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

Introduction to Python
Exercise Solutions, Session 6

Ex. 6.1 Given the following code:

```
numlist = [1, 13, 23, 3, 9, 16]
```

Sort the list in reverse numeric order and print it.

Suggested Solution:

```
numlist = [1, 13, 23, 3, 9, 16]

sortlist = sorted(numlist, reverse=True)

print(sortlist)
```

The reverse=True parameter argument to sorted() simply reverses the sort order, last to first. True can be placed literally or it can even be a variable of the value True.

Ex. 6.2 Given the following code:

```
charlist = ['a', 'z', 'b', 'c', 'd', 'f']
```

Sort the list in alphabetic order and print it.

Suggested Solution:

```
charlist = ['a', 'z', 'b', 'c', 'd', 'f']

sortlist = sorted(charlist)

print(sortlist)
```

This exercise is simply demonstrating that when asked to sort strings, sorted() sorts them in an an alphabetic order (as long as all strings are the same case).

Ex. 6.3 Given the following code:

```
charList = ['F', 'e', 'c', 'a', 'B', 'D']
```

Sort the list using standard sort and print it.

Suggested Solution:

```
charList = ['F', 'e', 'c', 'a', 'B', 'D']

sortlist = sorted(charList)

print(sortlist)
```

Demonstrates that when strings are a mix of uppercase and lowercase, sorted() defaults to "asciibetic", which refers to each character's position on the ascii character table (google "ascii table" and choose "image" to see examples). As you can see, uppercase letters come before lowercase letters in this table.

Ex. 6.4 Given the following code:

```
charList = ['F', 'e', 'c', 'a', 'B', 'D']
```

Sort the list in alphabetical order and print it.

Suggested Solution:

```
charList = ['F', 'e', 'c', 'a', 'B', 'D']

sortlist = sorted(charList, key=str.lower)

print(sortlist)
```

The str.lower method, when passed to sorted(), causes sorted() to lowercase each string before sorting (more accurately, to sort each string by its lowercase value), so that case no longer matters when sorting strings.

Ex. 6.5 Given your understanding that the key= argument to sorted() will in a sense process each element through whatever function we pass to it, sort these strings by their length, and print the sorted list.

Suggested Solution:

```
mystrs = ['I', 'was', 'hanging', 'on', 'a', 'rock']

sorted_list = sorted(mystrs, key=len)

print(sorted_list)
```

As you know, len() returns the integer length (# of characters) of a string. Therefore by passing the len function to sorted(), we are asking sorted() to pass each string through len() and sort that string by the returned value -- thus all strings are sorted by their respective lengths.

Ex. 6.6 Given the following numbers, which were retrieved from a file as strings:

```
mynums = ['5', '101', '10', '1', '3']
```

Sort these strings numerically without creating a new list.

Suggested Solution:

```
mynums = ['5', '101', '10', '1', '3']

sorted_list = sorted(mynums, key=int)

print(sorted_list)
```

To solve this, you would want to ask what function would take an element and return its integer value. Of course int() does this, so by passing int to sorted() we are asking sorted() to pass each string to int() and then sort that string by the return value of int(), or its integer value.

Ex. 6.7 Experimental exercise. See the discussion in the exercise description for details.

Ex. 6.8 Given the following code:

```
line_list = [
   'the value on this line is 3',
   'the value on this line is 1',
   'the value on this line is 4',
   'the value on this line is 2',
]
```

Sort line_list by the number at the end of each line. Loop through and print the sorted list.

Suggested Solution:

```
line_list = [
   'the value on this line is 3',
   'the value on this line is 1',
   'the value on this line is 4',
   'the value on this line is 2',
]

def by_end_num(line):
    words = line.split()
    return words[-1]

for line in sorted(line_list, key=by_end_num):
    print(line)
```

Again you should begin by identifying first what is being sorted (in this case, each string line), and the value by which one of these elements should be sorted (in this case, the integer value of the number at the end of the string). So your function's job is to expect a string line as argument, to split off the number at the end, to convert that number from a string to an int, and then to return that int.

Don't forget that a sort function's job is to process all elements, but it does so only one element at a time -- sorted() will feed each element to the function and keep track of the return value so that it can sort an element by your function's return value.

Ex. 6.9 Sort the lines of the file pyku.txt by the number of words on each line.

Suggested Solution:

```
pyku = '../python_data/pyku.txt'

def by_num_words(line):
    words = line.split()
    return len(words)

fh = open(pyku)
lines = fh.readlines()
for line in sorted(lines, key=by_num_words):
    line = line.rstrip()
    print(line)
```

Again, the steps are: 1) identify the elements that will be sorted a list of strings -- in this case the lines from the file; 2) decide what value the line should be sorted -- in this case the number of words in the line, which can be obtained with len(line.split()); 3) inside the function, take the single line as argument, split the line and take the integer len() of the resulting list, and return that integer.

Ex. 6.10 Sort the lines of revenue.txt by the numeric value in the last field by passing the readlines() of the file to sorted() and using a custom sort sub similar to an earlier exercise.

Suggested Solution:

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

def by_last_float(line):
    words = line.split(',')
    return float(words[-1])

fh = open(revenue)
lines = fh.readlines()
for line in sorted(lines, key=by_last_float):
    line = line.rstrip()
    print(line)
```

This brings us closest to what we will do in the homework. Again, if an item to be sorted is a line from the file, and the value by which it should be sorted is the integer value of the last field, then our sort function needs only to split the line into items, convert the last item to a float, and return the float.

Ex. 6.11 Loop through and print each row (i.e., each list) as a whole.

Suggested Solution:

```
mylist = [
  [ 'a', 'b', 'c', 'd' ],
  [ 1, 2, 3, 4 ],
  [ 'alpha', 'beta', 'gamma', 'delta' ],
  [ 'Torchy', 'Thing', 'Girl', 'Fantastic' ]
]

for this_list in mylist:
    print(this_list)
```

This exercise encourages us to work with a multidimensional structure in a very basic way. A list of lists is just that - if we loop through the items in mylist, then each item in mylist will be a list. Just printing each item gives us a print of each list as a whole (i.e., without looping through the items in each of the "inner" lists.

Ex. 6.12 Loop through mylist and print only the 2nd element of each list.

Suggested Solution:

```
mylist = [
  [ 'a', 'b', 'c', 'd' ],
  [ 1, 2, 3, 4 ],
  [ 'alpha', 'beta', 'gamma', 'delta' ],
  [ 'Torchy', 'Thing', 'Girl', 'Fantastic' ]
]

for this_list in mylist:
    print(this_list[1])
```

Taking it a step further, we see how access to each "inner" list means that we can access any of the elements of the "inner" list. So if we think of a list of lists as a list of rows, we can do things like print a single column, just by printing the same element of each list.

Ex. 6.13 Print out the word 'beta' from mylist in one simple statement (do not use a loop).

Suggested Solution:

```
mylist = [
  [ 'a', 'b', 'c', 'd' ],
  [ 1, 2, 3, 4 ],
  [ 'alpha', 'beta', 'gamma', 'delta' ],
  [ 'Torchy', 'Thing', 'Girl', 'Fantastic' ]
]

print(mylist[2][1])
```

Since each list is addressible by subscript, we can also pinpoint a single element in the structure by targeting the index "address" of each element.

The way to think about this is first to identify the location of the "inner" structure where the targeted element is (in this case, the 3rd "inner" list). You could even print this "inner" structure as a whole, i.e. print mylist[2]. This shows you that the list with the element you want is indeed at that location. You can then think of that "inner" list as accessible through mylist[2] and work with it in the same way you would a named structure (i.e., assigned to a name like mylist). It is then a simple matter to access the targeted element within the "inner" structure by again using a subscript -- in this case, "beta" is in the 3rd element (index 2) within the "inner" structure. You can treat mylist[2] as the list itself (because we access it through an index rather than through a name) and simply subscript this inner list in the same way you would a named structure -- mylist[2][1].

Ex. 6.14 Loop through the list of dicts, printing each one on a separate line.

Suggested Solution:

```
lod = [
   {
      'name': 'Apex Pharma',
      'city': 'Louisville',
      'state': 'KY',
   },
   {
      'name': 'Beta IT',
      'city': 'New York',
      'state': 'NY',
   },
   {
      'name': 'Gamma Husbandry',
      'city': 'Lancaster',
      'state': 'PA',
   },
]

for this_dict in lod:
    print(this_dict)
```

Same as an earlier exercise: if the list contains dicts, you can print each dict by looping through the list and printing each dict. This will lead us into accessing individual elements, etc.

Note that each dict in the supplied list has the same keys. This list of dicts is meant to represent rows in a table -- the key names indicate the column names from the table.

Ex. 6.15 Loop through the list of dicts, printing just the company names from each dict.

Suggested Solution:

```
lod = [
   {
      'name': 'Apex Pharma',
      'city': 'Louisville',
      'state': 'KY',
   },
   {
      'name': 'Beta IT',
      'city': 'New York',
      'state': 'NY',
   },
   {
      'name': 'Gamma Husbandry',
      'city': 'Lancaster',
      'state': 'PA',
   },
]

for this_dict in lod:
    print(this_dict['name'])
```

This exercise is analogous to printing a particular element in each of a list of lists. It can also be considered as the printing of each value in a column of the table from which this structure was derived. Here we are printing the 'name' column values.

Ex. 6.16 Loop through the list of dicts, printing the info in a formatted form. (I've added an extra blank print statement to separate records.)

Suggested Solution:

```
lod = [
   {
      'name': 'Apex Pharma',
      'city': 'Louisville',
      'state': 'KY',
   },
   {
      'name': 'Beta IT',
      'city': 'New York',
      'state': 'NY',
   },
   {
      'name': 'Gamma Husbandry',
      'city': 'Lancaster',
      'state': 'PA',
   },
]

for this_dict in lod:
    print(this_dict['name'])
    print("{}, {}".format(this_dict['city'], this_dict['state']))
    print()
```

Now we're putting the values in each dict to use -- as you loop through each item in the list, you have access to each "row" of data (that originally came from a table) in a very convenient form - to get the 'name' for that row, use the 'name' key, for 'city' for that row use the 'city' key, etc. We are only printing the values in a different form, but we can of course do other things with the data -- sum up a column, create a set of cities, etc.

Ex. 6.17 Without looping, print the info of the last company only:

Suggested Solution:

```
lod = [
   {
      'name': 'Apex Pharma',
      'city': 'Louisville',
      'state': 'KY',
   },
   {
      'name': 'Beta IT',
      'city': 'New York',
      'state': 'NY',
   },
   {
      'name': 'Gamma Husbandry',
      'city': 'Lancaster',
      'state': 'PA',
   },
]

print(lod[2]['name'])
print("{}, {}".format(lod[2]['city'], lod[2]['state']))
```

Now we'll do a targeted access with a dict, same as we did with the list of lists -- to use the subscripts to access particular elements within the data. As with the lists of lists, in order to target single elements in an inner structure you must first identify the structure in which they reside, and the inner structure's "address" within the outer structure. In this case, the "last company" means the last dict in the list, or the 3rd element in the list -- so to access this dict we use lod[2]. We can then subscript the dict the way we would with a named dict, for example lod[2]['city'], etc.

Ex. 6.18 Loop through the pages for whateva.com

Suggested Solution:

```
sites_pages = {
   'example.com':   [
                      'index.html',
                      'about_us.html',
                      'contact_us.html'
                    ],
   'something.com': [
                      'index2.html',
                      'prizes.html',
                      'puppies.html',
                    ],
   'whateva.com':   [
                      'main.html',
                      'getting.html',
                      'setting.html',
                    ],
}

for page in sites_pages['whateva.com']:
    print(page)
```

Looping through an "inner" list -- this simply reinforces the idea that an "inner" or "anonymous" structure can be accessed through the "outer" structure. Since the key for 'whateva.com' is associated with a list, you can access this list through sites_pages['whateva.com'], and thus anything that can be done with a named list can be done through the outer dict at this key. Notice the syntax for the solution is no different from an ordinary list looping, except that instead of the list name we use the location of the list in the outer dict -- site_pages['whateva.com'].

Ex. 6.19 In one statement and without looping, print just the name puppies.html

Suggested Solution:

```
sites_pages = {
   'example.com':   [
                      'index.html',
                      'about_us.html',
                      'contact_us.html'
                    ],
   'something.com': [
                      'index2.html',
                      'prizes.html',
                      'puppies.html',
                    ],
   'whateva.com':   [
                      'main.html',
                      'getting.html',
                      'setting.html',
                    ],
}

print(sites_pages['something.com'][2])
```

Again, this is "inner" element access as described in the earlier exercise.

Ex. 6.20 Loop through the entire sites_pages dict of lists, and print each site and page. Use spaces and empty print statements to enhance formatting.

Suggested Solution:

```
sites_pages = {
   'example.com':   [
                      'index.html',
                      'about_us.html',
                      'contact_us.html'
                    ],
   'something.com': [
                      'index2.html',
                      'prizes.html',
                      'puppies.html',
                    ],
   'whateva.com':   [
                      'main.html',
                      'getting.html',
                      'setting.html',
                    ],
}

for sitename in sites_pages:
    print(sitename)
    for page in sites_pages[sitename]:
        print('  ' + page)
    print()
```

Finally we use a nested for loop (i.e., a for loop inside a for loop) to get through an entire structure (as you will do in the homework). You can begin a process like this by simply looping through the outer structure (as we did at the start of this section of exercises). This is a dict, so we can simply print the key and value in the dict, which will render for us a string key and a list value.

Once this is established and understood, and we can see that each "inner" list is accessed through sites_pages[sitename] (where sitename is the current key in ths looping), we can then loop through the inner list by simply looping through site_pages[sitename]. However, we must remember that we are addressing each key/value pair one at a time, so the loop through the "inner" list must be done inside the loop of the "outer" dict's keys.

Send me questions!