```
# - ProblemSet3.py *- coding: utf-8 -*-
"""
Problem 3_1:
Write a function that reads a text file and counts the number of characters in
it.  Print both the text file and the number of characters. Do this so that the
printout isn't doubled space (use an end=""  argument in the print statement).
Also, skip a line before printing the count. Note that it is easy to get the
number of characters in each line using the len() function.
Here is my run for HumptyDumpty.txt.  Let me point out one thing that is not
visible here and is a bit technical.  At the end of each of the first three
lines there is a <newline> character.  These are invisible.  If you do the count
by eye, you are likely to come up short by these three characters, but they
are visible to len() and you should count them -- they are part of the 141
"letters" in the humptydumpty.txt file.  Counting them makes this an easier
function for you to write.

Your output should look just like this for the autograder:

problem3_1("humptydumpty.txt")
Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.

There are 141 letters in the file.

"""
#%%
def problem3_1(txtfilename):
    pass # replace this pass (a do-nothing) statement with your code

#%%
"""
Problem 3_2:
Below you see three objects which are of collection data type: a list, a tuple,
and a string. Although different in many ways, you can write a 'for' loop that
steps through a "collection" and it will work with all three. This problem
is started for you. Finish it by writing the loop that will step through the
collection and print out its items, one per line. Test it and make sure that
it works for all three of the following data objects.

Be sure that your code does not include the name of any one of these data
collections. That would stop it from being general enough to deal with a
different collection.  The grader will use different data.

There is a printout of my run after the problem starter.
"""
#%%
nlis = [23,64,23,46,12,24]          # list
atup = ("c","e","a","d","b")        # tuple
str1 = "Rumplestilskin"             # string

#%%
def problem3_2(collection):
    pass # replace this pass (a do-nothing) statement with your code
#%%
"""
My runs
problem3_2(nlis)
23
64
23
46
12
```

```
24

problem3_2(atup)
c
e
a
d
b

problem3_2(str1)
R
u
m
p
l
e
s
t
i
l
s
k
i
n

"""
"""
Problem 3_3:
Write a function that will convert a date from one format to another.
Specifically, 06/10/2016 should convert to June 17, 2016.  Actually, you
will input the 6, the 17, and the 2016 as separate integers (numbers) and
the function will assemble and print the date as June 17, 2016.  I suggest
that you create a tuple months = ("January", "February", "March", ...) to
store the names of the months.  Then it is easy to access the name February
as months[1] and so on.

Here is printout of my run.
problem3_3(6,17,2016)
June 17, 2016

*** Note: for simplicity use 6 and not 06 for numbers; otherwise, the
function will confuse Python and will have to be more complex to work. ***
*** Tip: In print statements, commas create a space.  So you may have difficulty
avoiding a space between the 17 above and the following comma.  I suggest
that you build the output as a single string containing the properly formatted
date and then print that.  You can convert any number to string by using str()
and tie the parts together using +. Duplicate the format of the example output
exactly. Everything you need to do this is covered in the lectures. ***
"""
#%%
def problem3_3(month, day, year):
    """ Takes date of form mm/dd/yyyy and writes it in form June 17, 2016
        Example3_3: problem3_3(6, 17, 2016) gives June 17, 2016 """
    pass # replace this pass (a do-nothing) statement with your code

#%%
"""
Problem 3_4:
Write a function that is complementary to the one in the previous problem that
will convert a date such as June 17, 2016 into the format 6/17/2016.  I
suggest that you use a dictionary to convert from the name of the month to the
number of the month.  For example months = {"January":1, "February":2, ...}.
Then it is easy to look up the month number as months["February"] and so on.
Note that the grader will assume that month names begin with capital letters.
*** Tip: In print statements, commas create a space.  So you may have difficulty
```

```
avoiding a space between the 7, 17, and 2016 below and the following comma.  I
suggest that you build the output as a single string containing the properly
formatted date and then print that.  You can convert any number to string by
using str() and tie the parts together using +. Duplicate the format of the
example output exactly. Everything you need to do this is covered in the
lectures. ***

Here is a printout of my run for June 17, 2016.

problem3_4("July",17, 2016)
7/17/2016

"""
#%%
def problem3_4(mon, day, year):
    """ Takes date such as July 17, 2016 and write it as 7/17/2016 """
    pass # replace this pass (a do-nothing) statement with your code

#%%
"""

Problem 3_5:
Write a function that will look up a phone number given a name.  Use this
dictionary of phone numbers in your program, so that the grader will know
what phone numbers are available.  In it's simplest form, the program will
crash if a name that isn't in its dictionary is asked for.

Here is a printout of one of my runs.

problem3_5("james")
(212) 567-8149

Below is the start of the program including the dictionary.
"""
#%%

def problem3_5(name):
    """ Looks up the phone number of the person whose name is name """

    phone_numbers = {"abbie":"(860) 123-4535", "beverly":"(901) 454-3241", \
                     "james": "(212) 567-8149", "thomas": "(795) 342-9145"}
    pass # replace this pass (a do-nothing) statement with your code

#%%
"""
Problem 3_6:
Write a program (not just a function, but a stand alone program or script) that
reads through a file and writes another file that gives the length of each line
in the first file.  If line is the line that you've read into your program, use
line.strip("\n") to strip away the invisible newline character at the end of
each line.  Otherwise, your count will be one higher than the autograder's.
Note that since this is a program running from the Command Window (or terminal
or cmd.exe), it won't be runnable as our usual functions are by entering
Shift-Enter.  You should use the File menu in Spyder to create you own file.
But, if you prefer, there is a starter file called problem3_6starter.py.

Here is a run of my solution program using the HumptyDumpty.txt file. The run
is followed by a printout of HumptyDumpty.txt and the written file
HumptyLength.txt. Note that your program does not print anything out.  It does
write a text file though.  To see these files we have to use type on a PC ( but
it would be cat for Mac or Linux).

C:>python problem3_6.py humptydumpty.txt humptylength.txt

C:>type humptydumpty.txt
```

```
Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall.
All the king's horses and all the king's men
Couldn't put Humpty together again.
C:>type humptylength.txt
28
31
44
35

Problem 3_7:
Write a function that would read a CSV file that looks like this, flowers.csv:

petunia,5.95
alyssum,3.95
begonia,5.95
sunflower,5.95
coelius,4.95

and look up the price of a flower and print out that price.  Remember to import
the necessary library.

Here is my run on the above CSV file:
problem3_7("flowers.csv","alyssum")
3.95

Solution starter:
"""
#%%
def problem3_7(csv_pricefile, flower):
    pass # replace this pass (a do-nothing) statement with your code

#%%
```