve a problem, that is OK)
  - Example: if you cooerce a list to a set, it will eliminate duplicates
    - set([1,2,3,4,5,1,2,3]) → {1, 2, 3, 4, 5}
    - list({1, 2, 3, 4, 5}) → [1, 2, 3, 4, 5]

Set Syntax, Methods, Functions

- Making an empty set
  - my_set = set()
- Curly brackets around elements with commas in between
  - my_set ={1,2,3}
- Adding elements (.add and .update)
  - my_set.add(4) ## duplicate elements do not get added
  - my_set.update({3,4}) ## like **list.append,** but for sets
- Removing an element
  - my_set.discard(9) ## no error if element is not there
  - my_set.remove(9) ## error if element is not there
- Set operations from math (returns union, intersection and difference)
  - set1.union(set2)
  - set1.intersection(set2)
  - set1.difference(set2)

More Set Syntax, Methods, Functions

- Boolean tests for membership, subset, superset
  - 1 in set1 ## membership
  - set1.issubset(set2)
  - set1.issuperset(set2)
- len(set1) → number of elements
- loop for item in set: ## similar to list, order is not important
  print(item)
- set(sequence) ## coerces sequences into a set

Dictionaries

- Syntax – instances of key + : + value, separated by commas
  - Keys must be immutable: string, number, tupple, etc.
- A dictionary can be initialized as follows:

  my_dictionary = {'eat' : 'verb', 'book' : 'noun', 'orange' : 'adjective'} # with values

  my_dictionary2 = {} # empty
- Items can be looked up in a dictionary, as follows:

  my_dictionary['eat']

  my_dictionary.get('eat')
  - If key is not in dictionary, these give error
  - Keys in dictionaries function similarly to the way indices function in lists (lookup, setting values)
- To check if item is in dictionary:

  'eat' in my_dictionary
- Items can be added or changes as follows:

  my_dictionary2['book'] = 'noun'
- Dictionaries are mutable (like lists)
  - Calling a function with a dictionary as an argument can change the dictionary
- A key can only occur once in a dictionary, i.e., there is one set of keys for a dictionary

Dictionaries 2

- What is a dictionary in Python?
  - Dictionaries consists of sets of key/value pairs
    - For example, the key 'book' currently has the value 'noun'
  - Not ordered. Look up by keyword, not by index as with sequences.
- Why have a dictionary?
  - To store fairly large amounts of information that has no natural order.
- Examples
  - Words: part of speech (noun, verb, etc.), definitions, pronunciation, etc. (Lexicons)
  - Sports player: statistics
  - Food: ingredients
  - Medicine: side effects
  - Telephone books (or reverse telephone books)

Dictionary Methods and Functions

- Methods for returning contents of dictionary:
  - my_dictionary.keys(), dictionary.values(), dictionary.items()
    - items = key,value pairs
- Methods for changing dictionary:
  - my_dictionary.clear() ## erases contents of dictionaryt
  - my_dictionary.update([['bear','common_noun'],['friendly','adjective']]) ## argument can be sequence or dictionary – all pairs are added
  - my_dictionary.pop('book') ## removes item/returns value
  - my_dictionary.popitem() ## removes/returns arbitrary key/value pair
- Copying: my_dictionary.copy() ## returns copy of dictionary
- len(dictionary) → number of items in a dictionary
- for key in dictionary: ## for loop treats dictionary as list of keys

  print(key,dictionary[key])
- 'book' in dictionary ## True if 'book' is a key in dictionary; False otherwise

Dictionaries are Hash Tables

- The dictionary data structure is Python's implementation of a hash table, a popular data structure for mapping keys to values
- There is a special “hash” function which maps keys to (ideally) unique places in memory. This makes it possible to “look up” the values efficiently, even though the items are not stored in sequential order.
  - Example (pretending that only sorted strings are considered and ignoring efficiency)
    - assign each character a unique prime number
    - for strings of multiple characters, multiply these prime numbers together
- Due to efficiency concerns, most hash functions do not map keys uniquely, but use heuristics (compromises) to get around this problem.
- Hash function are derived from keys – this can only work if the key doesn't change (different keys have different hash values).
  - Immutable objects can be hash keys (strings, numbers, tuples, etc.)
  - Mutable objects (lists, dictionaries, etc.) cannot be hash keys
  - Imagine trying to looking up a word in a physical (book) dictionary while somebody added and subtracted letters from the beginning and end of the word. Since the dictionary is in alphabetical order, its position in the dictionary would keep changing and you would have trouble finding it.
- For details see (for example) Wikipedia's entries for hash table and hash function

Class Exercise

1. Set a global variable called telephone_book to an empty dictionary

   telephone_book = {}

2. Write a simple function for users to add phone numbers into telephone_book. Remember to add *key+value* to *dictionary*, the command is:

   dictionary[key] = value

   The keys should be names (strings) and the values should be strings consisting of numbers and hyphens, e.g., '212-123-4567'

3. Add several names (at least 5) and phone numbers to your dictionary using the function you wrote.
4. Write a function to save the phonebook in a text file
5. Revise the program so it can handle cases where one name is associated with multiple phone numbers

Random Sentence Generation Program

- Demonstration of the program
- Where the words came from
  - 1000 most common words according to: http://www.bckelk.ukfsn.org/words/uk1000.html
    - Sorted them automatically into: nouns, plural_nouns, adjectives,verbs, adverbs
    - according to some dictionaries
    - also edited to work OK for simple sentences
  - Added some proper nouns also (person names, organization names, etc.)

What the Sentence Generator Does

- Generates sentences according to a set of rules (aka, a grammar).
- These rules are called phrase structure rules, common used in both linguistics and computer science
- The rules consist of two types of symbols: Terminals & Nonterminals
- Each rule consists of a symbol (a Nonterminal) followed by an arrow and a list of symbols (Terminal or Nonterminal)
- The interpretation is that the symbol before the arrow can be replaced by the symbols following the arrow.
- Terminal symbols can be replaced with random words of that type.
- The system automatically does this replacement until a list of words results.

Rules Used by Sentence Generator

- S → NP VP
- NP → DetP ADJ_SEQ NOUN_SEQ
- NP → DetP NOUN_SEQ
- NP → ADJ_SEQ plural_noun
- NP → plural_noun
- NP → Proper_noun
- DetP → NP pos
- DetP → determiner
- pos → 's
- PP → preposition NP

More Rules Used by Sentence Generator

- ADJ_SEQ → adjective
- ADJ_SEQ → adjective ADJ_SEQ
- NOUN_SEQ → plural_noun
- NOUN_SEQ → singular_noun
- VP → verb
- VP → verb NP
- VP → verb NP PP
- VP → adverb verb
- VP → adverb verb NP
- VP → adverb verb NP PP

Some Linguistic Background

- A simple sentence is:
  - A Noun Phrase (the subject) plus a Verb Phrase
- A Verb Phrase is a verb plus a bunch of other phrases, mostly following it.
- A Noun Phrase is:
  - a determiner (optional for plurals) plus some adjectives, plus a noun (and possibly other stuff)
  - or a name (which can consist of multiple words)
- This is a simplified view of things, but sufficient for understanding this program.
- The program will produce sentences that are nearly grammatically well-formed sentences (some will be OK).
- It also produces mostly semantically strange sentences.

Dictionary Lookup for Rules

- The dictionary maps each symbol before the arrow in the rules to each of the list of the possible sequences following the arrows.
- For example, the entry for 'NP' is the following list:
  - ***[['DetP','ADJ_SEQ','NOUN_SEQ'],['DetP','NOUN_SEQ'], ['ADJ_SEQ','plural_noun'], ['proper_noun'],['plural_noun']]***
- Each terminal symbol (***singular_noun, plural_noun***, etc.) is linked in the dictionary to a list of all the words of that class.

Generating a Random Sentence

- To generate a random sentence, keep replacing non_terminal symbols with terminal symbols until the whole phrase consists of words. Then print out those words.
- Start by looking up S (sentence) in the dictionary
  - There is only one expansion: ***S → NP VP***
  - The system starts with the "stack" set to ***['VP','NP']***
  - Right most item (***top*** of the stack) is popped and looked up in the dictionary: 1 of the 4 possible expansions is chosen randomly and added back in the stack, e.g., the stack may now be: ***['VP', 'NOUN_SEQ', 'ADJ_SEQ' 'DetP']***
  - The loop repeats, this time the next item ***'DetP'*** is looked up.
  - Any time a word is output, it has no dictionary entry and it gets added to a new list called result. In the end we have a list of words, which is essentially the output.
- Demo with trace

Summary

- Sets and Dictionaries are both unordered collections of items
  - They are mutable
  - Items in sets and dictionaries are unique (unlike lists)
- Sets model the math object – not a main focus of this class
- Dictionaries provide a way of looking up items without reference to sequences
  - Even though some dictionaries could be ordered (words in alphabetic order), this would be inconvenient/inefficient for very large sequences
  - Other dictionary keys have no natural order
  - Useful for modeling systems requiring arbitrary lookup
- Modeling actual problems may require datastructures that combine the various data types that we have covered in this class

Homework

- https://cs.nyu.edu/courses/spring18/CSCI-UA.0002-004/hw9.html