Python Dictionaries

Kelly M. Thayer

Lecture 9

Week 10

2017-04-11

Contact info:
kthayer@wesleyan.edu
Office Hours: M 8:30-10:40, Tue 10:10-11:10,
Wed 9:40-10:40, 5:00-6:00
Office: Exley 322/325

What is a Python Dictionary?

- **Dictionary** –a data structure in Python consisting of an unordered set of key:value pairs
- **Keys** – the indices of the dictionary.
  - Used to look up the values.
  - Must be unique for each dictionary
- **Value** – to what the key is mapped
- One key maps to one value

Example: colors in English and Spanish

key -> value
orange -> anaranjado
yellow -> amarillo
blue -> azul

Preview of Dictionaries: What are they good for?

```
# using lists
def match_by_list(search):
    # figure out the index of the search term
    i=0
    match=-1000 # initialize it to arbitrary value
    while i < len(engcolor):
        if engcolor[i] == search:
            match=i
            break
        i+=1
    if match !=-1000: # if it got updated bc a match was found
        # look in the corresponding list for the term.
        # assume the lists preserve the order of the matching pairs
        return spcolor[match]
    else:
        print('Sorry, your match term was not found.')
        return

engcolor=['red', 'orange',    'yellow', 'green', 'blue','violet']
spcolor= ['rojo','anaranjado','amarillo','verde','azul','violeta']

print("using match_by_list for 'red'")
print(match_by_list('red'))
print("using match_by_list for 'pink'")
print(match_by_list('pink'))
```

Match found

```
>>>
using match_by_list for 'red'
rojo
using match_by_list for 'pink'
Sorry, your match term was not found.
None
```

```
# using a dictionary
colors={'red':'rojo','orange':'anaranjado','yellow':'amarillo',\
        'green':'verde','blue':'azul','violet':'violeta'}
print(colors['red'])
```

Match found

```
>>>
using a dictionary for 'red'
rojo
using a dictionary for 'pink'
Traceback (most recent call last):
  File "D:\09_Python_course  S17\01_L
ecture\lecture09_dictionaries_code.py
", line 41, in <module>
    print(colors['pink'])
KeyError: 'pink'
```

not in dictionary

Application of Dictionaries

- Any time you want to have a collection of pairwise associated elements
- Especially useful when numeric indices would not work (i.e. when you can not make a list)
- When you need to make a histogram

Examples of Dictionary Applications:

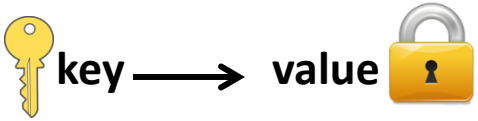
- Biology: DNA -> RNA -> Protein
- Chemistry: element -> physical property
- Physics: formula parameter -> value
- Foreign Lang: English word -> Spanish word
- Everyday: store -> [shopping_list]
- Music: Scale transposition
- Statistics: Making a histogram of a dataset

Learning Objectives

- Understand when to use the dictionary data structure
- Mapping and dictionaries
- Basic syntax for dictionaries
- How to perform common tasks related to dictionaries
  - Loop over a dictionary
  - Reverse lookup
  - Find out if a value is in a dictionary
- Use dictionaries as a set of counters

BASIC CONCEPTS AND SYNTAX

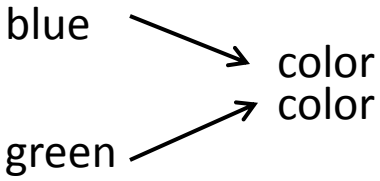
Uniqueness of Keys and Mapping



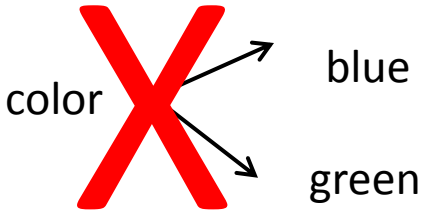
sí ⟶ yes

hola ⟶ hi

Each mapping is unique



Two keys may refer to a value that happens to be the same



Not possible:

- One key refers to two different values;
- Two values have the same key

Dictionary Syntax: Create a Dictionary

- Use curly brackets **{}** to indicate the dictionary data structure
- Associate a key with its value using a colon **:**
- Separate key-value pairs with commas **,**

Syntax:

```
# Create a dictionary
<mydict>={<key1>:<value1>, <key2>:<value2>, …,
<keyn>:<valuen>}
```

Example:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> eng2sp
{'three': 'tres', 'one': 'uno', 'two': 'dos'}
>>>
```

Dictionary Syntax: Adding a New Entry

Syntax:

add a key:value pair entry
<mydict>[<key>]=<value>

Example:

```
>>> eng2sp = {'one':'uno','two':'dos','three':'tres'}
>>> eng2sp['four']='cuatro'    # entry 'four':'cuatro' is added to the dictionary
>>> eng2sp
{'one': 'uno', 'two': 'dos', 'three': 'tres', 'four': 'cuatro'}
```

Dictionary Syntax: Looking up a Value in the Dictionary

- **Look up** – to find the value that corresponds to the key

look up a value for a particular key

<mydict>[<key>]

Example:

```
>>> eng2sp
{'one': 'uno', 'two': 'dos', 'three': 'tres', 'four': 'cuatro'}
>>> eng2sp['one']
'uno'
```

Dictionary Syntax: Printing the Dictionary

Syntax:

```
# print the contents of a dictionary
print(<mydict>)
```

Example:

```
>>> print(eng2sp)
{'three': 'tres', 'one': 'uno', 'four': 'cuatro', 'two': 'dos'}
>>>
```

#Note the order of pairs may not be preserved. Hold that thought…

Dictionary Syntax: Creating an Empty Dictionary

#create a dictionary with the dictionary command

Syntax:

<mydict> = dict()

Example:

```
>>> eng2sp = dict()
>>> eng2sp
{}
>>>
```

create a dictionary using curly brackets {}

Syntax:

<mydict> = {}

Example:

```
>>> eng2sp = {}
>>> eng2sp
{}
>>>
```

Order of Entries

- The order of entries in a dictionary does not matter
- The mapping of the key:value pairs determines if the dictionaries are equal or not

```
>>> mydict1={"A":1,"B":2}
>>> mydict2={"B":2,"A":1}
>>> mydict3={"A":2,"B":1}
>>> mydict1 == mydict2
True
>>> mydict1 == mydict3
False
```

Mutability

- Dictionaries are mutable
- However, the keys are not mutable
- The value to which a key is mapped can be changed

Syntax:

<dictionary>[<existing_key>]=<new_value>

Example:

```
>>> colors={'red':'rojo','orange':'anaranjado','yellow':\
'amarillo','green':'verde','blue':'azul','violet':'violeta'}
>>> colors['red']='rouge'
>>> colors
{'blue': 'azul', 'yellow': 'amarillo', 'red': 'rouge', 'violet'
: 'violeta', 'orange': 'anaranjado', 'green': 'verde'}
```

`len` and Dictionaries

the `len` function returns the number of key:value pairs

`len(<mydict>)`

```
>>> eng2sp
{'one': 'uno', 'three': 'tres', 'two': 'dos', 'four': 'cuatro'}
>>> len(eng2sp)
4
>>>
```

`in` and Dictionaries

the in operator tells if the input appears in the keys

<input> in <mydict>

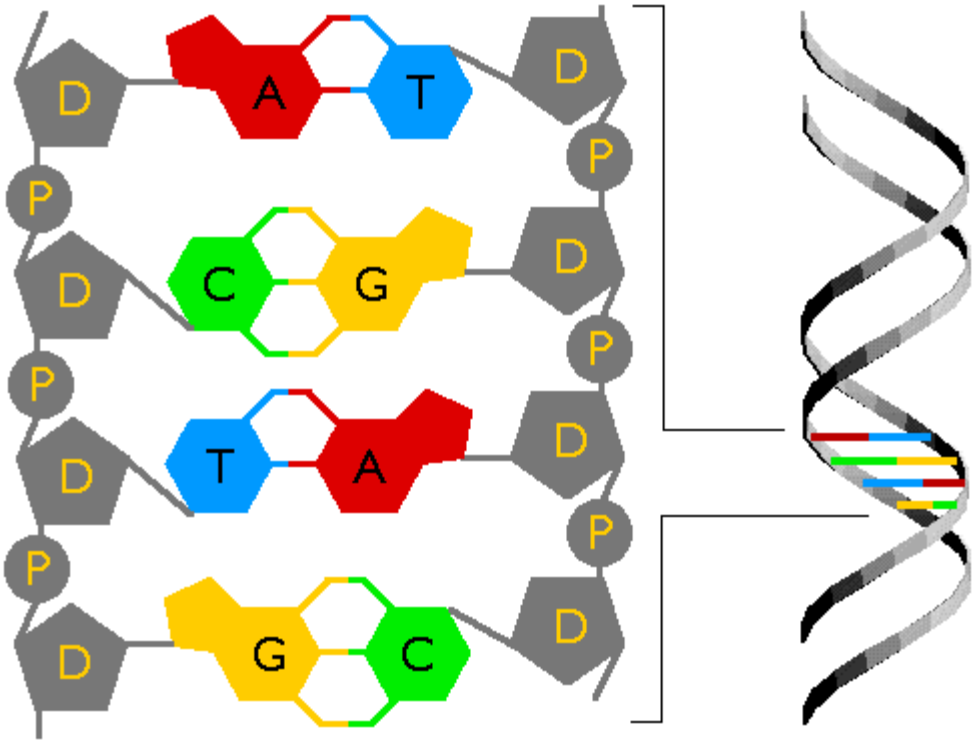
```
>>> 'two' in eng2sp
True
```

however it will not find the input in the values.

```
>>> 'dos' in eng2sp
False
```

```
>>> eng2sp
{'one': 'uno', 'three': 'tres', 'two': 'dos', 'four': 'cuatro'}
```

Ex: DNA Base Pairing as a Dictionary



Mapping

A -> T
C -> G
T -> A
G -> C

DNA base pairs in a predictable pattern due to its chemical properties

Example: Get DNA Complement Using a Dictionary

```
def get_complement(DNA):
    # signature: str -> str
    # computes the complement of a DNA sequence
    match=""
    for base in DNA:
        match += DNAbp[base]
    return match

# define the dictionary
DNAbp = {'A':'T','T':'A','G':'C','C':'G'}
# DNA sequence needing complement
DNA='AGCT'
# print the given and complement DNA seqences
print('DNA+', DNA)
print('DNA-', get_complement(DNA))
```

```
>>>
 RESTART: F:/09_Python_course  S17/01_Lecture/lecture09_dictionaries_code.py
DNA+ AGCT
DNA- TCGA
```

Dictionary Syntax: Keys Associated with Lists

- A dictionary key can be associated with a value which is a list

Syntax:

<mydict>={<key1>:[element1.1, element1.2, …element1.n], <key2>:[element2.1, element2.2, …element2.n], …,<keym>:[elementm.1, elementm.2, …m.n]}

Example:

```
wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}
```

Dictionary Syntax: Keys Associated with Lists

- Obtain the element of a list of a dictionary using the dictionary key and then specifying which element you want to be returned.

Syntax

```
<mydict>[<key_to_list>][<list_index_or_slice>]
```

Example:

```
>>> wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}
>>> print(wesPlaces['Freeman'][0])
athletic center              # returns the list entry
>>> print(wesPlaces['Freeman'][0:2])
['athletic center', '34']  # returns a list
```

Coding Exercise: Wesleyan Campus Tour

Write a function **`get_description`** by using the **`wesPlaces`** dictionary (below) to help prospective students and their families learn about the stops on the campus tour of the day. The dictionary contains a brief description of Wesleyan destinations and the corresponding number on the campus map. Design your function to look up a description of the stops on the tour itinerary and print the name of destination and its description. A sample input and output is given.

```
wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}

todaystour=["Admission","Olin","Freeman","Neon","Exley","Butterfield"]
get_description(todaystour)

>>> ================================ RESTART ================================
>>>
Admission - undergraduate admissions office
Olin - main library
Freeman - athletic center
Neon - Please ask your tour guide.
Exley - natural sciences and mathematics
Butterfield - dorms
```

Coding Exercise Solution: Wesleyan Campus Tour

```
def get_description(tour):
    # signature: list(str) -> noneType
    # looks up the description of Wesleyan tour destinations
    for destination in tour:
        if destination in wesPlaces:
            print (destination, '-',wesPlaces[destination][0])
        else:
            print(destination, '-','Please ask your tour guide.')

wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}

todaystour=["Admission","Olin","Freeman","Neon","Exley","Butterfield"]
get_description(todaystour)
```

```
>>> ================================ RESTART ================================
>>>
Admission - undergraduate admissions office
Olin - main library
Freeman - athletic center
Neon - Please ask your tour guide.
Exley - natural sciences and mathematics
Butterfield - dorms
```

`for` loops, reverse lookup, `values` method

MORE THINGS TO DO WITH DICTIONARIES

`for` Loops and Dictionaries

- A **`for`** loop can be used to traverse a dictionary's keys
- <variable> takes on the values of each key in turn

```
for <variable> in <mydict>:
        <loop_item1>
        <loop_item2>
        …
        <loop_itemn>
```

```
>>> eng2sp = {'one':'uno','two':'dos','three':'tres'}
>>> for k in eng2sp:
        print(k)
```

```
three
two
one
```

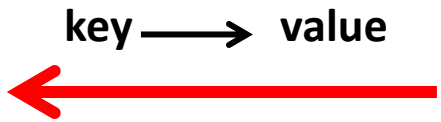
Sample Code with `for` Loop: Predict the Output

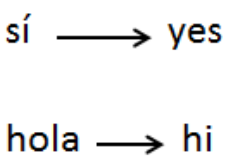
```
def process():
    acc=[]
    for place in wesPlaces:
        acc.append(wesPlaces[place][1] + ' ' + place)
    acc.sort()
    return acc

wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}

processed = process()
i=0
while i < len(processed):
    print (processed[i])
    i+=1
```

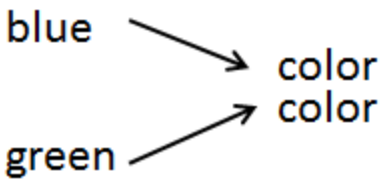
```
def process():
    acc=[]
    for place in wesPlaces:
        acc.append(wesPlaces[place][1] + ' ' + place)
    acc.sort()
    return (acc)

wesPlaces={"Exley":["natural sciences and mathematics","17"], \
      "Olin":["main library","23"], "Usdan":["food court","64"], \
      "Freeman":["athletic center","34"],"Butterfield":["dorms","50"],\
      "Admission":["undergraduate admissions office","65"],\
      "WesWings":["dining","25"],"Shanklin":["Biology","29"],\
      "Hall-Atwater":["Chemistry and Molecular Biology","19"]}

processed = process()
i=0
while i < len(processed):
    print (processed[i])
    i+=1

RESTART: F:/09_Python_course  S17/01_Lecture/lecture09_dictionaries_code.py
17 Exley
19 Hall-Atwater
23 Olin
25 WesWings
29 Shanklin
34 Freeman
50 Butterfield
64 Usdan
65 Admission
```

Reverse Lookup

- **Reverse Lookup -** given a value, find the key to which it corresponds.
- Any potential problems??





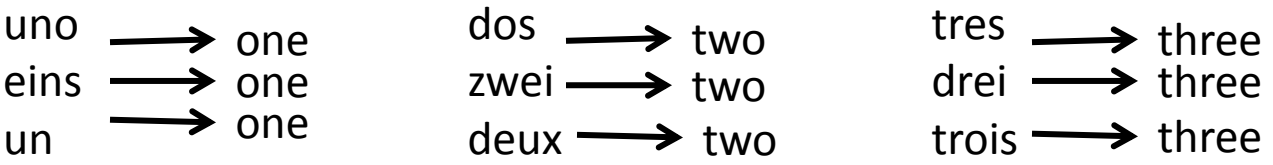


Reverse Lookup

- Could be more than one key that matches the value
- Deal with that in a problem specific way
  - Sometimes there may be a criterion to decide which key you want
  - Sometimes you will want all keys having that value
- You have to write the program yourself specific to the application of interest.

Reverse Lookup Example: Foreign Languages to English Dictionary

```
fl2eng={'uno':'one', 'dos':'two', 'tres':'three', \
        'eins':'one', 'zwei':'two', 'drei':'three', \
        'un':'one','deux':'two','trois':'three'}
```



Example:
Find All Keys Having the Value 'two'

```
def rev_look(mydict, val):
    # performs reverse lookup
    match_key=[]
    for key in mydict:
        if mydict[key] == val:#if current value == val we are looking for
            match_key.append(key) # save the key to match_key list
    return match_key

# define the dictionary
fl2eng={'uno':'one', 'dos':'two', 'tres':'three', \
        'eins':'one', 'zwei':'two', 'drei':'three', \
        'un':'one','deux':'two','trois':'three'}

# function call and report answer
answer =rev_look(fl2eng, 'two')
print(answer)
```

```
>>>
===== RESTART: E:/09_Python_course/01_Lecture/lecture10_reverselookup.py =====
['dos', 'deux', 'zwei']
>>>
```

`values` Method and Dictionaries

- The method **`values()`** returns a list of al the values available in a given dictionary

Syntax:

`<mydict>.values()`

Example:

```
>>> colors
{'blue': 'azul', 'yellow': 'amarillo', 'red': 'rouge', 'violet':
'violeta', 'orange': 'anaranjado', 'green': 'verde'}
>>> colors.values()
dict_values(['azul', 'amarillo', 'rouge', 'violeta', 'anaranjado'
, 'verde'])
```

Find Out If a Value Is in a Dictionary

- use **`values ()`** method to find out if a value is in a dictionary.

Syntax:

`<searchterm> in mydict.values()`

Example:

```
>>> colors={'red':'rojo','orange':'anaranjado','yellow':\
'amarillo','green':'verde','blue':'azul','violet':'violeta'}
>>> 'azul' in colors.values()
True
>>> 'red' in colors.values()
False          # 'red' is a key, not a value
>>> 'morado' in colors.values()
False          # 'morado' is not a value in the dictionary
```

DICTIONARIES AS COUNTERS

Counting: Ice Cream Shop

- Suppose an ice cream shop just started selling smoothies and wants to find out which flavor is the most popular
- Take a survey.
- How can the results be tallied by Python?

Some Possible Implementations

- Make a series of separate counters
  - Each counter is a variable
  - use looping and if statements to increment the counter corresponding to the letter
- Make a counter out of a list
  - The elements of the list record the number of matches
  - Assign a list position to be a counter for a particular letter
  - Traverse the sequence and increment the counter; keep track of the letter, figure out its index for the list, and increment that position
- **Consider using a dictionary**

Implementation – a way of performing a computation or algorithm

Counting

```
def count_with_vars(survey):
    #signature: list(str) -> noneType

    choc=0
    vanl=0
    stby=0
    othr=0
    for i in survey:
        if i == 'c':
            choc+=1
        elif i == 'v':
            vanl += 1
        elif i == 's':
            stby += 1
        else:
            othr +=1
    tally=[choc,  vanl,   stby,   othr]
    label=['choc','vanl','stby','othr']

    j = 0
    while j < len(tally):
        print(label[j],tally[j])
        j+=1
    return

survey=['v','c','v','v','s']
count_with_vars(survey)
```

```
# output
choc 1
vanl 3
stby 1
othr 0
```

Dictionary as a Counter

```
def count_with_dict(survey):
    dictFlavor={}
    for flavor in survey:
        if flavor not in dictFlavor:
            dictFlavor[flavor]=1 # create key and counter
        else:
            dictFlavor[flavor]+=1 # increment existing counter
    return dictFlavor

survey=['v','c','v','v','s']
print(count_with_dict(survey))
```

```
 RESTART: F:\09_Python_course  S17\01_Lecture\lecture09
{'v': 3, 's': 1, 'c': 1}
```

Exercise: Sequence Analysis

Write a function **`tally_letter`** that returns a dictionary with the tallies of the letters in a sequence **`seq`**. Then, write a second function **`percent`** that computes and prints the percentage of each letter. Sample input and output is given.

```
seq='GAATTCGCG'
myDict = tally_letter(seq)
percent(myDict)
```

```
>>> ================================ RESTART =====
>>>
G 33.33333333333333
T 22.22222222222222
C 22.22222222222222
A 22.22222222222222
```

Solution: Sequence Analysis

```
def tally_letter(seq):
    # takes a string of DNA bases and computes % composition
    myDict={}
    for letter in seq:
        if letter not in myDict:
            myDict[letter]=1 # create key and set value to 1
        else:
            myDict[letter] +=1 # increment existing
    return myDict

def percent(myDict,seq):
    # signature: dict -> noneType
    for letter in myDict:
        print(letter, myDict[letter]/len(seq)*100)

someseq='GAATTCGCG'
someDict = tally_letter(someseq)
percent(someDict,someseq)

>>> ==================
>>>
G 33.33333333333333
T 22.22222222222222
C 22.22222222222222
A 22.22222222222222
```

CONCLUDING REMARKS

The Big Picture: Comparing Dictionaries and Lists

Similarities between Lists and Dictionaries

- Contain multiple entries
- Entries can be extracted using their indices
- An entry may itself be a list

Differences

Lists

- Entries are stored in a specific order; the position in the list is important
- Entries can be accessed using the slice operator with [] notation; each item has an associated position in the list.

Dictionaries

- Entries are not stored in a specific order; the pair association is important
- Entries are looked up using the key.
- Reverse lookup also possible.
- Can be used for making histograms

Summary and Review

- Understand when to use the dictionary data structure
- Mapping and dictionaries
- Basic syntax for dictionaries
- How to perform common tasks related to dictionaries
  - Loop over a dictionary
  - Reverse lookup
  - Find out if a value is in a dictionary
- Use dictionaries as a set of counters

Summary: Syntax

- <mydict>=dict() # declare a dictionary
- <mydict>={} # declare a dictionary
- <mydict>[<key>]=<value> # add or append one key:value pair
- <mydict>={<key1>:<value1>,<key2>:<value2>,…<keyn>:<valuen>} # declare a dictionary with multiple entries
- print (<mydict>) # print dictionary
- <mydict>[<key>] # look up value of a key
- len(<mydict>) # number of key:value pairs
- <input> in <mydict> # Boolean search of keys
- for key in <mydict>: # loop over keys
- <mydict>.values() # reports all values in dictionary
- <searchterm> in mydict.values() # search for a value in dict

To Do This Week:

- Do the reading:
  - Chapter 11: Dictionaries
- Attend your lab section

END