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

Introduction to Python
Exercises, 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.

Expected Output:

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

Ex. 6.2 Given the following code:

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

Sort the list in alphabetic order and print it.

Expected Output:

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

Ex. 6.3 Given the following code:

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

Sort the list using standard sort and print it (note the output below).

Expected Output:

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

Ex. 6.4 Given the following code:

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

Sort the list in alphabetical order and print it. (Hint: this list will not sort alphabetically by default. We need a key= function that can modify each element in the list, and make them all the same case. The string upper() method or lower() method can do the job. To refer to this method, you can say str.upper or str.lower. However, make sure not to actually call the method (which is done with the parentheses). Instead, you simply refer to the method, i.e., mention the method without using the parentheses.)

If you see this message:

```
TypeError: descriptor 'upper' of 'str' object needs an argument
```

it means that you added parentheses and attempted to call the str.upper or str.lower method. Keep in mind that we don't call this method -- we give it to the sorted() function to call. We're doing this because sorted() will use whatever function we wish, to modify each element for the purposes of sorting. If we give it str.upper it will sort 'a' as 'A' and 'B' as 'B' and 'c' as 'C' -- this should indicate to you how we are able to use it for alphabetic sorting with mixed-case strings.

Expected Output:

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

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:

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

Expected Output:

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

If you see this message:

```
TypeError: len() takes exactly one argument (0 given)
```

it means that you added parentheses and thus attempted to call the len function. Keep in mind that we don't call this method -- we give it to the sorted() function to call. We're doing this because sorted() will use whatever function we wish, to modify each element for the purposes of sorting. If we give it len it will sort 'I' as 1, 'was' as 3, 'hanging' as 7, etc -- perfect for our purposes.

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. (Hint: based on your understanding that the key= argument to sorted() will in a sense process each element through whatever function we pass to it, use a function that converts numeric strings to integers.

If you see a message similar to the one mentioned in the previous exercise, it means you've done something similar to the issue mentioned there. Remember that key= references a function, it does not call it (thus we will not use parentheses).

Expected Output:

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

Ex. 6.7 Experimental exercise. Please run the following code:

```
def my_element_modifier(arg):
    lower_arg = arg.lower()
    print('sorting element "{}" by value "{}"'.format(arg, lower_arg))
    return lower_arg

sorted_list = sorted(['e', 'c', 'D', 'B', 'a'], key=my_element_modifier)
print(sorted_list)
```

Note the output:

```
sorting element "e" by value "e"
sorting element "c" by value "c"
sorting element "D" by value "d"
sorting element "B" by value "b"
sorting element "a" by value "a"

['a', 'B', 'c', 'D', 'e']
```

This exercise demonstrates that my_element_modifier() was called once for every element, in this case 5 times. arg is assigned each element in turn. And the value returned from my_element_modifier() is the value by which each element is sorted -- so 'a' is sorted by the value 'a', 'B' is sorted by the value 'b', 'c' by 'c', 'D' by 'd'. This facilitates the alphabetic sorting of these values. The values themselves don't change, but Python sorts according to the value returned from the function.

If this makes sense to you, please go back to the previous examples and link this understanding to the other functions and methods we've used before -- using int, len, str.upper, etc. They are all passing a function to be applied to each element, and Python is sorting by the value returned from the function or method.

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. (Hint: call sorted() on line_list, and make your key= value the name of a custom function that takes the line as an argument and returns the value of the number at the end of the line. Your custom function will simply take an arg (that will be a string, the line from the file), split the line into elements, and return the last element as an integer.

Expected Output:

```
the value on this line is 1
the value on this line is 2
the value on this line is 3
the value on this line is 4
```

Ex. 6.9 Sort the lines of the file pyku.txt by the number of words on each line. (Hint: write a custom sort function that takes a single line of text, splits the line into a list, and returns the length of the list. Pass the readlines() of the file to the sorted() function.) Also here I am using rstrip() to strip each line before printing.

Expected Output:

```
We're out of gouda.
Spam, spam, spam, spam, spam.
This parrot has ceased to be.
```

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. (I am also rstrip()ping each line before printing it.)

Expected Output:

```
Dothraki Fashions,NY,5.98
Hipster's,NY,11.98
Awful's,PA,23.95
Westfield,NJ,53.90
The Clothiers,NY,115.20
The Store,NJ,211.50
Haddad's,PA,239.50
```

For the next few exercises start with this structure:

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

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

Expected Output:

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

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

Expected Output:

```
b
2
beta
Thing
```

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

Expected Output:

```
beta
```

For the next few exercises, start with this structure:

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

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

Expected Output:

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

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

Expected Output:

```
Apex Pharma
Beta IT
Gamma Husbandry
```

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.):

Expected Output:

```
Apex Pharma
Louisville, KY

Beta IT
New York, NY

Gamma Husbandry
Lancaster, PA
```

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

Expected Output:

```
Gamma Husbandry
Lancaster, PA
```

For the next few exercises, start with this structure:

```
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',
                    ],
}
```

Ex. 6.18 Loop through the pages for whateva.com

Expected Output:

```
main.html
getting.html
setting.html
```

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

Expected Output:

```
puppies.html
```

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.

Expected Output:

```
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
```