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

Introduction to Python
Exercise Solutions, Session 3

Ex. 3.1 open the FF-abbreviated.txt file, reading it line-by-line. print each line

Suggested Solution:

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

fh = open(filename)     # open(filename) returns a file object

for line in fh:         # 'line' is a string, each line of the file
    print(line)          # 'line' is reassigned for each line in the file
```

The first step in our homework goal of compiling a sum: this is the standard way to read a file line-by-line.

(Note: the filepath used in open() above assumes that we are running our script in a directory that sits in the same parent directory as the python_data/ directory. If you are running your script in a different location you will need to change the path there.)

Ex. 3.2 building on the above program, set up a counter that is set to 0 before the loop begins, and counts 1 for each line in the file. print each line in the file, and at the end report the count.

Suggested Solution:

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

fh = open(filename)     # open(filename) returns a file object

count = 0               # int counter must be set before the loop

for line in fh:         # 'line' is a string, each line of the file
    count = count + 1   # executed once for each iteration of the loop
    print(line)          # we see each line in the file, one by one

print(count)             # counter must be printed only after loop ends
```

This a simple way to demonstrate the use of a "summary" variable (i.e., the line counter count). The variable must be initialized (set to 0) before the loop begins, because otherwise it would be re-initialized at every loop iteration. We modify (i.e., add one to) the summary variable in order to count the number of iterations. Then once the loop is complete, we can print the result. The summary variable shouldn't be printed until after the loop is complete, otherwise we will see it multiple times (i.e., once per iteration of the loop).

Ex. 3.3 building on the above program, print the value of the counter in front of each line, so that each line is printed with its line number. (To print the number alongside of the line, you can use string formatting, concatenation with str(), or a comma.)

Suggested Solution:

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

fh = open(filename)    # open(filename) returns a file object

count = 0              # int counter must be set before the loop

for line in fh:        # 'line' is a string, each line of the file

    count = count + 1  # executed once for each iteration of the loop
    print(count, line)

print(count)
```

This is just a simple addition, to allow you to visualize the progression of the for loop through the file as well as the progression of the "summary variable", the counter count. (Of course I have said that the "summary variable" would not normally be printed inside the loop; this is being done here for illustration purposes.)

Ex. 3.4 new program: open the FF-abbreviated file and print just the year from each line, so you see just a 4-digit year from each line. (hint: take a 4-digit slice of each line and instead of printing the line, print the slice)

Suggested Solution:

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

fh = open(filename)     # open(filename) returns a file object

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    print(year)
```

This exercise puts the "slice" syntax to work, and is leading us to performing this work in the first homework assignment. We need to isolate the year on each line in order to select only lines of a particular year, so we slice each line to obtain the year value for that line.

Ex. 3.5 building on the previous program, set a string variable to '1928'. comparing the string to the slice, print out only those lines where the strings are equivalent (i.e., the year for the line is '1928'). Keep in mind that the == operator works with numbers as well as strings

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'
        print(line)          # print the entire line
```

Adding another step necessary for the first homework, we hard-code a year value (instead of taking it from the user, which we will eventually do) and then compare it to the sliced value from the line. This 'if' gate will allow us to select only those lines that apply to the year 1928.

Ex. 3.6 building on the previous program, add the counter from one of the earlier programs and this time count just the lines that match the year '1928'. print the total count at the end. (Hint: make sure that the counter is incremented inside the if block, i.e., only if the year matches.)

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

count = 0               # int counter must be set before the loop

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'

        count = count + 1   # increment the counter
        print(line)          # print the entire line

print(count)             # counter must be printed only after loop ends
```

Now we add a count, which we will also need for the homework. Keep in mind that in the homework we are asking for an average value for a particular year, which requires that we get a sum of values and a count of values. This program performs the year selection and the counting parts.

Ex. 3.7 new program: open the FF-abbreviated file and split each line so the individual columns are separated into a list. print the list from each line. (Hint: the string split() method without any argument splits on whitespace.)

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

fh = open(filename)          # open(filename) returns a file object

for line in fh:              # 'line' is a string, each line of the file
    elements = line.split()  # split the line on whitespace->list of strs
    print(elements)           # print the entire list of strs
```

In this exercise we focus on the task of isolating an individual 'column' or 'field' value from each of the lines. As you know in a delimited file, this is done with split(), which divides a string into a list, using the delimiter to determine where to divide elements. In the case of split() with no argument, the string is split on 'whitespace', which means spaces, tabs or newlines -- when python reads a line from left to right and reaches spaces, it removes the spaces and stores the characters read up to there into a list element. split() then returns the list of strings from the line.

Ex. 3.8 building on the previous program, instead of printing the entire list, print just the 1st column (the year-month-day) of each line. (Hint: since the string split() method returns a list, you can easily print just one column from that list by using a subscript (i.e., index number inside square brackets after the list name).

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

fh = open(filename)          # open(filename) returns a file object

for line in fh:              # 'line' is a string, each line of the file
    elements = line.split()  # split the line on whitespace->list of strs
    print(elements[0])        # print just the first element (YYYYMMDD)
```

Continuing from previous, we now add the step of selecting one of the string elements returned from the split() -- in this case, the YYYYMMDD value, or 1st column, in each line.

Ex. 3.9 adjusting the previous program, print the the 2nd column instead of the 1st column.

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

fh = open(filename)          # open(filename) returns a file object

for line in fh:              # 'line' is a string, each line of the file
    elements = line.split()  # split the line on whitespace->list of strs
    print(elements[1])        # print the second element (MktRF float)
```

Same as previous, except now we're highlighting the second column or field. Of course for the homework we will be summing up these values for a given year.

Ex. 3.10 building on the previous program, now add the 'year selection' functionality from earlier and print only the 2nd column values whose lines match the year '1928'.

Note on efficiency: when adding in this functionality, you should make the line splitting and column selecting happen only if the year from the line is 1928. This means that the loop block will start with the slicing, then the 'if' test asking if the slice is equal to 1928, then inside that 'if', splitting the line, selecting the 2nd column, and printing the 2nd column value. The reason we want to follow this order is because we want the program to do as little work as possible: there's no point in splitting the line or selecting the value if the year doesn't match -- we'll be ignoring those lines anyway.

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'
        els = line.split()  # split the line on whitespace->list of strs
        print(els[1])        # print the second element (MktRF float)
```

Now we start to put previous steps together. This program prints all of those float values for the selected year (1928).

Ex. 3.11 building on the previous program, convert each of the 2nd column values to a float, and multiply that value * 2. Print the column value and then the doubled value on the same line (using a comma between them in a print statement is probably the easiest way to print a number and string together).

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'

        els = line.split()  # split the line on whitespace->list of strs
        mktrf = els[1]      # select 2nd element of list (Mktrf value)
        fel1 = float(mktrf) * 2  # convert the Mktrf str to a float
        print(mktrf, fel1)       # print original float value, doubled val
```

We're only doubling the float value in order to validate the fact that we've converted the float value from the line into a float type (it comes to us as a string, of course) for the purposes of summing them up (an intended step in the homework).

Ex. 3.12 building on the previous program, add in the counter, but increment the counter only if the year is 1928, so you're only counting the 1928 lines. print each count number, the float value, and the doubled float value on the same line.

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

count = 0               # int counter must be set before the loop

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'
        count = count + 1   # increment count by 1
        elements = line.split()   # split line on whitespace->list of strs
        fel1 = float(elements[1]) # convert 2nd element of list to float
        print(count, elements[1], fel1 * 2)
```

Still assembling prior steps -- we're now selecting the slice based on the slice from the line (i.e., if it is year is '1928'), incrementing a counter, splitting and selecting the float value from the line and converting the value to a float type.

Ex. 3.13 start a new program based on the logic of the previous one: create a sum of float values. before the loop begins, initialize a "floatsum" variable to 0. then looping through the data, if the year is 1928, select out the 2nd column (the 1st column of float values), convert it to a float, and add it to the floatsum variable. report the sum at the end.

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

sel_year = '1928'

fh = open(filename)     # open(filename) returns a file object

floatsum = 0            # sum value must be set before the loop

for line in fh:         # 'line' is a string, each line of the file
    year = line[0:4]    # slice first 4 chars of the line (i.e., the year)
    if year == sel_year:    # if the sliced year is equivalent to '1928'
        elements = line.split()  # split line on whitespace->list of strs
        fel1 = float(elements[1])   # convert 2nd element of list to float
        floatsum = floatsum + fel1  # add float value to sum value

print(floatsum)          # print sum value after loop is done
```

This brings us almost right up to the solution to the first homework: we're now summing up the selected float values. Please be well, and happy.

Ex. 3.14 Open the pyku.txt file and use the file read() method to read it into a string. Print the length of this string using len()

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

fh = open(filename)     # open(filename) returns a file object

text = fh.read()        # read() returns the entire file text as a string

print(len(text))         # len() of the file string shows the entire len
                        # of the file
```

All files can be expressed as strings because they are essentially sequences of characters. read() returns the file as a string, and len() can measure the length of a string. Thus we are able to measure the length of the file in terms of # of characters.

Ex. 3.15 Building on the previous program, count the number of times the word spam occurs in the file. (Hint: file read() the file into a string and use the str count() method.)

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

fh = open(filename)      # open(filename) returns a file object

text = fh.read()         # read() returns the entire file text as a string

print(text.count('spam')) # returns int showing # of times 'spam' appears
```

This is a side illustration not directly related to the "word count" solution: since we can easily store the entire file in a single string, we can use the string "inspector" methods to determine things about the string. In this case, we can use the count() method to count the number of occurrences of 'spam' (or any other string) in the file.

Ex. 3.16 Open the pyku.txt file and file read() it into a string. Use the string splitlines() to split the string into a list of lines. Print the type of the list returned from splitlines(), then print the first and last line from the file using list subscripts.

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

fh = open(filename)     # open(filename) returns a file object

text = fh.read()        # read() returns the entire file text as a string

lines = text.splitlines()  # splitlines() returns list of strs (lines)
print(type(lines))          # print the type of the list
print(lines[0])             # the first line (first element of list)
print(lines[-1])            # the last line (last element of list)
```

This exercise illustrates the use of splitlines(): to take a multiline string (like the one produced by read()) and split it into its constituent lines (i.e., this is largely the same as split('\n') on Unix machines). The rest of the exercise just illustrates that the return value is a list of strings.

Ex. 3.17 Open a file and use the file read() method to read it into a string. Use the string split() method to split the entire string into a list of words. Print the type of the list returned from split(), then print the first and last word from the file using list subscripts.

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

fh = open(filename)     # open(filename) returns a file object

text = fh.read()        # read() returns the entire file text as a string

words = text.split()    # split entire file on whitespace->list of words
print(type(words))       # print the type of the list
print(words[0])          # the first word in the file
print(words[-1])          # the last word in the file
```

This exercise is intended to work in the same way as the previous one, except by splitting the string into words rather than into lines. The rest of the exercise just illustrates that the return value is a list of strings.

Ex. 3.18 Starting with this list:

```
var = ['a', 'b', 'c', 'd']
```

print the length of the list. (Hint: do not use a loop and a counter. Use len())

```
var = ['a', 'b', 'c', 'd']   # initialize a list of str objects

print(len(var))               # print len of list (4)
```

Lists have a len()!