Στο αρχείο **addition.py** υπάρχει μία μόνο συνάρτηση, η *add*, η οποία προσθέτει 2 αριθμούς που παίρνει ως ορίσματα, βάζει το αποτέλεσμα στην μεταβλητή sum και την επιστρέφει.

Στο επόμενο αρχείο **buyLotsOfFruit.py**, έχουμε τη συνάρτηση *buyLotsOfFruit* η οποία υπολογίζει το συνολικό κόστος μιας παραγγελίας. Η παραγγελία βρίσκεται σε μία λίστα από *tuples*, η οποία περνιέται ως όρισμα στη συνάρτηση, και η οποία προσπελαύνεται και για κάθε *tuple*, παίρνω το δεύτερο στοιχείο της, το οποίο είναι η ποσότητα του φρούτου που θέλω να αγοράσω και χρησιμοποιώ το πρώτο στοιχείο, το οποίο είναι το όνομα του φρούτου, για να βρω την τιμή του φρούτου στο *dictionary fruitPrices*. Πολλαπλασιάζω αυτές τις 2 τιμές και τις προσθέτω στην μεταβλητή *totalCost*. Αφού γίνει η ίδια διαδικασία για όλα τα φρούτα της παραγγελίας, επιστρέφω το *totalCost*.

Στο αρχείο **shopSmart.py**, έχουμε τη συνάρτηση *shopSmart*, η οποία παίρνει ως όρισμα 2 λίστες, *orderList* και *fruitShops*. Η συνάρτηση με βάση την παραγγελία που βρίσκεται στην *orderList* ψάχνει να βρει σε ποιο *shop* συμφέρει να αγοράσουμε τα φρούτα της παραγγελίας. Αρχικά προσπελαύνω τη λίστα *fruitShops* και για κάθε *shop* βρίσκω την συνολική τιμή της παραγγελίας χρησιμοποιώντας τη συνάρτηση *getPriceOfOrder* της κλάσης *fruitShop*. Κάθε τιμή που βρίσκω τη βάζω σε μία λίστα κι αφού βρω τις τιμές για όλα τα *shop*, βρίσκω τη μικρότερη τιμή χρησιμοποιώντας τη συνάρτηση *min*. Τέλος χρησιμοποιώντας τη συνάρτηση *index*, βρίσκω σε ποια θέση της λίστας είναι η μικρότερη τιμή κι έτσι την αντιστοιχίζω με το σωστό *shop*. Με αυτόν τον τρόπο το *return fruitShops[index]* επιστρέφει το shop με τη μικρότερη τιμή για την παραγγελία.

Τέλος, το αρχείο **PriorityQueue.py**, περιέχει μια κλάση *PriorityQueue* με τα εξής χαρακτηριστικά: ένα *heap* το οποίο αρχικοποιείται ως κενή λίστα και έναν *counter* που αρχικοποιείται ως 0 και μας δείχνει πόσα στοιχεία έχει μέσα το *heap* και 5 συναρτήσεις.
Η συνάρτηση **push** βάζει ένα στοιχείο μέσα στο *heap* στην κατάλληλη θέση με βάση το priority του στοιχείου, χρησιμοποιώντας τη συνάρτηση *heappush* της βιβλιοθήκης *heapq* και αυξάνω τον counter κατά 1. Επειδή όμως, η *heappush* παίρνει 2 ορίσματα, το 1ο είναι η λίστα και το 2ο πρέπει να είναι ακέραιος, όταν την καλώ περνάω ως 2ο όρισμα ένα *tuple* της μορφής (*priority, item*) έτσι ώστε να γίνουν και οι έλεγχοι σωστά οι οποίοι απαιτούν ακέραιο ο οποίος πρέπει να είναι το πρώτο στοιχείο του *tuple*.
Μετά, η **pop** ελέγχει με τη βοήθεια του *counter* αν το heap είναι κενό και αν δεν είναι, τότε καλεί την *heappop* της *heapq* η οποία μειώνει το *counter* κατά 1, βγάζει το στοιχείο με το μικρότερο *priority* και επιστρέφει το όνομα του.
Επόμενη συνάρτηση είναι η **isEmpty**, η οποία πολύ απλά ελέγχοντας την τιμή του *counter* μας λέει αν το *heap* είναι άδειο.
Έπειτα, έχουμε την **update**, η οποία παίρνει ως όρισμα ένα στοιχείο και ελέγχει αν υπάρχει ήδη στο *heap*. Αν υπάρχει, και το priority του είναι μικρότερο του *priority* του καινούριου, τότε η συνάρτηση επιστρέφει χωρίς να κάνει τίποτα, ενώ αν είναι μεγαλύτερο, τότε απλά αλλάζει την τιμή του *priority* του υπάρχοντος και την κάνει

ίση με του καινούριου. Αν το στοιχείο δεν υπάρχει, τότε καλείται η συνάρτηση *push* και το στοιχείο μπαίνει στο *heap*.
Τελευταία είναι η συνάρτηση **PQSort**, η οποία παίρνει ως όρισμα μία λίστα ακεραίων και την επιστρέφει ταξινομημένη κατά αύξουσα σειρά. Αυτό το πετυχαίνει χρησιμοποιώντας τις συναρτήσεις *push* και *pop*. Κάθε στοιχείο της λίστας μπαίνει με την *push* στο *heap* και μετά με τη χρήση της *append*, βάζω σε μια νέα λίστα, με τη σειρά κάθε στοιχείο που κάνω *pop*. Έτσι η νέα λίστα περιέχει τα στοιχεία της πρώτης αλλά ταξινομημένα και η συνάρτηση επιστρέφει αυτή τη λίστα.
Δεν ετοίμασα κάποια συνάρτηση **main** αλλά στο τέλος του αρχείου θα βρείτε σε σχόλια κάτι δοκιμές που έκανα για έλεγχο. Αν θέλετε μπορείτε να τις χρησιμοποιήσετε και εσείς αλλιώς κάντε τις δικές σας.