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

Introduction to Python
Exercises, Session 3

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

Expected Output:

```
19260701    0.09    0.22    0.30   0.009

19260702    0.44    0.35    0.08   0.009

19260706    0.17    0.26    0.37   0.009
...intermediate output omitted...
19280301    0.23    0.04    0.12   0.011

19280302    0.07    0.01    0.66   0.011

19280303    0.49    0.01    0.64   0.011
```

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.

Expected Output:

```
19260701    0.09    0.22    0.30   0.009

19260702    0.44    0.35    0.08   0.009

19260706    0.17    0.26    0.37   0.009

...intermediate output omitted...

19280301    0.23    0.04    0.12   0.011

19280302    0.07    0.01    0.66   0.011

19280303    0.49    0.01    0.64   0.011

26
```

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, as in

```
print(count, line)
```

Expected Output:

```
1 19260701    0.09    0.22    0.30   0.009

2 19260702    0.44    0.35    0.08   0.009

3 19260706    0.17    0.26    0.37   0.009

...intermediate output omitted...

24 19280301    0.23    0.04    0.12   0.011

25 19280302    0.07    0.01    0.66   0.011

26 19280303    0.49    0.01    0.64   0.011

26
```

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)

Expected Output:

```
1926
1926
1926
1926
1926
1926
1926
1926
1926
1927
1927
1927
1927
1927
1927
1927
1927
1928
1928
1928
1928
1928
1928
1928
1928
1928
```

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

Expected Output:

```
19280103    0.43    0.90    0.20   0.010

19280104    0.14    0.47    0.01   0.010

19280105    0.71    0.14    0.15   0.010

19280201    0.25    0.56    0.71   0.014

19280202    0.44    0.15    0.18   0.014

19280203    1.12    0.48    0.42   0.014

19280301    0.23    0.04    0.12   0.011

19280302    0.07    0.01    0.66   0.011

19280303    0.49    0.01    0.64   0.011
```

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.)

Expected Output:

```
19280103    0.43    0.90    0.20   0.010

19280104    0.14    0.47    0.01   0.010

19280105    0.71    0.14    0.15   0.010

19280201    0.25    0.56    0.71   0.014

19280202    0.44    0.15    0.18   0.014

19280203    1.12    0.48    0.42   0.014

19280301    0.23    0.04    0.12   0.011

19280302    0.07    0.01    0.66   0.011

19280303    0.49    0.01    0.64   0.011

9
```

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.)

Expected Output:

```
['19260701', '0.09', '0.22', '0.30', '0.009']
['19260702', '0.44', '0.35', '0.08', '0.009']
['19260706', '0.17', '0.26', '0.37', '0.009']
['19260802', '0.82', '0.21', '0.01', '0.010']
['19260803', '0.46', '0.39', '0.38', '0.010']
['19260804', '0.35', '0.15', '0.32', '0.010']
['19260901', '0.54', '0.41', '0.08', '0.010']
['19260902', '0.04', '0.06', '0.23', '0.010']
['19260903', '0.48', '0.34', '0.09', '0.010']
['19270103', '0.97', '0.21', '0.24', '0.010']
['19270104', '0.30', '0.15', '0.73', '0.010']
['19270201', '0.00', '0.56', '1.09', '0.012']
['19270202', '0.72', '0.23', '0.18', '0.012']
['19270203', '0.17', '0.22', '0.08', '0.012']
['19270301', '0.38', '0.07', '0.57', '0.011']
['19270302', '1.12', '0.10', '0.22', '0.011']
['19270303', '1.01', '0.11', '0.04', '0.011']
['19280103', '0.43', '0.90', '0.20', '0.010']
['19280104', '0.14', '0.47', '0.01', '0.010']
['19280105', '0.71', '0.14', '0.15', '0.010']
['19280201', '0.25', '0.56', '0.71', '0.014']
['19280202', '0.44', '0.15', '0.18', '0.014']
['19280203', '1.12', '0.48', '0.42', '0.014']
['19280301', '0.23', '0.04', '0.12', '0.011']
['19280302', '0.07', '0.01', '0.66', '0.011']
['19280303', '0.49', '0.01', '0.64', '0.011']
```

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).

Expected Output:

```
19260701
19260702
19260706
19260802
19260803
19260804
19260901
19260902
19260903
19270103
19270104
19270201
19270202
19270203
19270301
19270302
19270303
19280103
19280104
19280105
19280201
19280202
19280203
19280301
19280302
19280303
```

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

Expected Output:

```
0.09
0.44
0.17
0.82
0.46
0.35
0.54
0.04
0.48
0.97
0.30
0.00
0.72
0.17
0.38
1.12
1.01
0.43
0.14
0.71
0.25
0.44
1.12
0.23
0.07
0.49
```

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.

Expected Output:

```
0.43
0.14
0.71
0.25
0.44
1.12
0.23
0.07
0.49
```

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).

Expected Output:

```
0.43 0.86
0.14 0.28
0.71 1.42
0.25 0.5
0.44 0.88
1.12 2.24
0.23 0.46
0.07 0.14
0.49 0.98
```

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.

Expected Output:

```
1 0.43 0.86
2 0.14 0.28
3 0.71 1.42
4 0.25 0.5
5 0.44 0.88
6 1.12 2.24
7 0.23 0.46
8 0.07 0.14
9 0.49 0.98
```

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.

Expected Output:

```
3.88
```

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

Expected Output:

```
80
```

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.)

Expected Output:

```
4
```

Ex. 3.16 Open the pyku.txt file and use the file read() method to read it into a string. Use str 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.

Expected Output:

```
<class 'list'>
We're out of gouda.
Spam, spam, spam, spam, spam.
```

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.

Expected Output:

```
<class 'list'>
We're
spam.
```

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())

Expected Output:

```
4
```