ΠΛΗ211
Εργαλεία Ανάπτυξης Λογισμικού και Προγραμματισμός Συστημάτων
Αναφορά Δεύτερης Άσκησης

Παντουράκης Μιχαήλ ΑΜ 2015030185
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Ημερομηνία Παράδοσης: 30 Νοεμβρίου 2019

Επισκόπηση Δομής Υλοποίησης computeSales.py

Η παρούσα αναφορά περιγράφει την συνοδευόμενη υλοποίηση του προγράμματος computeSales.py, το οποίο (με επιλογή του χρήστη) διαβάζει αρχεία αποδείξεων με συγκεκριμένη δομή, και εκτυπώνει στατιστικά για τις συνολικές πωλήσεις ανά πελάτη (με κλειδί τον ΑΦΜ του) και ανά προϊόν (με κλειδί το όνομά του).

Στο Σχήμα 1 παραθέτω το διάγραμμα κλάσεων του προγράμματος. Πιο συγκεκριμένα, το computeSales.py περιέχει δύο κλάσεις, τις Receipt και ReceiptEntry, οι οποίες μοντελοποιούν τα δεδομένα των αποδείξεων που διαβάζονται από τα αρχεία εισόδου. Αντικείμενα των κλάσεων-μοντέλων δημιουργούνται από την κλάση ReceiptParser, η οποία και ελέγχει το κείμενο εισόδου για την ορθότητα του. Τα αντικείμενα της Receipt χρησιμοποιούνται στη συνέχεια από την StatsHandler, η οποία ενημερώνει και διατηρεί τις απαραίτητες δομές δεδομένων για τα ζητούμενα στατιστικά πωλήσεων των επιλογών 2 και 3. Τέλος, η κλάση MenuHandler χρησιμοποιεί τις δύο προαναφερθείσες controller κλάσεις, αναλαμβάνοντας την πλοήγηση του χρήστη στο μενού και την κλήση των επιλεγμένων κάθε φορά λειτουργιών. Στις επόμενες ενότητες εξηγώ και αιτιολογώ με περισσότερη λεπτομέρεια τη δομή των κλάσεων αυτών.

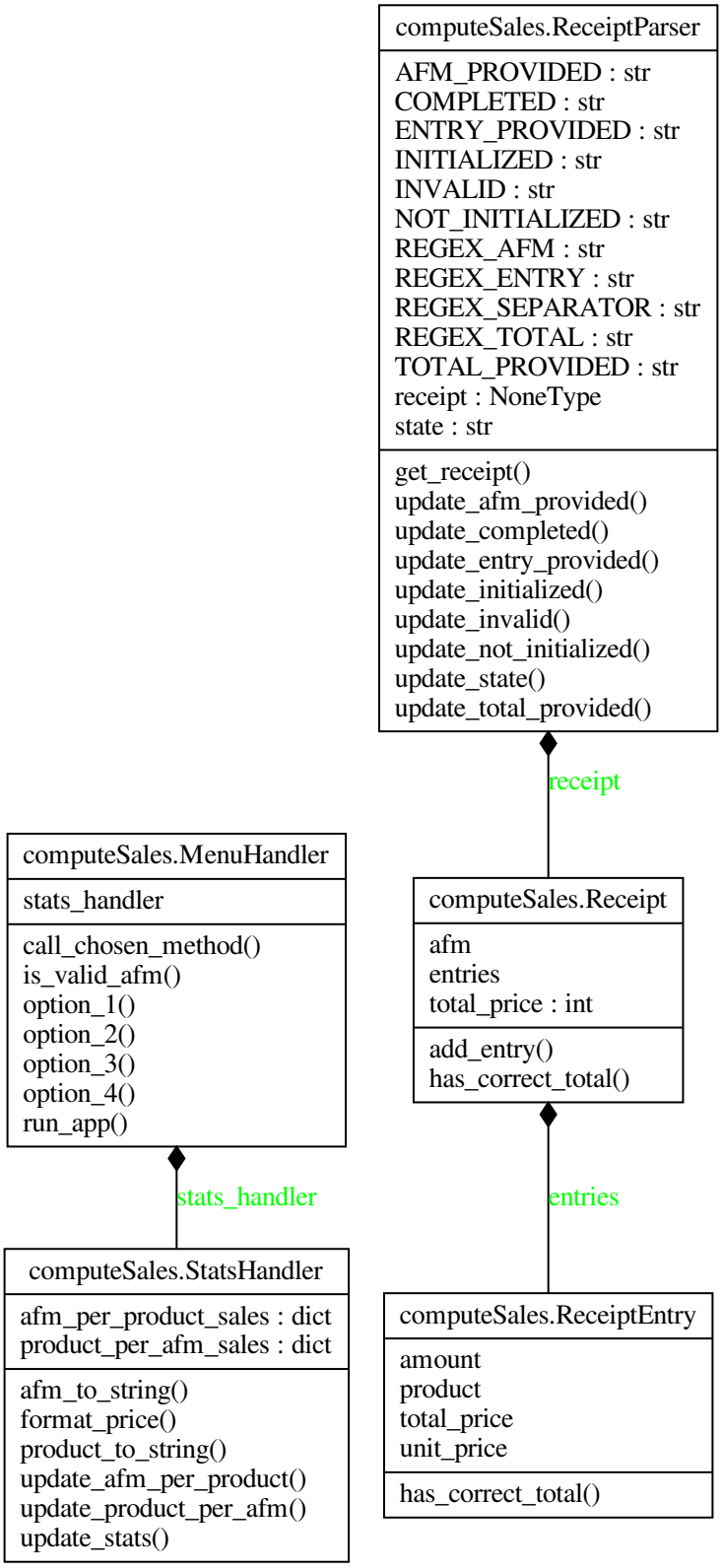
Κεντρικό Μενού

Το πρόγραμμα εκκινεί με τη συνάρτηση run_app() της MenuHandler. Αυτή η συνάρτηση εκτελεί επαναληπτικά την ανάγνωση της επιλογής εισόδου από το χρήστη. Η αριθμητική είσοδος του χρήστη δίνεται στην call_chosen_method, η οποία και την χρησιμοποιεί για να καλέσει δυναμικά την αντίστοιχη συνάρτηση (option_1, option_2, option_3, option_4). Η επαναληπτική διαδικασία του μενού (και το ίδιο το πρόγραμμα) σταματάει όταν καλεστεί η option_4.

Στις επιλογές 2 και 3 ο χρήστης πληκτρολογεί το όνομα του προϊόντος ή το ΑΦΜ του οποίου το σύνολο πωλήσεων επιθυμεί να δει. Στην πρώτη περίπτωση η είσοδος του χρήστη κανονικοποιείται (σε κεφαλαία), ενώ στη δεύτερη περίπτωση η is_valid_afm() παρέχει ένα στοιχειώδη έλεγχο εγκυρότητας του ΑΦΜ (θετικός ακέραιος 10 ψηφίων). Στην επόμενη ενότητα αναπτύσσω λεπτομερώς την υλοποίηση της επιλογής 1, μαζί με τα πλεονεκτήματα και μειονεκτήματα των επιλογών μου στη χρήση μνήμης, χρόνο εκτέλεσης και αναγνωσιμότητα/ευελιξία του κώδικα.

Ανάγνωση και Έλεγχος Ορθότητας Αποδείξεων

Η συνάρτηση option_1 αναλαμβάνει την ροή του προγράμματος κατά την ανάγνωση των αρχείων εισόδου. Η ανάγνωση γίνεται γραμμή προς γραμμή, και επαναληπτικά για κάθε γραμμή ελέγχεται: 1) η εγκυρότητα της εισόδου ως απόδειξη, 2) η επιστροφή αντικειμένου Receipt αν αυτή είναι έγκυρη (χωρίς σφάλματα που παραβιάζουν την εκφώνηση της άσκησης), και 3) η ενημέρωση των στατιστικών που διατηρεί η StatsHandler.



Σχήμα 1: Διάγραμμα κλάσεων του προγράμματος computeSales.py (δημιουργήθηκε με Pyreverse).

Τις διεργασίες αυτές περιβάλει ένα try-except που φροντίζει να επιστρέφει τη ροή στο αρχικό μενού σε περιπτώσεις εσφαλμένου ονόματος αρχείου (FileNotFoundError), εσφαλμένου τύπου αρχείου (UnicodeDecodeError) ή εξάντλησης της διαθέσιμης μνήμης (MemoryError). Όσον αφορά τη μνήμη κατά τη διάρκεια της ανάγνωσης, η ανάγνωση του αρχείου γραμμή-γραμμή εξασφαλίζει ότι όσο μεγάλο και αν είναι το αρχείο η built-in συνάρτηση open δεν θα συναντήσει προβλήματα μνήμης, παρά μόνο στην ακραία περίπτωση όπου το μέγεθος μίας γραμμής ξεπερνάει τη συνολική διαθέσιμη μνήμη.

Όπως ανέφερα και στην πρώτη ενότητα, η (ReceiptParser) παρέχει όλη τη λειτουργικότητα ελέγχου και δημιουργίας των αντικειμένων Receipt. Πιο συγκεκριμένα, το Σχήμα 2 απεικονίζει το διάγραμμα καταστάσεων του ελέγχου εγκυρότητας μίας απόδειξης. Εκκινώντας από την κατάσταση Not Initialized για ένα νέο αρχείο, δημιουργείται σταδιακά ένα αντικείμενο receipt, περιέχοντας όσα receipt_entries έχουν δωθεί. Στα βέλη φαίνονται οι συνθήκες μετάβασης για το string εισόδου και τα ποσά που περιλαμβάνουν. Για κάθε κατάσταση έχει υλοποιηθεί μία συνάρτησης ανανέωσης της κατάστασης (με είσοδο μία γραμμή του αρχείου εισόδου). Η συνάρτηση update_state που καλείται για κάθε γραμμή, καθορίζει δυναμικά με βάση την τωρινή κατάσταση ποια συνάρτηση update θα καλεστεί.

Για την εγκυρότητα της γραμμής εισόδου, χρησιμοποιούνται έλεγχοι με regular expressions, με τη χρήση του built-in πακέτου re. Για ευκολία στην κατανόησή τους, έχουν οριστεί σαν σταθερές REGEX_SEPARATOR, REGEX_AFM, REGEX_ENTRY και REGEX_TOTAL. Σημειωτέον ότι - σύμφωνα με τα δοσμένα παραδείγματα - στα ποσά υποθέτω πάντα την ύπαρξη δύο δεκαδικών ψηφίων, ενώ πεπερασμένος αριθμός κενών γίνεται δεκτός μόνο ανάμεσα στις λέξεις, και όχι στην αρχή και τέλος κάθε γραμμής. Επιπλέον, έγκυρα ονόματα θεωρούνται όλα όσα αποτελούνται από αλφαριθμητικούς χαρακτήρες και _, ενώ δεν γίνεται κάποια διαχείριση για τους τόνους μιας και δεν ζητείται στην άσκηση (έτσι π.χ. τα ονόματα 'ποικιλία' και 'ποικιλια' θεωρούνται διαφορετικά).

Έλεγχοι επίσης πραγματοποιούνται για την ορθότητα των συνόλων που περιέχει μία απόδειξη: είτε για το γινόμενο τιμής μονάδας και αριθμού προϊόντων με την συνάρτηση ReceiptEntry.has_correct_total(), είτε για το συνολικό άθροισμα που περιέχει μία απόδειξη με την συνάρτηση Receipt.has_correct_total(). Στην περίπτωση που αυτοί οι έλεγχοι είναι λανθασμένοι, η ReceiptParser καταλήγει στην κατάσταση Invalid.

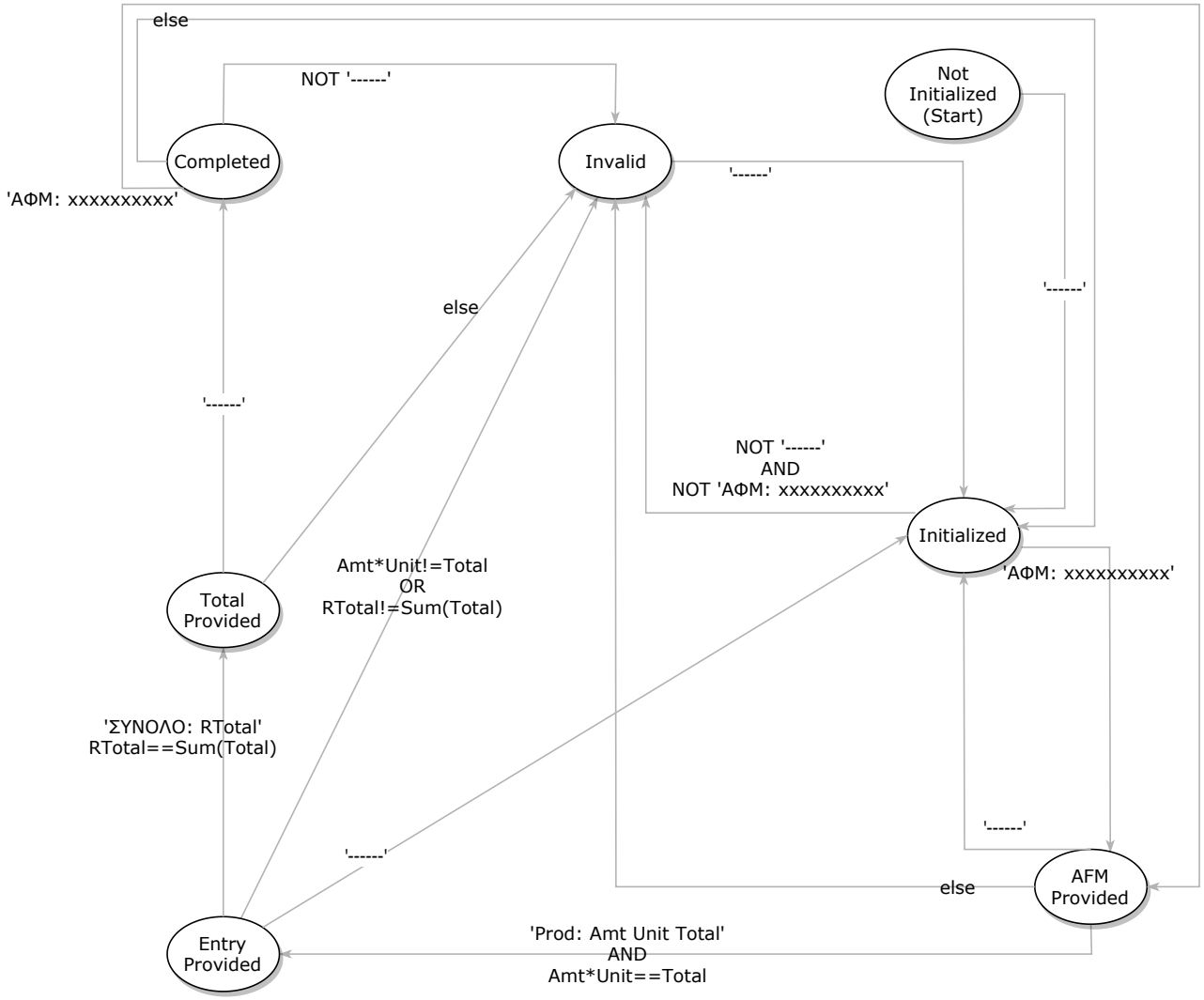
Μία απόδειξη τελικά επιστρέφεται στη MenuHandler με τη μέθοδο ReceiptParse.get_receipt() μόνο στην περίπτωση που είναι ορθά ολοκληρωμένη (κατάσταση Completed). Αυτή η απόδειξη, όπως προανέφερα, δίνεται στη συνέχεια ως είσοδος στην StatsHandler.update_stats(), όπου ενημερώνονται οι κατάλληλες δομές δεδομένων για τις επιλογές 2 και 3 του μενού. Στην επόμενη ενότητα περιγράφω αναλυτικότερα τη λειτουργία αυτή.

Διαχείριση Στατιστικών Πωλήσεων

Τα ερωτήματα 2 και 3 απαιτούν την εύρεση στατιστικών πωλήσεων για ένα δεδομένο προϊόν ή ΑΦΜ που ζητάει ο χρήστης. Τέτοιου είδος ερωτήματα στην Python απαντώνται ταχύτατα όταν τα δεδομένα αποθηκεύονται σε λεξικά (dictionaries, με κλειδί το προϊόν ή ΑΦΜ αντίστοιχα), καθώς τα λεξικά χρησιμοποιούν πίνακες κατακερματισμού (hash tables). Συνεπώς, το ρόλο των λεξικών στην παρούσα υλοποίηση αναλαμβάνουν τα πεδία product_per_afm_sales και afm_per_product_sales. Μιας και τα ερωτήματα των επιλογών 2 και 3 ενδιαφέρονται απλά για το συνολικό ποσό πωλήσεων, δεν χρειάζεται να κρατάμε στη μνήμη αναλυτικά κάθε απόδειξη, παρά μόνο τα ποσά που αντιστοιχούν σε κάθε συνδυασμό ΑΦΜ-προϊόντος.

Στην προηγούμενη ενότητα είδαμε ότι οι αποδείξεις χρησιμοποιούνται μία-μία ως είσοδο στην StatsHandler.update_stats(). Αυτό εξασφαλίζει ότι δεν κρατάμε στη μνήμη πολλά αντικείμενα της κλάσης Receipt, αλλά το πολύ ένα κάθε φορά. Η λογική αυτή όμως επιβάλει την επιλογή μίας δομής που επιτρέπει τη γρήγορη εύρεση υπαρχόντων ζευγών ΑΦΜ-προϊόντος, ώστε να προστίθεται κάθε φορά το ποσό της νέας απόδειξης στο τρέχον.

Η προφανής λύση και για τα δύο λεξικά είναι να περιέχουν με τη σειρά τους λεξικά με το προϊόν/ΑΦΜ - συνολικό ποσό ως κλειδί - τιμή. Πιο συγκεκριμένα, το product_per_afm_sales θα περιέχει για κάθε προϊόν λεξικό με τα διαφορετικά ΑΦΜ - συνολικά ποσά, ενώ το afm_per_product_sales αντίστοιχα κάθε ΑΦΜ έχει ως τιμή λεξικό με ζεύγη προϊόν - συνολικό ποσό. Σε αυτήν όμως την περίπτωση, σπαταλιέται δύο φορές μνήμη, όχι μόνο για τα ζεύγη προϊόντων - ΑΦΜ (κάτι θεμιτό για το σημαντική βελτίωση στη



Σχήμα 2: Διάγραμμα καταστάσεων που χρησιμοποιεί η ReceiptParser κατά την ανάγνωση του αρχείου εισόδου. Για την απλότητα του σχήματος παραλείπονται τα self-loops για οποιαδήποτε άλλη είσοδο πέρα από τις περιπτώσεις που απεικονίζονται (π.χ. για είσοδο `—- η κατάσταση Not Initialized παραμένει).

χρονική πολυπλοκότητα που προσφέρουν οι πίνακες κατακερματισμού και στις δύο επιλογές) αλλά και για το ποσό των ζευγών (που προσφέρει ασήμαντη χρονική βελτίωση για δυσανάλογα μεγάλο κόστος μνήμης). Επομένως, μιας και στην πραγματικότητα ο αριθμός των διαφορετικών ΑΦΜ αναμένεται να είναι μεγαλύτερος από αυτόν των προϊόντων, επέλεξα να κρατήσω στο product_per_afm_sales την πληροφορία για τα ποσά με εμφολιασμένα λεξικά, και αντίθετα να αφήσω το afm_per_product_sales με κλειδιά τα ΑΦΜ και τιμές μία απλή λίστα προϊόντων. Τα αποτελέσματα του Πίνακα 1 αιτιολογούν την επιλογή μου αυτή.

Σημαντική ήταν και η απόφαση της επιλογή του τύπου δεδομένων για την αναπαράσταση των αριθμών. Για την αποφυγή αριθμητικών σφαλμάτων κατά την εκτέλεση πράξεων με floats της Python, κάθε τιμή αποθηκεύεται ως ακέραιος (int). Έτσι, γνωρίζοντας φυσικά ότι όλα τα χρηματικά ποσά στην είσοδο αποτελούνται από δύο δεκαδικά ψηφία, γνωρίζουμε αυτόματα ότι στην πράξη οι αποθηκευμένοι ακέραιοι αντιστοιχούν στα πραγματικά ποσά πολλαπλασιασμένα με 100. Η κατάλληλη επαναφορά των πραγματικών ποσών στην εκτύπωση των αποτελεσμάτων εκτελείται από τη format_price(). Τέλος, οι option_3, option_2 του μενού καλούν τις συναρτήσεις product_to_string και afm_to_string αντίστοιχα, οι οποίες χτίζουν σε αμελητέο χρόνο το αποτέλεσμα σε μορφή string, ακόμη και για πολύ μεγάλο αριθμό δεδομένων.

afm_per_product_sales composed of:	Nested dictionaries	Nested lists
Χρόνος Εκτέλεσης Ανάγνωσης Αρχείου (sec)	29.281	28.909
Μέγεθος product_per_afm_sales (bytes)	42075769	42075769
Μέγεθος afm_per_product_sales (bytes)	95230721	63528813
Χρόνος Αναζήτησης Προϊόντοσ='τζατζικι' (sec)	0.036	0.036
Χρόνος Αναζήτησης ΑΦΜ='0000010000' (sec)	< 0.0001	< 0.0002

Πίνακας 1: Απόδοση επιλογών με χρήση λεξικών ή λιστών στο λεξικό afm_per_product_sales (bytes). Οι δοκιμές έγιναν σε VMware, Ubuntu 18.04.3, 2.9 GB, AMD Ryzen 7 1700, χρησιμοποιώντας το αρχείο 600000 αποδείξεων, με 50000 διαφορετικά ΑΦΜ και 35 διαφορετικά προϊόντα του συμφοιτητή Ιπποκράτη Στρατάκη από τη Συζήτηση του μαθήματος. Οι χρόνοι εξαρτώνται από την κατάσταση του λειτουργικού, αλλά είναι ενδεικτικοί.