```
# -Exercises3.py *- coding: utf-8 -*-

"""
TUPLES
Tuple (like the list type this is a data collection type)
Note a pair of numbers 3,4 is referred to as a tuple. There are tuples of
various sizes: 3,4 and 3,4,5 and 3,4,5,6 are all tuples. They do not have to
be numbers. The following are also tuples: 'a,','b' and x, y, z and
"Morning", "Afternoon", "Evening". In Python we can set tuples equal to other
tuples as follows:
"""
#%%
x,y = 3,4
print (x,y)
#%%
a,b,c = 10,11,12
print(a,b,c)
print(b)
#%%
"""
The tuple 4,6,8 is written by Python with parentheses: (4,6,8)
"""
#%%
4,6,8
#%%
"""
The items in tuples can be accessed much like those in a list, but unlike a
list, a tuple cannot be changed. One cannot add to a tuple or remove an item
from a tuple. It is said to be immutable.

Let's define a tuple and look at how to access its items:
"""
#%%
tup = ('a','e','i','o','u')
#%%
""" The following accesses are just like in a list. """
print("tup is", tup)
print(tup[0])
print(tup[1])
print(tup[-1])
print(tup[-2])
print(tup[1:3])
print(tup[:3])
#%%
"""
DICTIONARIES
Dictionary data type (another collection datatype along with list and tuple):
d = {} creates an empty dictionary
d={key1:value1, key2:value2, key3:value3} - use key to find the value
d[key2] gives value2, etc.
Lists and strings are in order and you can use slicing to single out an item:
lis[3] or strg[3] give the item with index 3 in the list or string. This won't
work with dictionaries. So you use d[key] to get the value that goes with key.
Example:
"""
#%%
d = {"Johnny": "5 years old", "Sally": "7 years old", "Eva":"10 years old",
 "Peggy": "7 years old"}
#%%
"""
Try out the following:
d[0] gives an error
d['Sally'] gives '7 years old'
d prints out as
```

```
{'Eva': '10 years old',
 'Peggy': '7 years old',
 'Sally': '7 years old',
 'Johnny': '5 years old'}
d.items()         # this gives the items in the form of a list of tuples
d.keys()          # this gives the keys
d.values()        # this gives the values
Note that the order is not the order entered.
Note also that the keys must be unique but the values don't have to be

Accessing. All the following two loops do the same or a similar thing:
"""
#%%
for key,value in d.items():
    print(key, "--> ", value)
#%%
for item in d.items():
    print(item)
#%%
for item in d.items():
    print(item)
    print(item[0], "--> ", item[1])
#%%
for item in d.items():
    print(item[0], "--> ", item[1])
#%%
for key in d:
    print(key, "--> ", d[key])
#%%
for key in d.keys():
    print(key, "--> ", d[key])
#%%
"""
If you want to see only the values:
"""
#%%
for value in d.values():
    print(value)
#%%
"""
The following will add the key:value pair "Ted":"5 years old" to d or change
the value if 'Ted' is already a key in d:
"""
#%%
d['Ted'] = "5 years old"
#%%
d
#%%
"""
d["sally"] gives an error because "s" is not capitalized
d["Jim"] gives an error because "Jim" is not a key.
d["5 years old"] gives an error because "5 years old" is a value not a key
"""
"""
Exercise:
Consider the following dictionary of US News and World Reports list of best
affordable sportscars (ascars). Execute the line below so that you have the
dictionary in your IPython console. Then answer (retrieve) the following.
Press <esc> if you lock up on some step.
a) Type ascars to display the dictionary
b) Using the key retrieve the Nissan sportcar.
b) Using the key retrieve the Chevy sportscar.
c) Change the MINI Cooper car to a "Coupe". Display ascars to check that it
worked.
d) Write a small 2 line loop to display all the values and only the values
```

```
e) Write a small 2 line loop to display all the keys (and only the keys).

Here is the list to execute:
"""
#%%
ascars = {"Ford" : "Mustang","Mazda" : "Miata", "Scion" : "FR-S",
"Subaru" : "BRZ","Dodge" : "Challenger", "Nissan" : "370Z", "Chevy" : "Camaro",
"Hyundai" : "Genesis Coupe" , "MINI Cooper" : "Roadster"}

#%%
"""
Solutions
"""
#%%
"a"
#%%
ascars["Nissan"]
#%%
"b"
#%%
ascars["Chevy"]
#%%
"c"
#%%
ascars["MINI Cooper"]="Coupe"
#%%
"d"
#%%
for value in ascars.values():
    print(value)
#%%
"e"
#%%
for key in ascars.keys():
    print(key)
#%%
"""
Some summary facts:
Lists can be appended to and items can be addressed by item number.
Tuples are immutable (can't be changed). Item can be addressed by item number.
Dictionaries can be appended to. Items cannot be retrieved by item number,
  because the items have no inherent order. The values can be retrieved by
  using their keys.
"""
"""
Exercise:
Lists, tuples and dictionaries
Let's examine the differences.
First, lists and tuples are ordered, so this makes sense: lis[2], tup[3]
Dictionaries are not ordered, so dict[0] returns an error.
Dictionaries don't look items up by index number, but by key; you can't do
this with lists and tuples.
[] is an empty list; () is an empty tuple; and {} is an empty dictionary.
Tuples can't be modified, but lists and dictionaries can.
You can stride lists and tuples, but not dictionaries. Dictionaries have no
notion of order and hence cannot be sliced ([3:7] doesn't mean anything) and
they cannot be strided -- it might appear that you can, but the order is
unpredictable.
Try the following out and see what works:
"""
#%%
# Execute this cell to create these variables
namelist = ["George", "Sally", "Catherine", "James", "Peggy"]
x,y,z = "George","Sally","Catherine"
mytuple = x,y,z
```

```
agedict = {"George":"17","Sally":"19",
            "Catherine":"18"}
#%%
# Which work (you might copy each into IPython and see whether it works)?
namelist[1]   # yes or no
mytuple[1]    # yes or no
agedict[1]    # yes or no
#%%
# Which work?
namelist.append("Rod")   # yes or no
mytuple.append("Rod")    # yes or no
agedict.append("Rod")    # yes or no
#%%
# Which work?
namelist[1]="Rod"    # yes or no
mytuple[1] = "Rod"   # yes or no
agedict["Rod"]="23"  # yes or no

#%%
"""
OPERATING SYSTEM COMMANDS
Some useful operation system commands for the rest of this lesson:
Mac/unix/Linix
pwd        prints the current working directory
ls         lists the files in the current directory (options -l, -F, -a, -A)
Windows
cd ,       prints the current working directory
dir        lists the files in the current directory (one column)
dir/w      lists the files in the directory "wide" (multi-column)

In an IPython window you can execute many operating system commands by
putting an ! in front of them. On my Windows pc, the above Unix style commands
work without the ! in front.

These might be helpful when reading and writing files, because you either
have to give the full directory path or assume the file is in the
current working directory and use a relative path. '!cd ,' or '!pwd' will show
you what the current working directory is.

Another note: there is a button to the right of the Global Working Directory
(this is the path displayed in the upper right of the toolbar) to set the
displayed path as the current working directory.
"""

"""
READING AND WRITING TEXT FILES

Reading/writing files summary:
infile = open(filename)  # For reading. Also infile = open(filename,'r')
infile.close()
outfile = open(filename,"w")  "Open for writing
outfile.write("string to write")
outfile.close()
"""
#%%
def print_file(filename):
    """ Opens file and prints its contents line by line. """
    infile = open(filename)

    for line in infile:
        print(line, end="") # the file has "\n" at the end of each line already

    infile.close()
#%%
def copy_file(infilename, outfilename):
```

```
    """ Opens two files and copies one into the other line by line. """
    infile = open(infilename)
    outfile = open(outfilename,'w')

    for line in infile:
        outfile.write(line)

    infile.close()
    outfile.close()
#%%
"""
Exercise:
Write a function write_to_file(filename,myname,myage,major) that opens the
file <filename> and writes 3 lines in it using the data given. Here is a sample
of what could be in the file. Call the file 'namefile.txt', so that it is
identifiable as a text file:

My name is George
My age is 21
I am majoring in Physics
Tips:
1. write() can take only a string and, in fact, only one. So whatever you
write has to be put together into one string (using +, for example).
2. Add +"\n" on the end of each write to put a newline at the end of each
line. Otherwise, everything will be jammed together on one line.
3. Convert your age (a number) to a string -- use str() to do that. Then join
it to the other parts of the string before writing.
4. When running the function, use quotes around every argument except the
number.

Use notepad (on pc) or textedit (on Mac) to look at the file to confirm it. An
outline of what needs to be done is given as comments.
"""
#%%
def write_to_file(filename, myname, myage, major):
    # open file first
    outfile.write("My name is "+ myname + " \n")
    # write out the age and major in two lines
    # close the file


#%%
"""
RUNNING STANDALONE SCRIPTS OR PROGAMS.

So far we have been writing Python functions and running them. Now let's
write a Python program (actually Python calls them 'scripts').

To run from command line open a command window in one of two ways:
On windows, from the Start Menu, enter cmd.exe
  -- in this case use cd to change to the directory with print_file.py in it
or
From Spyder, go to menu Tools>Open command prompt
  -- in this case you should be sure that print_file.py is in the directory at
  -- the top of the screen (Global working directory) or do as in the
  -- cmd.exe case and use cd to get there.

Once you are in the correct directory, enter the following command at the
">" prompt: python printfile.py <nameoffile to print>

"""
"""
NOTE: SOME OF YOUR COMPUTERS MAY HAVE PYTHON 2.7 INSTALLED ON THEM.  THE PATHS
ARE SET TO RUN VERSION 2.7. WITHOUT RESETTING THE PATH, THOSE COMPUTERS MAY
ATTEMPT TO RUN PROGRAMS USING PYTHON 2.7 INSTEAD OF PYTHON 3.4. Macintoshes,
```

```
for example, come with Python 2.7 installed and will have this issue. The
following works from the Spyder Command prompt regardless, so we will use that.
The environment can be set to python 3, but I'm not going into that right now.
"""
#%%  # Note the following cell can NOT be executed by Ctrl-Enter. Copy to a file.
# - print_file.py *- coding: utf-8 -*-
""" Opens file and prints its contents line by line. """

import sys     # we need this library to deal with operating system

filename = sys.argv[1]

infile = open(filename)

for line in infile:
    print(line,end="") # the file has "\n" at the end of each line already

infile.close()
#%%
"""
Exercise:
Now take copy_file from above and convert it to a stand alone program called
copy_file.py.  Actually, I've provided you a starter file named copy_file.py.
Open it and modify it. Then save it and run it in a terminal or command
window.  Now modify it by adding the import statement above. Since you have two
filenames now, you will need to use sys.argv[1] and sys.argv[2] to
get them. Here is copy_file.py that you need to rework:
"""
#%%
# -copy_file.py *- coding: utf-8 -*-
"""
Exercise: Convert this function to a standalone program or script that
takes two file names from the command line and copies one to the other.

Steps:
1. Delete "Def" line. You don't need it.
2. Use Edit menu of Spyder to Unindent all the lines.
3. import the system library sys
4. sys.argv is a list of the filenames following the program name.
   sys.argv[0] is the program name, sys.argv[1] is first argument, etc.
   Get the infilename and outfilename from this list.
5. Save the program
6. Run the program from a terminal window (Mac) a cmd.exe window (PC) or
   a command prompt within Spyder (use Tools>Open command prompt)

Here is how running it should look:
>python copy_file.py humptydumpty.txt newhumpty.txt
"""

def copy_file(infilename, outfilename):
    """ Opens two files and copies one into the other line by line. """
    infile = open(infilename)
    outfile = open(outfilename,'w')

    for line in infile:
        outfile.write(line)

    infile.close()
    outfile.close()
#%%

"""
Solution is in copy_file_worked.py
"""
"""
```

```
lwc.py below is a program that counts the number of lines, words, and
characters in a text file.  Similar to linux wc command.
Copy this to a separate file named lwc.py or download ours.
"""
#%%
# - lwc.py *- coding: utf-8 -*-
import sys

filename = sys.argv[1]
# print("\n",filename,"\n")  # You can check that the filename is correct

text_file = open(filename)     # open the file for reading

# initialize line, word, and char counters to 0
linect = 0
wordct = 0
charct = 0

for line in text_file:           # step through each line in the text file
    linect = linect + 1
    for word in line.split():    # split into a list of words
        wordct = wordct + 1
    charct = charct + len(line)

text_file.close()

print(linect, wordct, charct )
#%%
"""
Reads through a text file and counts the number of different words.
Uses a dict (dictionary data type)
d = {key1:value1,  key2:value2,  key3:value3}
d[key2] gives value2, etc.
The key in this case is a word and the value is the number of times it occurs.

The plan is to read through a text file, split each line into its constituent
words, add each word to a dictionary, then add one to the number of times the
word occurs in the dictionary. Finally, we sort the dictionary and print it
out listing each word (key) and its count (value).
"""
#%%
def count_words(filename):
    """ Makes a list of the words in the file filename and the number of times
    each word appears. This program is modified from one by Mark Summerfield in
    his excellent book, "Programming in Python 3" """

    text_file = open(filename)     # open the file for reading

    # Set up an empty dictionary to start a standard design pattern loop
    words_dic = {}

    # This loop adds each word to the dictionary and updates its count. Change
    # all words to lower case so Horse and horse are seen as the same word.
    for line in text_file:         # step through each line in the text file
        for word in line.lower().split():   # split into a list of words
            word = word.strip("'?,.;!-/\"") # strip out the stuff we ignore
            if word not in words_dic:
                words_dic[word] = 0      # add word to words with 0 count
            words_dic[word] = words_dic[word] + 1    # add 1 to the count

    text_file.close()

    # Sorts the dictionary words into a list and then print them out
    print("List of words in the file with number of times each appears.")
    word_list = sorted(words_dic)
```

```
    for word in word_list:
        print(words_dic[word], word)

#%%
"""
Exercise:
Save this cell as a separate file called count_words.py (or download
count_words.py)  and run as instructed below. This is a standalone version of
the above function.
NOTE: THIS WILL WORK FROM SPYDER COMMAND PROMPT, BUT MAY NOT ON A MAC FROM
TERMINAL WINDOW.  SEE RUNNING STANDALONE PROGRAMS ABOVE.
"""
#%%
# -count_words.py *- coding: utf-8 -*-
"""
Reads through a text file and counts the number of appearances of each word.

To run from command line open a command window in one of two ways:
On windows, from the Start Menu, enter cmd.exe
  -- in this case use cd to change to the directory with count_words.py in it
or
From Spyder, go to menu Tools>Open Command prompt
  -- in this case you should be sure that count_words is in the directory at
  -- the top of the screen (Global working directory) or do as in the
  -- cmd.exe case and use cd to get there.

Once you are in the correct directory, enter the following command at the
">" prompt:

python count_words.py <filename>

where <filename> is the name of a file (in quotes if it contains spaces)

"""
# -count_words.py *- coding: utf-8 -*-
import sys

filename = sys.argv[1]
# print("\n",filename,"\n")  # You can check that the filename is correct

text_file = open(filename)     # open the file for reading

# Set up an empty dictionary to start a standard design pattern loop
words_dic = {}

# This loop adds each word to the dictionary and updates its count. Change
# all words to lower case so Horse and horse are seen as the same word.
for line in text_file:         # step through each line in the text file
    for word in line.lower().split():   # split into a list of words
        word = word.strip("'?,.;!-/\"") # strip out the stuff we ignore
        if word not in words_dic:
            words_dic[word] = 0      # add word to words with 0 count
        words_dic[word] = words_dic[word] + 1    # add 1 to the count

text_file.close()

# Sorts the dictionary words into a list and then print them out
print("List of words in the file with number of times each appears.")
word_list = sorted(words_dic)
for word in word_list:
    print(words_dic[word], word)

# Sorts the dictionary words into a list and then print them out
print("List of words in the file with number of times each appears.")
word_list = sorted(words_dic)
```

```
for word in word_list:
    print(words_dic[word], word)
#%%

"""
CSV FILES
"""
"""
We now turn to reading/writing CSV files, that is Comma Separated Value files.
Text files lack the structure that we need for certain applications.  CSV files
can be read or written by spreadsheet programs such as Excel. They may be the
most common way of transferring files from one application to another.
Summary of CSV file statements:

import csv

infile = open(filename)    # For reading. Also infile = open(filename,'r')
infile.close()             # An open file locks other applications out
rows = csv.reader(infile)            # Read row

f = open(filename, 'w', newline='')  # Open for writing
csv.writer(f).writerows(rowlist)     # Write all rows at once
csv.writer(f).writerow(row)          # Write one row
f.close()

"""
"""
Example of reading from a CSV file and writing each row as a list.
"""
#%%
import csv

def read_csv_file(filename):
    """Reads a CSV file and prints each row, which is a list. """
    f = open(filename)
    for row in csv.reader(f):
        print(row)
    f.close()
#%%
"""
Example of reading from a CSV file and appending to a list.
"""
#%%
import csv

def read_csv_file1(filename):
    """Reads a CSV file and print it as a list of rows."""
    f = open(filename)
    data = []
    for row in csv.reader(f):
        data.append(row)
    print(data)
    f.close()
#%%
"""
Exercise:
Rewrite read_csv_file(filename), call it read_csv_file2(filename), so that you
print each row without the list bracket. You will print each item in the row
separately instead of printing the whole row. This requires you to know
before-hand how many columns are in the csv file. In the case of booksread.csv,
there are 3 items in each row: How do you address each item in the row list
named row? They are row[?] and row[??] and row[???], where you fill in the
?, ??, and ??? values.
Here's what the output should look like:
```

```
read_csv_file2("booksread.csv")
Beckert, Sven Empire of Cotton history
Buckley, Carla The Deepest Secret mystery
Carcaterra, Lorenzo Chasers mystery
Catton, Bruce The Army of the Potomac: The Glory Road military
Cohen, Gabriel The Ninth Step mystery
Darwin, Charles Origin of Species science
Ho, Yong China: An Illustrated History history
James, Henry Daisy Miller novel
Larsson, Stieg The Girl who played with fire novel
Lewis, Michael Liar's Poker: rising through the wreckage on Wall Street economics
Messenger, Bill Elements of Jazz: From Cakewalks to Fusion music
Paulos, John Allen Innumeracy mathematics
Penzler, Otto, ed. Murder at the Racetrack  mystery
Pintoff, Stefanie Secret of the White Rose mystery
Post, Robert C. Democracy, Expertise, Academic Freedom law
Solzhenitsyn, Alexander One Day in the Life of Ivan Denisovich novel
Torrence, Bruce F. and Eve A. The Student's Introduction to Mathematica mathematics
Woods, Stewart Mounting Fears novel

"""
"""
Solution Starter:
"""
#%%
import csv

def read_csv_file2(filename):
    """Reads a CSV file and prints each row without list brackets. """
    f = open(filename)
    for row in csv.reader(f):
        pass # replace this line with your code
    f.close()
#%%
"""
End solution
"""
"""
Example of writing to a CSV file from a list looping and writing one
row at a time. newline='' keeps csv file from writing a newline at the end of
each line. This keeps the file from being double spaced.
"""
#%%
def write_csv(filename):
    import csv

    L = [['Date', 'Name', 'Notes'],
         ['2016/1/18', 'Martin Luther King Day', 'Federal Holiday'],
         ['2016/2/2','Groundhog Day', 'Observance'],
         ['2016/2/8','Chinese New Year', 'Observance'],
         ['2016/2/14','Valentine\'s Day', 'Obervance'],
         ['2016/5/8','Mother\'s Day', 'Observance'],
         ['2016/8/19','Statehood Day', 'Hawaii Holiday'],
         ['2016/10/28','Nevada Day', 'Nevada Holiday']]

    f = open(filename, 'w', newline='')
    for item in L:
        csv.writer(f).writerow(item)
    f.close()
#%%
"""
We can see that it has been written, by double-clicking it in File Explorer in
Windows (or in the Finder on a Mac) so that our spreadsheet program opens it.
We can also see it by entering !type <filename> in the IPython Console
(!cat <filename> on a Mac), where filename is the name we gave in the function
```

```
call above.
"""
"""
Exercise:
This is a modification of write_csv(filename). Instead of getting the data
from a list, we're going to input it from the keyboard. This new function
name_phone(csv_filename) will put a friend's name and his/her phone number
into a csv file. Here is a start. We are going to add features to the program
so that each step is manageable and we don't get confused.
1) Run this program and see that you can enter names and they'll print.
   To exit, you enter a blank name (i.e., just press return or enter).
   Note the loop runs forever, so we have to use 'break' to get out.
2) Add an input statement to collect a phone number and a line to print the
   phone number. Carefully place these so that the program runs nicely; that
   is, don't print anything until you have the phone number and don't get the
   phone number until you know the user isn't quiting.
3) Now add writing to a CSV file. After each step you can try running the
   function to make sure everything works. We're "growing" the program making
   sure that we have a working program each step of the way. You can check
   the work by double clicking on the created file and using Excel to peek into
   it. In the following be careful to put the right statements inside the loop
   and the others outside the loop.
   a) First add any necessary import statements.
   b) Add a line opening the csv file for writing and a line closing the file.
   c) As each line written to the CSV file must be a list, create an empty list
      like this: line = []. Note this should be inside the loop. Why?
   d) append the name to line; append the phone to line
   e) write the line to the CSV file
"""
"""
Solution starter:
"""
#%%
def name_phone(csv_filename):

    # open the csv file here

    while True:
        nextname = input("Enter a friend's name, press return to end: ")
        if nextname == "":
            break              # break jumps out of the loop
        print(nextname)

        # add lines here to build a row (that is, a list) and append these
        # two pieces of data to it.  Write to the csv file

    # don't forget to close the csv file

#%%
"""
Example run:
name_phone("myphones.csv")

Enter a friend's name, press return to end: Jerry Seinfeld

Enter your friend's phone: (212) 434-1234
Jerry Seinfeld
(212) 434-1234

Enter a friend's name, press return to end: Elaine Benes

Enter your friend's phone: (212) 123-6543
Elaine Benes
(212) 123-6543
```

```
Enter a friend's name, press return to end:

!type myphones.csv
Jerry Seinfeld,(212) 434-1234
Elaine Benes,(212) 123-6543

"""
 """
Updating a csv file

Let us read a CSV file containing a person's daily weights and compute the
average weight and write that into the csv file.
Here we copy the old file to a new one and add an additional line that
contains the average of all the weight values.
Note that the first row has a header so we skip it.
Note also that the values are read in as strings and we must convert to float
(i.e., read number) before using a number for arithmetic.
Note that we are writing a new line at the end that contains the average.

In developing this program, we could somehow open a file and not manage to
close it. In which case, we may have to get out of Spyder to unlock the file.
Using a new name for the new file (we're writing to it), can also fix the
problem.

Here is my run on a PC. On a Mac, the command "type" should be "cat":
update_csv("weights.csv","xweights.csv")

!type xweights.csv
Date,Weight
5/1/2016,142
5/2/2016,143
5/3/2016,140
5/4/2016,141
5/5/2016,142
5/6/2016,141
5/7/2016,143
Average,141.71428571428572

"""
#%%
def update_csv(old_name, new_name):
    import csv

    fin = open(old_name)
    fout = open(new_name,'w',newline = '')
    ct = 0
    tot_weight = 0.0
    for row in csv.reader(fin):
        if row[0]!="Date":
            ct = ct + 1
            tot_weight = tot_weight + float(row[1])
        csv.writer(fout).writerow(row)
    row = ["Average", tot_weight/ct]
    csv.writer(fout).writerow(row)
    fin.close()
    fout.close()
#%%
```