Dictionaries

CSE 1310 – Introduction to Computers and Programming
Vassilis Athitsos
University of Texas at Arlington

Keys and Values

- Oftentimes we need lists of pairs, that associate values to specific keys.
  - E.g.: product codes are keys, and prices are values.
  - E.g.: names are keys, and phone numbers are values.

```
phone_list = [['mary', 2341], ['joe', 5423]]
```

Life Without Dictionaries

- Oftentimes we need lists of pairs, that associate values to specific keys.
  - E.g.: product codes are keys, and prices are values.
  - E.g.: names are keys, and phone numbers are values.

```
phone_list = [['mary', 2341], ['joe', 5423]]
```

- Then, finding the value for a specific key is doable, but a bit of a pain.

Life Without Dictionaries

- Oftentimes we need lists of pairs, that associate values to specific keys.
  - E.g.: product codes are keys, and prices are values.
  - E.g.: names are keys, and phone numbers are values.

```
phone_list = [['mary', 2341], ['joe', 5423]]
```

- Then, finding the value for a specific key is doable, but a bit of a pain.

```
for item in phone_list:
    if (item[0] == 'joe'):
        print(item[1])
```

Life With Dictionaries

- With dictionaries, dealing with key-value pairs becomes much easier.

```
phonebook = {'mary' : 2341, 'joe' : 5423}
```

- Then, finding the value for a specific key is very simple:

```
print(phonebook['joe'])
```

The **in** Operator

- *key* **in** *dictionary* returns true if the specified **key** is a key in the dictionary.
- IMPORTANT: for dictionaries, the **in** operator only looks at keys, not values.

```
>>> phonebook
{'mary': 59013, 'joe': 23432}
>>> 'joe' in phonebook
True
>>> 23432 in phonebook
False
>>> 'bill' in phonebook
False
```

The **del** Function

- You can delete dictionary entries using the **del** function. If your dictionary is stored in a variable called **dictionary_name**, and you want to delete the entry associated with the specific **key**, use this syntax:

  del(dictionary_name[key])

```
>>> phonebook
{'mary': 59013, 'joe': 23432}
>>> del(phonebook['mary'])
>>> phonebook
{'joe': 23432}
```

The **len** Function

- As in lists and strings, the **len** operator gives you the number of elements (i.e., the number of key-value pairs) in the dictionary.

```
>>> phonebook
{'mary': 59013, 'joe': 23432}
>>> len(phonebook)
2
```

The **items** Method

- *dictionary.***items()** returns the set of key-value pairs in *dictionary*.

```
>>> phonebook
{'mary': 59013, 'joe': 23432}
>>> entries = phonebook.items()
>>> for entry in entries:
      print(entry)

('mary', 59013)
('joe', 23432)
```

The **keys** Method

- *dictionary.***keys()** returns the set of keys in *dictionary*.

```
>>> elements = phonebook.keys()
>>> for element in elements:
      print(element)

mary
joe
```

The **values** Method

- *dictionary.***values()** returns the set of values in *dictionary*.

```
>>> elements = phonebook.values()
>>> for element in elements:
      print(element)

59013
23432
```

Example: Frequencies of Words in Text

- Suppose that we have a text file, and we want to:
  - count how many unique words appear in the text.
  - count how many times each of those words appears.

Step 1: Count Frequencies

```
def count_words(filename):
    in_file = open(filename, "r")

    # initialize the dictionary to empty
    result = {}
    for line in in_file:
        words = line.split()
        for word in words:
            if (word in result):
                result[word] += 1
            else:
                result[word] = 1

    return result
```

dictionary operations shown in red

Step 2: Printing the Result

```
def main():
    filename = "file1.txt"
    dictionary = count_words(filename)
    print(dictionary)
```

```
{'final': 1, 'men,': 1, 'brought': 1, 'met': 1, 'and': 5,
'here,': 2, 'Four': 1, 'years': 1, 'nor': 1, 'any': 1,
'not': 5, 'it,': 1, 'nation,': 3, 'say': 1, 'God,': 1,
'unfinished': 1, 'have': 5, 'battlefield': 1, 'nation': 2,
'or': 2, 'come': 1, 'nobly': 1, 'vain-that': 1,
'proposition' …
```

This printout is hard to read

Step 2 Revised

```
def print_word_frequencies(dictionary):
    print()
    for word in dictionary:
        frequency = dictionary[word]
        print(word + ":", frequency)

    print()
```

- We have defined a function that prints the dictionary in a nicer format.
- Remaining problems?

OUTPUT:

nor: 1
fought: 1
last: 1
hallow: 1
endure.: 1
can: 5
highly: 1
rather: 1
of: 5
men,: 1
in: 4
here,: 2
brought: 1
here.: 1
The: 2
on: 2
our: 2
or: 2
…

Step 2 Revised

```
def print_word_frequencies(dictionary):
    print()
    for word in dictionary:
        frequency = dictionary[word]
        print(word + ":", frequency)

    print()
```

- Remaining problems?
  - Result must be case-insensitive.

  E.g., "The" and "the" should not be counted as separate words.

OUTPUT:

nor: 1
fought: 1
last: 1
hallow: 1
endure.: 1
can: 5
highly: 1
rather: 1
of: 5
men,: 1
in: 4
here,: 2
brought: 1
here.: 1
The: 2
on: 2
our: 2
or: 2
…
the: 9
…

Step 2 Revised

```
def print_word_frequencies(dictionary):
    print()
    for word in dictionary:
        frequency = dictionary[word]
        print(word + ":", frequency)

    print()
```

- Remaining problems?
  - We should ignore punctuation.

  E.g., "here" and "here." should not be counted as separate words.

OUTPUT:

nor: 1
fought: 1
last: 1
hallow: 1
endure.: 1
can: 5
highly: 1
rather: 1
of: 5
men,: 1
in: 4
here,: 2
brought: 1
here.: 1
The: 2
on: 2
our: 2
or: 2
…

Step 2 Revised

```
def print_word_frequencies(dictionary):
    print()
    for word in dictionary:
        frequency = dictionary[word]
        print(word + ":", frequency)

    print()
```

- Remaining problems?
  - Would be nice to sort, either alphabetically or by frequency.

OUTPUT:

nor: 1
fought: 1
last: 1
hallow: 1
endure.: 1
can: 5
highly: 1
rather: 1
of: 5
men,: 1
in: 4
here,: 2
brought: 1
here.: 1
The: 2
on: 2
our: 2
or: 2
…

Making Result Case Insensitive

- To make the results case-insensitive, we can simply convert all text to lower case as soon as we read it from the file.

Making Result Case Insensitive

```
def count_words(filename):
    in_file = open(filename, "r")

    # initialize the dictionary to empty
    result = {}
    for line in in_file:
        line = line.lower()
        words = line.split()
        for word in words:
            if (word in result):
                result[word] += 1
            else:
                result[word] = 1

    return result
```

PREVIOUS OUTPUT:

in: 4
here,: 2
brought: 1
here.: 1
The: 2
on: 2
our: 2
or: 2
…
the: 9
…

156 words found

Making Result Case Insensitive

```
def count_words(filename):
    in_file = open(filename, "r")

    # initialize the dictionary to empty
    result = {}
    for line in in_file:
        line = line.lower()
        words = line.split()
        for word in words:
            if (word in result):
                result[word] += 1
            else:
                result[word] = 1

    return result
```

NEW OUTPUT:

…
devotion-that: 1
field,: 1
honored: 1
testing: 1
far: 2
the: 11
from: 2
advanced.: 1
full: 1
above: 1
…

153 words found

Ignoring Punctuation

- To ignore punctuation, we will:
  - delete from the text that we read all occurrences of punctuation characters (periods, commas, parentheses, exclamation marks, quotes).
  - We will replace dashes with spaces (since dashes are used to separate individual words).

- To isolate this processing step, we make a separate function for it, that we call **process_line**.

Ignoring Punctuation

```
def process_line(line):
    line = line.lower()
    new_line = ""

    for letter in line:
    if letter in """,.!"'()""":
            continue
        elif letter == '-':
            letter = ' '
        new_line = new_line + letter

    words = new_line.split()
    return words
```

Ignoring Punctuation

```
def count_words(filename):
    in_file = open(filename, "r")

    # initialize the dictionary to empty
    result = {}
    for line in in_file:
        words = process_line(line)
        for word in words:
            if (word in result):
                result[word] += 1
            else:
                result[word] = 1

    return result
```

PREVIOUS OUTPUT:

people,: 3
under: 1
those: 1
to: 8
men,: 1
full: 1
are: 3
it,: 1
…
for: 5
whether: 1
men: 1
sense,: 1
…

153 words found

Ignoring Punctuation

```
def count_words(filename):
    in_file = open(filename, "r")

    # initialize the dictionary to empty
    result = {}
    for line in in_file:
        words = process_line(line)
        for word in words:
            if (word in result):
                result[word] += 1
            else:
                result[word] = 1

    return result
```

NEW OUTPUT:

fathers: 1
people: 3
forth: 1
for: 5
men: 2
ago: 1
field: 1
increased: 1
…

138 words found

Sorting by Frequency

- We want to sort the results by frequency.
- Frequencies are **values** in the dictionary.
- So, our problem is the more general problem of sorting dictionary entries by value.
- To do that, we will create an inverse dictionary, called **inverse**, where:
  - inverse[frequency] is a list of all words in the original dictionary having that frequency.
- To isolate this processing step, we make a separate function for it, that we call **inverse_dictionary**.
- We use **inverse_dictionary** in **print_word_frequencies**.

Sorting by Frequency

```
def inverse_dictionary(in_dictionary):
    out_dictionary = {}
    for key in in_dictionary:
        value = in_dictionary[key]
        if (value in out_dictionary):
            list_of_keys = out_dictionary[value]
            list_of_keys.append(key)
        else:
             out_dictionary[value] = [key]

    return out_dictionary
```

Sorting by Frequency

```
def print_word_frequencies(dictionary):
    print()
    inverse = inverse_dictionary(dictionary)
    frequencies = inverse.keys()
    frequencies = list(frequencies)
    frequencies.sort()
    frequencies.reverse()

    for frequency in frequencies:
        list_of_words = inverse[frequency]
        list_of_words.sort()
        for word in list_of_words:
            print(word + ":", frequency)
```

Sorting by Frequency

```
def print_word_frequencies(dictionary):
    print()
    inverse = inverse_dictionary(dictionary)
    frequencies = inverse.keys()
    frequencies = list(frequencies)
    frequencies.sort()
    frequencies.reverse()

    for frequency in frequencies:
        list_of_words = inverse[frequency]
        list_of_words.sort()
        for word in list_of_words:
            print(word + ":", frequency)
```

PREVIOUS OUTPUT:

live: 1
above: 1
but: 2
government: 1
gave: 2
note: 1
remember: 1
advanced: 1
world: 1
whether: 1
equal: 1
seven: 1
task: 1
they: 3
…

153 words found

Sorting by Frequency

```
def print_word_frequencies(dictionary):
    print()
    inverse = inverse_dictionary(dictionary)
    frequencies = inverse.keys()
    frequencies = list(frequencies)
    frequencies.sort()
    frequencies.reverse()

    for frequency in frequencies:
        list_of_words = inverse[frequency]
        list_of_words.sort()
        for word in list_of_words:
            print(word + ":", frequency)
```

NEW OUTPUT:

that: 13
the: 11
we: 10
here: 8
to: 8
a: 7
and: 6
can: 5
for: 5
have: 5
it: 5
nation: 5
not: 5
of: 5
…

153 words found