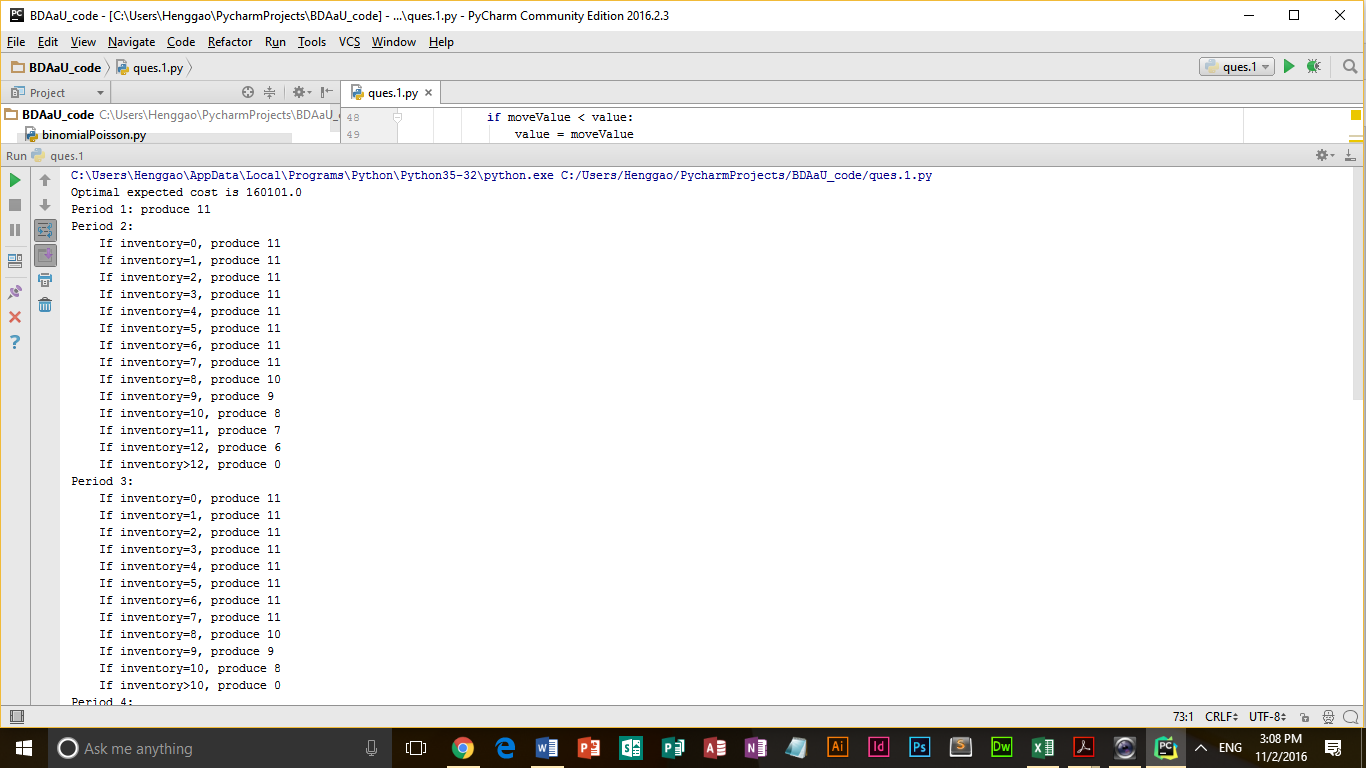
Homework 6

Henggao Cai

November 2, 2016

Question 1:

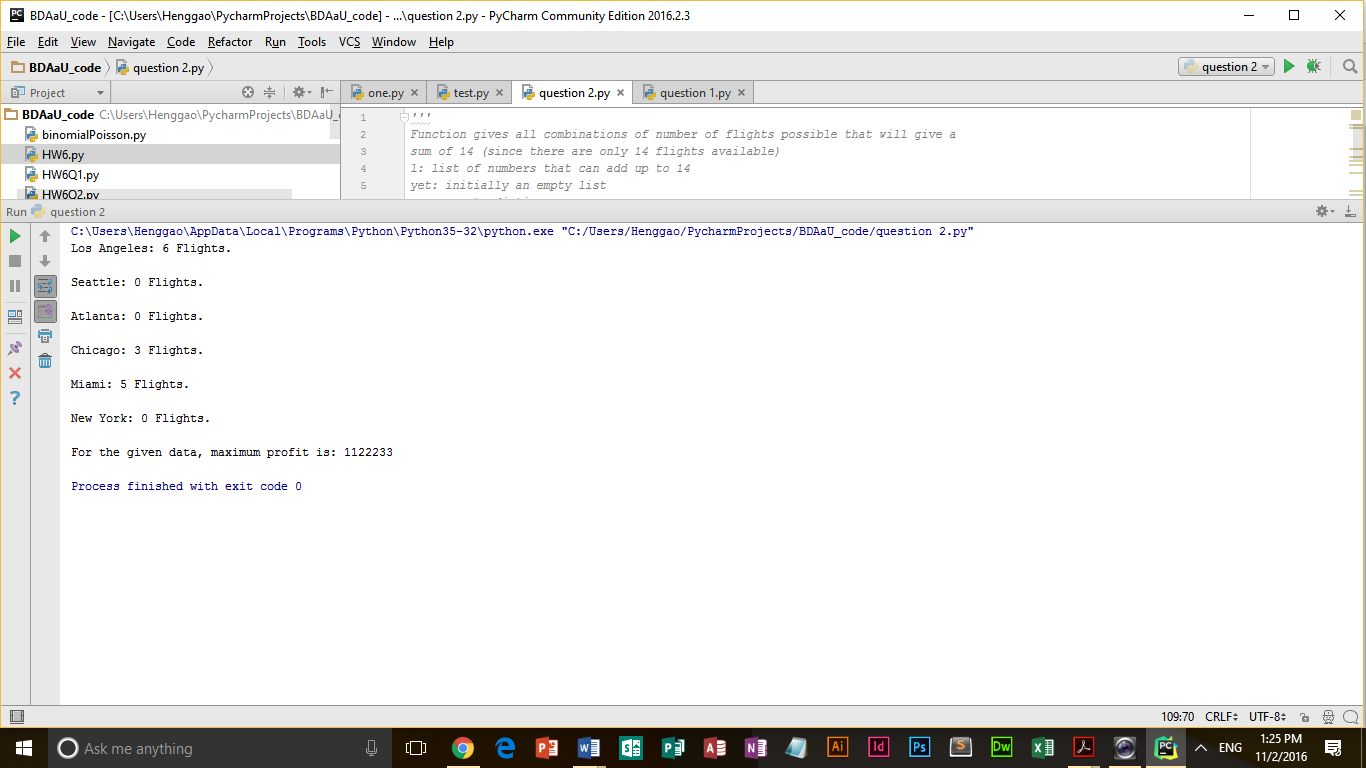
The optimal expected cost is $160,101.



**import** numpy  
  
hugeNumber = float(**"inf"**)  
unused = -1000  
  
stages = 6  
inventory = 4  
inventoryCapacity = 15  
productionCapacity = 15 - inventory  
  
setupCost = numpy.array([unused, 120, 125, 130, 125, 120, 110]) *# "unused" is for element 0,*unitCost = numpy.array([unused, 8.00, 10.00, 9.50, 9.50, 10.00, 10.00]) *# which we don't use*holdingCost = numpy.array([unused, 0.50, 0.60, 0.60, 0.70, 0.75, 0.70])  
shortageCost = numpy.array([unused, 35.00, 40.00, 35.00, 30.00, 25.00, 25.00])  
itemDemand = numpy.array([unused, 3, 4, 5, 9, 11, 7])  
  
minDemand = 3  
maxDemand = 11  
  
*# End of input data section*f = numpy.zeros([stages + 2, inventoryCapacity + 1])  
x = numpy.zeros([stages + 1, inventoryCapacity + 1], dtype=int)  
  
**for** t **in** range(stages, 0, -1):  
  
 **for** i **in** range(inventoryCapacity + 1):  
  
 minProduction = max(0, maxDemand - i)  
  
 maxProduction = min(productionCapacity, inventoryCapacity - i + minDemand)  
 value = hugeNumber  
 bestMove = unused *# Nothing meaningful in here yet* **for** p **in** range(minProduction, maxProduction + 1):  
  
 *# Compute production cost* productionCost = unitCost[t] \* p  
 **if** p > 0:  
 productionCost += setupCost[t]  
 inventory = inventory + p - itemDemand[t]  
 productionCost += (inventory + p - itemDemand[t]) \* holdingCost[t] / 2  
 moveValue = productionCost  
 **for** d **in** range(minDemand, maxDemand + 1):  
 **if** (inventory + p < itemDemand[t]):  
 moveValue += (itemDemand[t] - p - inventory) \* (shortageCost[t])  
 **if** moveValue < value:  
 value = moveValue  
 bestMove = p  
  
 *# End of p loop* f[t, i] = value  
 x[t, i] = bestMove  
  
 *# End of i loop  
  
# End of t loop*print(**"Optimal expected cost is "** + str(f[1, 4]))  
print(**"Period 1: produce "** + str(x[t, 4]))  
**for** t **in** range(2, stages + 1):  
 print(**"Period "** + str(t) + **":"**)  
 sumOfxt = sum(x[t, :])  
 **for** i **in** range(inventoryCapacity + 1):  
 print(**" If inventory="** + str(i) + **", produce "** + str(x[t, i]))  
 sumOfxt -= x[t, i]  
 **if** sumOfxt <= 0:  
 print(**" If inventory>"** + str(i) + **", produce 0"**)  
 **break**

Question 2:

The optimal strategy involves six flights to Los Angeles, three to Chicago, and five to Miami.

*'''  
Function gives all combinations of number of flights possible that will give a  
sum of 14 (since there are only 14 flights available)  
l: list of numbers that can add up to 14  
yet: initially an empty list  
a: an empty dictionary  
'''***def** total(l, yet, a):  
 *# sum of all elements in the list - yet* s = sum(yet)  
 **if** s == 14:  
 a[string(yet)] = 1 *# if s is 14, save it in the dictionary a* **return  
  
 elif** s > 14: *# if s exceeds 14, ignore it* **return** n = len(l) *# length of numbers in list l* **for** i **in** range(n):  
 *# call the fucntion recursively with l not having the current element and yet having it* total(l[i + 1:], yet + [l[i]], a)  
  
  
**'''  
Function returns a string concatenating the numbers present in list l  
'''  
  
  
def** string(l):  
 s = **''  
 for** num **in** l:  
 s += str(num)  
 **return** s  
  
  
**'''  
Function to be called first  
'''  
  
  
def** flights(costs):  
 *# maximum number of 1 number of flights is 6 since 6 < 14  
 # maximum number of 2 number of flights is , since 6\*2 < 14  
 # similarly for 6, maximum is 2, since 6\*2 < 14  
 # l is the list of the numbers that can add up to 14* l = [1] \* 6 + [2] \* 6 + [3] \* 4 + [4] \* 3 + [5] \* 2 + [6] \* 2  
  
 *# empty dictionary* a = {}  
 *# function called, dictionary a will be edited* total(l, [], a)  
  
 *# maximum profit initialized* max = 0  
 *# all combinations present in a are checked* key = **''  
 for** k **in** a:  
 p = profit(k, costs)  
 **if** p > max:  
 max = p  
 key = k  
  
 displaySchedule(key, costs)  
 print(**"For the given data, maximum profit is: "**+key)  
  
  
**def** profit(k, costs):  
 *# initialize index of each number of flight.* indexes = [0] \* 6  
 p = 0  
 **for** c **in** k:  
 i = int(c)  
 *# chose the highest profit first* p += costs[i - 1][indexes[i - 1]][0]  
 *# increment the index, so that the second highest profit can be taken* indexes[i - 1] += 1  
  
 **return** p  
  
  
**def** displaySchedule(k, costs):  
 indexes = [0] \* 6  
 *# List of city names* cities = [**"New York"**, **"Los Angeles"**, **"Miami"**, **"Chicago"**, **"Seattle"**, **"Atlanta"**]  
 *# number of flights initialised to zero for each city* sched = {**"New York"**: 0, **"Los Angeles"**: 0, **"Miami"**: 0, **"Chicago"**: 0, **"Seattle"**: 0, **"Atlanta"**: 0}  
 *# for each character in k. the character is converted to an int  
 # the index of the highest profit flight is noted and is used to access the city name  
 # the number of flights taken from the city is then incremented in the dictionary sched* **for** c **in** k:  
 i = int(c)  
 ind = costs[i - 1][indexes[i - 1]][1]  
 sched[cities[ind - 1]] += i  
 indexes[i - 1] += 1  
  
 **for** city **in** sched:  
 print(city + **": "** + str(sched[city]) + **" Flights."**)  
 print(**" "**)  
  
  
*# data saved in this way  
# costi is the list of flight costs for all cities when i number of flights are taken  
# each of these lists have a tuple which signifies the index of the city*costs1 = [(80, 1), (100, 2), (90, 3), (120, 4), (70, 5), (80, 6)]  
costs2 = [(150, 1), (195, 2), (180, 3), (200, 4), (160, 5), (175, 6)]  
costs3 = [(210, 1), (275, 2), (265, 3), (240, 4), (190, 5), (245, 6)]  
costs4 = [(250, 1), (325, 2), (310, 3), (245, 4), (230, 5), (280, 6)]  
costs5 = [(270, 1), (300, 2), (350, 3), (290, 4), (250, 5), (340, 6)]  
costs6 = [(280, 1), (250, 2), (320, 3), (300, 4), (290, 5), (330, 6)]  
*# saving all these lists in the list cost*costs = [costs1,costs2,costs3,costs4,costs5,costs6]  
  
*# sort all the lists present in the list cost in decreasing order***for** cost **in** costs:  
 cost.sort(reverse=**True**)  
  
flights(costs)