Ονοματεπώνυμο: Δημήτριος Φούντας
A.M:1112201600236
Τμήμα: Μαθηματικών
ΕΡΓΑΣΙΑ ΤΕΧΝΗΤΗΣ ΝΟΗΜΟΣΥΝΗΣ

Θα ξεκινήσουμε επεξηγώντας το πρώτο μέρος της εργασίας. Στην συνάρτηση add διορθώνουμε την τιμή που επιστρέφει η συνάρτηση και επιστρέφουμε το άθροισμα των στοιχείων που μας δίνονται στην είσοδο. Στην συνάρτηση buyLotsOfFruit παίρνω τις πλειάδες φρούτων και κιλών που μας δίνονται σαν είσοδο, αν όλα τα φρούτα του orderlist υπάρχουν στα κλειδιά του λεξικού fruitPrices, υπολογίζω το κόστος της αγοράς μου με τις τιμές που μου δίνει το λεξικό fruitPrices. Σε διαφορετική περίπτωση εμφανίζω Error.
Στη συνάρτηση shopSmart αρχικοποιούμε το πρώτο fruitshop ως το φθηνότερο για την αγορά μας και αρχικοποιούμε και το κόστος του ως ελάχιστο κόστος, στην συνέχεια διαπερνώ καθε fruitshop εκτός του πρώτου που χρησιμοποιήσαμε , βρίσκουμε το κόστος και το όνομα τους. Αν το καινούργιο κόστος είναι μικρότερο απο αυτό που όρισαμε ως ελάχιστο τότε ορίζεται αυτο ως ελάχιστο και αποθηκεύεται το όνομα του , η διαδικασία αυτή συνεχίζεται μεχρι να βρεθεί το πραγματικό ελάχιστο κόστος και να αποθηκευτεί το όνομα του αντίστοιχου μαγαζιού , το οποίο θα επιστρέφουμε.

Στο δεύτερο μέρος της εργασίας κάνουμε μια υλοποίηση της ουράς με την χρήση σορού. Ξεκινάμε φτιάχνοντας μέθοδο η οποία κάθε φορά που δημιουργούμε ενα στοιχείο PriorityQueue αρχικοποιεί ταυτόχρονα έναν κενό σορό και έναν μετρητή στον οποίο δίνει την τιμή 0. Στην συνέχεια φτιάχνουμε μέθοδο push με την οποία μπορούμε να εισάγουμε στοιχείο πλειάδα (προτεραιότητα, αντικείμενο) με ιδιοχαρακτηριστικά την προτεραιότητα και το αντικείμενο μέσα στον σορό. Ταυτόχρονα αυξάνουμε κατά 1 τον μετρητή μας count καθώς πλέον ο σορός έχει ένα επιπλέον στοιχείο. Στην συνέχεια δημιουργούμε μια μέθοδο pop η οποία μας επιστρέφει μόνο το ιδιοχαρακτηριστικό του αντικειμένου από την πλειάδα.Μετά δημιουργούμε την μέθοδο isEmpty η οποία ελέγχει με την χρήση του μετρητή count αν ο σορός μας είναι άδειος. Τέλος δημιουργούμε την μέθοδο update η οποία ζητάει μια πλειάδα του τύπου (αντικείμενο,προτεραιότητα) και χρησιμοποιεί την λογική μετάβλητη inside για να ελέγξει αν το αντικείμενο που μας δίνεται κατά την είσοδο βρίσκεται στον σορό μας . Αν υπάρχει το αντικείμενο που μας δώθηκε στην είσοδο μέσα στον σορό και η νέα προτεραιότητα που δώθηκε στην είσοδο είναι μικρότερη απο αυτήν που είχε εξ αρχής τότε σπρώχνω(heappush) την πλειάδα της (νέας προτεραιότητας, αντικείμενο) στον σορό μου και ταυτόχρονα αλλάζω την τιμή της λογικης μεταβλητής inside . Αλλιώς προκειμένου να μην χάσουμε κανένα απο τα στοιχεία που βγάλαμε απο τον σορό, τα εισάγω(append) ένα-ένα σε μια λίστα. Στην συνέχεια όλα τα αποθηκευμένα στοιχεία(πλειάδες) που είχα εισάγει στην λίστα τα αφαιρώ απο αυτή και τα σπρώχνω όλα πάλι πίσω στον σορό. Αν δεν υπάρχει στον σορό μας το αντικείμενο που μας δώθηκε κατα την είσοδο , τότε απλά σπρώχνουμε(heappush) την πλειάδα που δώθηκε στην είσοδο,μέσα στον σορό και ανεβάζουμε κατά μια μονάδα τον μετρητή που μετράει τα στοιχεία του σορού.

Τέλος δημιουργούμε συνάρτηση που δέχεται μια λίστα l ακεραίων και την επιστρέφει ταξινομημένη κατά αύξουσα σειρά με χρήση ουράς. Αυτό επιτυγχάνεται δημιουργόντας ένα αντικειμενο PriorityQueue , εισάγουμε τα στοιχεία (νούμερα της) λίστας ένα-ένα στην ουρά με την μορφή πλειάδας κάθε μία από τις οποίες έχουν ως αντικείμενο και ως προτεραιώτητα το ίδιο το στοιχείο της λίστας. Αμέσως μετά αδειάζουμε την λίστα για να την ξαναγεμίσουμε με την σειρά που θέλουμε. Τέλος διατρέχουμε όλα τα στοιχεία που έχει μέσα η ουρά, τα αφαιρούμε ένα-ένα με την χρήση του .pop() , αυτό θα έχει ως αποτέλεσμα να βγει πρώτα το μικρότερο αντικείμενό και μετά το αμέσως μεγαλύτερο και θα συνεχίσει

όμοια η διαδικασία. Κάθε αντικείμενο που βγαίνει απο τον σορό το εισάγουμε πίσω στην λίστα , η οποία με αυτόν τον τρόπο θα γεμίσει με την σειρά που θέλουμε.

Εργασία Μέρος 2

```
from _heapq import heappush, heappop, heapreplace

                                    #Αρχικοποιούμε τον σορό και τον μετρητή
class PriorityQueue:
    def __init__(self):
        self.pq = []
        self.count = 0
#Με την συνάρτηση push εισάγουμε στοιχείο πλειάδα με ιδιοχαρακτηριστικά την προτεραιότητα και το αντικείμενο και τα εισάγω στον σορό. Ταυτόχρονα αυξάνουμε κατα 1 τον μετρητή μας count  καθώς πλέον ο σορός έχει ένα επιπλέον στοιχείο
    def push(self, item, priority):
        self.item = item
        self.priority = priority
        heappush(self.pq, (priority, item))
        self.count += 1
# Εμφανίζουμε με την μέθοδο αυτή μόνο το αντικείμενο από την πλειάδα
    def pop(self):
        self.tup = heappop(self.pq)
        self.count -= 1
        return (self.tup[1])
# ελέγχουμε με την χρήση μετρητή count αν ο σορός μας είναι άδειος
    def isEmpty(self):
        return (self.count == 0)
    def update(self, item, priority):
        inside = False #λογική μεταβλητή που ελέγχει αν το αντικείμενο που μας δώθηκε στην είσοδο υπαρχει στην λίστα
        l = []
        while True and len(self.pq): #ανοίγω επανάληψη μέχρι να βρω στον σορό το αντικείμενο που δώθηκε στην είσοδο ή αν δεν υπάρχει , όταν έχω ψάξει ολα τα αντικείμενα του σορού
            temp = heappop(self.pq) # βγάζω ένα-ένα τα στοιχεία από τον σορό
            if item == temp[1] and priority < temp[0]: #αν βρω το αντικείμενο που μου δώθηκε στην είσοδο και η νέα προτεραιότητα που δώθηκε στην είσοδο είναι μικρότερη απο αυτήν που ήδη έχει εισάγω την πλειάδα της (νέας προτεραιότητας, αντικείμενο) στον σορό μου
                heappush(self.pq, (priority, item)) #εισάγω την πλειάδα που μας δώθηκε, στον σορό στην θέση της παλιάς πλειάδας που είχε ως ιδιοχαρακτηριστικό το αντικείμενο που μας δώθηκε
                inside = True
                break
            else:
                l.append(temp) #αλλιώς προκειμένου να μην χάσω κανένα απο τα στοιχεία που έβγαλα απο τον σορό, τα αποθηκεύω ένα-ένα σε μια λίστα
```

```
    for i in range(len(l)): #αδειάζω όλα τα απείραχτα αποθηκευμένα στοιχεία(πλειάδες) που είχα εισάγει στην λίστα και τα σπρώχνο όλα πάλι πίσω στον σορό
        tup = l.pop()
        heappush(self.pq, tup)
    if inside == False: #αν δεν υπάρχει στον σορό μας το αντικείμενο που μας δώθηκε κατα την είσοδο , τότε απλά σπρώχνουμε την πλειάδα που δώθηκε στην είσοδο,μεσα στον σορό και ανεβάζουμε κατά μια μονάδα τον μετρητή που μετράει τα στοιχεία του σορού
        self.count += 1
        heappush(self.pq, (priority, item))

#Φτιάχνουμε συνάρτηση που δέχεται μια λίστα l ακεραίων και την επιστρέφει ταξινομημένη κατα αύξουσα σειρά με χρήση ουράς
def PQSort(l):
  h = PriorityQueue() #αρχικοποιούμε μια κενή ουρά
  count=0 #μετράμε τα στοιχεία που θα εισάγουμε στην ουρά
  for number in l:
    h.push(number, number)#εισάγουμε τα νούμερα της λίστας ενα-ενα στην ουρά
    count+=1
  l.clear() #αδειάζουμε την λίστα για να την ξαναγεμίσουμε με την σειρά που θέλουμε
  for i in range(count): #για όλα τα στοιχεία που έχει μέσα η ουρά
    sort=h.pop()  #τα αφαιρούμε απο την ουρά με την χρήση του .pop(), και αφού τα αντικείμενα εχουν item==priority θα βγει πρωτα το μικρότερο ντικείμενό και μετά το αμέσως μεγαλύτερο και θα συνεχίσει όμοια
    l.append(sort)  #επιστρέφω πλέον τα στοιχεία πίσω την λίστα αυτήν την φορά με την σειρά που θέλουμε
  return l
```

Εργασία Μέρος 1

Addiction.py

```
def add(a, b):
  "Return the sum of a and b"
  print("Passed a = %s and b = %s, returning a + b = %s" % (a, b, a + b))
  return a + b
b=add(1,2)
```

buyLotsOfFruit.py

```
from __future__ import print_function

fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75,
               'limes': 0.75, 'strawberries': 1.00}


def buyLotsOfFruit(orderList):

    totalCost = 0.0
    for fruit,numprices in orderList:#παίρνω τις πλειάδες φρούτων και κιλών που θέλουμε να αγοράσουμε
        if fruit in fruitPrices:#Υπολογίζω το κόστος της αγοράς μου με τις τιμές που μου δίνει το λεξικό  fruitPrices αν όλα τα φρούτα του orderlist υπάρχουν στα κλειδιά του λεξικού fruitPrices σε διαφορετική περίπτωση εμφανίζω Error
```

```
            totalCost+=fruitPrices[fruit]*numprices
        else:
            print("Error")
    return  totalCost



# Main Method
if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orderList = [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)]
    print('Cost of', orderList, 'is', buyLotsOfFruit(orderList))
```

shopSmart.py

```
rom __future__ import print_function
import shop


def shopSmart(orderList, fruitShops):
        mincost=fruitShops[0].getPriceOfOrder(orderList)#Αρχικοποιώ το
πρώτο fruitshop ως το φθηνότερο για την αγορά μου και αρχικοποιώ και το
κόστος του ως ελάχιστο κόστος
        minshop=fruitShops[0]
        for i in range(1,len(fruitShops)):# διαπερνώ καθε fruitshop εκτος
του πρώτου που χρησιμοποίησα
            cost=fruitShops[i].getPriceOfOrder(orderList)
            name=fruitShops[i] #βρίσκω το κόστος και το όνομα τους
            if cost<mincost: #Αν το καινούργιο κόστος είναι μικρότερο απο
αυτό που όρισα ελάχιστο τότε ορίζεται αυτο ως ελάχιστο και αποθηκεύεται το
όνομα του , η διαδικασία αυτή συνεχίζεται μεχρι να βρεθεί το πραγματικό
ελάχιστο κόστος και να αποθηκευτεί το όνομα του αντίστοιχου μαγαζιού , το
οποίο επιστρέφω
                minshop=name
                mincost=cost
        return minshop

if __name__ == '__main__':
    "This code runs when you invoke the script from the command line"
    orders = [('apples', 1.0), ('oranges', 3.0)]
    dir1 = {'apples': 2.0, 'oranges': 1.0}
    shop1 = shop.FruitShop('shop1', dir1)
    dir2 = {'apples': 1.0, 'oranges': 5.0}
    shop2 = shop.FruitShop('shop2', dir2)
    shops = [shop1, shop2]
    print("For orders ", orders, ", the best shop is", shopSmart(orders,
shops).getName())
    orders = [('apples', 3.0)]
    print("For orders: ", orders, ", the best shop is", shopSmart(orders,
shops).getName())
```

Ονοματεπώνυμο: Δημήτριος Φούντας
Α.Μ:1112201600236
Τμήμα: Μαθηματικών