PROBLEM 1 : (***What is the output? (16 points)***)

Part A. What is the output of the following code segments?
Write the output to the right. Note that there is only output for the print statements.

```
lista = ['S', 'Y', 'C']
lista.append('B')
lista.extend(['U','E'])
print lista
lista = ['S', 'Y', 'C']
lista.insert(1,'J')
print lista
lista.remove('Y')
print lista
lista.pop()
print lista
# -------------------------
seta = set([3,1,5,1,3,7,5])
seta.add(4)
seta.add(5)
print sorted(list(seta))
seta = set([3,1,5,1,3,7,5])
seta.remove(5)
print sorted(list(seta))
# -------------------------
seta = set([4, 2, 8])
setb = set([8, 4, 3, 7])
print seta.difference(setb)
print setb.union(seta)
print seta&setb
# -------------------------
d = {'J':8, 'M':2, 'D':6, 'F':8}
print sorted(d.keys())
d['H'] = 6
d['J'] = 7
print sorted(d.keys())
print sorted(d.values())
print sorted(d.items())
```

Part B. What is the output of the following code segment? **Write the output after the code segment**. Note that there is only output for the print statements.

```
nums = [60, 50, 30, 50, 50, 30, 50, 70]
dict = {}
for n in nums:
    if n not in dict:
        dict[n] = 1
    else:
        dict[n] += 1
print dict.keys()
print dict.values()
print max(dict.values())
```

What is the output?

PROBLEM 2 : (***Short Code/Answer (17 points)***)

For parts that involve writing code, your code should also work if the given list was modified with different values.

A. (3 pts) Consider the following code.

What is the value of `result` after this line executes?

```
words = ["python", "meerkat", "giraffe", "rabbit", "cat", "ferret"]
result = [len(v) for v in words if len(v) < 7]
```

B. (3 pts) Consider the following code.

What is the value of `result` after this line executes?

```
words = ["python", "meerkat", "giraffe", "rabbit", "cat", "ferret"]
result = ["".join(sorted(list(w))) for w in words if 'a' in w]
```

C. (5 pts) **Write one line of code that includes a list comprehension** to assign to the variable `result` a new word formed from every other letter from `word` starting with the first such letter. The resulting word should have the letters in the same order as the original word.

For example, if `word` is `'giraffe'`, then after executing the list comprehension, then `result` would be the word `grfe`.

Write the list comprehension below.

```
result =
```

D. (6 pts) Write a function named `lastNames` that has one parameter `namelist` that is a list of strings representing names. Each name in `namelist` is two or more words separated by a blank. This function returns a list of ordered tuples. For each unique last name a tuple is created. The first part of the tuple is a last name and the second part is the number of times that last name appears in namelist as a last name. The list of tuples returned are **sorted** by last names.

For example, suppose `namelist` has the following value:

```
namelist = ["Abraham Lincoln", "Grace Kelly", "Michael Jackson",
        "Grace Murray Hopper", "Janet Damita Jo Jackson"]
```

Then the call `lastNames(namelist)` would return the list
`[('Hopper', 1), ('Jackson', 2), ('Kelly', 1), ('Lincoln', 1)]`

```
def lastNames(namelist):
```

PROBLEM 3 : (*Shopping for food (15 points)*)

Consider the following data file of information on buying items at a grocery store. Each line in the file represents one purchase by a customer. The format of each line is in one of two formats depending on whether the item has a price or is sold by weight and thus has a weight and a price per pound. Here is the format of a line. The first item is the customerID, followed by colon, followed by the item to purchase, followed by colon, followed by the letter P or W. If the letter is P, it is followed by a colon and then a price. If the letter is W, it is followed by a colon, followed by a weight, followed by a colon, followed by a price per pound.

An example of the data file is shown below. For example, in the first line, the customer id is 45623, the item to purchase is apples, the letter is W, meaning the 2.4 is the weight and 5.00 is the price per pound. In the second line, the customer id is 7634, the item to purchase is peanut butter, the letter is P, and the price is 4.00.

```
45623:apples:W:2.4:5.00
7634:peanut butter:P:4.00
45623:plums:W:1.5:2.50
45623:spinach:W:1.0:3.50
2375:eggs:P:5.20
7634:oats:W:0.5:3.00
45623:oj:P:3.75
7634:bananas:W:3.2:1.50
2375:yogurt:P:3.25
```

A. (7 pts) Write the function `setupList` that has one parameter `filename` which represents the name of the file. This function returns a list of lists of three things, where each list has the customer id, the item to purchase, and the total price of the item.

For example, the line `data = setupList("purchaseData.txt")` where "purchaseData.txt" is the file above would result in data having the value on the next page. For those items with the letter W, you need to calculate the total price for the item. For apples that would be 2.4 * 5 = 12.0.

```
data = [ ['45623', 'apples', 12.0],
['7634', 'peanut butter', 4.0],
['45623', 'plums', 3.75],
['45623', 'spinach', 3.5],
['2375', 'eggs', 5.2],
['7634', 'oats', 1.5],
['45623', 'oj', 3.75],
['7634', 'bananas', 4.80],
['2375', 'yogurt', 3.25] ]
```

Complete the function setupList below.

```
def setupList(filename):
    f = open(filename)
```

B. (8 pts) Write the function `groceryPurchases` that has three parameters, `data`, `custid` and `amount`, where `data` is the list of lists in the format resulting from Part A, `custid` is a customer id, and `amount` is a the amount of money the customer has.

This function determines which items the customer can purchase based on the desired items to purchase in `data` and the amount of money they have. That is, this function returns a list of tuples representing items the customer wanted and has enough money to purchase. Process the items in the order they are in `data`.

For example, assume *data* is the lists of lists of three items on the previous page. The result of calling `groceryPurchases(data, "45623", 16.00)` is [('apples', 12.0), ('plums', 3.75)]. Customer "45623" could buy apples and plums, but then has only 0.25 cents left and cannot purchase spinach or oj, two other items they were interested in purchasing.

```
def groceryPurchases(data, custid, amount):
```

PROBLEM 4 : (*Flying High (44 points)*)

Suppose you have data about airline flights in the format of a list of lists, where each list represents one flight. In particular, each list has **five strings**: the identifying flight information (which is two words, the airline and a number), the three letter airport code for the departing city, the three letter airport code for the arrival city, the number of seats on the plane, and the estimated number of minutes in the air.

For example, suppose `datalist` is the list below. The first item in the first list in `datalist` represents the flight "Delta 165" (where the airline is the first word "Delta" and the flight number is "165") The second item is the departure city "RDU", the third item is the arrival city "ATL", the fourth item is the total number of seats on the flight, 172, and the fifth item is the estimated number of minutes for the flight, 50.

```
datalist = [ ['Delta 165', 'RDU', 'ATL', '172', '50'],
['JetBlue 1862', 'RDU', 'DTW', '190', '109'],
['Southwest 175', 'RDU', 'DEN', '220', '235'],
['American 1567', 'RDU', 'DEN', '290', '232'],
['JetBlue 4576', 'DTW', 'DEN', '190', '190'],
['Delta 526', 'ATL', 'RDU', '78', '55'],
['Southwest 562', 'ATL', 'DEN', '290', '200'],
['American 1274', 'RDU', 'DEN', '290', '232'],
['Delta 1452', 'PHX', 'ATL', '350', '209'],
['Southwest 157', 'DTW', 'ATL', '260', '115'],
['American 237', 'RDU', 'DEN', '451', '192'],
['Delta 275', 'RDU', 'ATL', '50', '90'],
['JetBlue 422', 'DTW', 'PHX', '340', '160'] ]
```

In writing any of these functions, **you may call any other function you wrote for this problem.** Assume that function is correct, regardless of what you wrote.

A. (7 pts) Write the function named `arrivalCities` which has two parameters, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier, and the string `dcity`, which is a departure city. This function returns a list of the **unique, sorted** names of the arrival cities for flights departing from `dcity`.

For example, if `datalist` is the example list of lists mentioned earlier then the call `arrivalCities(datalist, 'RDU')`

would return the list `['ATL', 'DEN', 'DTW']`

```
def arrivalCities(datalist, dcity):
```

B. (7 pts) Write the function `longFlights` which has three parameters, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier, a string `airline` representing an airline, and an integer `time`. This function returns a **sorted unique** list of departure cities for `airline` whose flights are longer than `time`.

For example, if `datalist` is the list of lists mentioned earlier, then `longFlights(datalist, 'Southwest', 120)` would return the list ['ATL', 'RDU']. Both of these departure cities have at least one flight on Southwest that takes more than 120 minutes. Note the resulting list of cities are unique and in sorted order. Note that DTW also has a Southwest flight but it is shorter than 120 minutes.

```
def longFlights(datalist, airline, time):
```

C. (7 pts) Write the function `citiesNotFlownTo` which has two parameters, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier, and a list named `cities` that is a list of popular cities. This function returns a **sorted unique list** of the popular cities that are not in datalist. Consider all cities in datalist, arrivals and departures.

For example, if `datalist` is the list of lists mentioned earlier and if `popular` is the list of cities `['SEA', 'SFO', 'ATL', 'DEN', 'IAH']`

then the call `citiesNotFlownTo(datalist, popular)` would return the list `['IAH', 'SEA', 'SFO']` .

```
def citiesNotFlownTo(datalist,cities):
```

D. (8 pts) Write the function `cityMostFlights` which has one parameter, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier. This function computes the city from datalist that has the most flights, in or out of the city. You can assume there are no flights where the departure and arrival cities are the same. **You must build a dictionary as part of solving this problem.**

For example, if `datalist` is the list of lists mentioned earlier, then the call `cityMostFlights(datalist)` would return the city 'RDU', which has eight flights in datalist, the most number of any of the cities in datalist. If there was a tie, then you can return any one of the cities that tied.

```
def cityMostFlights(datalist):
```

E. (7 pts) Write the function `dictAirlines` which has one parameter, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier. This function returns a dictionary where each airline is mapped to a list of tuples of information for flights for that airline, where each tuple is a flight (airline and number) and the estimated minutes for that flight.

For example, if datalist is the list of lists mentioned earlier, then the call `dictAirlines(datalist)` would return a dictionary with several entries. One entry would have key 'American' with value [('American 1567', 232), ('American 1274', 232), ('American 237', 192)] as these are all the flights for American.

```
def dictAirlines(datalist):
```

F. (8 pts) Write the function `airlineLargestMinutes` which has one parameter, `datalist`, that is a nonempty list of lists of five strings in the format mentioned earlier. This function calculates the airline with the most total time in the air (the sum of the minutes of all the flights for this airline) and returns a tuple of three items. The first item is the name of such airline and the second item is the total minutes of all the flights for this airline. The third item is a list of tuples of the flights (name and number) and their estimated minutes for all flights for this airline. Assume there is no tie.

For example, if datalist is the list of lists mentioned earlier, then `airlineLargestMinutes(datalist)` would return `('American', 656, [('American 1567', 232), ('American 1274', 232), ('American 237', 192)])` . Note the airline with the most estimated minutes in the air is American, and the total minutes is 656. The list of tuples are all the individual flights for this airline, each with their estimated number of minutes. The 656 is the sum of all the minutes in that list.

```
def airlineLargestMinutes(datalist):
```