1. Suppose D = { 5 : 'hello', 'hi' : 5, (1,2) : {'un' : 'one, 'deux' : 'two'} }.

(a) list(D.keys()) = **[5, 'hi', (1,2) ]**

(b) list(D.values()) = **['hello',5,{'un':'one','deux':'two'}]**

(c) list(D.items()) = **[(5,'hello'), ('hi',5), ((1,2),{'un' : 'one, 'deux' : 'two'})]**

(d) What will be the result of executing the following:

>>> D = { 5 : 'hello', 'hi' : 5, (1,2) : {'un' : 'one, 'deux' : 'two'} }

>>> D\_reversed = { v:k for v in D.values() for k in D.keys() if D[k] == v }

**Exception: one of the values (the dictionary) is mutable**

2. (a) Write a function named price\_calculator() that prompts for and inputs the number of a certain item to be purchased; then prompts for and inputs the price per item; and returns **three values**: the number of items, the price per item and the total price. You may assume the user input is valid (no exceptions required).

**def price\_calculator():  
 numitems = int(input('Enter the number of items: '))   
 price = float(input(Enter the price per item: '))  
 return numitems, price, numitems\*price**

(b) *(String formatting)*

*Note: due to the number of students without the Python Pocket Reference, control of the number of digits past the decimal point for floats is not used.*

Write a code segment that calls the above function, then uses the output to print a message showing the details of the purchase. It should follow the format of the following example (user input underlined). The first two lines were produced by the call to price\_calculator; the last line is what your print statement should print.

Enter the number of items: 3   
Enter the price per item: 1.75   
For 3 items at $1.75 per item, the total price is $5.25

(i) using the C-style (%) formatting method

**count, price, total** = price\_calculator()  
print( **'For %s items at $%s per item, the total price is $%s'   
 %(count,price,total) )**

(ii) Using the str.format() method.

**count, price, total** = price\_calculator()

print(**'For {} items at ${} per item, the total price is ${}'.format(count,price,total))**

3. (*Dictionaries, Exceptions*)

Defintion: a substring w of a string s is said to be a *word* of s if it is a maximal substring containing no whitespace characters. Thus w is at the beginning of s (prefix) or is preceded by a whitespace character; and is either is a terminal substring of s (suffix) or is followed by a whitespace character. For the string   
 s = "this is that and that is this and that is all!"  
the list of words of s is  
['this','is','that','and','that','is','this','and',that','is','all!']

You are to write two functions, wordcounter1(fileobject) and wordcounter2(filename). For both functions, if no errors occur, the function returns a dictionary such that

* the keys are the words of the file specified by the argument; and
* the value for a word is the number of times that word appears in the file.

If an error occurs, None is returned. Also, any files that are opened by workcounter2 must be closed regardless of any errors.

For wordcounter1, fileobject is a file object which has been opened, but may not be opened for reading; attempting to read from this file might fail.

for wordcounter2, filename is a path which may or may not correspond to an actual file. The function wordcounter2 should call wordcounter1.

Example: if the file contents equal the string "this is that and that is this and that is all!", then the dictionary returned would be   
  
 **{'this' : 2, 'that ': 3, 'is' : 3, 'and' : 2, 'all!' : 1}**

*(see next page)*

def wordcounter1(f):  
 # f is a file object that has already been opened  
 D = {}  
 try:  
 words = f.read().split()  
 for w in set(words):  
 D[w] = len([x for x in words if x == w])

return D  
 except Exception as e:  
 print(e)

def wordcounter2(fname):  
# fname is a path which may or may not correspond to file  
# if a file is opened, it must be closed before the function   
# terminates.

fh = None  
try:  
 fh = open(fname,'r')  
 return wordcounter1(fh)  
except FileNotFoundError as e:  
 print(e)  
finally:  
 if fh:  
 fh.close()

4. ( *Slices, Comprehensions, whole slice assignment)*

Let L be a list of integers.

(a) Write a slice of L consisting of every other element of L starting at the first  
 L[ : : 2]

(b) Write a list comprehension whose elements will be double the elements of the above slice.

K = [ 2\*k for k in L[ : : 2] ]

(c) Write a **single line** that will double every other element of L, starting at the first.

L[ : : 2] = K

*Note: there is not a single correct answer for this part.*

5. Write the code for the following function. It is similar to the BigFiles assignment. Add any needed imports.

**import os**

**def getFilesBySize(startingdir, minsize):**

**''' Returns a list of the paths of the files in startingdir and all directories below startingdir whose size exceeds the integer minsize.'''**

**pathlist = [] #array that will contain the result**

**for dirname,subdir,files in os.walk(startingdir):**

**for fname in files:**

**currfilepath = dirname**

**currfilepath = os.path.join(currfilepath, fname)**

**if os.path.getsize(currfilepath) > minsize:**

**pathlist.append(currfilepath)**

**return pathlist**