New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Introduction to Python
Exercise Solutions, Session 5

Ex. 5.1 Given the following dict:

```
mydict = {'a': 1, 'b': 2, 'c': 3}
```

In two statements, add two key/value pairs to mydict (continuing the series values indicated). Print the dict.

Suggested Solution:

```
mydict = {'a': 1, 'b': 2, 'c': 3}    # initialize a dict

mydict['d'] = 4                      # set key/value pair 'd':4 in dict
mydict['e'] = 5                      # set key/value pair 'e':5 in dict

print(mydict)
```

Be sure that you are using the above subscript syntax to add a key/value pair to a dict, rather than by attempting to use a dict method (like the dict update() method, which is used to combine dictionaries). Subscript is the preferred way to add single key/value pairs.

Comparing the supplied dict with the dict after you have added the two key/value pairs, note that the keys are out of order. (Using Python 3.6 or later, however, dictionaries will keep their order.)

Ex. 5.2 Given the following list:

```
mylist = ['Hey', 'there', 'I', 'am', 'amazing!']
```

Initialize an empty dictionary. Loop through each element in mylist, and add a key/value pair to the dict where the key is the word and the value is the len() of the word. Print the dict.

Suggested Solution:

```
mylist = ['Hey', 'there', 'I', 'am', 'amazing!']

mydict = {}                    # initialize an empty dict

for word in mylist:            # for each string in list of strings
    mydict[word] = len(word)   # set key/value pair: the str : its length

print(mydict)
```

This exercise simply gets you started adding key/value pairs to an empty dictionary from a "looping source", in this case values from a list. (The len() value is just there so you can associate some related value to each string, i.e. its integer length.) It's a step towards the goal of looping through a file, splitting out an element from each line, and then adding that element (along with an associated value) to a dict.

As with last week's container work, we must be careful to initialize the dict before the loop begins, and print the dict only after the loop ends.

Again, note that before Python 3.6, dictionary keys appear in "random" order (more specifically, in an order that is useful only to Python.)

Ex. 5.3 Modify the above script: at the end of the script, instead of printing the dict directly, loop through the dict and print each key and value pair with a descriptive message.

Suggested Solution:

```
mylist = ['Hey', 'there', 'I', 'am', 'amazing!']

mydict = {}                    # initialize an empty dict

for word in mylist:            # for each string in list of strings
    mydict[word] = len(word)   # set key/value pair: string : its length

for key in mydict:             # for each key in dict (set above)
    print('"{}" is len {}.'.format(key, mydict[key]))   # print key and val
```

This exercise adds the step of looping through a dictionary that has been built earlier in the script and reporting its keys and values.

Ex. 5.4 Modify the above script: at the end of the script, instead of looping through the dict, use input() to ask for one of the words. If the user inputs one of the words, the program prints the word and its length.

Suggested Solution:

```
mylist = ['Hey', 'there', 'I', 'am', 'amazing!']

mydict = {}                    # initialize an empty dict

for word in mylist:            # for each string in list of strings
    mydict[word] = len(word)   # set key/value pair: string: its length

uinput = input('please enter a word:  ')   # keyboard input return str

input_val = mydict[uinput]      # retrieve the dict value for this word
print(('the word "{}" is len {}'.format(uinput, input_val)))
```

This exercise illustrates how , once the dict has been built, you can look up the value associated with a key in the dict.

Ex. 5.5 Extend the above script to allow for a missing key. If the user's input word does not exist in the dict, print a statement to that effect (hint: use the **in** operator to test for the existence of a key).

Suggested Solution:

```
mylist = ['Hey', 'there', 'I', 'am', 'amazing!']
mydict = {}                    # initialize an empty dict

for word in mylist:            # for each string in list of strs
    mydict[word] = len(word)   # set key/value pair:  string:  its length

uinput = input('please enter a word:  ')   # keyboard input return str

if uinput in mydict:            # test to see if input str a key in dict
    input_val = mydict[uinput]      # retrieve the dict value for this word
    print('the word "{}" is len {}'.format(uinput, input_val))
else:                          # if input str not key in dict
    print('the word "{}" does not exist in the dictionary'.format(uinput))
```

Of course if we are looking up a key in the dict there is always the possibility that the key will be missing, which would cause a KeyError exception if we don't account for that possibility. So this exercise shows you how to use **in** to test to see if the key exists in the dict before trying to read the value for the key.

Ex. 5.6 Start with an empty dict. Opening and looping through the file revenue.txt, load the name of the store as the key and the decimal point value as the value in the dict. Print the dict.

Suggested Solution:

```
revenue = '../python_data/revenue.txt'

rdict = {}                    # initialize an empty dict

fh = open(revenue)            # open a file, return a filehandle

for line in fh:               # for each string line in file
    items = line.split(',')   # split str line on comma -> list of strs
    name = items[0]           # 1st item of split line:  company name
    revenue = items[2]        # 3rd item of split line:  revenue val
    rdict[name] = revenue     # set key/val pair in dict: name and revenue

print(rdict)
```

Stepping up to a data source, we're able to loop through each line and split it as we have been in recent exercises, and then add a key/value pair to the dict using two of the fields in from the split line: the name of the company (the first field from the line) and the revenue figure (the 3rd field). This dict is simply matching the unique value from the data source (the company) to the value we're interested in seeing for each company. We can then presumably look up the value for a given company, find which company has the largest or smallest revenue, etc.

Ex. 5.7 Fix the above script by removing the newline from each value (hint: remove it when it is still part of the line by calling rstrip() on the line, before splitting).

Suggested Solution:

```
revenue = '../python_data/revenue.txt'

rdict = {}                    # initialize an empty dict

fh = open(revenue)            # open a file, return a filehandle

for line in fh:               # for each string line in file
    line = line.rstrip()      # a new str line without newline at end
    items = line.split(',')   # split str line on comma -> list of strs
    name = items[0]           # 1st item of split line:  company name
    revenue = items[2]        # 3rd item of split line:  revenue figure
    rdict[name] = revenue     # pair company name key with 'revenue' value

print(rdict)
```

The same program as previously, except we are stripping the newline from the line before splitting. Truth be told, this step is pretty much moot since we aren't making use of the end of the line in our dict key or value. But it is a good practice because we certainly wouldn't ever want the newline if we are parsing the data.

Ex. 5.8 Given the following string:

```
personal_info = "595-33-9193:68:Columbus, OH"
```

Split the data and in a single statement insert it into a string so that it appears as shown in the output.

Suggested Solution:

```
personal_info = "595-33-9193:68:Columbus, OH"

els = personal_info.split(':')   # split str on colon -> list of strs

mystr = "SS#:  {}\nAge:  {}\nResidence:  {}".format(els[0],   # insert
                                                   els[1],   # each
                                                   els[2])   # element
print(mystr)
```

A simple exercise in splitting a string and making use of individual elements in a formatted string using format(). It's recommended to use format() over string concatenation because it's much clearer to read.

Ex. 5.9 Starting with this list of non-unique values:

```
cities = ['Boston', 'Chicago', 'New York', 'Boston', 'Chicago', 'Boston']
```

Count the occurrence of each value in the list by compiling a count of cities in the dict, by storing the city as a key in the dict and an integer count of the number of times that city occurred. However, don't attempt to loop through the list more than once. Hint: as your loops encounters each element in the list, use the **in** membership test to see if the value is a key in the dict. If it isn't, add it as a key with a value of 0. Then, still in the block, and whether or not it is in the dict, add one to the value for that key, and assign the new +1 value back to the dict for that key.

Suggested Solution:

```
cities = ['Boston', 'Chicago', 'New York', 'Boston', 'Chicago', 'Boston']
city_count = {}                             # empty 'counting' dict

for city in cities:                         # each string in list
    if city not in city_count:              # if str not a key in dict
       city_count[city] = 0                 # set str key and int 0 val
    city_count[city] = city_count[city] + 1 # add one to int value for
                                            # this str
print(city_count)
```

This exercise demonstrates the counting dictionary reading data from a file.

Ex. 5.10 Start with an empty dictionary. Opening and reading student_db.txt (make sure to account for the header line by slicing readlines()), build a dictionary of states and a count for each state. Print the dictionary.

Suggested Solution:

```
students = '../python_data/student_db.txt'

state_count = {}                # initialize an empty dict

fh = open(students)             # open a file, return a filehandle
lines = fh.readlines()          # read entire file as a list of str lines
wanted_lines = lines[1:]        # slice lines from 2nd item to end:
                                # effectively, slicing out the header

for line in wanted_lines:       # for each string line in sliced list
    items = line.split(':')     # split str line on colon -> list of strs
    state = items[3]            # 4th item of split line:  the state
    if state not in state_count: # if state key is missing from dict
        state_count[state] = 0  # set state key in dict paired with 0

    state_count[state] = state_count[state] + 1    # add 1 to value
                                                  # for this state key

print(state_count)
```

This exercise demonstrates the counting dictionary reading data from a file.

Ex. 5.11 Now reading from revenue.txt, sum up the values associated with each state.

Suggested Solution:

```
filename = '../python_data/revenue.txt'

sumdict = {}                    # initialize an empty dict

fh = open(filename)             # open a file, return a filehandle

for line in fh:                 # for each string line in file

    line = line.rstrip()        # new string line with newline removed
    items = line.split(',')     # split str line on comma -> list of strs
    state = items[1]            # 2nd item of split line:  the state
    revenue = items[2]          # 3rd item of split line:  the revenue val

    if state not in sumdict:    # if state key is missing from dict
        sumdict[state] = 0      # set state key in dict paired with 0
    sumdict[state] = sumdict[state] + float(revenue)   # add revenue value
                                                       # as float to
                                                       # to current value
                                                       # for this state

print(sumdict)
```

This solution is just like a counting dictionary, except adding float values to the value associated with the id, so it sums instead of counting.

Ex. 5.12 Starting with the following dictionary:

```
mydict = {'c': 0.3, 'b': 7, 'a': 5}
```

Sort this dictionary by letter.

Suggested Solution:

```
mydict = {'c': 0.3, 'b': 7, 'a': 5}    # init a dict

sorted_keys = sorted(mydict)      # dict passed to sorted() returns a list

for key in sorted_keys:           # for each string key in list
    print("{} => {}".format(key, mydict[key]))  # print key, val from dict
```

This is a simple sorting of a dict by keys. When you put a dict into a sorted() function, it handles the keys. So the keys are sorted alphabetically. Sorted returns a list of sorted keys, then we can loop through that list of keys and use each key to get the value in the dict (note that we're not looping through the dict anymore, but a sorted list of its keys).

Ex. 5.13 Modifying the previous solution, sort the dictionary by value.

```
mydict = {'c': 0.3, 'b': 7, 'a': 5}           # init a dict

sorted_keys = sorted(mydict, key=mydict.get)   # key= arg will sort dict
                                               # by value, sorted()
                                               # returns a list of keys

for key in sorted_keys:                        # for each str key in list
    print("{} => {}".format(key, mydict[key]))  # print key, val from dict
```

The simple technique of using key=dict.get (where 'dict' is the dictionary you want to sort()) will sort a dict by value.

Ex. 5.14 Modifying the previous solution, reverse the sort using the reverse=True argument to sorted().

```
mydict = {'c': 0.3, 'b': 7, 'a': 5}           # init a dict

sorted_keys = sorted(mydict, key=mydict.get, reverse=True)

                                    # sorted() returns a list of keys
                                    # key= arg will sort dict by value,
                                    # reverse= arg reverses the sort order

for key in sorted_keys:                          # for each key in
                                                 # list of keys
    print("{} => {}".format(key, mydict[key]))    # print key,val from dict
```

The additional reverse=True is all that is needed to reverse a sort.

Ex. 5.15 Modifying the previous solution, allow the user to enter an argument indicating which direction the sort should go (the user can enter "ascending" or "descending").

Suggested Solution:

```
mydict = {'c': 0.3, 'b': 7, 'a': 5}             # initialize a dict

uinput = input('please enter a direction: ')  # str input from keyboard

if uinput == 'ascending':      # set value of rev to boolean True or False
    rev = False                # depending on value taken from user input
elif uinput == 'descending':
    rev = True

else:                                         # handle error in input --
    exit('enter "ascending" or "descending"') # if input not True or False

sorted_keys = sorted(mydict, key=mydict.get, reverse=rev)

                                    # sorted() returns a list of keys
                                    # key= arg will sort dict by value,
                                    # reverse= arg reverses the sort order

for key in sorted_keys:             # for each key in list of str keys
    print("{} => {}".format(key, mydict[key]))
```

This attempts to show that the True and False in reverse=True or reverse=False are simply object values like any other, and so one can substitute a variable for True or False in that expression rather than having to enter it literally.

Ex. 5.16 Extending the summing dictionary (or the counting dictionary) taken from revenue.txt, once the loop is complete, sort the dictionary using a standard sort (which will sort the dict by state names).

```
filename = '../python_data/revenue.txt'

sumdict = {}                    # initialize an empty dict

fh = open(filename)             # open a file, return a filehandle

for line in fh:                 # for each string line in file

    line = line.rstrip()        # new string line with newline removed
    items = line.split(',')     # split str line on comma -> list of strs
    state = items[1]            # 2nd item of split line:  the state
    revenue = items[2]          # 3rd item of split line:  the revenue
    if state not in sumdict:    # if state key is missing from dict
        sumdict[state] = 0      # set state key in dict paired with 0
    sumdict[state] = sumdict[state] + float(revenue)   # add revenue value
                                                       # as float to
                                                       # to current value
                                                       # for this state


sorted_keys = sorted(sumdict)   # pass dict to sorted(), returns sorted
                                # list of str keys (states)

for key in sorted_keys:         # for each key in list of states

    print("{} => {}".format(key, sumdict[key]))  # print key, val from dict
```

We're just doing a standard sort as described earlier... any dict placed into sorted() will sort its keys...

Ex. 5.17 Now sort the summing dictionary by value.

```
filename = '../python_data/revenue.txt'

sumdict = {}                    # initialize an empty dict

fh = open(filename)             # open a file, return a filehandle

for line in fh:                 # for each string line in file
    line = line.rstrip()        # new string line with newline removed
    items = line.split(',')     # split str line on comma -> list of strs
    state = items[1]            # 2nd item of split line:  the state
    revenue = items[2]          # 3rd item of split line:  the revenue
    if state not in sumdict:    # if state key is missing from dict
        sumdict[state] = 0      # set state key in dict paired with 0
    sumdict[state] = sumdict[state] + float(revenue)   # add revenue value
                                                       # as float to
                                                       # to current value
                                                       # for this state

sorted_keys = sorted(sumdict, key=sumdict.get)  # sort dict keys by value

for key in sorted_keys:                         # for each key in list
    print("{} => {}".format(key, sumdict[key]))  # print key,val from dict
```

...and now applying the dict.get() method to sort the keys by value.