

*Πολυτεχνείο Κρήτης - Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών*



**ΠΟΛΥΤΕΧΝΕΙΟ  
ΚΡΗΤΗΣ**

## *Εργαλεία Ανάπτυξης Λογισμικού και Προγραμματισμός Συστημάτων*

*Χειμερινό Εξάμηνο 2019-2020*

*Αναφορά 2ης Εργασίας*

Σπυριδάκης Χρήστος  
Ζαχαριουδάκης Νικόλας

2014030022  
2016030073

Παρασκευή 29 Νοεμβρίου 2019

# 1 Εισαγωγή

Στο συγκεκριμένο εργαστήριο χρειάστηκε να δημιουργήσουμε σε python έναν parser ο οποίος θα μπορεί να διαβάσει επαναλαμβανόμενα αποδείξεις πώλησης προϊόντων από επιχειρήσεις και στην συνέχεια θα μας απαντάει σε στατιστικά στοιχεία που του ζητούνται. Οι αποδείξεις πρέπει να τηρούν το pre-defined format αποδεκτών αποδείξεων, σε αντίθετη περίπτωση δεν συνυπολογίζονται στις απαντήσεις. Ενώ κατά την απάντηση των αποτελεσμάτων, αυτά ωφελούν να εμφανίζονται σε αύξουσα σειρά του πρωτεύοντος στοιχείου (είτε αριθμητικά είτε αλφαριθμητικά ανάλογα με το είδος της πληροφορίας).

Πρώτο βήμα πριν ξεκινήσουμε την υλοποίηση ήταν να σχεδιάσουμε την λειτουργικότητα του προγράμματος που θα δημιουργήσουμε. Για αυτό το λόγο, και τα δύο μέλη της ομάδας μεμονωμένα έκαναν μία προσωρινή σχεδίαση και στην συνέχεια συμπίχθηκαν αυτές σε μία, κρατώντας τα 'καλύτερα' σημεία του καθενός. Δεν μπορούν να δοθούν, συγκεκριμένες συναρτήσεις ή source lines of code με τα οποία ασχολήθηκε ο κάθε φοιτητής καθώς και οι δύο λίγο πολύ έκαναν διορθώσεις ή έδωσαν συμβουλές για τις υλοποιήσεις του άλλου μέλους. Αλλά μπορούμε να πούμε ότι περισσότερη προσοχή έδωσε ο Ζαχαριουδάκης στον τρόπο αποθήκευσης των δεδομένων και υπολογισμό των στατιστικών αποτελεσμάτων, ενώ ο Σπυριδάκης στο διάβασμα του αρχείου και έλεγχο των αποδεκτών αποδείξεων. Ενώ όσον αφορά με την αναφορά περισσότερη έμφαση έδωσε ο Σπυριδάκης στο **Section - 4** ενώ ο Ζαχαριουδάκης το **Section - 2** και **Section - 3**.

Για να είναι όσο δυνατόν πιο συνοπτική αλλά ταυτόχρονα και αναλυτική η αναφορά, έχει χωριστεί σε τρεις ενότητες. Την Περιγραφή, στην οποία θα αναλυθεί ο τρόπος προσέγγισης του προβλήματος. Την Υλοποίηση, στην οποία θα αναφέρουμε τα 'εργαλεία' και τον τρόπο που κάναμε develop το πρόγραμμα σε python. Ενώ τέλος υπάρχει ο Έλεγχος Λειτουργίας στο οποίο παρουσιάζουμε και πειραματικά την ορθή λειτουργία βάσει της σχεδίασης, στο οποίο επαληθεύεται ότι το πρόγραμμα τρέχει βάσει των requirements.

**Σημείωση:** Να αναφερθεί ότι κατά την υλοποίηση της πρώτης εργασίας, τα μέλη της ομάδας ήταν δύο ανεξάρτητες ομάδες. Ύστερα από συνεννόηση με τον κύριο Δεληγιαννάκη, δημιουργήθηκε η συγκεκριμένη ομάδα δύο ατόμων για την παράδοση των εναπομεινάντων σετ ασκήσεων. Κατά την πρώτη σετ ο κωδικός της ομάδας του φοιτητή Σπυριδάκη ήταν LAB21142505 ενώ του φοιτητή Ζαχαριουδάκη LAB21143105, του οποίου ο κωδικός ομάδας κρατήθηκε.

## 2 Περιγραφή

Πρώτο ζητούμενο της συγκεκριμένης άσκησης είναι να μπορεί το πρόγραμμα να ανοίγει ένα αρχείο και να αποθηκεύει τις αποδεκτές αποδείξεις που υπάρχουν σε αυτό. Αρχικά από την εκφώνηση γίνεται κατανοητό ότι ένας κάτοχος ΑΦΜ μπορεί να έχει στον κατάλογο του εκατοντάδες αν όχι παραπάνω κωδικούς προϊόντων. Γνωρίζοντας ότι το πρόγραμμα θα πρέπει να μπορεί να αντεπεξέλθει σε αρχεία τα οποία μπορεί να περιέχουν εκατοντάδες χιλιάδες γραμμές. Ήταν σημαντικό λοιπόν να σκεφτούμε με ποιον τρόπο μπορούμε να διαβάσουμε τα αρχεία εισόδου όσο πιο γρήγορα γίνεται και να αποθηκεύσουμε τα δεδομένα σε αποδοτικές δομές δεδομένων.

Ιδανικά σε προβλήματα με μεγάλο αριθμό πληροφορίας εισόδου ψάχνουμε να χρησιμοποιήσουμε αλγόριθμους ή δομές δεδομένων, με τις οποίες μπορούμε να απαντήσουμε το πολύ σε  $O(n \log n)$  χρόνο. Επιπλέον, αναγκαστικά καταλαβαίνουμε ότι θα πρέπει να διαβάσουμε όλο το αρχείο καθώς δεν γνωρίζουμε εκ των προτέρων αν και σε ποιες γραμμές υπάρχουν μη έγκυρες αποδείξεις. Επίσης καλό θα ήταν να μπορούμε σε  $O(1)$  χρόνο να απαντάμε σε όλα τα ερωτήματα, πράγμα που στον πραγματικό κόσμο δεν είναι δυνατόν.

Αποφασίσαμε λοιπόν να αποθηκεύσουμε τα δεδομένα σε δύο λεξικά το ένα εμφωλεμένο με το άλλο (όπως φαίνεται στο **Figure 1**), γνωρίζοντας ότι στην python υλοποιούνται ως hash tables, πράγμα που θα μας πρόσφερε  $O(1)$  χρόνο εισαγωγής στην average case για κάθε απόδειξη με συγκεκριμένο ΑΦΜ. Με τον συγκεκριμένο τρόπο θα είχαμε άμεση πρόσβαση στα προϊόντα τα οποία έχει κάποιος πωλητής, άρα και άμεση απάντηση στο τρίτο ερώτημα του μενού. Αυτό το οποίο δεν μας εξυπηρετούσε τόσο είναι η ταχύτητα απαντήσεων για τα ΑΦΜ που έχουν πουλήσει κάποιο προϊόν, καθώς θα πρέπει να ψάξουμε για κάθε ΑΦΜ αν υπάρχει το συγκεκριμένο προϊόν σε αυτό, με βάση την συγκεκριμένη προσέγγιση. Μία λύση θα ήταν να δημιουργήσουμε μία επιπλέον δομή ίδια με αυτή, έχοντας ως βασικό key για τα dictionary το όνομα του προϊόντος, πράγμα που όμως θα ήταν σπατάλη μνήμης.

Ο λόγος που δεν μας επηρεάζει τόσο αυτή η προσέγγιση (και το είδαμε και από τα πειράματα) είναι ότι θεωρητικά ο αριθμός των συνολικών δυνατών κωδικών είναι μεγαλύτερος από τον αριθμό των δυνατών ΑΦΜ. Το μεγάλο μειονέκτημα σε αυτή την υλοποίηση είναι ότι αποθηκεύουμε ονόματα προϊόντων ξανά παρόλο που μπορεί το ίδιο όνομα να το χρησιμοποιεί και κάποιο άλλο ΑΦΜ. Για μεγαλύτερα αρχεία από αυτά που αναφέρονται στην εκφώνηση πιθανόν, καλύτερη λύση θα ήταν να δημιουργήσουμε μία δομή σαν ένα μεταβλητού μεγέθους δι-διαστατό πίνακα (θα έμοιαζε με 2D hash table) όπου η μία διάσταση θα είναι τα ΑΦΜ και η άλλη τα προϊόντα ώστε αρχικά να μην έχουμε την όποια επανάληψη πληροφορίας μεταξύ των values (ονόματα προϊόντων για διαφορετικά ΑΦΜ), όπως επίσης να είχαμε καλύτερο χρόνο εκτέλεσης και για το δεύτερο ερώτημα.

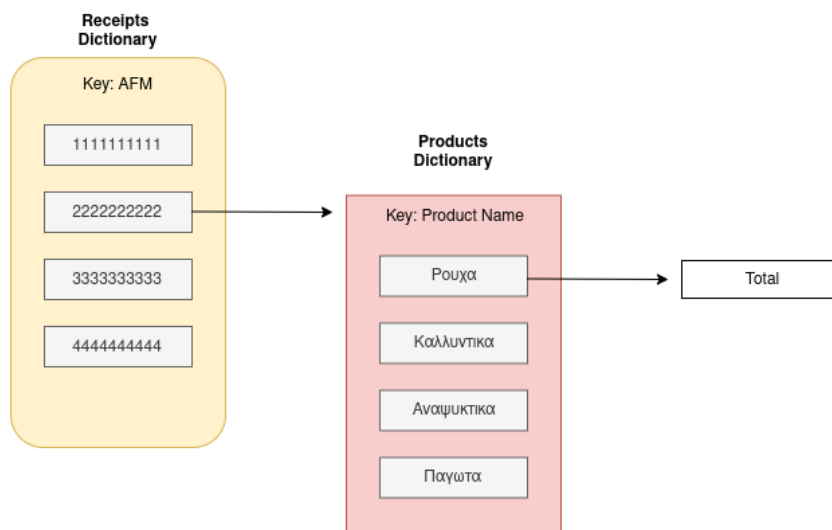


Figure 1: Data Structure for Receipts

Αφού έχουμε περιγράψει τον τρόπο που θα αποθηκεύουμε τις πληροφορίες, συνεχίζουμε με τον τρόπο που επιλέξαμε να διαβάσουμε το αρχείο και να κάνουμε τους ελέγχους ορθότητας. Όπως αναφέραμε θα έπρεπε να διαβάσουμε αναγκαστικά όλο το αρχείο, όμως έπρεπε να προσέξουμε να μην ξανά διαβάζουμε σημεία που ήδη είχαν επεξεργαστεί. Παρακάτω στον **Algorithm 1** φαίνεται ο ψευδοκώδικας της υλοποίησης μας.

Συγκεκριμένα αυτό που κάναμε είναι ότι διαβάζουμε όλο το αρχείο μέχρι να συναντήσουμε το End Of File, ή την πρώτη υποχρεωτική γραμμή για αποδεκτή απόδειξη (γραμμή μόνο με παύλες). Μόλις συναντήσουμε αυτήν την γραμμή βλέπουμε αν στην επόμενη υπάρχει το ΑΦΜ σε αποδεκτή μορφή και μόνο τότε ξεκινάμε να διαβάζουμε τα προϊόντα, να αποθηκεύουμε τις τιμές τους σε ένα προσωρινό dictionary, έως ότου συναντήσουμε την γραμμή που περιέχει το ΣΥΝΟΛΟ και στην συνέχεια πάλι το τέλος της απόδειξης (γραμμή μόνο με παύλες). Σε κάθε ενδιάμεσο στάδιο γίνονται έλεγχοι ορθότητας (περισσότερες πληροφορίες σχετικά με του ελέγχους υπάρχει στο **section - Υλοποίηση**

---

**Algorithm 1** File parser

---

```

1: procedure add_new_file(filename)
2:    $ln \leftarrow$  read first line
3:   while true do                                     ▷ Until EOF or first valid line for receipt
4:     if  $ln = EOF$  then Return
5:     else if  $ln$  is dash line then
6:        $ln \leftarrow$  read next line
7:       if  $ln$  is not an acceptable AFM line then continue           ▷ First loop
8:       products  $\leftarrow$  initialize dictionary
9:       while true do                                     ▷ For receipt's valid lines
10:         $ln \leftarrow$  read next line
11:        if  $ln$  is an acceptable product line then
12:          products  $\leftarrow$  save product or update existing products dictionary
13:        else if  $ln$  is an acceptable total line && products  $\neq \emptyset$  then
14:           $ln \leftarrow$  read next line
15:          if  $ln$  is dash line then
16:            receipts_dict  $\leftarrow$  save products or update value
17:            break                                           ▷ Go back to first loop
18:          else break                                         ▷ Go back to first loop
19:        else
20:           $ln \leftarrow$  read next line

```

---

### 3 Υλοποίηση

Listing 1: Menu

```
1 if __name__ == '__main__':
2     ... # Arguments parsing
3     receipts_dict = {}
4     while True:
5         print_menu()
6         try:
7             # Only 1, 2, 3 and 4 are valid -> 1.0, 2.00, two, 8, -1, +2, etc... are not!
8             user = str(input())
9         except Exception:
10            user = 0
11
12        if user == '1':
13            f_name = str(input("Give File name: "))
14            try:
15                add_new_file(f_name)
16            except Exception:
17                pass
18        elif user == '2':
19            give_statistics_for_product(str(input("Give a product name: ")).upper())
20        elif user == '3':
21            afm = str(input("Give an AFM: "))
22            if re.match(AFM, afm) is not None:
23                give_statistics_for_afm(afm)
24        elif user == '4':
25            exit(0)
```

Η παραπάνω συνάρτηση θεωρούμε ότι είναι πολύ απλή και θα κάνουμε μία γρήγορη εξήγηση στα καίρια σημεία. Αρχικά με τη χρήση `str()` για το `input` εξασφαλίζουμε ότι το `input` του χρήστη θα είναι μόνο 1,2,3 ή 4 διαφορετικά θα ξανατρέξει την τύπωση του μενού. Στην συνέχεια στην κλήση της `add_new_file` την κλείνουμε μέσα σε `try-except`, διότι όταν θα πάει να ανοίξει ένα αρχείο που δεν υπάρχει ή δεν έχει δικαίωμα ο χρήστης να το ανοίξει, αντί να σκάσει θα προχωρήσει και θα ξανατυπώσει το μενού. Τέλος πριν μπούμε στην συνάρτηση της 3ης επιλογής ελέγχουμε αν η είσοδος είναι τύπος ΑΦΜ (10 ψηφία), αν δεν είναι τότε εμφανίζουμε πάλι το μενού.

Listing 2: Give statistics for products

```
1 @calculate_time
2 def give_statistics_for_product(prod):
3     for afm in sorted(receipts_dict.keys()):
4         if prod in receipts_dict[afm]["products"]:
5             print(afm.__str__() + " " + str(format(receipts_dict[afm]["products"][prod], '.2f'))
6         )
```

Για την τύπωση στατιστικών στοιχείων για ένα δεδομένο προϊόν, αρχικά δουλεύουμε με ένα `loop` το οποίο τρέχει για όλα τα ΑΦΜ του `dictionary` μας, που όμως βρίσκονται μέσα σε μία σορταρισμένη λίστα. Εν συνεχεία μέσα απο το `receipts_dict[afm]["products"]` παίρνουμε ολόκληρο το nested Products dictionary όπως φαίνεται στο **Figure 1** και ελέγχουμε αν το όνομα του προϊόντος βρίσκεται μέσα στα `keys` του. Εφόσον υπάρχει τότε τυπώνουμε το ΑΦΜ και το `value` του `product` του nested dictionary, όπως μας δείχνει το σχεδιάγραμμα.

Listing 3: Give statistics for AFMs

```
1 @calculate_time
2 def give_statistics_for_afm(afm):
3     data = receipts_dict.get(str(afm))
4     if data is None:
5         return
6     data = data["products"]
7     for prods in sorted(data.keys()):
8         print(str(prods) + " " + str(format(data[prods], '.2f')))
```

Στην παρούσα συνάρτηση που καλείται όταν θέλουμε να τυπώσουμε στατιστικά στοιχεία για ένα δεδομένο ΑΦΜ, παίρνουμε το nested dictionary που περιέχει το products dictionary για το δεδομένο ΑΦΜ. Ακολούθως τυπώνουμε τα ζητούμενα με τον ίδιο τρόπο που ενεργήσαμε και στην παραπάνω συνάρτηση.

Listing 4: Add new file - open file

```

1 @calculate_time
2 def add_new_file(filename):
3     with open(filename, "r", encoding='utf-8') as file:
4         # Starting point of new file parsing
5         ln = file.readline()
6         while True:
7             #Listing 5: Add new file - Read Receipt

```

Listing 5: Add new file - Read Receipt

```

1 if not ln: # If END of file return
2     return
3 elif dash_pattern.match(ln): # Dash line PATTERN match
4     ln = file.readline().upper() # Read next line (must be AFM)
5     if afm_pattern.match(ln) is None: # AFM PATTERN check
6         continue # Wrong AFM PATTERN
7     afm = str(ln.strip().split(":").__getitem__(1).strip()) # Save AFM
8     products = {} # Initialize products dict
9     p_total = 0.0 # Initialize receipt total
10
11 # Starting point of new receipt parsing
12 while True: # For RECEIPT loop
13     ln = file.readline().upper() # Read next line (PROD or TOT)
14     if product_pattern.match(ln) is not None: # Product PATTERN check
15         name = str(ln.strip().split(":").__getitem__(0).strip())
16         spec = str(ln.strip().split(":").__getitem__(1))
17         amount = int(spec.strip().split().__getitem__(0).strip())
18         cost = float(spec.strip().split().__getitem__(1).strip())
19         total = float(spec.strip().split().__getitem__(2).strip())
20
21         # Checking if amount*cost per product equals product's total
22         if round((amount * cost), 2) != total:
23             break
24         p_total = round(p_total + total, 2) # Get the subtotal of the receipt products
25         if name in products:
26             products[name] = round(products[name] + total, 2)
27         else:
28             products[name] = total
29         continue
30     elif total_pattern.match(ln) is not None: # Total PATTERN check
31         rec_total = float(ln.strip().split("__getitem__(1).strip()) # Save total cost
32         # check if we have empty receipt or total != subtotal
33         if not bool(products) or p_total != float(rec_total):
34             break
35         ln = file.readline() # Read next line
36         if dash_pattern.match(ln): # Dash line
37             if afm in receipts_dict:
38                 update_afm_receipts({afm: {"products": products}}, afm)
39             else:
40                 receipts_dict.update({afm: {"products": products}})
41             break
42         else:
43             break
44     else:
45         break
46 # Read next line if this is not a dash line
47 else:
48     ln = file.readline()

```

Στους παραπάνω δύο πίνακες βλέπουμε την κυριότερη διαδικασία που έπρεπε να γίνει για το πρότζεκτ που δεν είναι άλλη από το διάβασμα του αρχείου και την αποθήκευση των δεδομένων στα προαναφερθέντα dictionaries. Το πρώτο κομμάτι γίνεται στο **Listing 4** και ουσιαστικά ανοίγει το αρχείο κρατώντας στην μεταβλητή `ln` το περιεχόμενο της 1ης σειράς του και στην συνέχεια ακολουθεί ένα `while` loop, που τρέχει μέχρι να σταματήσει από εσωτερική συνθήκη, και διαβάσει το αρχείο σειρά σειρά. Ο λόγος που έχουμε το `with open() as file` είναι γιατί με αυτόν τον τρόπο μόλις βγούμε από την συνάρτηση αυτομάτως κλείνει και το αρχείο μας. Για το δεύτερο κομμάτι τώρα, που γίνεται στο **Listing 5**, όπως φαίνεται ξεκινάει με δυο conditional statements το πρώτο είναι για τον έλεγχο End Of File, το δεύτερο είναι για το έλεγχο παρουσίας διαχωριστικού αποδείξεων, ενώ σε κάθε άλλη περίπτωση βλέπουμε να διαβάσει την επόμενη γραμμή του αρχείου και να συνεχίζει ξανά μέσα στο loop. Στην πιο αξιοσημείωτη περίπτωση που θα βρει dash lines (διαχωριστικό), αμέσως διαβάσει την επόμενη γραμμή και ελέγχει αν ταιριάζει με το regular expression που έχουμε δημιουργήσει `'^\s*AΦΜ\s*:\s*[0-9]{10}\s*$'` και σε αντίθετη περίπτωση με την εντολή `continue` συνεχίζει στο loop και τσεκάρει το περιεχόμενο της σειράς. Σε περίπτωση που έχουμε αποδεκτό ΑΦΜ στη σειρά 7 παίρνουμε, μέσω `split` και `strip` για να κόψουμε τα κενά, το ΑΦΜ και συνεχίζουμε στο διάβασμα των προϊόντων (σειρές 12-46). Η ανάγνωση των προϊόντων γίνεται μέσα σε ένα `while` loop που διαβάζουμε στην αρχή μία σειρά και ελέγχουμε αν η σειρά περιέχει μοτίβο προϊόντος (πρώτο conditional statement) ή μοτίβο συνόλου (δεύτερο conditional statement), πάλι με την χρήση regular expressions που παραλείπονται από την αναφορά λόγω του εκτενούς μήκους τους. Σε κάθε άλλη περίπτωση βλέπουμε πως σταματάει το εσωτερικό loop και συνεχίζει για αναζήτηση και ανάγνωση άλλης απόδειξης. Σημαντικό εδώ είναι να αναφέρουμε ότι για εξοικονόμηση μνήμης δεν αποθηκεύουμε τα προϊόντα στο βασικό μας λεξικό αλλά σε ένα temporary, για να αποφεύγουμε τις πολλές προσβάσεις και περιττή δέσμευση μνήμης στην περίπτωση που έχουμε σφάλμα στο τελικό αναγνωριστικό της απόδειξης ή σε κάποια άλλη περίπτωση που η απόδειξή μας θα πρέπει να είναι άκυρη. Η αποθήκευση στο temporary dictionary φαίνεται στις σειρές 25-29. Τελευταίο βήμα έχουμε όταν βρει το ΣΥΝΟΛΟ της απόδειξης ελέγχει αν υπάρχει τουλάχιστον ένα προϊόν στην απόδειξη και αν το τελικό σύνολο είναι όντως ίσο με αυτό των καταχωρημένων προϊόντων και έτσι αν είναι επιτυχής οι έλεγχοι προχωράει στην ανάγνωση της επόμενης σειράς με σκοπό να δούμε αν είναι dash lines καθώς αν δεν είναι πρέπει να προχωρήσει το loop μέχρι να βρει έγκυρη έναρξη νέας απόδειξης διαφορετικά στις γραμμές 36-41 έχουμε την εισαγωγή της απόδειξης στο κύριο λεξικό μας. Ο λόγος που βλέπουμε αν υπάρχει ήδη το ΑΦΜ στο λεξικό, είναι διότι σε αυτήν την περίπτωση πρέπει να γίνει διαφορετικό update από εκείνο που μας δίνει η συνάρτηση `update` των dictionaries, επειδή θέλουμε να κάνουμε update και το nested dictionary, και είναι αυτό που παρουσιάζεται στο **Listing 6**. Τέλος για να αποφύγουμε τυχόν προβλήματα με upper-lower cases αποφασίσαμε μόλις διαβάζουμε μία σειρά αρχείου να την μετατρέπουμε σε upper-case και να εισάγουμε τα δεδομένα μας με αυτόν τον τρόπο στα dictionaries, όπως και φαίνεται στην σειρά 4 του **Listing 5**

Listing 6: Update AFM Receipts

```

1
2 def update_afm_receipts(new_receipt, afm):
3     for key in new_receipt[afm]["products"]:
4         if key in receipts_dict[afm]["products"]:
5             receipts_dict[afm]["products"][key] = \
6                 round((receipts_dict[afm]["products"][key]+new_receipt[afm]["products"][key]), 2)
7         else:
8             receipts_dict[afm]["products"][key] = new_receipt[afm]["products"][key]

```

Τέλος, σε αυτήν την συνάρτηση ανανέωσης του dictionary μας γίνεται η εξής διαδικασία. Για όλα τα προϊόντα που περιέχονται μέσα στο λεξικό που θέλουμε να προσθέσουμε, ελέγχουμε αν υπάρχουν μέσα στο κύριο μας για το δεδομένο ΑΦΜ και αν ναι τότε κάνουμε update το value του προϊόντος. Σε διαφορετική περίπτωση κάνουμε απλή εισαγωγή στα προϊόντα του δεδομένου ΑΦΜ.

## 4 Έλεγχος Λειτουργίας

Προκειμένου να αναπτύξουμε και να ελέγξουμε το πρόγραμμα μας χρησιμοποιήσαμε τόσο τον integrated debugger του IDE που χρησιμοποιήσαμε. Προσθέσαμε όμως διάφορα DEBUG messages προκειμένου να μπορούμε και κατά την ορθή εκτέλεση να εμφανίσουμε γρήγορα σε κάθε περιβάλλον διάφορες σημαντικές πληροφορίες. Σε περίπτωση που το επιθυμείτε τρέχοντας την παρακάτω εντολή μπορείτε να δείτε το help menu του προγράμματος με όλα τα δυνατά flag που μπορούν να δοθούν σε αυτό.

```
# python3 computeSales.py -h
```

Επίσης ξεινήσαμε με το να κάνουμε απλά unit tests στα μεμονωμένα modules και στην συνέχεια να ελέγχουμε την συνολική λειτουργία σε λιγότερο τετριμμένες εισόδους.

Προκειμένου να ελέγξουμε την ορθή λειτουργία του κώδικα για πολύ μεγάλα αρχεία, δημιουργήσαμε το python script που φαίνεται στο **Listing 7**, αυτό που πρέπει να επισημάνουμε είναι ότι το AFMS και το PRODUCTS είναι δύο λίστες με περίπου 100 και 250 διαφορετικές τιμές το καθένα.

Listing 7: Create random files with valid receipts

```
1 import sys
2 import random
3 import tqdm
4
5 AFMS = [...] # Is just a list of valid AFMS
6 PRODUCTS = [...] # Is just a list of valid product names
7
8 # Code import tool that used does not support UTF-8, in python script these are in greek
9 AFM_V = 'AFM'
10 TOTAL_V = 'SYNOLO'
11
12 # Number of receipts to create
13 receipts = sys.argv[1]
14
15 for i in tqdm.tqdm(range(1, int(receipts)+1)): # tqdm creates a visual status bar
16     afm = AFMS[random.randrange(0, len(AFMS))]
17     line = "-"*random.randrange(1, 40) + '\n' + ' '*random.randrange(0, 10) + AFM_V + ' '*random.
18         randrange(0, 10) + ':' + ' '*random.randrange(0, 10) + str(afm)
19     print(line)
20     total = 0.0
21
22     max_products = random.randrange(1, 60)
23     for j in range(1, int(max_products)+1):
24         prod = PRODUCTS[random.randrange(0, len(PRODUCTS))]
25         quantity = random.randrange(1, 20)
26         price = round(random.uniform(0.20, 15.99), 2)
27         sum = round(quantity*price, 2)
28         total = round(total+sum, 2)
29         line = ' '*random.randrange(0, 10) + str(prod) + ' '*random.randrange(0, 10) + ':' + ' '*
30             random.randrange(0, 10) + str(quantity) + ' '*random.randrange(1, 10) + str(price) + ' '*random.
31             randrange(1, 10) + str(sum) + ' '*random.randrange(0, 10)
32         print(str(line))
33
34     line = ' '*random.randrange(0, 10) + TOTAL_V + ' '*random.randrange(0, 10) + ':' + ' '*random.
35         randrange(0, 10) + str(total)
36     print(line)
37     print('-'*random.randrange(1, 40))
```

### 4.1 Simple input

Όπως αναφέραμε πρώτο βήμα ήταν να δημιουργήσουμε ένα πολύ απλό αρχείο και να δούμε ότι δουλεύει όπως θα έπρεπε. Σε αυτό το παράδειγμα κάνουμε την χρήση κάποιων από των debug messages για να παρουσιάσουμε ότι ο κώδικας κάνει αυτό που σχεδιάσαμε.



```

1 -----
2 ΑΦΜ: 111111111
3 ΑΛΕΥΡΙ_ΣΙΚΑΛΗΣ : 6 7.00 42.00
4 ΚΡΑΣΙΡΟΖΕ : 15 8.41 126.15
5 ΣΥΝΟΛΟ: 168.15
6 -----
7 ΑΦΜ: 222222222
8 ΜΠΟΥΚΙΕΣ_ΚΡΙΘΙΝΕΣ : 18 0.65 11.7
9 ΠΙΠΕΡΟΠΙΤΑ : 7 3.4 23.8
10 ΚΡΑΣΙΡΟΖΕ: 2 4.32 8.64
11 ΠΙΠΕΡΟΠΙΤΑ : 3 2 6
12 ΣΥΝΟΛΟ: 50.14
13 -----

```

Figure 2: test1-simple data

```

cs@cs-l: ~/Desktop/Projects/0sDev/project-2 [master] python3 computeSales.py -dpic
Namespace(DEBUG=True, ERROR=False, ITEMS_DICTIONARY=True, MEMORY=False, PRODUCTS=True, TIME=False, UPD
ATE=False, VALID=True)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
1
Give File name: test1-simple
##### ADD NEW FILE #####
{AFM:2} - New Receipt, AFM: 111111111
{PRODUCT-INFO:3} - Name: ΑΛΕΥΡΙ_ΣΙΚΑΛΗΣ | Quantity: 6 | Cost: 7.0 | Sum: 42.0
{PRODUCT-INFO:4} - Name: ΚΡΑΣΙΡΟΖΕ | Quantity: 15 | Cost: 8.41 | Sum: 126.15
{TOTAL:5} - Total: 168.15
{VALID-RECEIPT !!!}
-----
{AFM:7} - New Receipt, AFM: 222222222
{PRODUCT-INFO:8} - Name: ΜΠΟΥΚΙΕΣ_ΚΡΙΘΙΝΕΣ | Quantity: 18 | Cost: 0.65 | Sum: 11.7
{PRODUCT-INFO:9} - Name: ΠΙΠΕΡΟΠΙΤΑ | Quantity: 7 | Cost: 3.4 | Sum: 23.8
{PRODUCT-INFO:10} - Name: ΚΡΑΣΙΡΟΖΕ | Quantity: 2 | Cost: 4.32 | Sum: 8.64
{PRODUCT-INFO:11} - Name: ΠΙΠΕΡΟΠΙΤΑ | Quantity: 3 | Cost: 2.0 | Sum: 6.0
{TOTAL:12} - Total: 50.14
{VALID-RECEIPT !!!}
##### EOF #####
{RECEIPTS-DICTIONARY} -----
1) AFM: 111111111
- ΑΛΕΥΡΙ_ΣΙΚΑΛΗΣ 42.0
- ΚΡΑΣΙΡΟΖΕ 126.15

2) AFM: 222222222
- ΚΡΑΣΙΡΟΖΕ 8.64
- ΜΠΟΥΚΙΕΣ_ΚΡΙΘΙΝΕΣ 11.7
- ΠΙΠΕΡΟΠΙΤΑ 23.8

{TIME} Function: add_new_file | Time: 0.001231432 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
2
Give a product name: ΚρασιΡοζε
111111111 126.15
222222222 8.64
-----
{TIME} Function: give_statistics_for_product | Time: 0.0001206398 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
3
Give an AFM: 111111111
ΑΛΕΥΡΙ_ΣΙΚΑΛΗΣ 42.00
ΚΡΑΣΙΡΟΖΕ 126.15
-----
{TIME} Function: give_statistics_for_afm | Time: 9.36985e-05 (sec)

```

Figure 3: test1-simple output

Αρχικά τρέχουμε το πρόγραμμα με ενεργοποιημένο το debug (περισσότερες πληροφορίες για τον χρόνο εκτέλεσης, κλπ.), το correct receipt, το products show και το items show (receipts' dictionary) flags. Αυτό σημαίνει ότι θα εμφανίζεται κάθε ενέργεια που γίνεται στον κώδικα όπως επίσης μόλις γίνει το insert ενός αρχείου στη μνήμη τότε θα εμφανίσει και την κατάσταση του dictionary. Μέσα στα άγκιστρα εμφανίζεται το είδος της ενέργειας (κάποιο error case ή έλεγχος που γίνεται και μετά την άνω-κάτω τελεία η γραμμή για την οποία αντιστοιχεί). Τέλος εμφανίζει τον χρόνο που χρειάστηκε για να πραγματοποιηθεί η συγκεκριμένη ενέργεια σε κάθε περίπτωση.

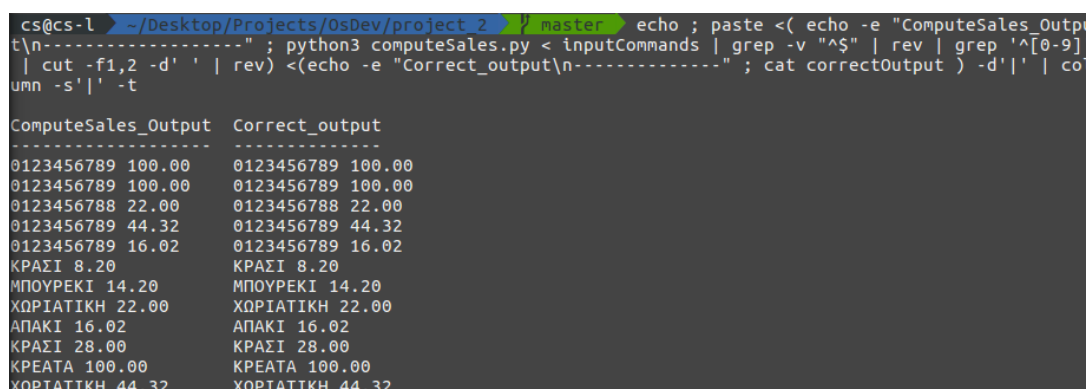
Ξεκινώντας με την λειτουργικότητα, βλέπουμε ότι αρχικά αφού έχει βρει ένα dash line και στο επόμενο είναι ένα valid ΑΦΜ θα το διαβάσει, μετά θα διαβάσει ένα-ένα τα προϊόντα που υπάρχουν σε αυτό, μέχρι να βρει το σύνολο και μετά πάλι το dash line. Το ίδιο θα επαναλάβει για κάθε receipt μέσα στο αρχείο. Αφού διαβάσει όλο το αρχείο μας εμφανίζει τα περιεχόμενα της δομής που χρησιμοποιήσαμε για να αποθηκεύουμε τις πληροφορίες και επαληθεύουμε ότι είναι αυτές που θα έπρεπε. Επόμενη ενέργεια είναι για το προϊόν ΚρασιΡοζε να μας εμφανίσει όλα τα ΑΦΜ στα οποία υπάρχει και για το καθένα το σύνολο των πωλήσεων (βλέπουμε ότι δεν είναι case sensi-

tive τα ονόματα επίσης). Ενώ τέλος για το πρώτο ΑΦΜ (111111111) εμφανίζει όλα τα προϊόντα και το σύνολο πωλήσεων για το καθένα. Ακόμα σε αυτό το παράδειγμα επαληθεύουμε ότι δεν παρουσιάζεται πρόβλημα αν σε μία απόδειξη εμφανίζεται με διαφορετική τιμή το ίδιο προϊόν σε άλλο σημείο.

**Σημείωση:** Οι ενέργειες του menu διαχωρίζονται η μία με την άλλη με τις γραμμές που περιέχουν underscore.

## 4.2 Given sample

Αφού είχαμε κάνει αυτό τρέξαμε τον κώδικα για το απλό παράδειγμα το οποίο μας δόθηκε μέσω του courses. Με αυτό το τρόπο μπορούσαμε να ελέγξουμε ότι γίνονται αποδεκτά πολλαπλά αρχεία και συνυπολογίζονται στα αποτελέσματα, όταν δεν έχουν πρόβλημα τα receipts. Στο παρακάτω **figure-4** εμφανίζεται το αποτέλεσμα. Να σημειωθεί ότι απλά έχουμε τρέξει την εντολή paste ώστε να εμφανίσουμε στα αριστερά το αποτέλεσμα του κώδικά μας και δεξιά το περιεχόμενο του αρχείου correctOutput, ενώ για να είναι πιο διακριτικός ο διαχωρισμός κάναμε pipe το αποτέλεσμα στο column -t για να τα εμφανίσει στοιχισμένα.



```
cs@cs-l ~/Desktop/Projects/DsDev/project_2 % master echo ; paste <( echo -e "ComputeSales_Output\n-----"; python3 computeSales.py < inputCommands | grep -v "^$" | rev | grep '[0-9]'\n | cut -f1,2 -d' ' | rev) <(echo -e "Correct_output\n-----"; cat correctOutput ) -d'|' | col\numn -s'|' -t
```

ComputeSales_Output	Correct_output
0123456789 100.00	0123456789 100.00
0123456789 100.00	0123456789 100.00
0123456788 22.00	0123456788 22.00
0123456789 44.32	0123456789 44.32
0123456789 16.02	0123456789 16.02
ΚΡΑΣΙ 8.20	ΚΡΑΣΙ 8.20
ΜΠΟΥΡΕΚΙ 14.20	ΜΠΟΥΡΕΚΙ 14.20
ΧΩΡΙΑΤΙΚΗ 22.00	ΧΩΡΙΑΤΙΚΗ 22.00
ΑΠΑΚΙ 16.02	ΑΠΑΚΙ 16.02
ΚΡΑΣΙ 28.00	ΚΡΑΣΙ 28.00
ΚΡΕΑΤΑ 100.00	ΚΡΕΑΤΑ 100.00
ΧΩΡΙΑΤΙΚΗ 44.32	ΧΩΡΙΑΤΙΚΗ 44.32

Figure 4: test2-simple output

## 4.3 Sample with errors

Επόμενο βήμα ήταν να δημιουργήσουμε ένα αρχείο το οποίο θα περιέχει λανθασμένες αποδείξεις, ώστε να επαληθεύσουμε ότι αναγνωρίζει τα λάθη και δεν θα πρέπει να τις συμπεριλαμβάνει. Τα λάθη που δοκιμάζουμε στο συγκεκριμένο αρχείο είναι τα εξής:

- Το dash line να περιλαμβάνει και άλλους χαρακτήρες πέρα του dash - (σειρά 8 μεταξύ 2ης και 3ης απόδειξης)
- Να μην ξεκινάει μία απόδειξη με dash line - (1η και 3η απόδειξη)
- Να μην τελειώνει μία απόδειξη με dash line - (2η και 13η απόδειξη)
- Το ΑΦΜ να ΜΗΝ είναι ένας αριθμός 10 ψηφίων - (4η απόδειξη)
- Να μην εμφανίζεται στην αρχή μίας απόδειξης το ΑΦΜ - (5η απόδειξη)
- Τα προϊόντα να περιέχουν κενό χαρακτήρα (αναφέρθηκε ότι δεν θα έπρεπε να περιλαμβάνεται) - (6η απόδειξη)
- Η ποσότητα αγοράς ενός αντικειμένου να είναι διαφορετικό από φυσικό αριθμό. - (7η απόδειξη)
- Το συνολικό κόστος ενός προϊόντος να είναι λανθασμένα υπολογισμένο  $\neq$  ποσότητα \* τιμή ανά μονάδα - (8η απόδειξη)
- Να μην υπάρχει κανένα προϊόν σε μία απόδειξη - (9η απόδειξη)
- Να μην εμφανίζεται στην τελευταία γραμμή της απόδειξης η γραμμή με το ΣΥΝΟΛΟ - (10η απόδειξη)
- Να είναι λανθασμένα υπολογισμένο το συνολικό κόστος (11η απόδειξη)

Για να δοθεί έμφαση ότι σε καμία από τις περιπτώσεις που δεν πρέπει δεν αποθηκεύουμε πληροφορία όλες οι αποδείξεις που είναι λανθασμένες έχουν ένα προϊόν με το ίδιο όνομα. Ενώ έχει προστεθεί και μία ορθή απόδειξη. Και σε αυτήν την περίπτωση χρησιμοποιούμε τα debug messages για να δείξουμε και στην πράξη αυτό που περιγράφουμε. Σημαντικό σημείο είναι στο τέλος όταν εμφανίζονται τα περιεχόμενα της δομής δεδομένων (dictionary) που αποθηκεύουμε όλες τις πληροφορίες ότι περιέχεται ΜΟΝΟ πληροφορίες από την αποδεκτή απόδειξη.

```

1  ΑΦΜ: 0000000001
2  ΠΡΟΙΟΝ: 1  2.00  2.00
3  ΣΥΝΟΛΟ: 2.00
4  -
5  ΑΦΜ: 0000000002
6  ΠΡΟΙΟΝ: 1  2.00  2.00
7  ΣΥΝΟΛΟ: 2.00
8  -----
9  ΑΦΜ: 0000000003
10 ΠΡΟΙΟΝ: 1  2.00  2.00
11 ΣΥΝΟΛΟ: 2.00
12 -----
13 ΑΦΜ: 0000000004
14 ΠΡΟΙΟΝ: 1  2.00  2.00
15 ΣΥΝΟΛΟ: 2.00
16 -----
17 ΠΡΟΙΟΝ: 1  2.00  2.00
18 ΣΥΝΟΛΟ: 2.00
19 -----
20 ΑΦΜ: 0000000006
21 ΠΡΟΙΟΝ: 1  2.00  2.00
22 ΣΥΝΟΛΟ: 2.00
23 -----
24 ΑΦΜ: 0000000007
25 ΠΡΟΙΟΝ: 1.1 2.00  2.20
26 ΣΥΝΟΛΟ: 2.20
27 -----
28 ΑΦΜ: 0000000008
29 ΠΡΟΙΟΝ: 1  2.00  100.00
30 ΣΥΝΟΛΟ: 100.00
31 -----
32 ΑΦΜ: 0000000009
33 ΣΥΝΟΛΟ: 0.00
34 -----
35 ΑΦΜ: 0000000010
36 ΠΡΟΙΟΝ: 1  2.00  2.00
37 -----
38 ΑΦΜ: 0000000011
39 ΠΡΟΙΟΝ: 1  2.00  2.00
40 ΣΥΝΟΛΟ: 100.00
41 --
42 ΑΦΜ: 0000000012
43 ΑΠΟΔΕΚΤΟ : 1  2.00  2.00
44 ΣΥΝΟΛΟ: 2.00
45 -----
46 ΑΦΜ: 0000000013
47 ΠΡΟΙΟΝ: 1  2.00  2.00
48 ΣΥΝΟΛΟ: 2.00
49

```

Figure 5: test3-errors data



```

cs@cs-l ~/Desktop/Projects/OsDev/project 2 master echo -n "" > test4-bigsingle ; python3 pr.p
y 1000000 > test4-bigsingle ; cat test4-bigsingle | wc -l
8000001
cs@cs-l ~/Desktop/Projects/OsDev/project 2 master python3 computeSales.py -tei
Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
1
Give File name: test4-bigsingle
----- {RECEIPTS-DICTIONARY} -----
1) AFM: 1111111111
  - PROION_1      1000000.0

2) AFM: 2222222222
  - PROION_2      2000000.0
-----
{TIME} Function: add_new_file | Time: 25.1236171722 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
2
Give a product name: PROION_2
2222222222 2000000.00
-----
{TIME} Function: give_statistics_for_product | Time: 0.0001094341 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
3
Give an AFM: 1111111111
PROION_1 1000000.00
-----
{TIME} Function: give_statistics_for_afm | Time: 6.50883e-05 (sec)

```

Figure 7: test4, two receipts \* 1.000.000 - Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

```

cs@cs-l ~/Desktop/Projects/OsDev/project 2 master cat test5 | wc -l
32989609
cs@cs-l ~/Desktop/Projects/OsDev/project 2 master head -n 5 test5
-----
      ΑΦΜ      : 7592514611
ΓΚΟΤΖΙ      : 8      5.27 42.16
ΣΑΛΑΜΙ      :19 4.93      93.67
ΠΑΤΑΤΕΣ_Τρίπολης : 15      15.0      225.0
cs@cs-l ~/Desktop/Projects/OsDev/project 2 master python3 computeSales.py -te
Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
1
Give File name: test5
-----
{TIME} Function: add_new_file | Time: 247.6994609833 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
2
Give a product name: σαλαμι
0248022159 62622.94
0254473700 63910.63

9646941745 60058.92
9757215606 66352.94
9784327407 59385.65
9886170933 60847.29
9925080196 61790.67
-----
{TIME} Function: give_statistics_for_product | Time: 0.0023858547 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
3
Give an AFM: 7592514611
ΑΓΟΥΡΕΛΑΙΟ_ΕΞΤΡΑ ΠΑΡΘΕΝΟ 53957.71
ΑΛΑΤΙ_ΑΝΕΠΕΞΕΡΓΙΑΣΤΟ 60930.93
ΑΛΑΤΙ_ΗΜΙΧΟΝΔΡΟ 66947.42
ΑΛΑΤΙ_ΘΑΛΑΣΣΙΝΟ 59570.37

ΧΟΡΤΟΠΙΤΑ 66603.75
ΧΟΡΤΟΠΙΤΑ_ΝΗΣΙΤΙΣΙΜΗ 65974.54
ΧΥΛΟΠΙΤΕΣ_ΠΙΠΕΡΙΑ_Κ_ΝΤΟΜΑΤΑ 62125.16
ΧΥΜΟΣ_ΜΕΙΓΜΑ_ΤΖΙΝΤΖΕΡ 62367.97
ΧΥΜΟΣ_ΡΟΔΙ 64201.76
-----
{TIME} Function: give_statistics_for_afm | Time: 0.0039930344 (sec)

```

Figure 8: test5, 1.000.000 random receipts - Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

```

cs@cs-l ~/Desktop/Projects/OsDev/project_2 master cat test5 | wc -l
32989609
cs@cs-l ~/Desktop/Projects/OsDev/project_2 master cat test4-bigsingle | wc -l
8000001
cs@cs-l ~/Desktop/Projects/OsDev/project_2 master python3 computeSales.py -te

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
1
Give File name: test5
-----
{TIME} Function: add_new_file | Time: 255.7208478451 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
1
Give File name: test4-bigsingle
-----
{TIME} Function: add_new_file | Time: 26.4209268093 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
2
Give a product name: Προιον_1
111111111 1000000.00
-----
{TIME} Function: give_statistics_for_product | Time: 0.0004708767 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
3
Give an AFM: 2222222222
ΠΡΟΙΟΝ_2 2000000.00
-----
{TIME} Function: give_statistics_for_afm | Time: 6.77109e-05 (sec)

Give your preference: (1: read new input file, 2: print statistics for a specific product, 3: print st
atistics for a specific AFM, 4: exit the program)
2
Give a product name: ΑΛΑΤΙ_θαλασσινό
0248022159 59568.05
0254473700 63033.18
0265979758 58599.73
0478451093 63410.42
0497620156 66962.11

```

Figure 9: test6, compine test4 and test5 - Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

Από τα παραπάνω παραδείγματα (παρόλο που δεν δίνεται ακριβές screenshot) μπορούμε να επαληθεύσουμε επίσης ότι για το ερώτημα 2ο τα ΑΦΜ είναι ταξινομημένα κατά αύξων αριθμητικά, ενώ για το 3ο ότι τα προϊόντα είναι ταξινομημένα κατά αύξων αλφαβητικά.