02-2 Dictionaries

CSS 500

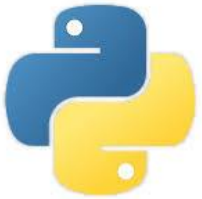
Spring 2018

Course material derived from:
Downey, Allen B. 2012. "Think Python, 2nd Edition". O'Reilly Media Inc., Sebastopol CA.

"How to Think Like a Computer Scientist" by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. Oct 2012
http://openbookproject.net/thinkcs/python/english3e/index.html

A dictionary is a mapping

- Recall: when using a list, the indexes must be integers
  - But when using a dictionary, the indexes can be almost anything
- Set of indexes are called "keys"
  - Each index has an associated "value"
  - These are called "key-value" pairs
- The dict() function creates a new dictionary
  - squiggly braces can also be used {}



```
# dictinary mapping English to Spanish
eng2sp = dict()
eng2sp
{ }

# let's add a word to our dictionary
eng2sp[ 'one' ] = 'uno'

eng2sp
{ 'one' : 'uno' }

# let's create a larger dictionary
eng2sp = {'one':'uno', 'two':'dos', 'three':tres}

# order is NOT preserved !
eng2sp
{ 'one':uno, 'three':'tres', 'two':'dos'}
```

Dictionaries and tuples

- Dictionaries have a built-in method called items that returns a sequence of tuples representing key-value pairs
  - the 'dict_items' object is an iterator
  - allows easy access to key-value pairs
- You can use a set of tuples to create a new dictionary
  - Combined with zip() and range(), this makes an easy way to create a dictionary

```
# use dictionary items method
d = { 'a':0, 'b':1, 'c':2 }
t = d.items()
t
dict_items( [ ('b', 1), ('a', 0), ('c', 2) ] )
for key, value in d.items():
        print( key, value )
b 1
a 0
c 2
# make a new dictionary from list of tuples
t = [ ('a',0), ('b',1), ('c',2) ]
t
[ ('a', 0), ('b', 1), ('c', 2) ]        # a list
d = dict( t )
d
{'b': 1, 'a': 0, 'c': 2}                # a dictionary

# slick way to make dictionary
d = dict( zip( 'abc', range(3))  )
d
{'b': 1, 'a': 0, 'c': 2}
```

Dictionaries as counters

- Dictionaries are ideal for tasks involving counting
  - count only items that are seen
  - don't need to know entire possible set of things in advance

- built-in "get" function
  - takes item (key) to find
  - returns number of times key is found
  - takes alternate value to return if key not found

```
# let's make a histogram

# function to count characters in a string
def histogram( s ):
    d = dict()
    for ch in s:
        if ch not in  d:
            d[ch] = 1
        else:
            d[ch] += 1
    return d

histogram('brontosaurus')
{'o': 2, 'b': 1, 't': 1, 'r': 2, 's': 2, 'n': 1, 'u': 2, 'a': 1}

h = histogram('brontosaurus')
h.get('r',  0)
2
h.get('z', 0)
0
```

Looping and dictionaries

- A for loop can be used to traverse the keys
  - keys are in no particular order
  - values come out willy-nilly
- To produce sorted output, traverse the keys in sorted order
  - keys sorted
  - values associated with sorted keys

```
# function to print a histogram
def print_hist( h ):
    for ch in h:
        print(ch, h[ch] )

h = histogram( 'parrot')
print_hist( h )
r 2
p 1
a 1
t 1
o 1

>>># function to print a sorted histogram
def print_sorted_hist( h ):
    for ch in sorted(h):
        print(ch, h[ch] )

print_sorted_hist( h )
a 1
o 1
p 1
r 2
t 1
```

Reverse lookup

- Given a dictionary and a key, it's easy to find a value
  - this is called "lookup"
- What about the reverse problem: how to find a key given a value?
  - might be more than one instance of value
  - We have to search through the dictionary
  - If not found, we "raise" an error, in this case a user-defined "LookupError"

```
# reverse lookup
def reverse_lookup( d, v):
    for k in d:
        if d[k] == v:
            return k
    raise LookupError()

h = histogram('parrot')
key = reverse_lookup(h, 2)
key
'r'             # we have two 'r's in  'parrot'

# what happens if there isn't data for a key?
key = reverse_lookup(h, 3)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    key = reverse_lookup(h, 3)
  File "C:/Users/slscott/Desktop/Downey
Python Examples/Week 4 examples v01.py",
line 36, in reverse_lookup
    raise LookupError()
LookupError
```

Dictionaries and Lists

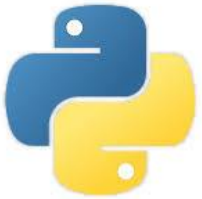
- Lists can be used as values in a dictionary
- example: "forward" dictionary
  - key is the letter
  - value is the number of occurrences
- example: "reverse" dictionary
  - key is the number of times something occurred
  - value is a list of letters occurring 'key' times

```
# reverse lookup
# invert dictionary
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse

word = 'supercaliragilisticexpialodocious'
h = histogram( word )
h
{'s': 3, 't': 1, 'f': 1, 'c': 3, 'e': 2, 'a': 3, 'r': 2, 'o': 1, 'g': 1,
'p': 2, 'l': 3, 'd': 1, 'i': 6, 'x': 1, 'u': 2}
inverse = invert_dict( h )
inverse
{1: ['t', 'f', 'o', 'g', 'd', 'x'], 2: ['e', 'r', 'p', 'u'], 3: ['s', 'c',
'a', 'l'], 6: ['i']}
```

Dictionaries as hash

- A "hash" is a function that takes a value of any type and returns an integer
- Dictionaries use these returned integer values as "keys" to store and look up values
- Keys must be immutable, or else the correspondence gets lost
- Ideally each value maps to a unique key (in practice this doesn't always work)



```
# a bad example of a hash function
def hashfunc( value ):
        return len(value)

print( hashfunc( 'bacon' ) )
5
print( hashfunc( 'ham' ) )
3
print( hashfunc( 'spam' ) )
4
print( hashfunc( 'eggs' ) )
4
```

key	value
5	bacon
3	ham
4	spam
4 (broken!)	eggs

Using dictionaries

- Let's use Python dictionary to count the number of words in a text
- General outline:
  - read in a text
  - break the text into a List of words
  - iterate over the list
    - if we find a new word, set the word's counter to 0
    - if we've seen the word before, increment the word's counter by 1
  - print out the words and their frequencies
- You probably should create a new text file for this example…

Our source text:

Humpty Dumpty sat on a wall
Humpty Dumpty had a great fall
All the king's horses and all the king's men
Couldn't put Humpty together again

Preprocessing the text

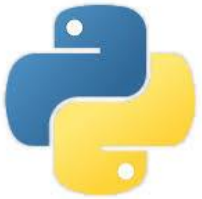
- Assume text is stored in a long character String
- we need to parse it into words
- use the split() operator, break on white space ' '
- also let's convert to lower case for consistency

```
text = "Humpty Dumpty sat on a wall \
Humpty Dumpty had a great fall \
All the King's horses and all the King's men \
Couldn't put Humpty together again"

words = []
for item in text.split(' '):    # split on ' '
    item = item.lower()         # make all lower case
    words.append(item)          # append word to list
```

Build the dictionary

- create an empty dictionary
- iterate over the word list
  - if not found in the dictionary's keys, then add word to dictionary with count = 1
  - if found in dictionary's keys, increment word count

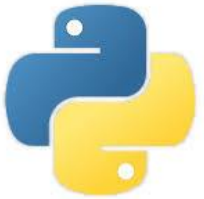


```
# create dictionary
word_freq = {}

# populate with words to frequencies
for word in words:
    word = word.lower()
    if word in word_freq.keys():
        word_freq[word] += 1
    else:
        word_freq[word] = 1
```

Print the results

- Print out the keys
  - these are the words we found
  - note: keys are not sorted by default
- Print out the values
  - these are the frequencies associated with each word



```
print(' ')
print('Results: words to frequencies')
print(word_freq.keys())
print(word_freq.values())
```

```
Results: words to frequencies

dict_keys(['humpty', 'dumpty', 'sat', 'on', 'a',
'wall', 'had', 'great', 'fall', 'all', 'the', "king's",
'horses', 'and', 'men', "couldn't", 'put',
'together', 'again'])

dict_values([3, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1,
1, 1, 1, 1, 1, 1])
```

Summary

- Dictionaries are a built-in Python data structure for key-value pairs
  - key: a hashable object used to identify a value
  - value: the value associated with the key
  - This concept is widely used in NO-SQL data architectures like JSON
- Helpful functions for dictionaries
  - dict() creates a new dictionary (can also use squiggly braces {})
  - d.keys() returns the keys
  - d.values() returns the values
  - d.items() returns a dict_item iterator object used to get the values