ΚΩΝΣΤΑΝΤΙΝΟΣ Π. ΓΙΑΛΟΥΡΗΣ

Προγραμματίζοντας σε Python

Κωνσταντίνος Γιαλούρης

10/10/2024

Περιεχόμενα

1 Εισαγωγή στο Δομημένο Προγραμματισμό

1.1 Δομημένος Προγραμματισμός

Ο όρος **Δομημένος Προγραμματισμός** χρησιμοποιήθηκε, για πρώτη φορά, από τον καθηγητή E. Dijkstra στα μέσα της δεκαετίας του 60 που πρότεινε τον περιορισμό της χρήσης της εντολής GO TO στις γλώσσες προγραμματισμού. Στόχος του δομημένου προγραμματισμού είναι να αλλάξει τη διαδικασία καταγραφής του αλγόριθμου από μια επίπονη λειτουργία (*δοκιμής και λάθους*) σε μια ποιοτική και ελεγχόμενη λειτουργία.

Ο Δομημένος Προγραμματισμός προϋποθέτει την ανάπτυξη του αλγόριθμου, έτσι ώστε να αποτελείται από ανεξάρτητα τμήματα με βάση ένα προκαθορισμένο σχέδιο και χρησιμοποιεί τις βασικές αλγοριθμικές δομές της **ακολουθίας**, της **επιλογής και της επανάληψης.** Στη συνέχεια η κωδικοποίηση του αλγόριθμου σε γλώσσα προγραμματισμού θα ακολουθήσει την ίδια τεχνική δημιουργώντας το τελικό πρόγραμμα.

Τα κύρια πλεονεκτήματα του Δομημένου Προγραμματισμού είναι:

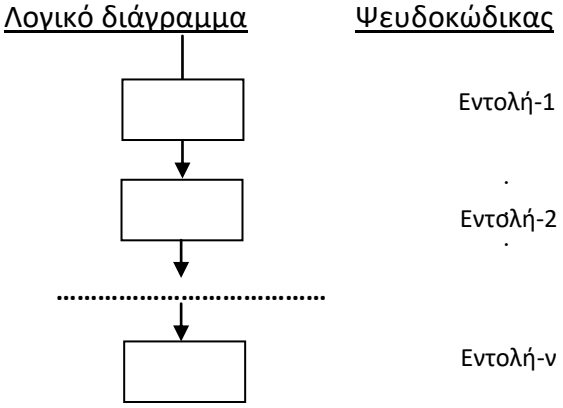
- Διευκόλυνση στην ανάπτυξη του αλγόριθμου κατά τμήματα.
- Ευκολία και ταχύτητα στην κωδικοποίηση.
- Καλύτερη ποιότητα προγραμμάτων.
- Ευκολία στις διορθώσεις και τη συντήρηση.
- Τεκμηρίωση που περιέχεται σχεδόν εξ' ολοκλήρου στο ίδιο το πρόγραμμα.

1.2 Βασικές Αλγοριθμικές Δομές

Υπάρχουν διάφοροι μέθοδοι για την αναπαράσταση των αλγορίθμων όπως η φραστική μέθοδος, ο ψευδοκώδικας και το λογικό διάγραμμα. Απ' αυτές έχουν καθιερωθεί το λογικό διάγραμμα και ιδιαίτερα ο ψευδοκώδικας ο οποίος αποτελεί σήμερα το βασικότερο τρόπο αναπαράστασης αλγορίθμων. Η λύση κάθε προβλήματος μπορεί να παρασταθεί με την χρήση των βασικών αλγοριθμικών δομών της **ακολουθίας**, της **επιλογής** και της **επανάληψης**:

1.2.1 Ακολουθία

Οι εντολές οι οποίες βρίσκονται σε **διαδοχή (sequence)** εκτελούνται κατά τη σειρά που είναι γραμμένες δηλαδή κάθε εντολή εκτελείται εφόσον έχει εκτελεστεί η προηγούμενή της.

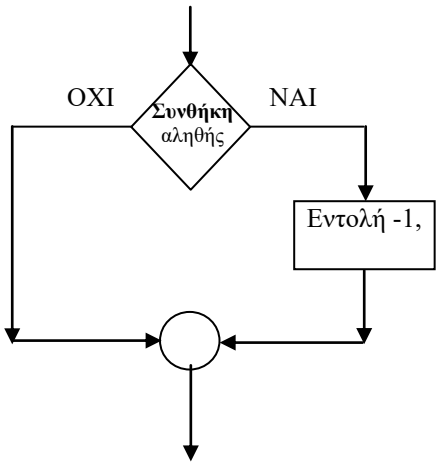


1.2.2 Επιλογή

Με τη δομή της **επιλογής** (Αν ... τότε ... αλλιώς ....) παρέχεται η δυνατότητα εκτέλεσης μιας η περισσοτέρων εντολών ανάλογα με το αποτέλεσμα ελέγχου μιας συνθήκης απλής ή σύνθετης.

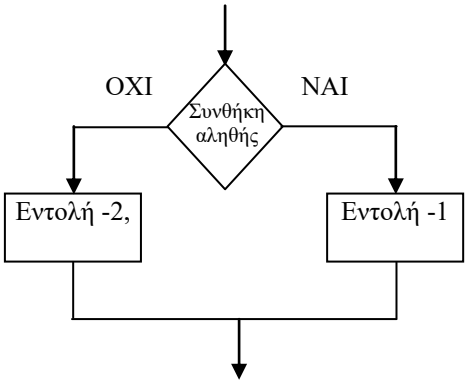
Παράδειγμα-1

Λογικό διάγραμμα επεξήγηση



Αν η **Συνθήκη** είναι αληθής
Τότε εκτελείται η **Εντολή -1**

Παράδειγμα-2



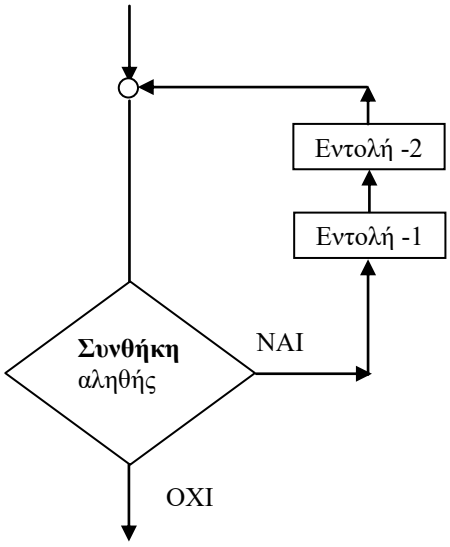
Αν η **Συνθήκη** είναι αληθής
Τότε εκτελείται η **Εντολή -1**
Αλλιώς εκτελείται η Εντολή-2

1.2.3 Επανάληψη

Μία δομή επανάληψης επιτρέπει την εκτέλεση ενός τμήματος προγράμματος πολλές φορές. Οι δομές επανάληψης μπορούν να έχουν τις παρακάτω αναπαραστάσεις:

1. Επανάλαβε όσο η συνθήκη είναι αληθής

Λογικό διάγραμμα: Επεξήγηση



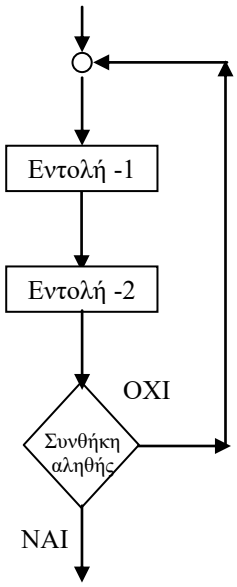
Η **Συνθήκη** είναι ένα κριτήριο ελέγχου εκτέλεσης της επαναληπτικής διαδικασίας Πρώτα γίνεται ο έλεγχος της Συνθήκης και αν είναι αληθής αρχίζει η εκτέλεση της επανάληψης

Η **Εντολή-1** και η **Εντολή-2** εκτελούνται εφόσον η συνθήκη είναι αληθής

Η **Εντολή-1** και η **Εντολή-2** δεν εκτελούνται αν η συνθήκη είναι ψευδής.

2.Επανάλαβε μέχρις ότου η συνθήκη να γίνει αληθής

Λογικό διάγραμμα:



Επεξήγηση:

Η **Συνθήκη** είναι ένα κριτήριο ελέγχου εκτέλεσης της επαναληπτικής διαδικασίας Αρχικά εκτελούνται η **Εντολή-1** και η **Εντολή-2.**
Στη συνέχεια γίνεται ο έλεγχος της **Συνθήκης** και αν η συνθήκη είναι **ψευδής** επαναλαμβάνεται η εκτέλεση των εντολών 1 και 2.

Το τέλος εκτέλεσης της επανάληψης πραγματοποιείται **όταν η Συνθήκη γίνει αληθής.**
Η **Εντολή-1** και η **Εντολή-2** εκτελούνται **τουλάχιστον μία φορά**

Με την εντολή ***επανάλαβε-μέχρις ότου*** έχουμε εκτέλεση τουλάχιστον μία φορά, των εντολών που περιέχει, έστω και αν η συνθήκη είναι αληθής (αφού ο έλεγχος της συνθήκης γίνεται στο τέλος των εντολών επανάληψης).

1.3 Τεχνικές Προγραμματισμού

Όπως έχει ήδη αναφερθεί, ο Δομημένος Προγραμματισμός ασχολείται με την ανάπτυξη αλγόριθμων με έναν συστηματικό τρόπο. Ο τρόπος αυτός χρησιμοποιεί τις βασικές αλγοριθμικές δομές της **ακολουθίας**, **επιλογής και επανάληψης.** Παρόλα αυτά, οι δομές από μόνες τους δεν είναι αρκετές για την ολοκληρωμένη παρουσίαση ενός δομημένου αλγόριθμου, αλλά πρέπει να αναπτυχθούν με την χρησιμοποίηση συγκεκριμένων τεχνικών. Οι τεχνικές που χρησιμοποιούνται περισσότερο είναι του **Τμηματικού Προγραμματισμού** (***Modular Programming***) και του **Ιεραρχικού Σχεδιασμού** ***(Hierarchical design)***

1.3.1 Τμηματικός Προγραμματισμός

Ο Τμηματικός Προγραμματισμός έχει ως βασικό στοιχείο το χωρισμό ενός προγράμματος σε ανεξάρτητες λογικές ενότητες - τμήματα (υποπρογράμματα). Τα τμήματα αυτά πρέπει να είναι όσο το δυνατόν μικρότερα, ώστε να διευκολύνεται ο καταμερισμός του προγράμματος σε μικρά τμήματα τα οποία είναι ευκολότερο να διορθωθούν. Με τον Τμηματικό Προγραμματισμό επιταχύνεται η ανάπτυξη μιας εφαρμογής μια και τα διάφορα υποπρογράμματα μπορούν να αναπτύσσονται παράλληλα από δύο ή περισσότερους προγραμματιστές.

1.3.2 Ιεραρχικός Προγραμματισμός

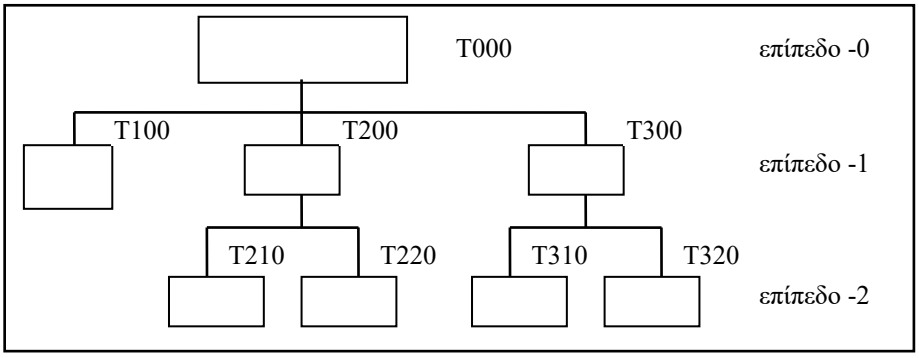
Η σχεδίαση προγράμματος σύμφωνα με τον Ιεραρχικό Προγραμματισμό δομείται από πάνω προς τα κάτω. Ξεκινάμε με μία πολύ απλή και σύντομη δήλωση για το ***τι κάνει το πρόγραμμα,*** π.χ. με τον υπολογισμό της μισθοδοσίας. Αυτό γίνεται στο αρχικό επίπεδο σχεδίασης του προγράμματος, ενώ στο επόμενο επίπεδο, εμφανίζονται με περισσότερες λεπτομέρειες οι επεξεργασίες που εκτελεί το πρόγραμμα. Καθεμιά απ' αυτές δεν είναι τελείως συμπληρωμένη, αλλά περιγράφεται με περισσότερες λεπτομέρειες σε κατώτερο επίπεδο του προγράμματος.

Η διαδικασία αυτή επαναλαμβάνεται βήμα προς βήμα και, σε κάθε κατώτερο επίπεδο, εμφανίζονται περισσότερες λεπτομέρειες. Η σε κατώτερα επίπεδα ανάλυση σταματάει, όταν η επεξεργασία περιέχει τόσες λεπτομέρειες, ώστε να μπορεί να κωδικοποιηθεί σε μία γλώσσα προγραμματισμού.

Τα παραπάνω συνιστούν την τεχνική του Ιεραρχικού Προγραμματισμού όπου, ξεκινώντας από τα γενικά σε ανώτερα επίπεδα, προχωρούμε προσθέτοντας λεπτομέρειες σε κατώτερα επίπεδα.

Η παράσταση αυτών των τεχνικών γίνεται με τα διαγράμματα HIPO (Hierachical Input Output Processing), που στα Ελληνικά μεταφράζεται ως *Ιεράρχηση Εισόδου - Επεξεργασίας - Εξόδου.*

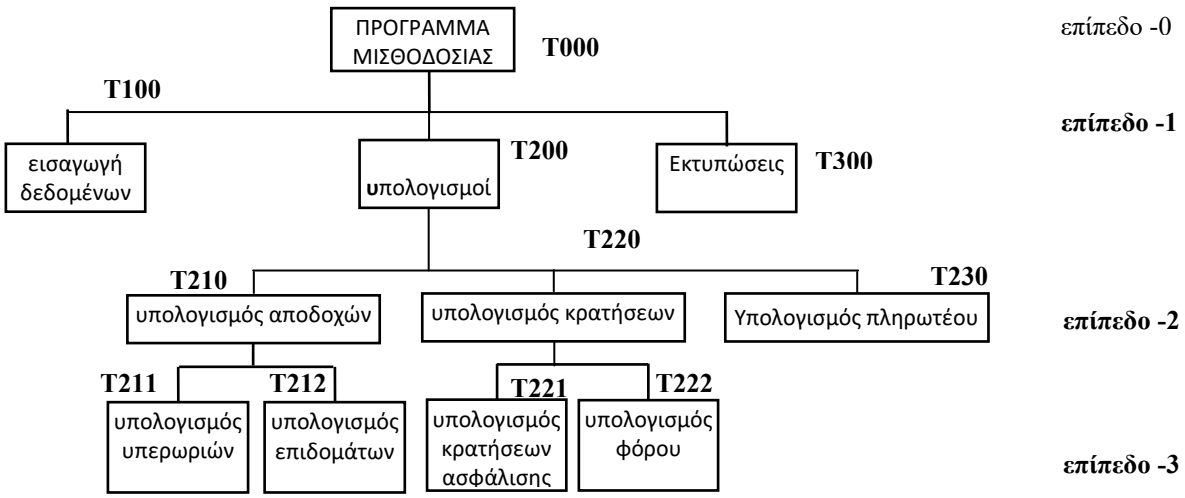
Ένα διάγραμμα HIPO αποτελείται από τρία βασικά τμήματα, τα οποία περιγράφουν την είσοδο δεδομένων, την επεξεργασία και την έξοδο αποτελεσμάτων που απαιτεί ένας αλγόριθμος. Στα διαγράμματα HIPO εμφανίζονται τα διάφορα επίπεδα ιεραρχίας και τα τμήματα του αλγόριθμου-προγράμματος με αριθμό σύμφωνα με τη σειρά εκτέλεσής τους. Στο ανώτερο επίπεδο υπάρχει μια περιγραφή της όλης διαδικασίας που αναπαρίσταται από το διάγραμμα HIPO. Στο αμέσως επόμενο επίπεδο διακρίνονται, σε τμήματα, οι τρεις βασικές λειτουργίες (είσοδος, επεξεργασία και έξοδος) οι οποίες μπορεί να αναλύονται περαιτέρω σε άλλα υπο-τμήματα.



Διάγραμμα HIPO.

Παρακάτω δίδεται ένα απλό διάγραμμα HIPO για τον υπολογισμό μισθοδοσίας εργαζομένων.

ΥΠΟΛΟΓΙΣΜΟΣ ΜΙΣΘΟΔΟΣΙΑΣ



Διάγραμμα HIPO υπολογισμού μισθοδοσίας

2 Εισαγωγή στην Python

Με τον όρο *γλώσσα προγραμματισμού* εννοούμε ένα υποσύνολο της αγγλικής γλώσσας, όπου επιτρέπει στον προγραμματιστή να δίνει στον υπολογιστή μονοσήμαντες εντολές. Μια γλώσσα προγραμματισμού είναι μια γλώσσα την οποία μπορεί να κατανοήσει ο υπολογιστής. Αντιθέτως, μια φυσική γλώσσα, όπως η Ελληνική ή η Αγγλική είναι πολύ γενική και διφορούμενη για τον υπολογιστή. Οι γλώσσες προγραμματισμού διακρίνονται σε δύο κατηγορίες:

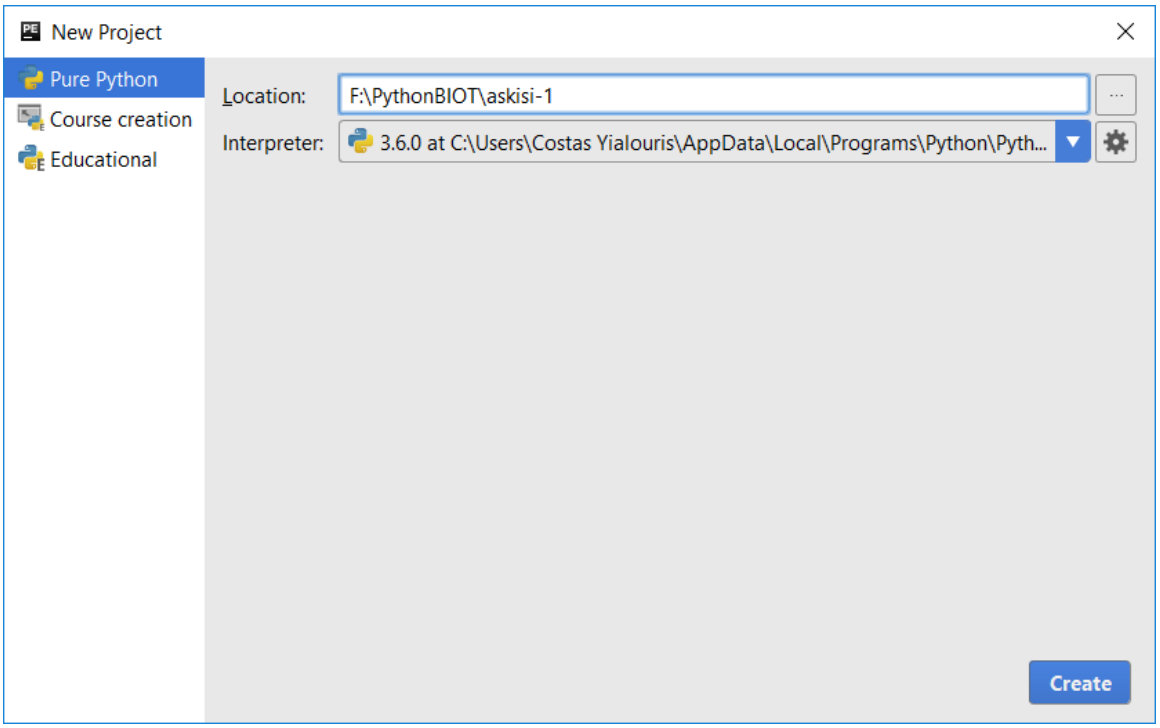
- Χαμηλού επιπέδου γλώσσες (low level languages) και
- Υψηλού επιπέδου γλώσσες (high level languages).

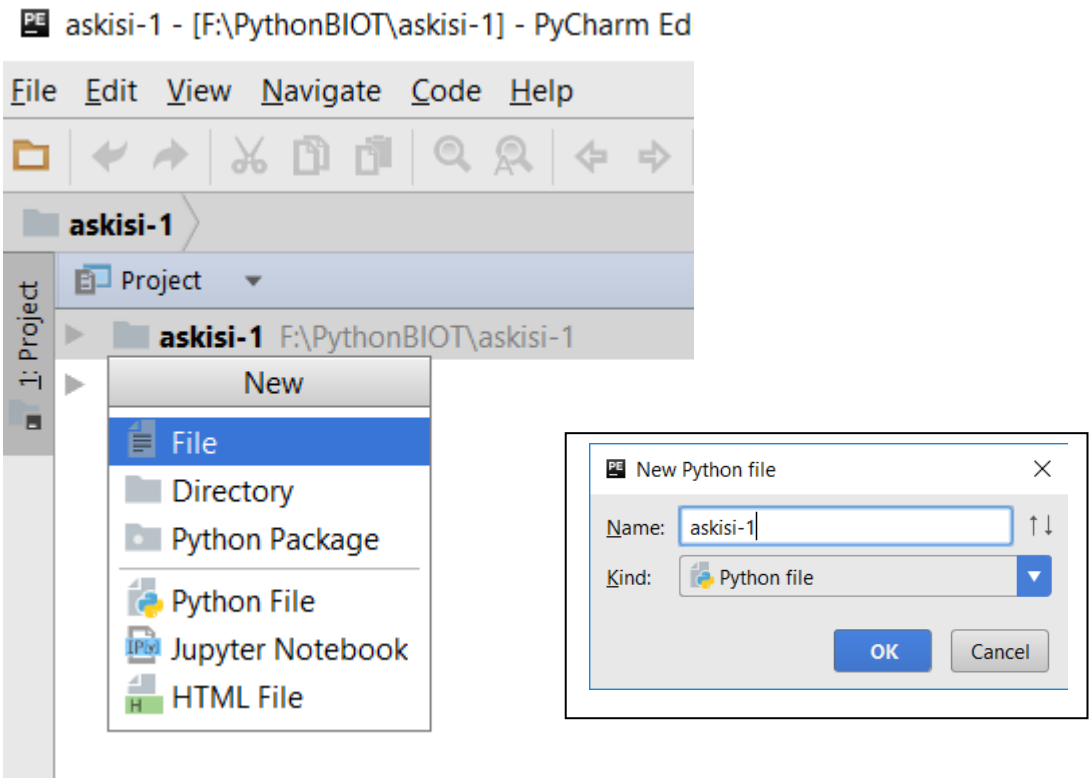
Οι γλώσσες χαμηλού επιπέδου είναι συγγενικές με τη "γλώσσα μηχανής", όπου συνδυασμοί δυαδικών αριθμών έχουν αντικατασταθεί με μνημονικούς συμβολισμούς. Η γλώσσα μηχανής είναι η μόνη γλώσσα που μπορούν να καταλάβουν οι υπολογιστές. Όμως, οι γλώσσες χαμηλού επιπέδου είναι ιδιαίτερα δύσκολες στη χρήση τους. Οι γλώσσες υψηλού επιπέδου αναπτύχθηκαν για να διευκολυνθεί η συγγραφή προγραμμάτων σε ιδιαίτερο περιβάλλον όπως επιστημονικό, επιχειρηματικό ή εκπαιδευτικό. Παραδείγματα τέτοιων γλωσσών είναι η Java, η Python, η C++, η Fortran, κ.ά. Είναι εύκολες στην εκμάθησή τους και συνήθως ανεξάρτητες από τα διάφορα συστήματα υπολογιστών.

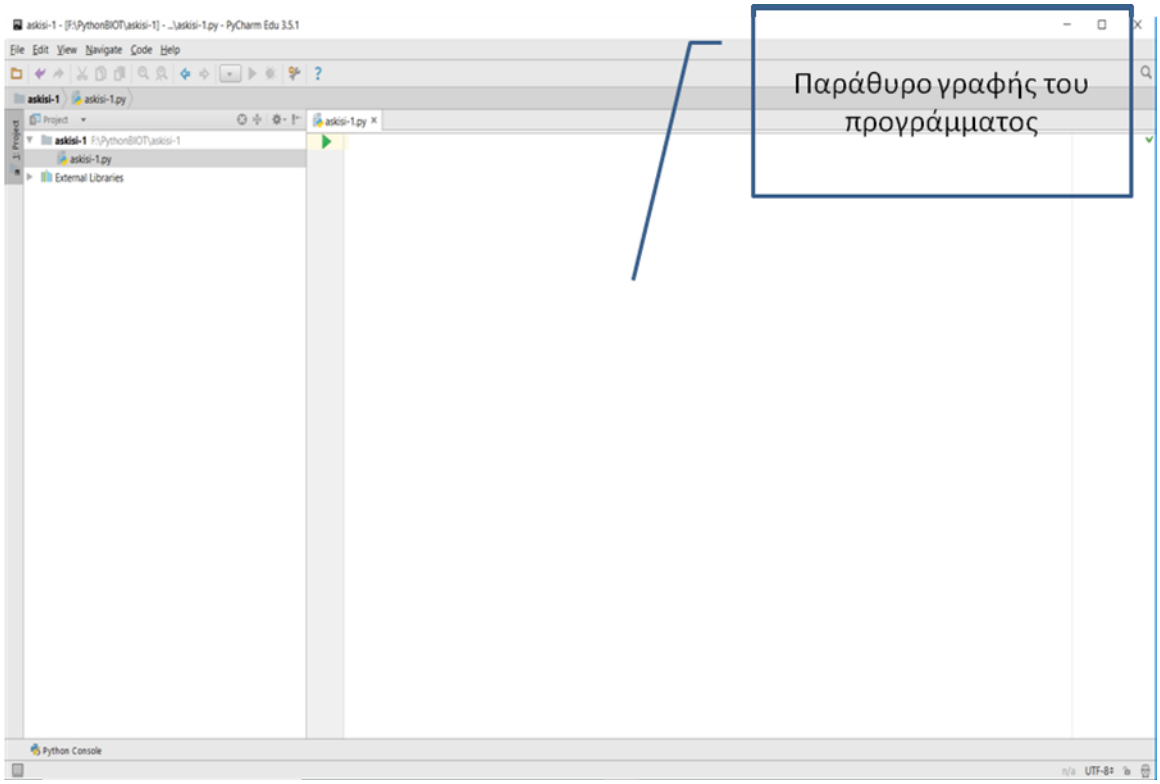
3 Εκκίνηση της Python

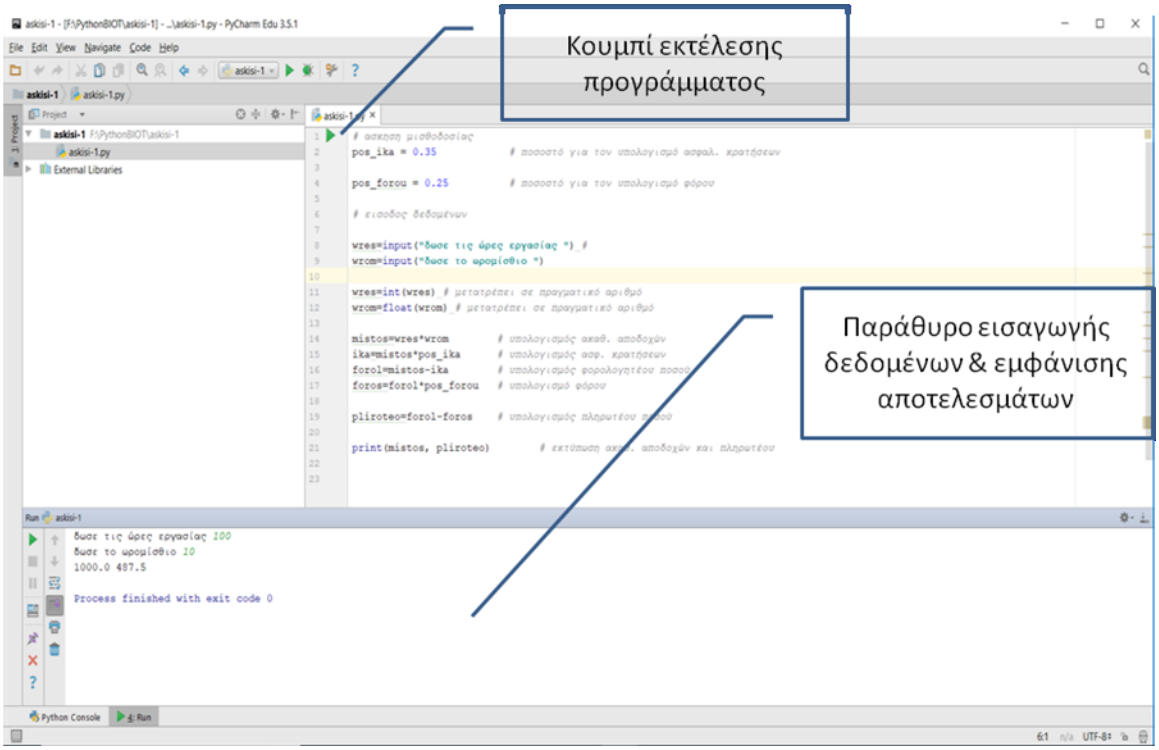
4 Περιβάλλον Ανάπτυξης PyCharm

Πριν μερικά χρόνια, ένα πρόγραμμα αντιπροσωπεύονταν από ένα αρχείο. Σήμερα όμως καθώς τα προγράμματα γίνονται όλο και πιο πολύπλοκα, ένα πρόγραμμα μπορεί να αποτελείται πολλά επιμέρους αρχεία. Στο περιβάλλον PyCharm το σύνολο των επιμέρους αρχείων αποτελεί ένα Project. Με τη δημιουργία ενός νέου project δημιουργείται ένας φάκελος με όνομα το όνομα του project που έδωσε ο χρήστης και στη συνέχεια ένα σύνολο διαφορετικών φακέλων και αρχείων.




askisi-1 - [F:\PythonBIOT\askisi-1] - PyCharm Ed
File Edit View Navigate Code Help
askisi-1
1: Project
Project
askisi-1 F:\PythonBIOT\askisi-1
New
File
Directory
Python Package
Python File
Jupyter Notebook
HTML File
New Python file
Name: askisi-1
Kind: Python file
OK
Cancel


Παράθυρο γραφής του προγράμματος



5 Μεταβλητές –Τύποι Δεδομένων

5.1 Μεταβλητές

Με τον όρο μεταβλητή εννοούμε μια περιοχή στη κεντρική μνήμη του υπολογιστή η οποία αναφέρεται ονομαστικά και μπορεί να αλλάξει το περιεχόμενό της κατά την εκτέλεση του προγράμματος. Στο κώδικα ενός προγράμματος μπορούμε να χρησιμοποιούμε μια ή περισσότερες μεταβλητές στις οποίες αποθηκεύονται κείμενο, αριθμοί, κλπ. **Σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού, στην Python οι μεταβλητές και ο τύπος τους, δεν δηλώνονται πριν χρησιμοποιηθούν**. Ο τύπος μιας μεταβλητής ορίζεται αυτόματα με την ανάθεση τιμής σε αυτήν και **μπορεί να αλλάξει κατά τη διάρκεια της εκτέλεσης**.

Το όνομα μιας μεταβλητής θα πρέπει να ακολουθεί τους παρακάτω κανόνες:

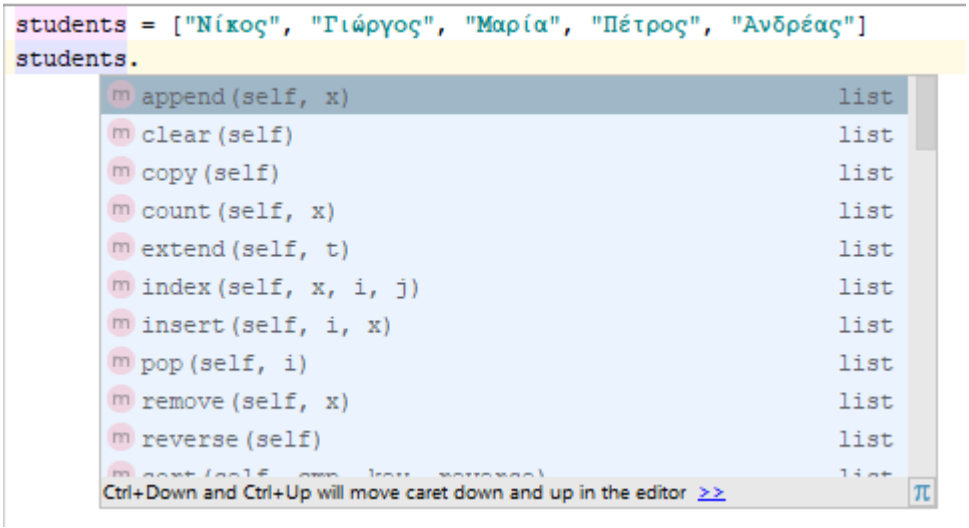
- Να αρχίζει με γράμμα.
- Να αποτελείται από λατινικά γράμματα, αριθμούς ή την κάτω παύλα (_). Δεν πρέπει να περιέχει κενά ή άλλα σύμβολα (π.χ. τελεία).
- Δεν μπορεί να περιέχει δεσμευμένες λέξεις ή ονόματα αντικειμένων ή ιδιοτήτων της Python (π.χ. `for, if, print` κλπ).

5.2 Τύποι Δεδομένων

Οι βασικοί τύποι δεδομένων, αν και στην ουσία δεν είναι τύποι δεδομένων αλλά κλάσεις και ο ορισμός μίας μεταβλητής μέσω της ανάθεσης τιμής σε αυτήν ορίζει αντικείμενο της αντίστοιχης κλάσης. Συνεπώς η μεταβλητή ως αντικείμενο θα συνοδεύεται από μία σειρά μεθόδων χειρισμου. Οι βασικοί τύποι που υποστηρίζει η Python δίνονται στον παρακάτω πίνακα.

Τύπος	Περιγραφή
bool	Λογικός τύπος με διακριτές τιμές True ή False
float	Πραγματικοί αριθμοί
int	Ακέραιοι αριθμοί
str	Χαρακτήρες
list	Λίστα
dict	Λεξικό
set	Σύνολο
tuple	Πλειάδα

Οι τύποι δεδομένων της Python διαφέρουν από άλλες γλώσσες προγραμματισμού γιατί είναι Κλάσεις. Ο ορισμός μίας μεταβλητής μέσω της ανάθεσης τιμής σε αυτήν ορίζει την μεταβλητή ως ένα αντικείμενο της αντίστοιχης Κλάσης. Συνεπώς η μεταβλητή ως αντικείμενο θα συνοδεύεται από μία σειρά μεθόδων χειρισμου, που υποστηρίζει η Python. Οι μέθοδοι χειρισμού της αντίστοιχης μεταβλητής εμφανίζονται στον προγραμματιστή γράφοντας το όνομα της μεταβλητής που έχει ορισθεί προηγουμένως και στο τέλος του ονόματος ακολουθεί η τελεία (.). π.χ.



Όπως αναφέρθηκε δεν δηλώνεται τύπος της μεταβλητής. Μπορούμε να δούμε τον τύπο της μεταβλητής χρησιμοποιώντας τη συνάρτηση type η οποία θα έχει ως όρισμα το όνομα της μεταβλήτης.

Παράδειγμα:

```
students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας"]
print(type(students))
```

Η εκτέλεση του παραπάνω προγράμματος θα έχει ως έξοδο:

```
<class 'list'>


Process finished with exit code 0
```

5.2.1 bool – Λογικός τύπος

Οι μεταβλητές λογικού τύπου μπορούν να πάρουν τιμή **True** ή **False**. Οι τιμή αυτή μπορεί να προκύπτει από παραστάσεις με σχεσιακούς ή συγκριτικούς τελεστές ή απ' ευθείας με ανάθεση τιμής.

5.2.2 Αριθμητικοί τύποι

Η αναπαράσταση αριθμητικών δεδομένων στην Python διακρίνονται σε δύο κατηγορίες στην ακέραια μορφή και στην μορφή κινητής υποδιαστολής (floating point) ή μορφή πραγματικού αριθμοί. Ετσι έχουμε τον τύπο των Ακεραίων (**int**) αριθμών και τον τύπο των Πραγματικών (**float**) αριθμών.

Μία μεταβλητή θεωρείται ότι είναι ακέραιου τύπου (**int**) αν περιέχει μόνο αριθμητικά ψηφία.

Μία μεταβλητή θεωρείται ότι είναι πραγματικού τύπου(**float**) αν περιέχει μόνο αριθμητικά ψηφία και την υποδιαστολή η οποία πρέπει να είναι η τελεία (.) και μπορεί να ακολουθείται από δεκαδικά ψηφία.

5.2.3 str – Χαρακτήρες

Ο τύπος str χαρακτηρίζεταιαπό την ιδιότητα ότι περιέχει δεδομένα μορφής κειμένου. Μπορεί να περιέχει γραμματα αριθμούς σύμβολα κλπ. Τα δεδομένα τύπου str περιλαμβάνονται μετα ξύ των μονών ή διπλών εισαγωγικών.

Πχ.

```
onoma="Γιώργος"
eponymo='Παναγιωτόπουλος'
```

Περισσότερα θα αναφερθούν στην ενότητα Βασικές μέθοδοι χειρισμού **string**.

5.2.4 list – Λίστα

Μία λίστα είναι ένα αντικείμενο το οποίο περιέχει πολλαπλά δεδομένα. Μία λίστα είναι ένας δυναμικός τύπος δεδομένων. Αυτό σημαίνει ότι μπορούμε να προσθέσουμε ή να αφαιρέσουμε στοιχεία σε μία λίστα. Η λίστα είναι ένα διατεταγμένο σύνολο δεδομένων, η τάξη/αρίθμηση των στοιχείων της αρχίζει από το 0 (μηδέν). Κάθε στοιχείο μίας λίστας μπορεί να προσπελαστεί μέσω ενός δείκτη που καθορίζει τη θέση του στη λίστα.

Παραδείγματα λίστας

```
ar=[2, 4, 6, 8,10]
onomata = ["Νικολάου", "Πέτρου", "Γεωργίου"]
vathmoi=["Πέτρου", 10, 8, 7]
```

Εκτύπωση λίστας

Έστω το παρακάτω πρόγραμμα Python

```
ar = [2, 4, 6, 8,10]  # ορισμός λίστας
onomata = ["Νικολάου", "Πέτρου", "Γεωργίου"]
print(ar) # εκτύπωση όλων των στοιχείων της λίστας ar
print(onomata) # εκτύπωση όλων των στοιχείων της λίστας onomata
print(onomata[2]) # εκτύπωση του 3ου στοιχείου (με τάξη 2) της λίστας onomata
```

Η εκτέλεση του προγράμματος θα δώσει ως αποτέλεσμα:

```
[2, 4, 6, 8, 10]
['Νικολάου', 'Πέτρου', 'Γεωργίου']
Γεωργίου
```

Περισσότερα θα αναφερθούν στην ενότητα **Βασικές μέθοδοι χειρισμού λιστών**.

5.2.5 Tuple – Πλειάδα

Μία **tuple (πλειάδα)** είναι μία αλληλουχία όπως και η λίστα. Διαφέρει ως προς τη λίστα όσον αφορά στο γεγονός ότι το περιεχόμενό της δεν μπορεί να αλλάξει. Πλεονέκτημα έναντι της λίστας είναι ότι α**υξάνει την ταχύτητα επεξεργασίας και υπάρχει ασφάλεια λόγω μη δυνατότητα μεταβολής του περιεχομένου της.**

Όπως και η λίστα η πλειάδα (tuple) υποστηρίζει τη διαχείριση των στοιχείων της μέσω δεικτών. Διατηρεί τις ίδιες λειτουργίες με τη λίστα εκτός αυτών που αλλάζει το περιεχόμενό της.

Παραδείγματα **tuple**:

```
a=("Νικος","Πέτρος","Μαρία")
b=(4,3,5,6,2,-3)
c=(1,)
```

Περισσότερα θα αναφερθούν στην ενότητα Βασικές μέθοδοι χειρισμού tuple.

5.2.6 Dictionary –Λεξικό

Μία μεταβλητή τύπου dictionary είναι συλλογή δεδομένων. Κάθε στοιχείο της συλλογής αυτής αποτελείται από δύο μέρη. Το πρώτο μέρος αναφέρεται ως **κλειδί (key)** και το δεύτερο μέρος ως **τιμή(value).**Το κλειδί χρησιμοποιείται για τον εντοπισμό μιας τιμής. Σε κάθε κλειδί αντιστοιχεί μία τιμή. Τα ζεύγη **key:value καταχωρούνται μέσα σε άγκιστρα { } και διαχωρίζονται με (,). Η τιμή μπορεί να είναι οποιουδήποτε τύπου(κλάσης).**

Παραδείγματα Dictionary

ονομα_dictionary={key1: value1, key2:value2,…..keyn:valuen}

```
phone_list={5182:"Γεωργίου",4175:"Πέτρου",4183:"Παύλου",4182:"Ιωάννου"}
```

Περισσότερα θα αναφερθούν στην ενότητα Βασικές μέθοδοι χειρισμού λεξικών.

5.2.7 Set - Σύνολο

Ένα σύνολο (**set**) είναι συλλογή δεδομένων από **μοναδικές τιμές**. Τα στοιχεία ενός συνόλου **δεν είναι** διατεταγμένα και **μπορεί να είναι διαφορετικού τύπου**

Για τη δημιουργία ενός συνόλο χρησιμοποιούμε τη συνάρτηση **set.**

Δημιουργία κενού συνόλου

```
synolo1=set()
```

Δημιουργία συνόλου με στοιχεία `'a','b','c','d'`

```
synolo2={'a','b','c','d'}
```

Δημιουργία συνόλου με στοιχεία μιας λίστας

```
synolo2=set(['a','b','c','d'])
```

Η εκτύπωση του περιεχομένου του παραπάνω συνόλου θα είναι:

```
{'a', 'c', 'b', 'd'}
```

Δημιουργία συνόλου με στοιχεία από μία μεταβλητή τύπου `str`

```
synolo3=set('LONDON')
```

Η εκτύπωση του περιεχομένου του παραπάνω συνόλου θα είναι:

```
{'L', 'N', 'O', 'D'}
```

όπως φαίνεται και από την εκτύπωση εκ του ορισμού του συνόλου το γράμμα Ο υπάρχει μία φορά μέσα στο σύνολο.

Περισσότερα θα αναφερθούν στην ενότητα "Μέθοδοι χειρισμού συνόλων".

6 Ανάθεση τιμής σε μεταβλητή -Εκφράσεις

Στις μεταβλητές καταχωρούνται τιμές με τη χρήση του τελεστή απόδοσης τιμής (=) και εκφράσεων. Με τον όρο έκφραση εννοούμε είτε μια συγκεκριμένη τιμή είτε συνδυασμό τιμών και μεταβλητών συνδεδεμένων με κατάλληλους τελεστές.

Γενικά η σύνταξη της εντολής ανάθεσης τιμής σε μια μεταβλητή είναι η εξής:

όνομαΜεταβλητής = Έκφραση

Παραδείγματα:

```
country = "Greece"
cost = 1000
tax = 100
```

όπου έκφραση μπορεί να είναι

- Μία σταθερά
- Μια μεταβλητή
- Μία συνάρτηση
- Μία παράσταση που χρησιμοποιεί μεταβλητές, σταθερές κλπ σε συνδιασμό με τελεστές πράξεων.

Η Python έχει και μία επιπλέον σύνταξη προκειμένου να αναθέτει τιμές σε δύο η περισσότερες μεταβλητές μέσα σε μία εντολή. Στην περίπτωση αυτή το αριστερό μέλος είναι μία σειρά μεταβλητών χωριζόμενες με (,) ενώ το δεξιό μέλος μπορεί να είναι:

- Σταθερές χωριζόμενες με (,)
- Μεταβλητές (ή οποίες έχουν ορισθεί σε προηγούμενη εντολή, χωριζόμενες με (,)
- Εκφράσεις χωριζόμενες με (,)

Πχ

```
a,b =3,5
x,y=3*a,b
```

Μαθηματικοί τελεστές

Οι μαθηματικοί τελεστές χρησιμοποιούνται για τον υπολογισμό και την αντιστοίχιση των αποτελεσμάτων κάποιων εκφράσεων σε μεταβλητές. Οι τελεστές αυτοί μοιάζουν και δρουν σαν τους αντίστοιχους τελεστές των μαθηματικών και δίνονται στον παρακάτω πίνακα:

Όνομα Τελεστή	Τελεστής	Παράδειγμα
Πρόσθεση	+	x+ y
Αφαίρεση	-	x- y
Πολλαπλασιασμός	*	3*x+5*y
Διαίρεση	/	17/5 δίνει αποτέλεσμα 3,4
Ακέραια Διαίρεση (πηλίκο)	//	17//5 δίνει αποτέλεσμα 3
Υπόλοιπο Διαίρεσης	%	17 % 5 δίνει αποτέλεσμα 2
Υψωση σε Δύναμη	**	x ** 2

Σε περίπτωση αριθμητικών μεταβλητών μπορούμε να χρησιμοποιούμε την παρακάτω σύνταξη:

όνομαΜεταβλητής **Θ**= Έκφραση

όπου Θ ένας οποιοσδήποτε τελεστής αριθμητικής πράξης (`+,-,*,/,//,%,**`)

Παραδείγματα εκφράσεων

Εντολή	Ισοδύναμη εντολή	επεξήγηση
`x += a`	`x = x+a`	Αυξάνει το περιεχόμενο της μεταβλητής **x** κατά **a**
`x /=5`	`x=x/5`	Διαιρεί το περιεχόμενο της μεταβλητής **x** κατά **5**
`x-=3*(y+a)`	`x=x-3*(y+a)`	Μειώνει το περιεχόμενο της μεταβλητής x κατά το αποτέλεσμα της παράστασης **3*(y+a)**

Σειρά εκτέλεσης των πράξεων

Κάθε τελεστής στην έχει κάποια σειρά προτεραιότητας. Από δύο γειτονικούς τελεστές εκτελείται πρώτα αυτός που έχει την ανώτερη προτεραιότητα. Στην περίπτωση που εξακολουθεί να υπάρχει αμφισημία, τότε η έκφραση αποτιμάται από αριστερά προς τα δεξιά. Η σειρά προτεραιότητας των πράξεων αρχίζοντας με την υψηλότερη προτεραιότητα είναι η εξής:

- Ύψωση σε δύναμη (**)
- Πολλαπλασιασμός και διαίρεση (*, /)
- Ακέραια διαίρεση (//)
- Υπόλοιπο (%)
- Πρόσθεση και αφαίρεση (+, -)

Διαμερισμός εντολών σε περισσότερες από μία γραμμές.

Η Python δίνει τη δυνατότητα να γραφεί μία εντολή σε δύο η περισσότερες γραμμές. Αυτό επιτυγχάνεται μέσω του χαρακτηρα της ανάστροφης μπάρας (\) στο τέλος της γραμμής ο χαρακτήρας αυτός ενημερώνει τον διερμηνέα ότι η εντολή αυτή συνεχίζεται στην επόμενη γραμμή.

7 Εντολή εισόδου `input`

Για την εισαγωγή δεδομένων σε ένα πρόγραμμα της Python μπορεί να χρησιμοποιηθεί η συνάρτηση **input**.

Η **input** είναι μια ειδική συνάρτηση με την οποία δίνει δυνατότητα στον χρήστη να εισάγει δεδομένα στο πρόγραμμα

Η βασική σύνταξή της είναι:

variableName =input (*"Μήνυμα"*)

Κατά την εκτέλεση του προγράμματος η συνάρτηση **input** εμφανίζει το Μήνυμα, που βρίσκεται εντός εισαγωγικών, στην οθόνη και περιμένει τον χρήστη να πληκτρολογήσει το δεδομένο εισόδου. Η διαδικασία εισόδου ολοκληρώνεται με το πάτημα του κουμπιού **Enter** του πληκτρολογίου. Τότε αναθέτει στην μεταβλητή μία τιμή που αντιστοιχεί σε **string** . Η συνάρτηση input αναθέτει στην μεταβλητή μία τιμή που αντιστοιχεί σε **string**. Σε περίπτωση που μας ενδιαφέρει ανάγνωση αριθμητικής τιμής τότε θα πρέπει να γίνει μετατροπή σε αριθμητικό τύπο με χρήση συνάρτησης μετατροπής **int** ή **float**

8 Εντολή εξόδου `print`

Η Python παρέχει στον προγραμματιστή πολλούς διαφορετικούς τρόπους εμφάνισης στην οθόνη της τιμής μιας μεταβλητής. Θα αναφερθούμε μόνο σε κάποιες περιπτώσεις. Η απλούστερη σύνταξη της εντολής εξόδου **print** είναι:

print(δεδομένα_εξόδου)

Όπου **δεδομένα_εξόδου** είναι μία λιστα από σταθερές ή μεταβλητές ή αριθμητικές παραστάσεις που διαχωρίζονται μεταξύ τους με (,).

Μία πιο συνθετη σύνταξη της **print** είναι:

print(δεδομένα_εξόδου, sep=string1, end=string2)

όπου:

string1: Ένας ή περισσότεροι χαρακτήρες οι οποίοι θα εμφανίζονται μεταξύ των εκτυπούμενων τιμών.Η προκαθορισμένη τιμή είναι ένα κενό διάστημα.

string2: Ένας ή περισσότεροι χαρακτήρες οι οποίοι θα εμφανίζονται στο τέλος της γραμμής εκτύπωση. Η προκαθορισμένη τιμή είναι ο χαρακτήρας διαφυγής **\n** ώστε η εκτύπωση να γίνει στην επόμενη γραμμή εκτύπωσης. Εάν δοθεί οποιοσδήποτε άλλος

χαρακτήρας εκτός του χαρακτήρα διαφυγής **\n, τότε η επόμενη εκτύπωση θα γίνει στην ίδια γραμμή.**

8.1 Διαμόρφωση εκτύπωσης

8.1.1 Χαρακτήρες διαφυγής

Χαρακτήρες διαφυγής	Λειτουργία
\n	Η εκτύπωση που θα ακολουθήσει θα γίνει στην επόμενη γραμμή
\t	Η εκτύπωση που θα ακολουθήσει θα γίνει στην tab θέση
\'	Θα γίνει η εκτύπωση του μονό εισαγωγικού '
\"	Θα γίνει η εκτύπωση του διπλού εισαγωγικού "
****	Θα γίνει η εκτύπωση της ανάποδη μπάρας \

8.1.2 Διαμόρφωση αριθμητικών τιμών με τη συνάρτηση `format`

Η συνάρτηση **`format`** χρησιμοποιείται προκειμένου να διαμορφώσει αριθμητικές τιμές.

Η σύνταξη της εντολής είναι:

`format(έκφραση, string_διαμόρφωσης)`

`όπου:`

`έκφραση` είναι μία αριθμητική

- σταθερά ή
- μεταβλητή ή
- παράσταση που χρησιμοποιεί αριθμητικές μεταβλητές, σταθερές κλπ, σε συνδυασμό με τελεστές αριθμητικών πράξεων

`string_διαμόρφωσης`

- Ένα **`string`** που δηλώνει το είδος διαμόρφωσης

Ως συνάρτηση **`format`** επιστρέφει μία τιμή η οποία είναι τύπου **`str`**. Κατά συνέπεια η μεταβλητή στην οποία θα ανατεθεί τιμή μέσω της **`format`** θα είναι τύπου **`str.`**

Πίνακας με τα κυριότερα string διαμόρφωσης αριθμητικών τιμών

String διαμόρφωσης	**Λειτουργία**

'.nf'	Πραγματικός αριθμός με **n** δεκαδικά ψηφία
'+.nf'	Πραγματικός αριθμός με n δεκαδικά ψηφία και με εμφάνιση του προσήμου
','	Πραγματικός ήακέραιος αριθμός διαχωρίζοντας τις χιλιάδες με (,)
'k,.nf'	Πραγματικός αριθμός με **n** δεκαδικά ψηφία διαχωριζοντας τις χιλιάδες με (,) και στοιχισμένος δεξια σε **str** που έχει μήκος **k**
'+.nf'	Πραγματικός αριθμός με n δεκαδικά ψηφία και με εμφανιση του προσήμου
'k,'	Ακέραιος αριθμός διαχωριζοντας τις χιλιάδες με (,) και στοιχισμένος δεξια σε **str** που έχει μήκος **k**
'.2%'	Μετατροπή ενός πραγματικού αριθμού σε μορφή ποσοστού με 2 δεκαδικά και στο τέλος το συμβολο %
'.ne'	Πραγματικός αριθμός σε εκθετική μορφή με n δεκαδικά ψηφία

8.1.3 Διαμόρφωση αριθμητικών τιμών με τη συνάρτηση `round`

Η διαμόρφωση αριθμητικών δεδομένων όσον αφορά στα δεκαδικά σημεία με την προηγούμενη μέθοδο με χρήση της συνάρτησης **`format`** απλά δεν εμφανίζει τα δεκαδικά στοιχεία. Εσωτερικά στις μεταβλητές διατηρούνται και στο ενδεχόμενο μίας επαναληπτικής διαδικασίας που αθροίζει αυτούς τους πραγματικούς αριθμούς το άθροισμα ως αποτέλεσμα δεν συνάδει με το άθροισμα των αριθμών όπως παρουσιάζονται στην εκτύπωση.

Προκειμένου να λυθεί το θέμα του αθροίσματος χρησιμοποιούμε την συνάρτηση **`round`**. Η συνάρτηση συντάσεται ως εξής:

`round`(έκφραση, πλήθος δεκαδικών)

Όπου έκφραση μπορεί να είναι:

- Πραγματικός αριθμός
- Μεταβλητή τύπου **`float`**
- Αριθμητική παράσταση που το αποτέλεσμα αναμένεται πραγματικός αριθμός

Και στρογγυλεύει τον πραγματικό αριθμό στο πλήθος δεκαδικών που ορίσθηκαν

Ο συνδυασμός της round και της format χρησιμοποιείται για την κατά το δυνατόν καλύτερη εμφάνιση αριθμητικών αποτελεσμάτων.

9 Δομές Αποφάσεων

Όταν ένα πρόγραμμα της Python εκτελείται στον υπολογιστή τότε οι εντολές του κώδικα προγράμματος εκτελούνται η μια μετά την άλλη σύμφωνα με τη σειρά που έχουν διατυπωθεί, μέχρι να ολοκληρωθεί και η τελευταία εντολή. Πολλές φορές όμως απαιτείται η εκτέλεση όχι της επόμενης εντολής, αλλά κάποιας άλλης, σε διαφορετικό σημείο του προγράμματος υπερπηδώντας ενδεχόμενα τις εντολές που παρεμβάλλονται. Αυτό επιτυγχάνεται με τις δομές αποφάσεων.

9.1 Σχεσιακοί Τελεστές

Οι σχεσιακοί τελεστές ή τελεστές σύγκρισης συγκρίνουν δεδομένα στο κώδικα προγράμματος και καθορίζουν το αποτέλεσμα της σύγκρισης με απαντήσεις του τύπου **`True`** (αληθές) ή **`False`** (ψευδές). Οι τελεστές σύγκρισης μπορούν να συγκρίνουν αριθμητικές και αλφαβητικές τιμές. Μέσω των σχεσιακών τελεστών πραγματοποιούνται συγκρίσεις μεταξύ ιδίου τύπου δεδομένων. Οι τελεστές σύγκρισης είναι οι ακόλουθοι:

Τελεστής	Περιγραφή
==	Ίσο με
!=	Διάφορο του
>	Μεγαλύτερο από
<	Μικρότερο από
>=	Μεγαλύτερο από ή ίσο από
<=	Μικρότερο από ή ίσο από

Παραδείγματα:

Συνθήκη	Αποτέλεσμα
5!=16	**True** διότι το 5 δεν είναι ίσο με 16.
posotita<120	**True** αν η μεταβλητή **posotita** είναι μικρότερη από 120. **False** αν η μεταβλητή **posotita** είναι μεγαλύτερη ή ίση με 120.

9.2 Τελεστής ελέγχου συμμετοχής

Ο τελεστής ελέγχου συμμετοχής ελέγχει αν μία τιμή είναι μέλος μίας αλληλουχίας δεδομένων όπως **`list, tuple, set`** κλπ. Διακρίνουμε δύο τελεστές.

Τελεστής	Παράδειγμα	Λειτουργία

`in`	Εστω η λίστα:
	`students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας", "Ελένη"]`
	
	`"Νίκος" in students`
	`"Μαίρη" in students`
		Επιστρέφει την τιμή **`True`** αν η τιμή της σταθεράς ή μεταβλητής αριστερά του in βρίσκεται εντός της λίστας αλλιώς επιστρέφει **`False`**
		
		Επιστρέφει `True`
		Επιστρέφει `False`

`not in`	Εστω η λίστα: `students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας", "Ελένη"]`	Επιστρέφει την τιμή **`False`** αν η τιμή της σταθεράς ή μεταβλητής αριστερά του **`in`** βρίσκεται εντός της λίστας αλλιώς επιστρέφει **`True`**

	"Νίκος" in students "Μαίρη" in students	**Επιστρέφει False** **Επιστρέφει True**

9.3 Λογικοί Τελεστές

Η Python επιτρέπει την σύνθεση δύο ή περισσοτέρων συνθηκών με τη χρήση λογικών τελεστών. Οι λογικοί τελεστές εκτελούν λογικές πράξεις που τελικά έχουν ως αποτέλεσμα την τιμή **True** ή **False**.

Η λειτουργία των τελεστών αυτών παρουσιάζεται παραδοσιακά με πίνακες αληθείας όπως φαίνεται στους παρακάτω πίνακες.

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

A	not A
False	True
True	False

Παραδείγματα:

Παράσταση	Αποτέλεσμα
flower="Τριαντάφυλλο" and price<800	**True** αν και οι δύο συνθήκες είναι **True**, διαφορετικά **False**.
flower="Γαρύφαλλο" or price<300	**True** αν μια από τις δύο συνθήκες είναι **True**, διαφορετικά **False**.
not (Price<=200)	**True** αν η μεταβλητή price έχει τιμή μεγαλύτερη από 200, διαφορετικά **False**

9.4 Η εντολή `if`

Η εντολή **if** επιτρέπει την εκτέλεση μίας ή περισσοτέρων εντολών υπό συνθήκη. Με τον όρο συνθήκη θεωρούμε ότι μπορεί να είναι μία μεταβλητή που έχει μία λογική τιμή ή μία απλή συνθήκη ορίζεται με τη χρήση σχεσιακών τελεστών ή μία πιο πολύπλοκη συνθήκη με χρήση σχεσιακών και λογικών τελεστών καθώς και λογικών μεταβλητών.

Η απλούστερη σύνταξη της δομής **if** … είναι η εξής:

```
if συνθήκη:
        εντολή_1
```

```
            εντολή_2
            ……
      επόμενη_εντολή
```

Εάν η *συνθήκη* είναι αληθής(**True**), τότε η Python εκτελεί όλες τις εντολές (εντολή-1, εντολή-2 κλπ) οι οποίες στοιχίζονται εσωτερικότερα του **if**. Εάν όμως η *συνθήκη* είναι ψευδής (**False**), τότε εκτελείται η εντολή (**επόμενη _εντολή**) που πρέπει να έχει στοίχιση αντίστοιχη του **if**.

Παραδείγματα:

```
if score >= 20:
      amount = 10000
      print("You Win!",amount)
```

9.5 if … : … else

Η δομή **if** συνθήκη**:** …**else** και **if** …**..elif** …**. else** επιτρέπει τον ακολουθιακό έλεγχο πολλών συνθηκών. Η σύνταξή της έχει την ακόλουθη μορφή.

```
if συνθήκη1 :
      εντολή(ές)1
elif συνθήκη2 Then
      εντολή(ες)2
…
else:
      εντολή(ες)3
```

Οι δεσμευμένες λέξεις **else** και **elif** επιτρέπουν το ορισμό πολλών συνθηκών σε μια δομή ελέγχου και επιλογής **if**. Χρησιμοποιούνται μόνο μέσα σε αυτή τη δομή **if**… και ποτέ ανεξάρτητα από αυτήν.

Εξετάζονται οι συνθήκες με τη σειρά. Εάν η συνθήκη είναι αληθής, εκτελούνται οι εντολές που βρίσκονται αμέσως μετά τη συνθήκη αυτή, μέχρι την επόμενη δεσμευμένη λέξη **elif** ή **else**. Εάν όμως η συνθήκη είναι ψευδής τότε εκτελείται η αμέσως επόμενη **elif**, **else** με υπερπήδηση των εντολών που ακολουθούν τη ψευδή συνθήκη. Εάν καμία από τις παραπάνω συνθήκες δεν είναι αληθής τότε ενεργοποιείται το **else** και εκτελούνται οι εντολές που βρίσκονται μετά το **else**.

Παραδείγματα

```
number = 62   #Αρχική τιμή
if number <10:
      digits =1
elif number <100:
      digits =2
elif number<1000:
      digits=3
else
      digits =4
print(number, "αριθμός ψηφίων ",digits)
```

Ημεταβλητή number έχει τιμή 62. Στη δομή **if**…**:...else** η πρώτη συνθήκη (number <10) είναι ψευδής ενώ η δεύτερη συνθήκη **number <100** είναι αληθής και συνεπώς θα

εκτελεστεί η εντολή digits=2 και έτσι θα ολοκληρωθεί αυτή η δομή. Στη συνέχεια το πρόγραμμα θα εκτελέσει την εντολή **`print(number, "αριθμός ψηφίων ",digits)`**

Παράδειγμα 1.

Να γραφεί πρόγραμμα το οποίο διαβάζει το εισόδημα ενός φορολογούμενου και υπολογίζει τον φόρου που αναλογεί σε μια υποθετική εκκαθάριση φορολογίας εισοδήματος, όπου ο υπολογισμός του φόρου υπολογίζεται κλιμακωτά με βάση τον παρακάτω πίνακα.

Σύνολο εισοδήματος	Ποσοστό φορολογίας
10000	0%
(10000,25000]	5%
(25000,30000]	15%
(30000,40000]	30%
>40000	40%

```
poso=input("δώσε το φορολογητέο εισόδημα :")
poso=float(poso)
if poso<=10000:
   foros=10000*0
elif poso<=25000:
   foros=10000*0+(poso-10000) *0.05
elif poso<=30000:
   foros=10000*0+15000*0.05+((poso-25000)*0.15)
elif poso<=40000:
   foros=10000*0+15000*0.05+5000*.15+10000*0.3+((poso-30000)*0.30)
else:
   foros=10000*0+15000*0.05+5000*.15+10000*0.3+10000*((poso-40000)*0.40)
print("εισόδημα  ", poso, " φορος ", foros)
```

Το παραπάνω παράδειγμα μπορεί να γραφεί ισοδύναμα με εμφωλευμένα **`if`** ως εξής:

```
poso=input("δώσε το φορολογητέο εισόδημα :")
poso=float(poso)
if poso<=10000:
  foros=10000*0
else:
  if poso<=25000:
    foros=10000*0+(poso-10000)*0.05
  else:
      if poso<=30000:
           foros=10000*0+15000*0.05+((poso-25000)*0.15)
      else:
           if poso<=40000:
               foros=10000*0+15000*0.05+5000*0.15+10000*0.3+((poso-30000)*0.30)
           else:
               foros=10000*0+15000*0.05+5000*0.15+10000*0.3+10000*((poso-40000)*0.40)
print("εισόδημα  ", poso, " φορος ", foros)
```

Παράδειγμα 2.

Να γραφεί πρόγραμμα σε Python το οποία να διαβάζει από το πληκτρολόγιο τη θερμοκρασία του περιβάλλοντος και να εμφανίζει κατάλληλο μήνυμα ως εξής:

- Εάν η θερμοκρασία είναι μικρότερη ή ίση με 8 βαθμούς εμφανίζει το μήνυμα *Κάνει Παγωνιά*,

- Εάν η θερμοκρασία είναι μεγαλύτερη από 8 και μικρότερη ή ίση από 15 εμφανίζει το μήνυμα *Κάνει Ψύχρα*,
- Εάν η θερμοκρασία είναι μεγαλύτερη των 15 και μικρότερη ή ίση από 28 εμφανίζει το μήνυμα *Έχει καλό καιρό*,
- Εάν η θερμοκρασία είναι μεγαλύτερη από 28 τότε εμφανίζει το μήνυμα *Κάνει πολύ ζέστη*.

Λύση 1η

```
thermo=input("δώσε τη θερμοκρασία του περιβάλλοντος ")
thermo=float(thermo)
if thermo<=8:
    print("κάνει παγωνιά")
elif thermo<=15:
    print("κάνει ψύχρα")
elif thermo<=28:
    print("έχει καλό καιρό")
else:
    print("Κάνει πολύ ζέστη")
```

Λύση 2η με εμφωλευμένα **if**

```
thermo=input("δώσε τη θερμοκρασία του περιβάλλοντος ")
thermo=float(thermo)
if thermo<=8:
    print("κάνει παγωνιά")
else:
    if thermo<=15:
        print("κάνει ψύχρα")
    else:
        if thermo<=28:
            print("έχει καλό καιρό")
        else:
            print("Κάνει πολύ ζέστη")
```

Λύση 3η με διακριτά **if** και χρήση λογικών τελεστών

```
thermo=input("δώσε τη θερμοκρασία του περιβάλλοντος ")
thermo=float(thermo)
if thermo<=8:
    print("κάνει παγωνιά")
if thermo>8 and thermo<=15:
    print("κάνει ψύχρα")
if thermo>15 and thermo<=28:
    print("έχει καλό καιρό")
if thermo>28:
    print("Κάνει πολύ ζέστη")
```

10 Επαναλήψεις

Η Python παρέχει τη δυνατότητα επανάληψης ενός συνόλου εντολών ειδικών εντολών επανάληψης. Η Python παρέχει τις ακόλουθες περιπτώσεις επαναλήψεων:

- Επανάληψη ενός συνόλου εντολών για προκαθορισμένο αριθμό επαναλήψεων (εντολή **for**).
- Επανάληψη ενός συνόλου εντολών όσο μια συνθήκη είναι αληθής (εντολή **while**)

10.1 for ..in …

Η εντολή **for** είναι μια απλή δομή επανάληψης που επαναλαμβάνει ένα block εντολών ένα συγκεκριμένο πλήθος επαναλήψεων. Η εντολή **for..in..** είναι μία εντολή βρόχου, η οποία είναι σχεδιασμένη να λειτουργεί τόσες φορές όσες προκύπτει από μία αλληλουχία δεδομένων.

Η σύνταξη της δομής επανάληψης **for**… είναι της μορφής:

```
for μεταβλητή_for  in αλληλουχία_δεδομένων :
   εντολή-1
   εντολή-2
   ……
   εντολή-ν
επομενη_μετα_for_εντολή
```

Η **αλληλουχία_δεδομένων** είναι ένα σύνολο διακριτών δεδομένων που προκύπτουν από:

- Μία συνάρτηση που ορίζει ένα κλειστό διάστημα ακεραίων αριθμών.
- Τους χαρακτήρες ενός **string**, τα στοιχεία μίας λίστας (**list**), μίας πλειάδας (**tuple**), ενός λεξικού (**dictionary**), ενός συνόλου(**set**), ή μιας πιο πολύπλοκης δομής.

Η **μεταβλητή_for** είναι το όνομα μιας μεταβλητής η οποία θα πάρει τιμές μέσα από αυτό που ονομάσθηκε **αλληλουχία_δεδομένων**.

10.1.1 Η συνάρτηση range

Η Python με τη συνάρτηση **range** παρέχει στην εντολή **for** παραμέτρους για την εκτέλεση ενός block επανάληψης. Η σύνταξη και οι λειτουργίες της **range** είναι :

- **range(n)** όπου n ακέραιος αριθμός
  - **Λειτουργία:** Δημιουργεί την αλληλουχία διαδοχικών αριθμών 0,1,2,3,4,…,n-1
- **range(from, to)** όπου **from, to** ακέραιοι αριθμοι
  - **Λειτουργία:** Δημιουργεί την αλληλουχία διαδοχικών αριθμών στο διάστημα [from, to-1]
- **range(from, to, step)** όπου **from , to, step** ακέραιοι αριθμοι
  - **Λειτουργία:** Δημιουργεί την αλληλουχία αριθμών στο διάστημα [from, to-step] ως εξής : **from,from+step,from+step+step,**……**,to-step**]

Παράδειγμα:

Να υπολογιστεί το άθροισμα των ακέραιων αριθμών από το 1 μέχρι το 10.
Λύση:

```
for k in range (1,11):
    sum=sum+k
print(sum)
```

Παράδειγμα:
Να υπολογιστεί το άθροισμα των ακέραιων αριθμών από το 5 μέχριο το 50 με βήμα 5.

```
sum=0
for k in range (5,55,5):
    sum=sum+k
print(sum)
```

Παράδειγμα: Επανάληψη όπου η αλληλουχία προκύπτει από τα στοιχεία μίας λίστας.

```
onomata=['Νίκος', 'Γιώργος', 'Μαρία','Ελένη']
for onoma in onomata:
    print(onoma)
```

Στο παραπάνω παράδειγμα η μεταβλητή **onoma** θα λάβει διαδοχικά τις τιμές :

'Νίκος', 'Γιώργος', 'Μαρία','Ελένη'

Προκειμένου να γίνει έξοδος από μία επανάληψη **for** πριν αυτή ολοκληρωθεί χρησιμοποιείται η εντολή **break**.

10.2 while

Η εντολή **while** είναι μία εντολή επανάληψης στην οποία ο έλεγχος της επανάληψης πραγματοποιείται μέσω μίας λογικής πρότασης.

Η δομή της εντολή είναι:

```
while λογική_προταση:
        Εντολή-1
        Εντολή-2
        Εντολή-3
        ……..
        Εντολή-ν
Επόμενη εντολή
```

Σε αυτή τη μορφή επανάληψης, οι εντολές που βρίσκονται στο block επανάληψης θα εκτελεστούν για μη προκαθορισμένο αριθμό επαναλήψεων.

Η επανάληψη τερματίζει με βάση την τιμή της λογικής πρότασης.

Με το όρο **λογική_πρόταση** θεωρείται ότι μπορεί να είναι:

- Μία μεταβλητή που μπορεί να πάρει τιμές **True** ή **False** ή

- μία απλή συνθήκη με χρήση σχεσιακών τελεστών ή
- μία σύνθετη συνθήκη με χρήση λογικών τελεστών ή.
- η σταθερά **True.**

Ο βρόχος **While** λειτουργεί με τον εξής τρόπο:

Εάν η συνθήκη είναι **False** (ψευδής) τότε η Python αγνοεί όλες τις εντολές που βρίσκονται στο εσωτερικό του βρόχου while και συνεχίζει την επόμενη εντολή εκτός **while** η οποία πρέπει να βρίσκεται στοιχισμένη με το **while**.

Εάν η λογική πρόταση είναι **True** (αληθής) τότε εκτελούνται οι εντολές που βρίσκονται στο εσωτερικό του βρόχου **while** (τουλάχιστον μια φορά). **Όταν η συνθήκη είναι πάντα αληθής τότε οι εντολές του βρόχου επαναλαμβάνονται επ άπειρον (ατέρμονας βρόχος).** Προκειμένου να τερματιστεί η επανάληψη πρέπει, με κάποιο τρόπο, να αλλάξουν κάποιες τιμές εντός της επανάληψης οι οποίες θα επιδράσουν στην λογική πρόταση προκειμένου αυτή να γίνει **False** ή να χρησιμοποηθεί η εντολή **break**

Παράδειγμα:1

Να γραφεί πρόγραμμα σε Python το οποίο να υπολογίζει και να εμφανίζει στην οθόνη το άθροισμα των περιττών αριθμών από 1 έως και 100.

```
k=1       # αρχική τιμή του μετρητή
sum=0     # μηδενισμό αθροιστή
while k<=100:  # οσο ο μετρητή είναι <=100
    sum=sum+k  # προσθέτει στον αθροιστή το περιεχόμενο του μετρητή
    k=k+2      # αυξηση του μετρητή κατά 2
print(sum)     # εκτύπωση του αθροίσματος
```

Παράδειγμα:2

Να γραφεί πρόγραμμα το οποίο να διαβάζει από την οθόνη:

- Το επώνυμο ενός φοιτητή
- Την βαθμολογία του σε ένα μάθημα ελέγχοντας ότι ο βαθμός περιέχεται στο διάστημα [0-10].

Η διαδικασία επαναλαμβάνεται μέχρι να δοθεί ως επώνυμο η λέξη “ΤΕΛΟΣ”

Μετά το τέλος εισαγωγής στοιχείων το πρόγραμμα εμφανίζει:

- Το ποσοστό των αρίστων φοιτητών (με βαθμό >=8.5)
- Το ποσοστό των επιτυχόντων φοιτητών (με βαθμό >=5)
- Το ονοματεπώνυμο του φοιτητή με τη μεγαλύτερη βαθμολογία

Λύση 1η

```
onoma='' # δίδεται αρχική τιμή για τον έλεγχο του while
pl_ar=0   # πλήθος αρίστων
pl_ep=0   # πλήθος επιτυγχόντων
plithos=0 # πλήθος φοιτητών
meg=-1    # μέγιστος βαθμός
kal=" "   # ονομα του φοιτητή με τον καλύτερο βαθμό
while onoma!="ΤΕΛΟΣ":
    onoma=input("δώσε το όνομα ή ΤΕΛΟΣ ")
    if onoma!="ΤΕΛΟΣ":
        vathmos=-1
        while vathmos<0 or vathmos>10:
            vathmos=float(input("δώσε τον βαθμό [0,10]"))
```

```
        if vathmos>meg:
            meg=vathmos
            kal=onoma
        plithos=plithos+1
        if vathmos>8.5:
            pl_ar =pl_ar+1
        if vathmos>=5:
            pl_ep=pl_ep+1
pos_ar=100*pl_ar/plithos
pos_ep=100*pl_ep/plithos
print("ποσοστό αρίστων ",pos_ar,"%")
print("ποσοστό επιτυγχόντων ",pos_ep,"%")
print("τον μεγαλύτερο βαθμό πήρε ο/η ",kal, " με βαθμό ", meg)
```

Λύση 2η

```
pl_ar=0    # πλήθος αρίστων
pl_ep=0    # πλήθος επιτυγχόντων
plithos=0  # πλήθος φοιτητών
meg=-1     # μέγιστος βαθμός
kal=" "    # ονομα του φοιτητή με τον καλύτερο βαθμό
while True:
    onoma=input("δώσε το όνομα ή ΤΕΛΟΣ ")
    if onoma!="ΤΕΛΟΣ":
        while True:
            vathmos=float(input("δώσε τον βαθμό [0,10]"))
            if vathmos>=0 and vathmos<=10:
                break # αν ο βαθμός είναι στο διάστημα [0,10]
                      # πραγματοποιείται έξοδος από το while
                      # εισαγωγής βαθμολογίας
        if vathmos>meg:
            meg=vathmos
            kal=onoma
        plithos=plithos+1
        if vathmos>8.5:
            pl_ar =pl_ar+1
        if vathmos>=5:
            pl_ep=pl_ep+1
    else:
        break  # αν όνομα είναι ΤΕΛΟΣ
               # πραγματοποιείται έξοδος από το while εισαγωγής δεδομένων

pos_ar=100*pl_ar/plithos
pos_ep=100*pl_ep/plithos
print("ποσοστό αρίστων ",pos_ar,"%")
print("ποσοστό επιτυγχόντων ",pos_ep,"%")
print("τον μεγαλύτερο βαθμό πήρε ο/η ",kal, " με βαθμό ", meg)
```

11 Βασικές μέθοδοι χειρισμού string

Η Python διαθέτει ένα αριθμό μεθόδων χειρισμού των **string** που διακρίνουνται σε μεθόδους τροποιητικές, ελέγχου ή αναζήτησης. Χρησιμοποιούνται σε συνδιασμό με το όνομα που δόθηκε στο συγκεκριμένο **string** κατά την ανάθεση τιμής.

Δηλ.

onomastring.methodos(orisma)

Παρακάτω θα αναφέρουμε τις κυριότερες μεθόδους **string.**

11.1 .capitalize

Επιστρέφει ένα ***string*** μετατρέποντας τον πρώτο χαρακτήρα ενός ***string,*** εφόσον είναι χαρακτήρας αλαφβήτου, σε κεφαλαίο χαρακτήρα. Το αρχικό ***string*** παραμένει αναλλοίωτο.

11.2 .join

Σύνταξη : `stringA.join(akolouthia)`

Η μέθοδος αυτή δέχεται ως όρισμα μία ακολουθία **(list, tuple,set)** που αποτελείται από **string** και επιστρέφει το ένα **string** το οποίο αποτελείται από την συννένωση καθένα από τα **string** που περιέχονται στην σχετική ακολουθία διαχωρίζοντας τα μεταξύ τους με βάση το **stringA**.

11.3 .upper

Επιστρέφει ένα ***string*** μετατρέποντας τους πεζούς ενός ***string*** σε κεφαλαίους χαρακτήρες. Το αρχικό ***string*** παραμένει αναλλοίωτο.

Πχ έστω:
```
met = 'Γεωπονικό Πανεπιστήμιο Αθηνών'
print(met.upper())
print(met)
```

Η εκτέλεση θα έχει ως αποτέλεσμα

```
ΓΕΩΠΟΝΙΚΌ ΠΑΝΕΠΙΣΤΉΜΙΟ ΑΘΗΝΏΝ
Γεωπονικό Πανεπιστήμιο Αθηνών
```

11.4 .lower

Επιστρέφει ένα string μετατρέποντας τα τους κεφαλαίους χαρακτήρες ενός ***string*** σε πεζούς. Το αρχικό ***string*** παραμένει αναλλοίωτο.

Πχ έστω:
```
met = 'ΓΕΩΠΟΝΙΚΌ ΠΑΝΕΠΙΣΤΉΜΙΟ ΑΘΗΝΏΝ'
 print(met.lower())
print(met)
```

Η εκτέλεση θα έχει ως αποτέλεσμα

```
Γεωπονικό Πανεπιστήμιο Αθηνών
ΓΕΩΠΟΝΙΚΌ ΠΑΝΕΠΙΣΤΉΜΙΟ ΑΘΗΝΏΝ
```

11.5 .count

Σύνταξη : `stringA.count(stringB)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα ***string*** πχ ***stringB*** και επιστρέφει το πλήθος των επαναλήψεων του ορίσματος μέσα στο ***stringA***.

Πχ έστω:

```
met = 'ΓΕΩΠΟΝΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ'
print(met.count('Ο'))
```

Η εκτέλεση θα έχει ως αποτέλεσμα

3

11.6 .find

Σύνταξη 1η : `stringA.find(stringB)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα string πχ ***stringB*** και επιστρέφει τη θέση του ορίσματος μέσα στο ***stringA***.

Σύνταξη 2η : `stringA.find(stringB, pos)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα string πχ ***stringB*** και έναν ακέραιο αριθμό πχ **pos** και επιστρέφει τη θέση του ορίσματος μέσα στο stringA μετά τη θέση ***pos*** του ***stringA***

Εάν δεν υπάρχει το **stringB** τότε η τιμή που επιστρέφει είναι -1

11.7 .replace

Σύνταξη : `stringA.replace(stringB, stringC)`

Η μέθοδος αυτή δέχεται δύο string ορίσματα πχ ***stringB*** και **stringC** και αντικαθιστά το **stringB** με το ***stringC***

11.8 .strip

Σύνταξη 1η : `stringA.strip()`

Η μέθοδος αυτή επιστρέφει ένα ***string*** στο οποίο έχουν αφαιρεθεί όλοι οι λεγόμενοι *λευκοί χαρακτήρες* δηλαδή τα κενά (**space**), οι στηλοθέτες (**\t**), οι χαρακτήρες αλλάγής γραμμής (**\n**) που βρίσκονται στην αρχή ή στο τέλος του ***stringA***.

Σύνταξη 2η : `stringA.strip(char)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα χαρακτήρα (***char***) και επιστρέφει ένα ***string*** στο οποίο έχουν αφαιρεθεί όλοι οι χαρακτήρες (***char***) που βρίσκονται στην αρχή ή στο τέλος του ***stringA***.

11.9 .rstrip

Σύνταξη 1η : `stringA.rstrip()`

Η μέθοδος αυτή επιστρέφει ένα ***string*** στο οποίο έχουν αφαιρεθεί όλοι οι λεγόμενοι *λευκοί χαρακτήρες* δηλαδή τα κενά (***space***), οι στηλοθέτες (**\t**), οι χαρακτήρες αλλαγής γραμμής (**\n**) που βρίσκονται στο τέλος του ***stringA***.

Σύνταξη 2η : `stringA.rstrip(char)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα χαρακτήρα (***char***) και επιστρέφει ένα string στο οποίο έχουν αφαιρεθεί όλοι οι χαρακτήρες (***char***) που βρίσκονται στο τέλος του ***stringA***.

11.10.lstrip

Σύνταξη 1η : `stringA.lstrip()`

Η μέθοδος αυτή επιστρέφει ένα ***string*** στο οποίο έχουν αφαιρεθεί όλοι οι λεγόμενοι *λευκοί χαρακτήρες* δηλαδή τα κενά (***space***), οι στηλοθέτες (\t), οι χαρακτήρες αλλαγής γραμμής (**\n**) που βρίσκονται στην αρχή του ***stringA***.

Σύνταξη 2η : `stringA.lstrip(`**`char`**`)`

Η μέθοδος αυτή δέχεται ως όρισμα ένα χαρακτήρα (***char***) και επιστρέφει ένα ***string*** στο οποίο έχουν αφαιρεθεί όλοι οι χαρακτήρες (***char***) που βρίσκονται στην αρχή του ***stringA***.

11.11 .split

Η μέθοδος αυτή δέχεται ως όρισμα ένα ***string*** (συνήθως ένα χαρακτήρα) και δημιουργεί μία λίστα η οποία περιέχει ως στοιχεία τα μέρη του ***string*** τα οποία διαχωρίζονται με το όρισμα.

Π.χ έστω το πρόγραμμα

```
alfa="Ιερά Οδός 75 11855 Αθήνα"
lexeis=alfa.split(" ")
for lexi in lexeis:
     print(lexi)
```

Το αποτέλεσμα της επεξεργασίας θα είναι:

```
Ιερά
Οδός
75
11855
Αθήνα
```

11.12 .isalnum

Η μέθοδος αυτή είναι μία μέθοδος ελέγχου. Δεν δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο αλφαβητικούς χαρακτήρες ήαριθμητικά ψηφία και τουλάχιστο ένα αλφαβητικό χαρακτήρα. Σε κάθε άλλη περίπτωση επιστρέφει **False**.

11.13 .isalpha

Η μέθοδος αυτή είναι μία μέθοδος ελέγχου. Δεν δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο αλφαβητικούς χαρακτήρες αλλιώς επιστρέφει **False**.

11.14 .isdigit

Η μέθοδος αυτή είναι μία μέθοδο ελέγχου. Δε δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο αριθμητικούς χαρακτήρες αλλιώς επιστρέφει **False**.

11.15 .islower

Η μέθοδος αυτή είναι μία μέθοδο ελέγχου. Δεν δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο πεζούς αλφαβητικούς χαρακτήρες . αλλιώς επιστρέφει **False**.

11.16 .isspace

Η μέθοδος αυτή είναι μία μέθοδο ελέγχου. Δεν δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο χαρακτήρες κενούς (space) ή στηλοθέτη (tab) . αλλιώς επιστρέφει **False**.

11.17 .isupper

Η μέθοδος αυτή είναι μία μέθοδο ελέγχου. Δεν δέχεται ως όρισμα. Επιστρέφει την τιμή **True** αν το υπό έλεγχο **string** περιέχει μόνο κεφαλαίους αλφαβητικούς χαρακτήρες. αλλιώς επιστρέφει **False**.

11.18 .format

Η μέθοδος **format()** χρησιμοποιείται στη διαμόρφωση ενός string μέσα από τη εισαγωγή παραμέτρων, σε αυτό και σε συγκεκριμένες θέσεις.

Η μέθοδος αυτή λειτουργεί ως εξής: Μέσα ένα **string** τοποθετούμε σε επιθυμητές θέσεις k ζεύγη αγκίστρων {}. Η μέθοδος δέχεται ως ορίσματα μια σειρά από k παραμέτρους. Οι παράμετροι μπορεί να είναι σταθερές ή μεταβλητές. Η μέθοδος format αναγνωρίζοντας τον τύπο των παραμέτρων που δέχεται, διαμορφώνει το string τοποθετώντας τις σταθερές ή τις τιμές των μεταβλητών στις αντίστοιχες θέσεις αγκίστρων παραλείποντας τα άγκιστρα. Το περιεχόμενων των αγκίστρων μπορεί να είναι κενό ή να περιέχει παραμέτρους για την επιμέρους διαμόρφωση του string. Ως μέθοδος επιστρέφει πάντα ένα **string**.

Η μέθοδος αυτή είναι χρήσιμη για τη δημιουργία ομοιόμορφης εκτύπωση όπως επίσης για το σχηματισμό εντολών πχ. SQL στο χειρισμό Βάσεων Δεδομένων.

Εντός των αγκίστρων μπορεί να υπάρχουν δείκτες οι οποίοι θα καθορίζουν το ποια παράμετρος θα εισαχθεί σε ποίο ζεύγος άγκιστρων. Η αρίθμηση των παραμέτρων ξεκινά από το 0.

Μπορούμε να διαμορφώσουμε ένα δεδομένο αριθμητικού τύπου εισάγωντας μέσα στο άγκιστρο το χαρακτηριστικό string διαμόρφωσης αριθμητικών δεδομένων όπως είδαμε στην παράγραφο 9.1.2. Στην περίπτωση αυτή θα ορισθεί ένας αριθμός ως δείκτης που θα αναφέρεται στην παράμετρο στη συνέχεια θα πρέπει να ακολουθεί η άνω και κάτω τελεία (:) και τέλος θα τοποθετηθεί το **string** διαμόρφωσης.

Παράδειγμα-1.

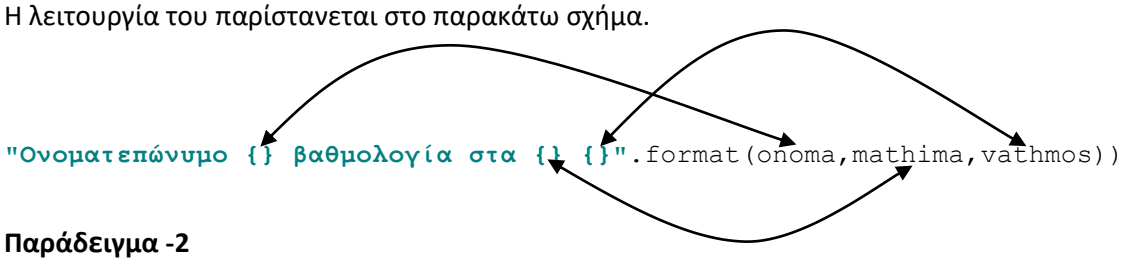
Εστω το πρόγραμμα:

```
onoma="Γεωργίου Ιωάννης"
mathima="Μαθηματικά"
vathmos=8.5

mathima2="Πληροφορική"
vathmos2=6

print ("Ονοματεπώνυμο {} βαθμολογία στα {} {}".format(onoma,mathima,vathmos))
```

Η εκτέλεσή του θα έχει ως αποτέλεσμα

```
Ονοματεπώνυμο Γεωργίου Ιωάννης βαθμολογία Μαθηματικά 8.5
```

Η λειτουργία του παρίστανεται στο παρακάτω σχήμα.



Παράδειγμα -2

Εστω το πρόγραμμα:

```
onoma="Γεωργίου Ιωάννης"
mathima="Μαθηματικά"
vathmos=8.5

print ("Ονοματεπώνυμο {} βαθμολογία στα {} {}".format(onoma,mathima,vathmos))

print ("Ονοματεπώνυμο {0:} βαθμολογία {2} στα {1:}".format(onoma,mathima,vathmos))
```

Η εκτέλεσή του θα έχει ως αποτέλεσμα

```
Ονοματεπώνυμο Γεωργίου Ιωάννης βαθμολογία στα Μαθηματικά 8.5
Ονοματεπώνυμο Γεωργίου Ιωάννης βαθμολογία  8.5 στα Μαθηματικά
```

Παρατηρούμε ότι παρόλο που η σειρά των παραμέτρων στο **format** δεν άλλαξε. Οι παράμετροι τοποθετήθηκαν σε σειρά με βάση τον δείκτη του αγκίστρου.

Παράδειγμα -3

Εστω το πρόγραμμα:

```
onoma="Γεωργίου Ιωάννης"
mathima="Μαθηματικά"
vathmos=8.5
print ("Ονοματεπώνυμο {0} βαθμολογία {2} στα {1}".format(onoma,mathima,vathmos))

onoma="Πέτρου Γεώργιος"
mathima="Πληροφορική"
vathmos=5
print ("Ονοματεπώνυμο {0} βαθμολογία {2:5.1f} στα {1}".format(onoma,mathima,vathmos))
```

Η εκτέλεσή του θα έχει ως αποτέλεσμα

```
Ονοματεπώνυμο Γεωργίου Ιωάννης βαθμολογία 8.5 στα Μαθηματικά
Ονοματεπώνυμο Πέτρου Γεώργιος βαθμολογία  5.0 στα Πληροφορική
```

Παρατηρούμε ότι μπορούμε να διαμορφώσουμε τα αριθμητικά δεδομένα των παραμέτρων χρησιμοποιωντας τους χαρακτήρες διαμόρφωσης αριθμητικών δεδομένων.

12 Βασικές μέθοδοι και συναρτήσεις χειρισμού λίστας

12.1 .append

Με τη μέθοδο **append** επιτυγχάνεται εισαγωγή ενός στοιχείου σε μία υπάρχουσα λίστα. Το στοιχείο που θα εισαχθεί να τοποθετηθεί στο τέλος της λίστας.

Σύνταξη:
lista.append(στοιχείο)

Παράδειγμα:
Έστω η λίστα **students** με τις παρακάτω τιμές:

students =["Νίκος","Γιώργος","Μαρία","Πέτρος","Ανδρέας"]
Μετά την εκτέλεσης της: students.append("Ελένη") η λίστα θα είναι η:

["Νίκος","Γιώργος","Μαρία","Πέτρος","Ανδρέας","Ελένη"]

12.2 .insert

Η μέθοδος **insert** δέχεται δύο ορίσματα. Το πρώτο όρισμα ορίζει την τάξη (θέση) μέσα στη λίστα ενώ το δεύτερο όρισμα είναι το προς εισαγωγή στοιχείο.

Σύνταξη:
lista.insert(δεικτης, στοιχείο)

Έστω η λίστα **students** με τις παρακάτω τιμές :

students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας", "Ελένη"]
Η εντολή **students.insert(3, "Τάσος")** θα εισάγει στο στοιχείο **"Τάσος"** στην 4η θέση (τάξη 3) και η το περιεχόμενο της λίστας θα είναι:

["Νίκος", "Γιώργος", "Μαρία", "Τάσος", "Πέτρος", "Ανδρέας", "Ελένη"]

12.3 .remove

Με τη μέθοδο **remove** επιτυγχάνεται διαγραφή ενός στοιχείου σε μία υπάρχουσα λίστα.
Εάν το στοιχείο υπάρχει περισσότερες από μία φορές διαγράφει το πρώτο που θα βρει
Εάν το στοιχείο δεν υπάρχει δημιουργείται λάθος εκτέλεσης του προγράμματος

Σύνταξη:
lista.remove(στοιχείο)

Παράδειγμα:

Έστω η λίστα **students** με τις παρακάτω τιμές :
students = ["Νίκος","Γιώργος","Μαρία","Πέτρος","Ανδρέας","Ελένη"]
Η εντολή **students.remove("Μαρία")** θα διαγράψει το στοιχείο με όνομα **"Μαρία".**

12.4 .index

Η μέθοδος **index** δέχεται ως όρισμα ένα στοιχείο μιας λίστας και επιστρέφει την τάξη του στη λίστα.

Έστω η λίστα `students` με τις παρακάτω τιμές

```
students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας", "Ελένη"]
```

Η εντολή **`print(students.index("Ελένη")`** θα εμφανίσει τον αριθμό **5**. Δεδομένου ότι η τάξη του πρώτου στοιχείου είναι το 0 (μηδέν)

12.5 .sort

Η μέθοδος **sort** ταξινομεί μία λίστα.

Σύνταξη:

```
lista.sort()                # ταξινόμηση αύξουσα
lista.sort(reverse=True)    # ταξινόμηση φθίνουσα
```

12.6 .reverse

Η μέθοδος **reverse** αντιστρέφει τη σειρά των στοιχείων μιας λίστας

Σύνταξη:

```
lista.reverse()
```

12.7 .count

Η μέθοδος **count** επιστρέφει το πλήθος των επαναλήψεων ενός στοιχείου μιας λίστας.

Σύνταξη:

```
lista.count(στοιχειο)
```

12.8 Βασικές συναρτήσεις λίστας

α/α	Ονομα συνάρτησης	Λειτουργία
1	**len**	Δέχεται ως όρισμα μία λίστα και επιστρέφει το πλήθος των στοιχείων της λίστας
2	**min**	Δέχεται ως όρισμα μία λίστα και επιστρέφει το μικρότερο των στοιχείων της λίστας
3	**max**	Δέχεται ως όρισμα μία λίστα και επιστρέφει το μεγαλύτερο των στοιχείων της λίστας
4	**sum**	Δέχεται ως όρισμα μία λίστα στην οποία τα στοιχεία της είναι αριθμοί και επιστρέφει το άθροισμα στοιχείων της λίστας

Παραδείγματα

Εστω το παρακάτω πρόγραμμα:

```
students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας"]
print(students)
students.reverse()
print(students)
```

```
print(students.index("Νίκος"))
students.append("Ελένη")
print(students)
print(students[4])
students.insert(3,"Τάσος")
print(students)
students.remove("Μαρία")
print(students)
del(students[2])
print("sorting")
students.sort()
print(students)
print("reverse sort")
students.sort(reverse=True)
print(students)
print (min(students),max(students))
```

Η εκέλεσή του θα έχει ως αποτέλεσμα το παρακάτω

```
['Νίκος', 'Γιώργος', 'Μαρία', 'Πέτρος', 'Ανδρέας']
['Ανδρέας', 'Πέτρος', 'Μαρία', 'Γιώργος', 'Νίκος']
4
['Ανδρέας', 'Πέτρος', 'Μαρία', 'Γιώργος', 'Νίκος', 'Ελένη']
Νίκος
['Ανδρέας', 'Πέτρος', 'Μαρία', 'Τάσος', 'Γιώργος', 'Νίκος',
'Ελένη']
['Ανδρέας', 'Πέτρος', 'Τάσος', 'Γιώργος', 'Νίκος', 'Ελένη']
sorting
['Ανδρέας', 'Γιώργος', 'Ελένη', 'Νίκος', 'Πέτρος']
reverse sort
['Πέτρος', 'Νίκος', 'Ελένη', 'Γιώργος', 'Ανδρέας']
Ανδρέας Πέτρος
```

12.9 Αναζήτηση και προσπέλαση σε λίστα

12.9.1 Έλεγχος ύπαρξης στοιχείου

Ο έλεγχος ύπαρξης ενός στοιχείου σε μία λίστα επιτυγχάνεται με τον τελεστή **in** και με χρήση της **if.**

Παράδειγμα:

Έστω η λίστα **students**

```
students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας"]
```

Ζητείται ο χρήστης να δίδει ένα όνομα και το πρόγραμμα να ελέγχει αν το όνομα αυτό υπάρχει στη λίστα εμφανίζοντας την τάξη του στη λίστα. Το πρόγραμμα θα έχει την παρακάτω μορφή:

```
students = ["Νίκος", "Γιώργος", "Μαρία", "Πέτρος", "Ανδρέας"]
onoma=input(" Δώσε το όνομα ")
if onoma in students:
    print("το όνομα ",onoma)
    print("βρίσκεται στη θέση ", students.index(onoma))
else:
    print("δεν υπάρχει αυτό το όνομα")
```

12.9.2 Πολλαπλή προσπέλαση στοιχείων

Εκτός της προσπέλαση μέσω ενός δείκτη που δηλώνει την θέση ενός στοιχείου σε μία λίστα που επιστρέφει ένα μόνο στοιχείο της λίστας υπάρχει και συνδυασμός δείκτη με το σύμβολο ‘:’ ή ‘::’ . Με τον τρόπο αυτό ο χρήστη μπορεί να προσπελάση περισσότερα του ενός στοιχεία μίας λίστας.

Παρακάτω δίδονται οι διάφοροι συνδυασμοί σύνταξης.

`lista[:n]`
Επιστρέφει τα **n** πρώτα στοιχεία της λίστας

`lista[-n:]`
Επιστρέφει τα **n** τελευταία στοιχεία της λίστας

`lista[n:]`
Επιστρέφει όλα τα στοιχεία της λίστας μετά την αφαίρεση των **n** πρώτων

`lista[n:-m]`
Επιστρέφει τα στοιχεία της λίστας αφαιρώντας το πρώτα **n** και τα τελευταία **m** στοιχεία

`lista[n:m:k]`
Επιστρέφει τα στοιχεία της λίστας από το **n**-στο στοιχείο μέχρι το **m**-στο και με βήμα **k**.

Αν n>m τότε το πρέπει να είναι k<0

`lista[::n]`
Προσπέλαση των στοιχείων της λίστας με επιλογή ανά **n** στοιχεία

Παράδειγμα:

```
li=[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36,
38]
print("εκτύπωση  των 5 πρώτων στοιχείων της λίστας")
print(li[:5]) #[0, 2, 4, 6, 8]
print("εκτύπωση των  στοιχείων της λίστας αφαιρώντας τα 5 πρώτα")
print(li[5:])# [10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
print("εκτύπωση ανά 3 των στοιχείων της λίστας")
print(li[::3])#[0, 6, 12, 18, 24, 30, 36]
print("εκτύπωση των στοιχείων της λίστας αφαιρώντας το 2 πρώτα και τα 3
τελευταία")
print(li[2:-3])#[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34]
print("εκτύπωση των 4 τελευταίων στοιχείων της λίστας")
print(li[-4:])#[32, 34, 36, 38]
print("εκτύπωση των στοιχείων από το 10ο ανά δύο μέχρι το 2ο")
print(li[10:2:-2]) #[20, 16, 12, 8]
print("εκτύπωση των στοιχείων από το 5ο ανά δύο μέχρι το 15ο")
print(li[5:15:2]) #[10, 14, 18, 22, 26]
```

12.10 Πράξεις με λίστες

12.10.1 Πρόσθεση λιστών

Με την έννοια πρόσθεση λιστών εννοούμε τη συνένωση δύο η περισσοτέρων λιστών.

```
a=[1,2,3,4]
b=[5,6,7,8,9]
x=a+b
```

Η λίστα x θα έχει τη μορφή

```
[1,2,3,4,5,6,7,8,9]
```

12.10.2 Πολλαπλασιασμός λιστών

Με την έννοια *πολλαπλασιασμού μιας λίστας με έναν ακέραιο* εννοούμε τη επανάληψη των στοιχείων όσες φορές προκύπτει από τον ακέραιο.

```
a=[1,2,3]
x=a*3
```

Η νέα λίστα **x** θα έχει τιμή:

[1,2,3,1,2,3,1,2,3]

13 Βασικές μέθοδοι χειρισμού λεξικών

Όπως έχουμε αναφέρει μία μεταβλητή(αντικείμενο) τύπου **dictionary** είναι συλλογή δεδομένων. Κάθε στοιχείο της συλλογής αυτής αποτελείται από δύο μέρη. Το πρώτο μέρος αναφέρεται ως **κλειδί (key)** και το δεύτερο μέρος ως **τιμή(value)**.Το κλειδί χρησιμοποιείται για τον εντοπισμό μιας τιμής και σε κάθε κλειδί αντιστοιχεί μία τιμή

13.1 .clear

Η μέθοδος αυτή διαγράφει το περιεχόμενο του λεξικού

Σύνταξη : **dictionary.clear()**

13.2 .get

Η μέθοδος αυτή επιστρέφει την τιμή (**value**) ενός κλειδιού με ονομα **kleidi** στο λεξικό **dictionary**

Σύνταξη : **timi=dictionary.get(kleidi)**

Παράδειγμα:

Έστω το πρόγραμμα:

```
lexiko={1:'Νίκος', 2:'Γιώργος', 3:'Μαρία',4:'Ελένη'}
kleidi=3
onoma=lexiko.get(kleidi)
print(onoma)
```

Η εκτέλεσή του θα έχει ώς αποτέλεσμα:

```
Μαρία
```

13.3 .keys

Η μέθοδος αυτή επιστρέφει όλα τα κλειδιά ενός λεξικού σ'ένα αντικείμενο που κάθε στοιχείο του αντικειμένου αυτού είναι ιδίου τύπου με το κλειδί όπως ορίσθηκε στο λεξικό.

Παράδειγμα:

```
lexiko={1:'Νίκος', 2:'Γιώργος', 3:'Μαρία',4:'Ελένη'}
onomata=lexiko.items()
kleidia=lexiko.keys()

dict_items([(1, 'Νίκος'), (2, 'Γιώργος'), (3, 'Μαρία'), (4, 'Ελένη')])
dict_keys([1, 2, 3, 4])
```

13.4 .items

Η μέθοδος αυτή επιστρέφει όλα τα κλειδιά & τιμές ενός λεξικού σ'ένα αντικείμενο που κάθε στοιχείο του αντικειμένου αυτού είναι μια **tuple**.

Παράδειγμα:

```
lexiko={1:'Νίκος', 2:'Γιώργος', 3:'Μαρία',4:'Ελένη'}
periexomeno=lexiko.items()
kleidia=lexiko.keys()
print(lexiko)

print(type(periexomeno))
print(periexomeno)
print(type(kleidia))
print(kleidia)
```

Το αποτέλεσμα της εκτέλεσης του παραπάνω προγράμματος είναι:

```
{1: 'Νίκος', 2: 'Γιώργος', 3: 'Μαρία', 4: 'Ελένη'}
<class 'dict_items'>
dict_items([(1, 'Νίκος'), (2, 'Γιώργος'), (3, 'Μαρία'), (4, 'Ελένη')])
<class 'dict_keys'>
dict_keys([1, 2, 3, 4])
```

13.5 .values

Η μέθοδος αυτή επιστρέφει τις τιμές(**values**) ενός λεξικού σ' ένα αντικείμενο τύπου **dict_values** που κάθε στοιχείο του αντικειμένου αυτού είναι ιδίου τύπου με την τιμή(**value**) όπως ορίσθηκε στο λεξικό.

Παράδειγμα
έστω το πρόγραμμα:

```
lexiko={1:'Νίκος', 2:'Γιώργος', 3:'Μαρία',4:'Ελένη'}
times=lexiko.values()
print(times)
```

Η εκτέλεσή του έχει ώς αποτέλεσμα:

```
dict_values(['Νίκος', 'Γιώργος', 'Μαρία', 'Ελένη'])
```

14 Παραδειγμα χειρισμού λεξικού

Εστω ότι μία εμπορική εταιρεία καταγράφει για κάθε πωλητή την αξία κάθε πώλησης που πραγματοποίησε σε μία ημέρα. Να γραφεί πρόγραμμα το οποίο για κάθε πωλητή να διαβάζει από την οθόνη:

το όνομα του πωλητή
και την αξία πωλήσης

Να αποθηκεύει τα παραπάνω στοιχεία σε ένα λεξικό.

Η διαδικασία να επαναλαμβάνει μέχρις ότου να δοθεί ως ονομα του πωλητή η λέξη **ΤΕΛΟΣ**.

Μετά το τέλος εισαγωγής των στοιχείων να εμφανίζει για κάθε πωλητή τη συνολική αξία πωλήσεων του.

Λύση

```
poliseis={} # Δημιουργία κενού λεξικού {key:value,....}
            # key:επώνυμο
            # value: συνολική αξία πωλήσεων
epan=True
while epan:
    eponymo=input("επώνυμο πωλητή ή ΤΕΛΟΣ ")
    if eponymo!="ΤΕΛΟΣ":
        axia=float(input("Αξία πωλήσεων "))
        if eponymo in poliseis: # αν υπάρχει εγγραφή στο λεξικό με αυτό το κλειδί
            s_axia=poliseis.get(eponymo) # διαβάζεται η τιμή (value) της εγγραφής
            s_axia= s_axia + axia # αυξάνεται κατά το περιεχόμενο της μεταβλητής axia
            poliseis[eponymo]=s_axia # η εγγραφή με το αντίστοιχο κλειδί
                                     #  ενημερώνεται με τη νέα τιμή
        else:
            poliseis[eponymo]=axia # εισάγεται μία νέα εγγραφή
    else: epan=False
for politi in poliseis:
    print(politi, poliseis.get(politi)) # για κάθε κλειδί του λεξικού
                                        # εμφανίζει το κλειδί και την τιμή της
                                        #εγγραφής (επωνυμο και αξία πωλήσεων)
```

15 Βασικές μέθοδοι χειρισμού συνόλων και πράξεις συνόλων

15.1 Βασικές μέθοδοι

.add

add προσθέτει ένα στοιχείο σε σύνολο

`myset.add(στοιχείο)`

.update

update προσθέτει σε υπάρχον σύνολο τα στοιχεία ενός **string**

`myset.update(`**`string`**`)`

.clear

clear διαγράφει το περιεχόμενο του συνόλου

`myset.clear()`

.discard

discard διαγράφει ένα συγκεκριμένο στοιχείο από το σύνολο

```
myset.discard(στοιχείο)
```

.remove

remove διαγράφει ένα συγκεκριμένο στοιχείο από το σύνολο

```
myset.remove(στοιχείο)
```

15.2 Πράξεις μεταξύ συνόλων

.union

union Ένωση δύο συνόλων

```
newset=myset.union(oldset)
```

.interection

intersection Τομή δύο συνόλων

```
newset=myset.intersection(oldset)
```

.difference

difference Διαφορά δύο συνόλων

```
newset=myset.difference(oldset)
```

.symmetric_difference

symmetric_difference Συμμετρική διαφορά δύο συνόλων

```
newset=myset.symmetric_difference(oldset)
```

In έλεγχος ύπαρξης στοιχείου σε σύνολο

```
if x in myset
```

>= έλεγχος για υποσύνολο

issubset

<= έλεγχος για υπερσύνολο

issuperbset

Παράδειγμα

Το παρακάτω πρόγραμμα παρουσιάζει όλες τις μεθόδους χειρισμού των Συνόλων

```
a={1,2,3,4,5,6,7,8,9,10,11,12,13,14}
print ('a=',a)
b={4,5,6,7,8,9,10,11,12,15,16,17,18}
print('b=',b)
tomi_a_b=a.intersection(b)
print ('a τομή b=',tomi_a_b)
c={33,34,35,36}
print('c=',c)
tomi_a_c=a.intersection(c)

print ('a τομή c=',tomi_a_c)
enosi_a_b=a.union(b)
```

```
print ('a ένωση b=',enosi_a_b)
diafora_a_b=a.difference(b)
print('διαφορά a-b ',diafora_a_b)
sym_diafora_a_b=a.symmetric_difference(b)
print('συμμετρική διαφορά a-b ',sym_diafora_a_b)

d={1,2,3,4}
print("d=",d)
if d.issubset(a):
    print('το d υποσύνολο του a ')

if a.issuperset(d):
    print('το a υπερσύνολο του d ')

arithmos=int(input ('δωσε ένα ακέραιο '))
if arithmos in a:
    print( "Ο ",arithmos, " υπάρχει στο συνολο a")
else:
    print( "ο ",arithmos, " δεν υπάρχει στο συνολο a")
```

Η εκτέλεση τουθα έχει ως αποτέλεσμα :

```
a= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
b= {4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18}
a τομή b= {4, 5, 6, 7, 8, 9, 10, 11, 12}
c= {33, 34, 35, 36}
a τομή c= set()
a ένωση b= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
διαφορά a-b  {1, 2, 3, 13, 14}
συμμετρική διαφορά a-b  {1, 2, 3, 13, 14, 15, 16, 17, 18}
d= {1, 2, 3, 4}
το d υποσύνολο του a
το a υπερσύνολο του d
δωσε ένα ακέραιο 6
   Ο 6  υπάρχει στο συνολο a
```

16 Λειτουργικές Μονάδες ή αρθρώματα Modules

Ένα module είναι ένα αρχείο που περιέχει ορισμούς ομάδων ή κλάσεων από συναρτήσεις που χρησιμεύουν για την επίλυση ενός συγκεκριμένου προβλήματος. Η Python κατά την εγκατάστασή της στον υπολογιστή αποθηκεύει στο δίσκο ένα φάκελο (**folder**) ένα μεγάλο αριθμό από modules, το καθένα με τα δικές του χαρακτηριστικές συναρτήσεις. Το όνομα του αρχείου καθορίζει και το όνομα του module και έχει επέκταση όπως και τα προγράμματα σε Python τα γράμματα py πχ **math.py** που αφορά σε Μαθηματικές Συναρτήσεις. Εκτός των αυτόματα εγκατεστημένων modules μπορούμε να γράψουμε τα δικά μας ή να κατεβάσουμε από το διαδίκτυο επιπλέον εξειδικευμένα modules όπως για παράδειγμα το **pypyodbc** το οποίο χρησιμοποιείται για τον χειρισμό Βάσεων δεδομένων παρέχοντας συναρτήσεις, εντολές για το σκοπό αυτό.

Η ενσωμάτωση ενός **module** στο πρόγραμμα επιτυγχάνεται μέσω της εντολής **import** στην αρχή το προγράμματος. Για την ενσωμάτωση και χρήση Μαθηματικών Συναρτήσεων χρειαζόμαστε το **math.py** που επιτυγχάνεται με την εντολή:

```
import math
```

Η χρήση των συναρτήσεων που υπάρχουν στο **module** γίνεται με την αναφορά στο module και στην επιθυμητή συνάρτηση.

Παράδειγμα

Οι συναρτήσεις του **math** καλούνται ως εξής:

math.συνάρτηση(παράμετροι)

π.χ.

```
y=math.cos(x) υπολογίζει το συνημίτονο του x και αναθέτει το αποτέλεσμα
              στη μεταβλητή y
print(math.pi)  τυπωνει τον αριθμο π=3.14159….
```

16.1 Το module math – Μαθηματικές Συναρτήσεις

Πολλές φορές είναι ανάγκη να υπολογίσουμε σε ένα πρόγραμμα το συνημίτονο ή την τετραγωνική ρίζα ενός αριθμού. Για αυτό το λόγο η Python περιέχει αρκετές ενσωματωμένες συναρτήσεις. Υπάρχουν πολλές ενσωματωμένες συναρτήσεις: Μαθηματικές Συναρτήσεις (αριθμητικές, τριγωνομετρικές, λογαριθμικές), συναρτήσεις μετατροπής δεδομένων και συναρτήσεις μορφοποίησης. Οι Μαθηματικές Συναρτήσεις βρίσκονται σε ένα **module** της Python που ονομάζεται **math**. Το **module math** της βιβλιοθήκης της **Python** περιέχει διάφορες μαθηματικές συναρτήσεις που μπορούν να χρησιμοποιηθούν σε μαθηματικούς υπολογισμούς. Εκτός του **math** με ορίσματα πραγματικούς αριθμούς υπάρχει το **cmath** με ορίσματα μιγαδικούς αριθμούς.

Στον επόμενο πίνακα δίνονται συνήθεις Μαθηματικές συναρτήσεις του math.

Συνάρτηση του Module	**Περιγραφή**
acos(x)	Τόξο συνημιτόνου x
asin(x)	Τόξο ημιτόνου x
atan(x)	Τόξο εφαπτομένης x
ceil(x)	Ο μικρότερος ακέραιος που είναι μεγαλύτερος ή ίσος του x
cos(x)	Συνημίτονο του τόξου x όταν αυτό εκφράζεται σε ακτίνια
degrees(x)	Επιστρέφει τις μοίρες ενός τόξου x όταν αυτό εκφράζεται σε ακτίνια
exp(x)	Εκφράζει τη δύναμη e^x
hypot(x,y)	Υπολογίζει το μήκος ενός ευθυγράμμου τμήματος ΑΒ με συντεταγμένες Α(0,0) και Β(x,y)
log(x)	Φυσικός λογάριθμος του x
log10(x)	Δεκαδικός λογάριθμος του x
radians(x)	Επιστρέφει τα ακτίνια ενός τόξου x όταν αυτό εκφράζεται σε μοίρες

sin(x)	Ημίτονο του τόξου x όταν αυτό εκφράζεται σε ακτίνια
sqrt(x)	Τετραγωνική ρίζα του x
tan(x)	Εφαπτομένη του τόξου x όταν αυτό εκφράζεται σε ακτίνια
e	Επιστρέφει τον αριθμό e=2.718281828459045
pi	Επιστρέφει το αριθμό 3.141592653589793

16.2 Το module random. Παραγωγή Τυχαίων αριθμών

Ένα άλλο χρήσιμο **module** είναι το **random**, που χρησιμοποιείται για την παραγωγή τυχαίων αριθμών. Οι πλέον συνήθεις συναρτήσεις του random παρουσιάζονται στον παρακάτω πίνακα.

Συνάρτηση του Module	**Περιγραφή**
random()	Παράγει ένα τυχαίο πραγματικό αριθμό στο διάστημα **(0,1)**
randint(x,y)	Παράγει ένα τυχαίο ακέραιο αριθμό στο διάστημα **[x,y]**
randrange(x)	Παράγει ένα τυχαίο ακέραιο αριθμό στο διάστημα **[0,x)**
randrange(x,y)	Παράγει ένα τυχαίο ακέραιο αριθμό στο διάστημα **[x,y)**
randrange(x,y,z)	Παράγει ένα τυχαίο ακέραιο αριθμό στο διάστημα **[x,y). Οι αριθμοί του διαστήματος αυτού** παράγονται από την ακολουθία **x, x+z, x+2z, x+3z** …. Το **z** δηλώνει το βήμα.
uniform(x,y)	Παράγει ένα τυχαίο πραγματικό αριθμό στο διάστημα **(x,y)**
choice(x)	Δέχεται ως όρισμα μία **list** ή μία **tuple** και επιστρέφει ένα τυχαίο στοιχείο της.

16.3 Το module os

Το **module os** χρησιμοποιείται για την κλήση και εκτέλεση εντολών του Λειτουργικού Συστήματος. Καλείται με την εντολή **import os** και περιέχει μία σειρά μεθόδων προκειμένου να εκτελεστούν οι αντίστοιχες εντολές. Η χρήση του module os προϋποθέτει τη γνώση των βασικών εντολών του Λειτουργικού Συστήματος σε περιβάλλον «***γραμμής εντολών»***. Οι κυριότερες μέθοδοι παρουσιάζονται παρακάτω.

1. **.chdir**

   Η μέθοδος αυτή χρησιμοποιείται αλλαγη του φακέλου εργασίας. Δέχεται ως όρισμα το όνομα την πλήρη διαδρομή για ένα φάκελο.

   Πχ. os.chdir('h:/app/python/projects')

Μετά την εκτέλεση της παραπάνω εντολής η μεταβλητή ο φάκελος εργασίας θα είναι ο: **projects** που βρίσκεται στο δίσκο **h:** μέσα στον φάκελο **python** που βρίσκεται μέσα στον φάκελο **app**.

2. .getwd

Η μέθοδος αυτή χρησιμοποιείται να ανάγνωση από το σύστημα του φακέλου εργασίας και την απόδοση του σε μία μεταβλητή.

Πχ. dir_ergasias=os.getwd()

3. .listdir

Η μέθοδος **listdir** δέχεται ώς όρισμα το όνομα ή την πλήρη διαδρομή ενός φακέλου και επιστρέφει σε μία λίστα τα όνόματα των αρχείων ή φακέλων που περιέχονται στον φάκελο αυτόν.

πχ. **filelist = os.listdir('h:/students').**
Μετά την εκτέλεση της παραπάνω εντολής η μεταβλητή **filelist** θα περιέχει τα ονόματα των αρχείων του φακέλου **students** που βρίσκεται στον δίσκο **h:**.

4. .mkdir

Η μέθοδος **.mkdir** χρησιμοποιείται για τη δημιουργία ενός folder/directory/φακέλου.

πχ. Έστω ότι ο φάκελος εργασίας είναι στο δίσκο **D:** και έχει όνομα **esoda**, η εντολή **os.mkdir("sales")** δημιουργεί μέσα στον φάκελο esoda ένα νέο **folder** με όνομα **sales**.

5. .makedirs

Η μέθοδος **.makedirs** χρησιμοποιείται για τη δημιουργία μίας ιεραρχικής δομής από φακέλους(**folders**).

πχ. Έστω ότι ο φάκελος εργασίας είναι στο δίσκο **D:** και έχει όνομα **apothiki** η εντολή **os.makedirs("sales/pelates")** δημιουργεί στο χρησιμοποιούμενο **φάκελο apothiki** ένα νέο φάκελο με όνομα **sales** και μέσα σε αυτόν το θα δημιουργήσει ένα νέο με όνομα **pelates**.

6. .path.isfile

Η μέθοδος αυτή δέχεται ως όρισμα το όνομα ενός αρχείου και επιστρέφει **True** αν υπάρχει στον φάκελο εργασίας.

Πχ.

print(os.path.isfile('keras.pdf'))

7. .path.isdir

Η μέθοδος αυτή δέχεται ως όρισμα το όνομα ενός φακέλου και επιστρέφει **True** αν υπάρχει στον φάκελο εργασίας αλλιώς επιστρέφει **False**.

Πχ.

print(os.path.isdir('h:/software/python/keras'))

8. .rename

Η μέθοδος **.rename** χρησιμοποιείται για την μετονομασία ενός αρχείου. Δέχεται δύο ορίσματα τύπου **str** που αντιστοιχούν στο παλαιό και στο νέο όνομα του αρχείου.

πχ. **rename ("poliseis.dat", "sales.dat").**

Μετά την εκτέλεση της παραπάνω εντολής το αρχείο με όνομα **"poliseis.dat"** μετονομάζεται σε **"sales.dat".**

17 Συναρτήσεις οριζόμενες από τον χρήστη

17.1 Η έννοια της Συνάρτησης

Μία συνάρτηση οριζόμενη από το χρήστη είναι ένα σύνολο εντολών που εκτελούν μία συγκεκριμένη λειτουργία. Οι συναρτήσεις καλούνται με τα ονόματα τους μπορούν να δεχθούν ορίσματα/παραμέτρους και κάθε υποπρόγραμμα εκτελεί μια συγκεκριμένη εργασία.

Τα σημαντικότερα πλεονεκτήματα των συναρτήσεων είναι ότι καταργούν την επανάληψη γραμμών κώδικα, βελτιώνουν την αναγνωσιμότητα του προγράμματος και απλοποιούν την ανάπτυξή του.

Ο χωρισμός ενός προγράμματος σε υποπρογράμματα, συναρτήσεις στην Python, έχει ως βασική προϋπόθεση την καλή ανάλυση του προβλήματος, τη διάσπασή του σε μικρότερα προβλήματα τα οποία μπορούν να αντιμετωπισθούν από πλευράς προγραμματισμού διακριτά και αυτόνομα με στόχο κάθε μικρότερο πρόβλημα να επιλύεται με μία συνάρτηση.

17.2 Ορισμός Συνάρτησης

Η βασική σύνταξη μίας συνάρτησης είναι η εξής:

```
def ονομαΣυνάρτησης (ορισμα1, ορισμα2,..όρισμαn):
      εντολη-1
      εντολη-2
      ……..
      εντολη-ν
      return val1
```

Ως **όνομαΣυνάρτησης** είναι ένα οποιοδήποτε έγκυρο όνομα για την Python όπως αναφέρθηκε στον ορισμό των μεταβλητών.

Σε μία συνάρτηση μπορεί να μεταβιβάζονται από το κύριο πρόγραμμα ή από μία άλλη συνάρτηση μία σειρά από μεταβλητές. Η μεταβίβαση των μεταβλητών αυτών επιτυγχάνεται μέσω των της λίστας των ορισμάτων/παραμέτρων που καθορίστηκαν κατά τη δημιουργία της συνάρτησης. Τα ορίσματα αυτά μπορούμε να τα χαρακτηρίσουμε σαν ***μεταβλητές εισόδου*** στη συνάρτηση.

Μία συνάρτηση μπορεί να επιστρέφει στο κύριο πρόγραμμα ή στην συνάρτηση που την κάλεσε μία ή περισσότερες τιμές που υπολογίστηκαν μέσα στη συνάρτηση. Η επιστροφή των τιμών αυτών επιτυγχάνεται μέσω των μεταβλητών που ακολουθύν την εντολή **return**.

Στη συνάρτηση μπορούμε να ορίζουμε μεταβλητές όπως και στο κύριο πρόγραμμα, οι μεταβλητές αυτές δεν είναι ορατές στο πρόγραμμα που καλεί τη συνάρτηση. Οι μεταβλητές αυτές είναι τοπικές της συνάρτησης. Αντίθετα οι μεταβλητές που έχουν ορισθεί στο κύριο πρόγραμμα είναι ορατές στη συνάρτηση. Βέβαια δεν είναι σωστή τακτική να χρησιμοποιούμε μεταβλητές του κύριου προγράμματος μέσα στη συνάρτηση χωρίς να περάσουν ως ορίσματα της συνάρτησης, επειδή η συνάρτηση είναι εξαρτημένη από το πρόγραμμα που την καλεί.

Μία συνάρτηση:

- Καλείται με το όνομά της.
- Μπορεί να μην έχει ορίσματα.
- Μπορεί να δεχθεί ένα ή περισσότερα ορίσματα ώς παραμέτρους εισόδου.
- Μπορεί να μην επιστρέφει καμία τιμή στο κύριο πρόγραμμα.
- Μπορεί να επιστρέφει στο κύριο πρόγραμμα μία περισσότερες τιμές. Για την επιστροφή τιμών στο κύριο πρόγραμμα χρησιμοποιούμε εντός της συνάρτησης την εντολή **`return`** η οποία ακολουθείται από μία η περισσότερες μεταβλητές. Όταν επιστρέφει μία η περισσότερες τιμές η κλήση του υποπρογράμματος γίνεται μέσω εντολής ανάθεσης τιμής. Εάν επιστρέφει περισσότερες από μια τιμές τότε στην ανάθεση τιμής στο αριστερό μέλος της εντολής αναγράφονται οι μεταβλητές στις οποίες επιστρέφουν οι τιμές διαχωριζόμενες με κόμα. π.χ

  `s_cost,axia_pol, p_fpa, p_forou,asfalisi,kerdos=ypologismos(cost)`
  βλέπε Παράδειγμα 2.

Συνάρτηση χωρίς επιστροφή τιμών

```
def ονομαΣυνάρτησης (ορίσματα):
      Εντολη-1
      Εντολη-2
      ……
      Εντολη-ν
```

Συνάρτηση με επιστροφή μίας τιμής

```
def ονομαΣυνάρτησης (ορίσματα):
      εντολη-1
      εντολη-2
      ……
      εντολη-ν
      return val1
```

Συνάρτηση με επιστροφή πολλών τιμών

```
def ονομαΣυνάρτησης (ορίσματα):
      εντολη-1
      εντολη-2
      ……
      εντολη-ν
      return val1,val2,..,valk
```

<u>Παράδειγμα-1</u>:

Να γραφεί πρόγραμμα το οποίο να διαβάζει ένα θετικό ακέραιο αριθμο n και να υπολογογίζει το παραγοντικό του αριθμού αυτού n!

n!= 1 2 3 4 …n και ισχύει 0!=1!=1

Λύση

```
def parag(k):
    par=1
    if k==0:
        par=1
    else:
        for m in range(1,k+1):
            par=par*m
```

```
    return  par

n=int(input("δώσε το  n:"))
n=abs(n)
paragontiko=parag(n)
print(paragontiko)
```

Όταν εκτελείται η εντολή : **paragontiko=parag(n)** συμβαίνουν τα παρακάτω:

Ο έλεγχος του προγράμματος ανατίθεται στη συνάρτηση **parag** ως εξής: Το όρισμα k της συνάρτησης **parag** παίρνει τιμή ίση με το περιεχόμενο της μεταβλητής **n**. Εκτελούνται οι εντολές της συνάρτησης. Με την εντολή **return par** το περιεχόμενο της μεταβλητής **par** ανατίθεται στη μεταβλητή **paragontiko** του κυρίου προγράμματος.

Παράδειγμα-2:

Να γραφεί πρόγραμμα στο οποίο δίνεται το **κόστος κατασκευής** ενός προϊόντος και υπολογίζει μέσω συνάρτησης.

- Τον ΦΠΑ 24% επί του κόστους κατασκευής
- Την εισφορά ασφάλισης 27% επί του κόστους κατασκευής
- Το φορός 35% επί του κόστους κατασκευής
- Το κέρδος 30% επί της συνολικού κόστους (κοστος κατασκευής +φοροι κλπ)
- Τη τελική αξία πωλησης του προϊόντος

```
def ypologismos(axia_katask):
    pos_fpa=0.24 # ποσοστό ΦΠΑ
    pos_forou=0.35 # ποσοστό φόρου
    pos_asf=0.27 # ποσοστό ασφαλ. εισφορών
    pos_kerd=0.30 # ποσοστό κέρδους

    poso fpa = axia katask * pos fpa
    foros= axia_katask * pos_forou
    asfal=axia_katask*pos_asf
    syn_axia= axia_katask + poso_fpa + asfal+foros
    kerd=pos kerd*syn axia
    teliki=syn_axia+kerd
    return syn_axia,teliki,poso_fpa,foros,asfal,kerd

cost=input("κόστος κατασκευής ")
cost=float(cost)
# s_cost    : Συνολικό κόστος
# axia_pol  : αξία πωλήσης
# p_fpa     : ποσό ΦΠΑ
# p forou   : ποσο Φόρου
# asfalisi  : ποσό ασφ.εισφοράς
# kerdos    : ποσό κέρδους
s_cost,axia_pol, p_fpa, p_forou,asfalisi,kerdos=ypologismos(cost)

print("Συνολικό Κοστος:{} ".format(s_cost))
print("ΦΠΑ:{} ".format(p_fpa))
print("Ασφ.Εισφ:{} ".format(asfalisi))
print("Αξία πώλησης:{}".format(axia_pol))
print("Κέρδος:{}".format(kerdos))
```

17.3 Αναδρομικές Συναρτήσεις

Αναδρομική συνάρτηση (**recursive function**) είναι μία συνάρτηση η οποία καλεί τον εαυτό της, δηλαδή η συνάρτηση αυτή ορίζεται κάνοντας χρήση την ίδια την συνάρτηση.

Παράδειγματα αναδρομής έχουμε από τα Μαθηματικά στον ορισμό της δύναμης, του παραγοντικού, των ακολουθιών κλπ:

Ορισμός δύναμης :$x^n=x^{n-1}$ x, με αρχική τιμή $x^0=1$

Με βάση τον παραπανω ορισμό η αναδρομική συνάρτηση υπολογισμού δύναμης θα είναι:

```
def dynami(k,n):
    if n==0:
        dyn=1
    else:
        dyn=k*dynami(k,n-1)
    return  dyn
```

Ορισμό του παραγοντικού ν!=(ν-1)! ν, με αρχική τιμή 0!=1

Με βάση τον παραπανω ορισμό η αναδρομική συνάρτηση υπολογισμού δύναμης θα είναι:

```
def parag(k):
    if k==0:
        par=1
    else:
        par=k*parag(k-1)
    return  par
```

Κάθε φορά που γίνεται κλήση μίας αναδρομικής συνάρτησης δεσμεύεται νέος χώρος στην μνήμη. Αυτό έχει ώς αποτέλεσμα να υπάρχει περίπτωση να μήν υπάρχει διαθέσιμη μνήμη όταν υπάρχει μεγάλος αριθμός αναδρομικών κλήσεων της συνάρτησης.

Παράδειγμα-1:

Να γραφεί πρόγραμμα το οποίο να διαβάζει ένα θετικό ακέραιο αριθμο n και να υπολογογίζει το παραγοντικό του αριθμού αυτού **n!**

n!= 1 2 3 4 …n και ισχύει 0!=1!=1

Λύση με αναδρομή

```
def parag(k):
    if k==0:
        par=1
    else:
        par=k*parag(k-1)
    return  par

n=int(input("δώσε το  n:"))
n=abs(n)
paragontiko=parag(n)
print(paragontiko)
```

18 Modules οριζόμενα από τον χρήστη

Ένα module το οποίο ορίζεται από τον χρήστη είναι ένα αρχείο το οποίο έχει ως επέκταση py (πχ. **ypologismoi.py**) και περιέχει ένα σύνολο συναρτήσεων ή κλάσεων που μπορούν να χρησιμοποιηθούν για την επίλυση ενός συγκεκριμένου προβλήματος. Το αρχείο αυτό πρέπει να είναι αποθηκευμένο στον φάκελο (**folder**) που είναι αποθηκευμένο το κύριο πρόγραμμα του χρήστη.

Η ενσωμάτωση ενός **module** στο πρόγραμμα επιτυγχάνεται μέσω της εντολής import στην αρχή το προγράμματος. Για την ενσωμάτωση στο κύριο πρόγραμμα και χρήση των συναρτήσεων που περιέχονται στο αντίστοιχο **module** πρέπει να γίνεται εισαγωγή στο κύριο πρόγραμμα. Αυτό επιτυγχάνεται με την εντολή **import** π.χ.

```
import ypologismoi
```

Η χρήση κάποια συνάρτησης του **module** επιτυχγάνεται με την αναφορά στο **module** και στην αντίστοιχη συνάρτηση.

Π.χ Εστω ένα **module** με ονομα ypologismoi για τον υπολογισμό διαφόρων μεταβλητών στη μισθοδοσία υπαλλήλων. Εστω επίσης ότι περιέχει εκτός των άλλων μία συνάρτηση με όνομα **yp_forou** που δέχεται ως παράμετρο εισόδου τις φορολογητέες απόδοχές του υπαλλήλου. Για να λάβουμε τον αντίστοιχο φόρο σε μία μεταβλητή με το όνομα **foros** η αναφορά στην συνάρτηση θα γίνει με την εξής εντολή:

```
foros=ypologismoi.yp_forou(apodoxes)
```

Ένα **module** μπορούμε να το καλέσουμε και να του δώσουμε ένα ψευδώνυμο, το οποίο αντικαθιστά το όνομα του **module** μεσα στο πρόγραμμά μας. π.χ.

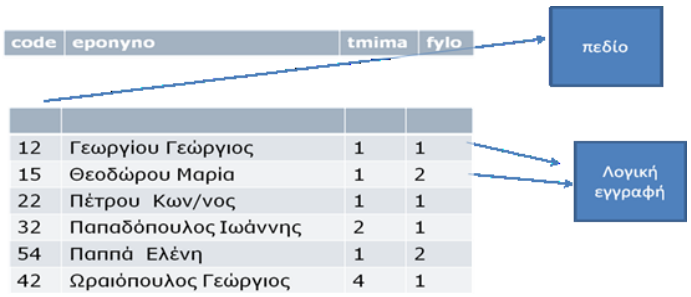
```
import ypologismoi  as yp
```

Στην περίπτωση αυτή η κλήση της συνάρτησης υπολογισμού του φόρου του προηγούμενου παραδείγματος θα είναι:

```
foros=yp.yp_forou(apodoxes)
```

19 Διαχείριση αρχείων

Για να είναι δυνατή η επεξεργασία μεγάλου αριθμού δεδομένων τα δεδομένα αποθηκεύονται σε ψηφιακά μέσα έχοντας μία κατάλληλη οργάνωση. Η αποθήκευση γίνεται σε αρχεία. Ένα αρχείο αποτελείται από μία σειρά ομοειδών δεδομένων που ονομάζονται *λογικές εγγραφές* (logical records ή records). Κάθε λογική εγγραφή ενός αρχείου αποτελείται από μία σειρά διακριτών δεδομένων που έχουν νόημα επεξεργασίας που ονομάζονται *πεδία* (fields).



Στην ενότητα αυτή θα ασχοληθούμε με την είσοδο και έξοδο δεδομένων σε σειριακά αρχεία, την απλούστερη μορφή αρχείων. Τα σειριακά αρχεία αποτελούνται από μια σειρά γραμμών κειμένου και συχνά ονομάζονται αρχεία ASCII. Στα αρχεία αυτά τα δεδομένα αποθηκεύονται χρησιμοποιώντας ένα byte για κάθε χαρακτήρα. Τα αρχεία αυτά είναι επεξεργάσιμα και από το πρόγραμμα **Σημειωματάριο** (**Notepad**).

19.1 Χειρισμός Σειριακών Αρχείων

19.1.1 Άνοιγμα– Κλείσιμο Αρχείων

Πριν επιχειρήσουμε οποιαδήποτε ενέργεια για ανάγνωση δεδομένων από αρχείο ή αποθήκευση δεδομένων σε αρχείο θα πρέπει πρώτα να εκτελέσουμε τη ενέργεια **Άνοιγμα Αρχείου**. Με το άνοιγμα ενός σειριακού αρχείου δηλώνουμε στο σύστημα το είδος της ενέργειας που θα εκτελέσουμε στο αρχείο (ανάγνωση ή αποθήκευση στοιχείων) και ταυτόχρονα αντιστοιχίζουμε ένα *συμβολικό όνομα* στο συγκεκριμένο αρχείο. Με βάση αυτό *συμβολικό όνομα* εκτελούμε τις εντολές ανάγνωσης ή εγγραφής στο αρχείο. Σημειώνεται ότι το *συμβολικό όνομα* συνδέεται με το αρχείο μόνο όσο το αρχείο είναι ανοικτό.

Σε ένα πρόγραμμα μπορούμε να χρησιμοποιήσουμε περισσότερα του ενός αρχεία η δε αντιστοίχιση τους συμβολικά ονόματα αφορούν μόνο στο τρέχον πρόγραμμα και δεν είναι δεσμευτικά για άλλα προγράμματα που χρησιμοποιούν τα ίδια αρχεία. Το άνοιγμα ενός αρχείου πραγματοποιείται μέσω της συνάρτησης **`open`** ως εξής:

```
file_object = open("filename", "mode")
```

Όπου

`file_object`	είναι το συμβολικό όνομα του αντικειμένου τύπου αρχείου
`filename`	"*filename*" δηλώνει τη διαδρομή και το όνομα του αρχείου που θα ανοίξει
`mode`	τρόπος χειρισμού του αρχείου με τιμές: **r** : Ανάγνωση αρχείου. **w** : Δημιουργία αρχείου και εγγραφή σε αυτό το αρχείο. **a** : Επέκταση υπάρχοντος αρχείου με νέες εγγραφές.

μέσω αυτή της εντολής δημιουργείται ένα **αντικείμενο τύπου αρχείου** για περαιτέρω χρήση.

Το αρχείο που ***ανοίχθηκε*** μέσω της **`open`** θα πρέπει να ***κλείσει*** μετά το τέλος της χρήσης του. Για το κλείσιμο ενός αρχείου χρησιμοποιείται η μέθοδος **`.close`** των αντικειμένων που είναι τύπου αρχείου. Η πλήρη μορφή της εντολής **`close`** είναι:

```
file_object.close()
```

19.1.2 Χειρισμός σειριακού αρχείου εξόδου

Σε ένα σειριακό αρχείο εξόδου μπορούμε **μόνο** να γράψουμε (αποθηκεύσουμε) στοιχεία. Διακρίνουμε δυο περιπτώσεις.

- Δημιουργία ενός αρχείου που δεν υπάρχει.
- Επέκταση ενός υπάρχοντος αρχείου με επιπλέον στοιχεία.

Οι βασικές ενέργειες χειρισμού ενός αρχείου εξόδου είναι:

- Άνοιγμα αρχείου.
- Αποθήκευση δεδομένων στο αρχείο.
- Κλείσιμο αρχείου.

Έστω η εντολή:

```
arx=open("d:\arxeia\apotel.txt", "w")
```

Η εντολή αυτή λέει στη Python να ανοίξει ένα αρχείο στη μονάδα αποθήκευσης **D** στο φάκελο **arxeia** (σημ. η μονάδα αποθήκευσης και ο φάκελος πρέπει να υπάρχουν) με όνομα **apotel.txt** στο οποίο πρόκειται να γίνει εγγραφή δεδομένων. Εάν η μονάδα αποθήκευσης **D** ή ο φάκελος **arxeia** δεν υπάρχει τότε η Python σταματά την εκτέλεση του προγράμματος και εμφανίζει λάθος. Εάν το αρχείο δεν υπάρχει ήδη, τότε το δημιουργεί. Εάν υπάρχει, διαγράφει τα περιεχόμενα που έχει ήδη το αρχείο αυτή τη στιγμή.

19.1.3 Αποθήκευση δεδομένων σε αρχείο

Όπως αναφέρθηκε μέσω της **open** ορίζεται ένα αντικείμενο τύπου αρχείου που ανατίθεται σε ένα συμβολικό όνομα μέσω της εντολής ανάθεσης τιμής. Το αντικείμενο αυτό ανήκει σε μία κλάση αρχείων και έχει μια σειρά από μεθόδους χειρισμού του. Για την αποθήκευση δεδομένων σε ένα αρχείο χρησιμοποιείται η μέθοδος **.write**. Η μέθοδος αυτή δέχεται **ένα μόνο όρισμα** ως λογική εγγραφή που θα αποθηκευτεί στο σχετικό αρχείο.

Εστω ότι ανοίγουμε το αρχείο **students** στο δίσκο Ε στον φάκελο **arxeia** προκειμένου να αποθηκεύσουμε στοιχεία φοιτητών. Η εντολή είναι:

```
arxeio = open("e:/arxeia/students.txt", 'w')
```

Έστω ότι κάθε εγγραφή που αφορά σε φοιτητή είναι ονομάζεται **record** και θέλουμε να την αποθηκεύσουμε στο αρχείο. Η εντολή είναι ώς εξής:

```
arxeio.write(record)
```

Η μεταβλητή **record** είναι αποκλειστικά τύπου **string**.

Για να γίνει η εγγραφή σε ένα αρχείο κειμένου κατά γραμμές πρέπει να εισαχθεί στο προς εγγραφή string το **"\n"** που δηλώνει την αλλαγή γραμμής.

Παράδειγμα δημιουργίας αρχείου:

Να γραφεί πρόγραμμα το οποίο διαβάζει από την οθόνη τα παρακάτω στοιχεία που αφορούν στη βαθμολογία ενός φοιτητή σε τρεις προόδους:

- Κωδικός φοιτητή
- Βαθμός-1
- Βαθμός-2
- Βαθμός-3

Και να τα γράφει σε ένα αρχείο με όνομα **students.txt**

Η διαδικασία να επαναλαμβάνεται μέχρι να δοθεί ως κωδικός φοιτητή ο αριθμός 0.

θεωρούμε ότι το αρχείο θα έχει ώς διαχωριστικό των πεδίων το κόμμα (,).

Ο κώδικας του προγράμματος είναι

```
arxeio = open("e:/arxeia/students.txt", 'w')  # ανοιγμα του αρχείου
epan = True
while epan: # επανάληψη μέχρι η epan να πάρει τιμή False
    code = input("δώσε κωδικό φοιτητή ")
    code = int(code)
    if code != 0:
        epon = input("επώνυμο φοιτητή ")
        ba1 = float(input("βαθμός 1ος:"))
        ba2 = float(input("βαθμός 2ος:"))
        ba3 = float(input("βαθμός 3ος:"))
        # δόμηση της λογικής εγγραφής  record διαχωρίζοντας τα πεδία με ,
        #record = str(code)+","+epon+","+str(ba1)+","+str(ba2)+","+str(ba3)+"\n"
        # ισοδύναμα ή καλυτερα μπορεί να γραφεί με την παρακάτω μορφή
        record = "{},{},{},{},{} \n".format(str(code),epon,str(ba1),str(ba2),str(ba3))
        # το : \n δηλώνει να γράφει τα δεδομένα σε μία γραμμή του αρχείου
        # και η επόμενη εγγραφή στο αρχείο θα είναι στην επόμενη γραμμή

        arxeio.write(record)
    else:
        epan = False
arxeio.close() # κλείσιμο αρχείου
```

19.1.4 Ανάγνωση σειριακού αρχείου

Αντίστοιχα με τη δημιουργία σειριακών αρχείων, απαιτούνται τρία βήματα για την ανάγνωση δεδομένων από σειριακά αρχεία, δηλαδή άνοιγμα αρχείου, ανάγνωση δεδομένων του αρχείου και κλείσιμο του αρχείου. Το αρχείο ανοίγει με την ακόλουθη εντολή:

```
arxeio = open("e:/arxeia/students.txt",'r')
```

Ο τρόπος χειρισμού του αρχείου **r** επιτρέπει την ανάγνωση δεδομένων από ένα υπάρχον αρχείο. Η Python παρέχει διαφορετικούς μεθόδους και τεχνικές για την ανάγνωση δεδομένων από αρχεία κειμένου. Παρακάτω θα παραθέσουμε δύο τεχνικές ανάγνωστης, μέσω της εντολής **for** και μέσω της εντολής **while** σε συνδιασμό με την μέθοδο **.readline** και την **.readlines** με τους δύο αυτούς τρόπους θα διαβάζουμε το αρχείο μια προς μία τις εγγραφές του δηλαδή μία προς μία τις γραμμές του αρχείου κειμένου. Το σημαντικό και στις δύο αυτές περιπτώσεις είναι ότι κάθε μία γραμμή του κειμένου που αντιστοιχεί σε μία **λογική εγγραφή (record)** ανατίθεται σε μία μεταβλητή τύπου string. Προκειμένου να διακρίνουμε τα πεδία της εγγραφής, τα οποία χωρίζονται με κόμμα, θα χρησιμοποιήσουμε τη μέθοδο **.split** των **string**, η οποία θα διαχωρίσει τα πεδία με βάση το **κόμμα** και θα τα αναθέσει σε μία λίστα.

Παράδειγμα ανάγνωσης αρχείου:

Να γραφεί πρόγραμμα το οποίο να διαβάζει κάθε εγγραφή του αρχείου students.txt που δημιουργήθηκε στο προηγούμενο παράδειγμα και να εμφανίζει τον κωδικό του φοιτητή, το ονοματεπώνυμό του και τον μέσο όρο της επίδοσής του.

Λυση 1η με χρήση for

Μέσω της εντολής open έχει δημιουργηθεί ένα αντικείμενο τύπου αρχείου το οποίο περιέχει όλες τις εγγραφές (γραμμές) του αρχείου. Κατά συνέπεια με τη εντολή **`for eggrafi in arxeio:`** επιλέγονται διαδοχικά μία προς μία όλες οι εγγραφές του αρχείου και αποδίδονται σε μία μεταβλητή τύπου **`string`** με το όνομα **`eggrafi`**. Επειδή τα πεδία της εγγραφής διαχωρίζονται με (,) με την εντολή **`pedia = eggrafi.split(',')`** κάθε πεδίο ανατιθεται ως τιμή μίας λίστα η οποία έχει όνομα **`pedia.`**

```
arxeio = open("g:/arxeia/students.txt", 'r')  # ανοιγμα του αρχείου
for eggrafi in arxeio:
    pedia=eggrafi.split(',')
    code=int(pedia[0])
    eponymo=pedia[1]
    bathmos1=float(pedia[2])
    bathmos2=float(pedia[3])
    bathmos3=float(pedia[4])
    mo=(bathmos1+bathmos2+bathmos3)/3
    print(code,eponymo,mo)
arxeio.close()
```

Λυση 2η με χρήση `while` και τη μέθοδο `.readline()`

Μέσω της εντολής `open` έχει δημιουργηθεί ένα αντικείμενο τύπου αρχείου. Με τη μέθοδο `.readline()` το πρόγραμμα διαβάζει από το αντίστοιχο αρχείο μία γραμμή χρησιμοποιόντας την εντολή τη εντολή **`eggrafi=arxeio.readline().`** Με την ανάθεση τιμής η γραμμή αυτή αποδίδεται σε μία μεταβλητή τύπου **`string`** με το όνομα **`eggrafi`**. Επειδή τα πεδία της εγγραφής διαχωρίζονται με (,) με την εντολή **`pedia = eggrafi.split(',')`** κάθε πεδίο ανατίθεται ως τιμή μίας λίστα η οποία έχει όνομα **`pedia.`**

```
arxeio = open("g:/arxeia/students.txt", 'r') # ανοιγμα του αρχείου
eggrafi=arxeio.readline() # διάβασμα για αρχικοποίηση της μεταβλητής eggrafi
while eggrafi!="": # όσο η μεταβλητή eggrafi δεν είναι κενή
    pedia=eggrafi.split(',')
    code=int(pedia[0])
    eponymo=pedia[1]
    bathmos1=float(pedia[2])
    bathmos2=float(pedia[3])
    bathmos3=float(pedia[4])
    mo=(bathmos1+bathmos2+bathmos3)/3
    print(code,eponymo,mo)
    eggrafi=arxeio.readline()
arxeio.close()
```

Λυση 3η με χρήση for και τη μέθοδο .readlines

Μέσω της εντολής open έχει δημιουργηθεί ένα αντικείμενο τύπου αρχείου το οποίο περιέχει όλες τις εγγραφές (γραμμές) του αρχείου. Με τη μέθοδο `.readlines()` και την εντολή **`grammes=arxeio.readlines()`** το πρόγραμμα διαβάζει όλες τις γραμμές του αρχείου κειμένου και τοποθετεί κάθε γραμμή σε μία λίστα όνομα **`grammes.`** Επειδή τα πεδία της γραμμης διαχωρίζονται με (,) με την εντολή **`pedia = grammi.split(',')`** κάθε πεδίο ανατίθεται ως τιμή μίας λίστα η οποία έχει όνομα **`pedia.`**

```
arxeio = open("g:/arxeia/students-1.txt", 'r') #ανοιγμα του αρχείου
grammes=arxeio.readlines()#διάβασμα όλου του αρχείου και δημιουργία της λίστας
                          # grammes

arxeio.close()#κλείσιμο του αρχείου
for grammi in grammes: #για κάθε στοιχείο της λίστας δηλ για κάθε γραμμή του αρχείου
    pedia=grammi.split(',')
    code=int(pedia[0])
    eponymo=pedia[1]
    bathmos1=float(pedia[2])
    bathmos2=float(pedia[3])
    bathmos3=float(pedia[4])
    mo=(bathmos1+bathmos2+bathmos3)/3
    print(code,eponymo,mo)
```

20 Διαχείριση Λαθών Προγράμματος-Εξαιρέσεις

Σε ένα πρόγραμμα μπορούν να παρουσιαστούν τρία είδη λαθών. (α) συντακτικά (syntax errors ή compiler errors), (β) λάθη κατά την εκτέλεση του προγράμματος (runtime errors) και (γ) λογικά λάθη (logic errors).

- Συντακτικά σφάλματα είναι σφάλματα προγραμματισμού (π.χ. λάθος ιδιότητα ή δεσμευμένη λέξη) τα οποία παραβιάζουν τους κανόνες της γλώσσας προγραμματισμού. Ένα συντακτικό σφάλμα προκαλείται όταν γραφεί μία εντολή η οποία παραβιάζει τη γραμματική και το συντακτικό μιάς γλώσσας. Για παράδειγμα, ένας αναγραμματισμός μίας εντολής, **`prnit`** αντί για **`print`** θα εντοπίστει από περιβάλλον αντάπτυξης ή το μεταφραστικό πρόγραμμα και θα εμφανίστεί στην οθόνη. Σε αυτή την περίπτωση θα πρέπει να διορθωθεί η ορθογραφία της εντολής για να εκτελεστεί το πρόγραμμα.
- Σφάλματα κατά την εκτέλεση του προγράμματος είναι σφάλματα που παρουσιάζονται είτε όταν το πρόγραμμα δέχεται δεδομένα τα οποία δεν ξέρει πώς να τα επεξεργαστεί είτε από εξωγενείς αιτίες, όπως μια περιφερειακή συσκευή που δεν λειτουργεί, μια κατεστραμμένη μονάδα εισόδου εξόδου, ένα αρχείο που δεν βρίσκεται στον δίσκο. Η διόρθωση αυτών των σφαλμάτων είναι συνήθως δυσκολότερη από τη διόρθωση των συντακτικών σφαλμάτων.
- Λογικά σφάλματα είναι ανθρώπινα λάθη. Ένα λογικό σφάλμα παρατηρείται όταν το πρόγραμμα δε λειτουργεί σωστά επειδή του οι εντολές μπορεί να είναι λάθος. Για παράδειγμα αντί για πρόσθεση δύο μεταβλητών έχετε δώσει πολλαπλασιασμό των μεταβλητών. Κατά την εκτέλεση ενός προγράμματος δεν είναι δυνατόν η Python να εντοπίσει σφάλματα που αναφέρονται στη λογική του προβλήματος διότι εύκολα μπορεί να παρουσιασθεί ένα πρόβλημα σωστό συντακτικά αλλά με λάθος λογική. Επειδή τα λογικά σφάλματα είναι δύσκολο να ανιχνευθούν, η μεγαλύτερη

προσπάθεια εξάλειψης των σφαλμάτων εστιάζεται στον τύπο αυτό των σφαλμάτων.

Η διαδικασία της διόρθωσης των σφαλμάτων ονομάζεται εκσφαλμάτωση(debugging). Το περιβάλλον ανάπτυξης της **Pycharm** της **Python** διαθέτει ορισμένα εργαλεία ενσωματωμένα που βοηθούν στην εύρεση και διόρθωση των λαθών κυρίως συντακτικών. Τα εργαλεία αυτά μειώνουν σημαντικά την προσπάθεια για την εκσφαλμάτωση των προγραμμάτων.

Ένας τρόπος για την αναγνώριση των σφαλμάτων είναι να εκτελείται το πρόγραμμα ανά μία γραμμή τη φορά και να εξετάζεται το περιεχόμενο των μεταβλητών καθώς αυτό μεταβάλλεται. Αυτό επιτυγχάνεται με την εισαγωγή σημείου διακοπής (Break point) κατά την εκτέλεση του προγράμματος. Η διαδικασία της διακοπής δίνει την δυνατότητα επισταμένου ελέγχου του προγράμματος ενώ αυτό εκτελείται.

Εάν κατά την εκτέλεση ενός προγράμματος παρουσιαστεί λάθος τότε το πρόγραμμα σταματά απότομα. Προκειμένου να αποφύγουμε τον απότομο τερματισμό ενός προγράμματος χρησιμοποιούμε τις εντολές **try** ….**except** για τον έλεγχο του προγράμματος.

Η απλούστερη δομή είναι:

```
try:
    Εντολή_try_1
    Εντολή_try_2
    ……
    Εντολή_try_n
except:
    Εντολή_except_1
    Εντολή_except_2
    …….
    Εντολή_except _m
```

Σύμφωνα με την παραπάνω δομή αν θεωρούμε ότι μπορεί να παρουσιαστεί λάθος κατά την εκτέλεση ενός συνόλου εντολών `Εντολή_try_1,…. Εντολή_try_n` αυτές οι εντολές τοποθετούνται μεταξύ της εντολής `try:` και της `Except..`Αν παρουσιαστεί λάθος κατά την εκτέλεση των εντολών αυτό το πρόγραμμα δεν σταματά απότομα αλλά εκτελούνται οι εντολές `Εντολή_except_1,… Εντολή_except_m` οι οποίες βρίσκονται μέσα στο block του `except.`

```
try:
    Εντολή-try-1
    Εντολή-try-2
    ……
except Exception as err:
    Εντολή exept_1.1
    Εντολή exept_1.2
    …….
```

Με την παραπάνω δομή μπορούμε να χρησιμοποιήσουμε την μεταβλητή **err** προκειμένου να εμφανίσουμε το είδους του λάθους. Μερικοί τύποι λαθών δίδονται στο παρακάτω πίνακα.

Ονομα	**Τύπος λάθους**
ZeroDivisionError	Διαίρεση δια του μηδενός
IOError	Σφάλμα εισόδου/εξόδου
ValueError	Σφάλμα αντιστοίχισης δεδομένων Πχ.Σφάλμα μετατροπής ενός **string** σε αριθμητική μεταβλητή
KeyboardInterrupt	Διακοπή της εκτέλεσης μέσω πληκτρολογίου
IndexError	Σφάλμα δείκτη για προσπέλαση σε χαρακτήρα **string**, ή στοιχείο **list** ή **tuple**

Παράδειγμα με επεξεργασία αρχείου.

```
try:
    #επιχειρείται το άνοιγμα και επεξεργασία του αρχείου
    arxeio = open("g:/arxeia/paragogi.txt",'r')
    for record in arxeio:
        print(record)
    arxeio.close()
except :
     # αν υπάρξει λάθος κατά το άνοιγμα του αρχείου
    print (" Λάθος όνομα αρχείου ")
```

```
try:
    #επιχειρείται το άνοιγμα και επεξεργασία του αρχείου
     arxeio = open("h:/arxeia/parag.txt",'r')
     for record in arxeio:
         print(record)
         pedia=record.split(",")
         code= int(pedia[0])
         eponymo=pedia[1]
         print(code, eponymo)
     arxeio.close()
except Exception as err:
     # αν υπάρξει λάθος κατά το άνοιγμα του αρχείου
     # εμφανίζει το σχετικό μήνυμα καθώς και το είδος του
     # run time error
     print (" Λάθος όνομα αρχείου ", err)
```

Μία πιο πολύπλοκη σύνταξη της **try**…**except είναι:**

```
try:
     Εντολή-try-1
     Εντολή-try-2
     ……
except Exception as err:
     Εντολή exept_1.1
     Εντολή exept_1.2
     …….
else:
     Εντολή exept_2.1
     Εντολή exept_2.2
     …….
finally:
     Εντολή final_1
     Εντολή final_2
```

Εκτελούνται όταν δεν παρουσιάζονται λάθη

Εκτελούνται όταν παρουσιάζονται λάθη συγκεκριμένου τύπου

Εκτελούνται όταν παρουσιάζονται λάθη εκτός του συγκεκριμένου τύπου

Εκτελούνται πάντα είτε παρουσιαστεί λάθος είτε όχι

21 Επεξεργασία Βάσεων Δεδομένων με Python

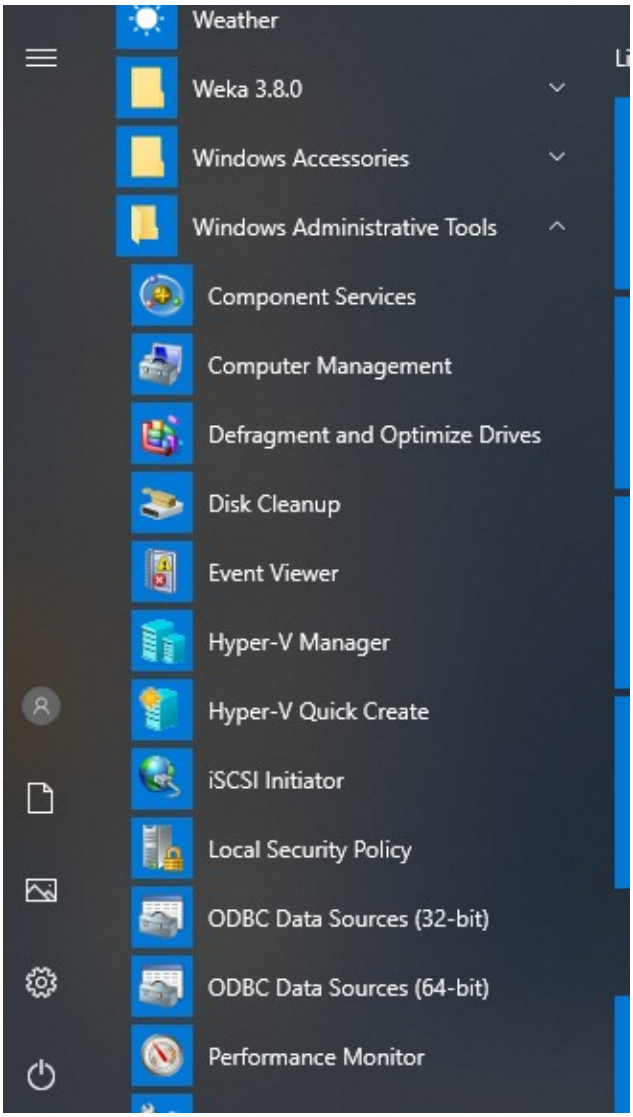
Ο χειρισμός και επεξεργασία μίας Σχεσιακής Βάσης Δεδομένων μέσα από ένα πρόγραμμα Python προϋποθέτει την εισαγωγή (import) στο πρόγραμμα ενός module που περιέχει όλες τις εντολές για το χειρισμό της Βάσης. Θα διακρίνουμε δύο κατηγορίες:

- Στον χειρισμό της Βάσεων Δεδομένων τύπου SQLite
- Στον χειρισμό της Βάσεων Δεδομένων που απαιτούν ρυθμιση του ODBC του υπολογιστή. Στην περίπτωση αυτή θα παρουσιαστούν όλες οι ρυθμίσεις για το Λειτουργικό Σύστημα Windows καθώς και οι αντίστοιχες εντολές για τη σύνδεση του προγράμματος με τη Βάση Δεδομένων.

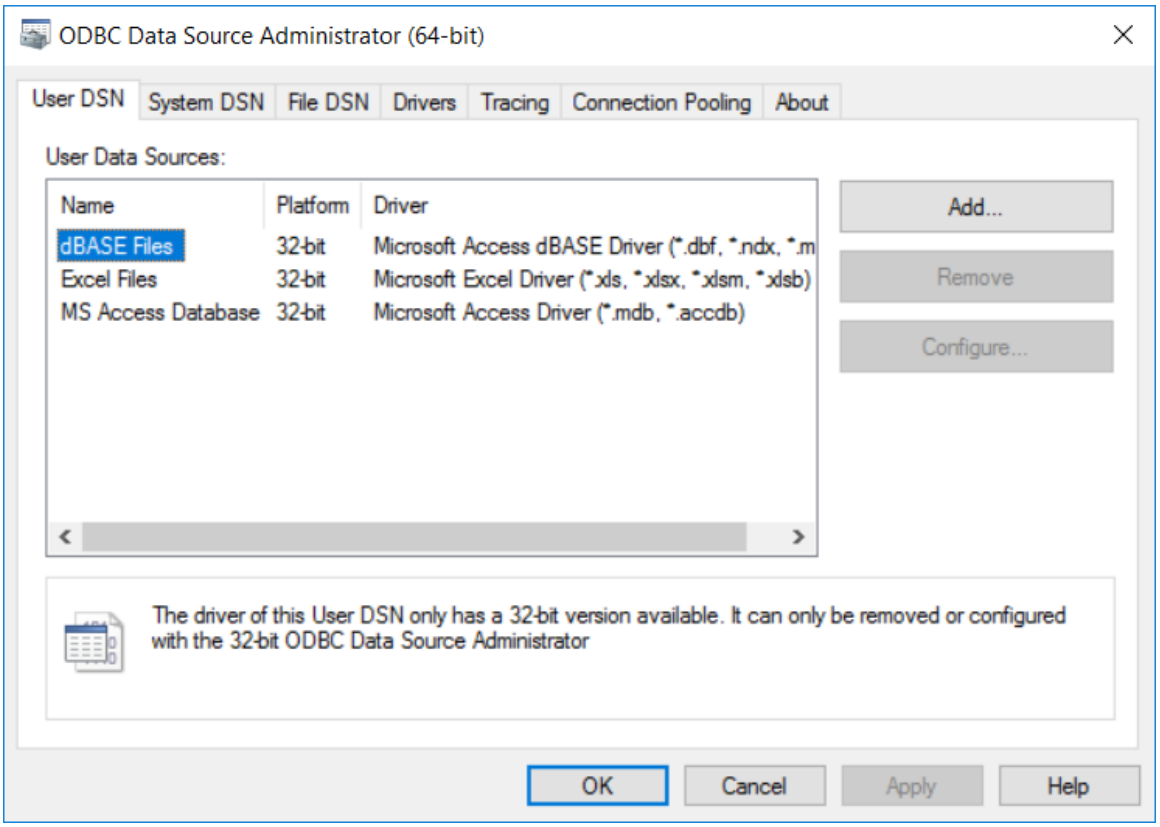
Αξίζει να σημειωθεί ότι αφού πραγματοποιηθεί η σύνδεση με τη Βάση Δεδομένων, οι λοιπες εντολές είναι όμοιες.

21.1 Ρυθμιση του ODBC του υπολογιστή

Από τα προγράμματα εφαρμογών των windows επιλέγουμε Windows Administrative Tools και στη συνέχεια ODBC Data Sources. Αναλογα με τον υπολογιστή και την έκδοση του αντίστοιχου driver επιλέγετε τα 32bits ή 64 bits.



Με την επιλογή των αντίστοιχου ODBC εμφανίζεται η οθόνη:



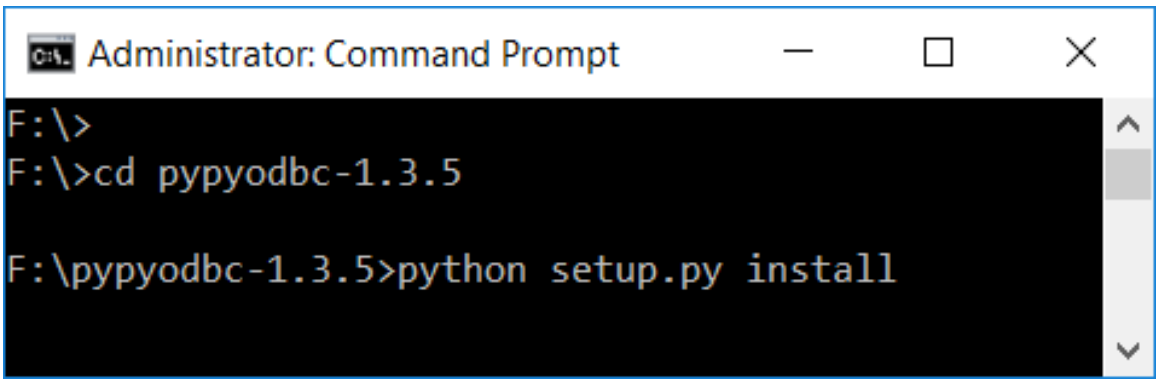
Στην παραπανω οθόνη θα μπορείτε να προσθέσετε τον αντίστοιχο driver για το είδος της Βάσης Δεδομένων που θα χρησιμοποιείσετε. Όσον αφορά την MS Access εάν δεν υπάρχει στο σύστημά σας μπορείτε να την αναζητήσετε στο site της Microsoft

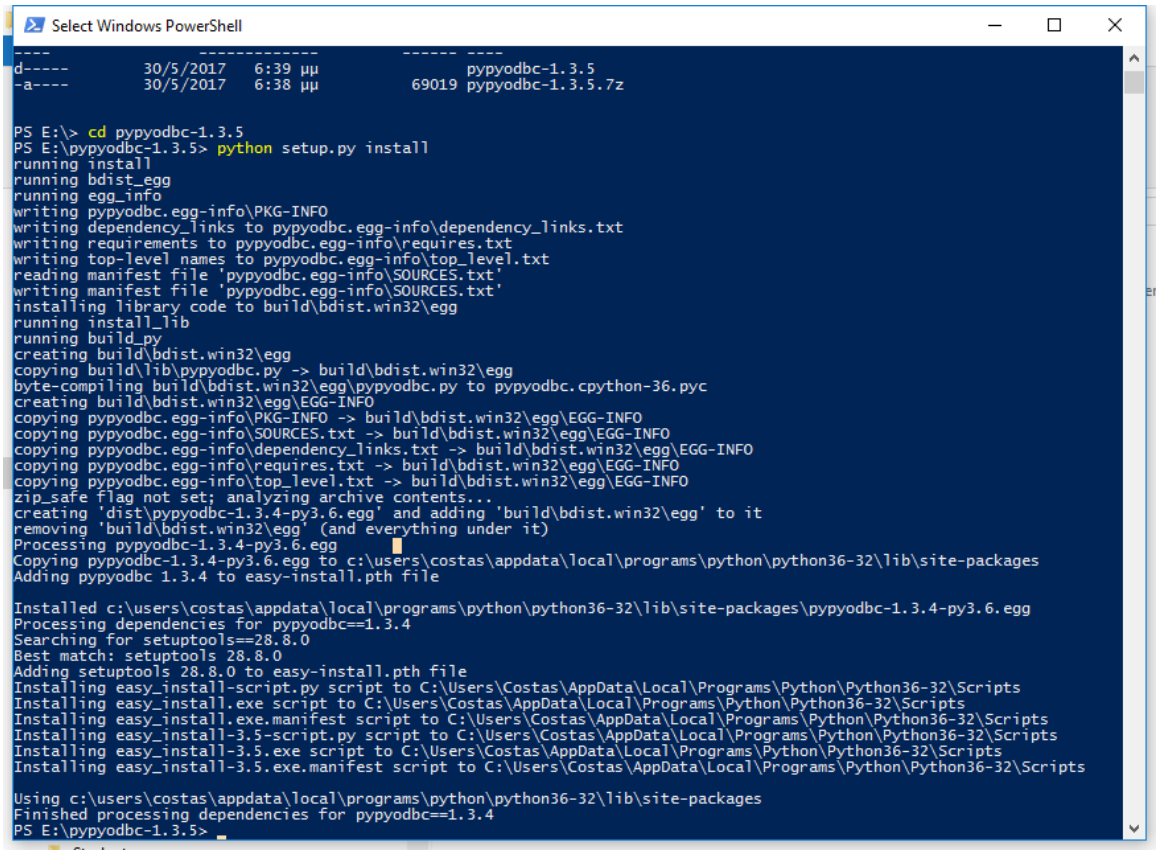
21.2 Ρυθμιση του pypyodbc

Η ρυθμιση του ODBC του υπολογιστή δεν επαρκεί για το χειρισμό μιας Βάσης Δεδομένων μέσω Python. Χρειάζεται στο πρόγραμμα μας να γίνει import το pypyodbc ή αντίστοιχο module για την Python.

Ακολουθούμε τα παρακάτω βήματα:

- ☐ Αναζητούμε το **pypyodbc** μέσω google
- ☐ Κατεβάζουμε το **pypyodbc**
- ☐ Αποσυμπιέζουμε το zip αρχείο
- ☐ Καλούμε το **command prompt** ή το **Windows PowerShell**
- ☐ Πηγαίνουμε στο folder του **`pypyodbc`** και γραφούμε: **`Python setup.py install`**



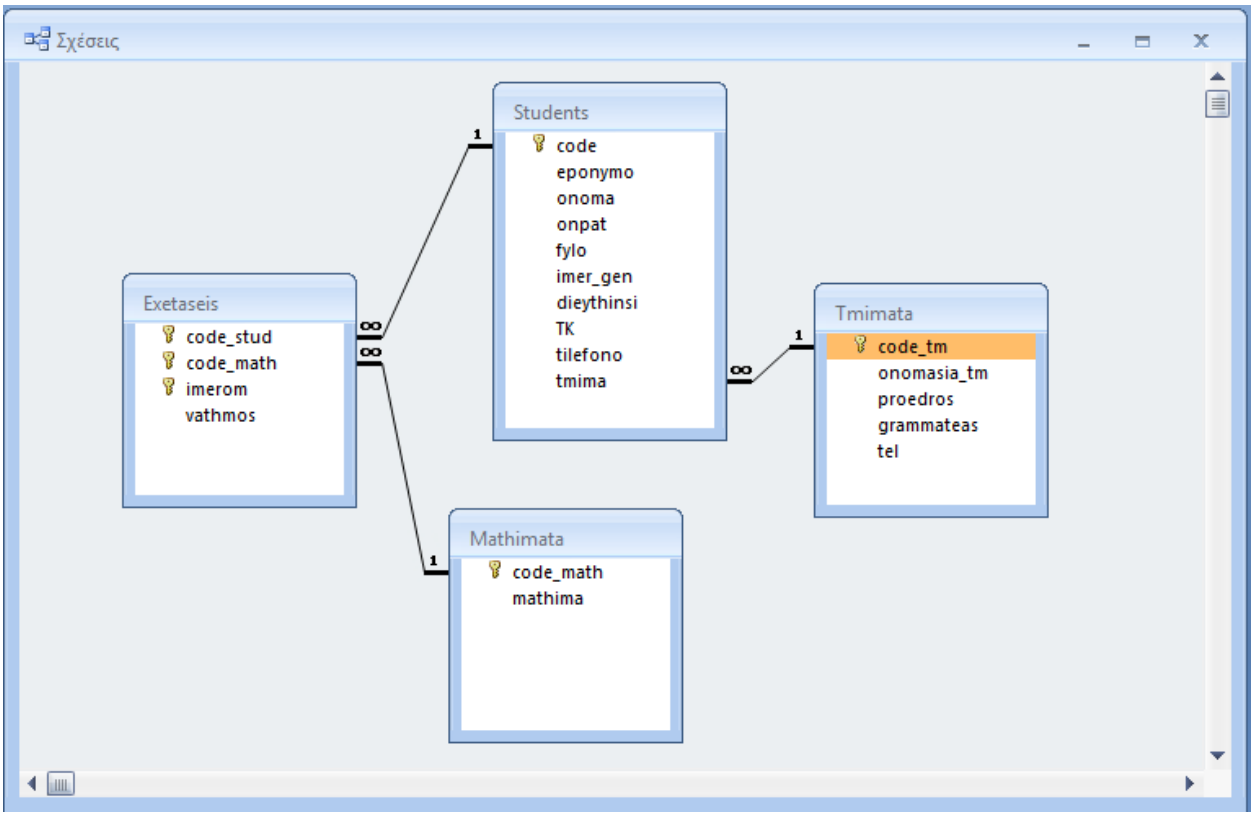


Στην οθόνη του Windows PowerShell εμφανίζονται τα αποτελέσματα επιτυχούς εγκατάστασης του pypyodbc στην εγκατάσταση της Python.

Μετά από την επιτυχή εγκατάσταση του pypyodbc είμαστε έτοιμοι για να γράψουμε πρόγραμμα σε Python για τη χρήση μιας Βάσης Δεδομένων.

21.3 Παραδείγματα με ACCESS

Ας θεωρήσουμε ότι εχουμε μία ΒΔ της Access με **`university.accdb`**. Η συνοπτική δομή των πινάκων της ΒΔ και οι σχέσεις των πινάκων παρουσιάζονται στις παρακάτω εικόνες.



Exetaseis

Όνομα πεδίου	Τύπος δεδομένων
code_stud	Κείμενο
code_math	Αριθμός
imerom	Ημερομηνία/Ώρα
vathmos	Αριθμός

Mathimata

Όνομα πεδίου	Τύπος δεδομένων
code_math	Αριθμός
mathima	Κείμενο

Tmimata

Όνομα πεδίου	Τύπος δεδομένων
code_tm	Αριθμός
onomasia_tm	Κείμενο
proedros	Κείμενο
grammateas	Κείμενο
tel	Κείμενο

Students

Όνομα πεδίου	Τύπος δεδομένων
code	Κείμενο
eponymo	Κείμενο
onoma	Κείμενο
onpat	Κείμενο
fylo	Κείμενο
imer_gen	Ημερομηνία/Ώρα
dieythinsi	Κείμενο
TK	Αριθμός
tilefono	Κείμενο
tmima	Αριθμός

21.3.1 Παράδειγμα ανάγνωσης ΒΔ με κριτήρια

Εστω ότι θέλουμε από τη ΒΔ αυτή να εμφανίσουμε τους φοιτητές ενός συγκεκριμένου Τμήματος του Πανεπιστημίου δίνοντας το κωδικό του Τμήματος. Στην περίπτωση αυτή διακρίνουμε τα εξής βασικά βήματα:

1. Ορισμός της Βάσης Δεδομένων.
2. Σύνδεση με τη Βάση Δεδομένων.
3. Δημιουργία SQL εντολής
4. Εκτέλεση της SQL εντολής
5. Λήψη αποτελεσμάτων της εντολής
6. Εμφάνιση αποτελεσμάτων της εντολής
7. Κλείσιμο της Βάσης Δεδομένων

Παρακάτω επεξηγούνται σχετικά απλά χωρίς λεπτομέριες που σχετίζονται με τις Βάσεις Δεδομένων και SQL τα παραπάνω βήματα.

Ορισμός της Βάσης Δεδομένων

Ο ορισμός της ΒΔ επιτυγχάνεται καθορίζοντας το όνομα της ΒΔ, της θέση της ΒΔ, τον χρήστη που θα συνδεθεί καθώς και το password του χρήστη. Κατά συνέπεια ορίζονται τρεις μεταβλητές τύπου **str**.

Πχ.

```
db_file = "g:/DB-examples/university.accdb"
user = 'admin'
password = ''
```

Σύνδεση με τη Βάση Δεδομένων.

Η σύνδεση με τη Βάση Δεδομένων έχει την παρακάτω ακολουθία

Δημιουργία connection string. Το **connection string** περιέχει τις προηγούμενες πληροφορίες της ΒΔ καθώς και τους **drivers** που χρειάζονται στο ODBC για να γίνει η προσπέλαση της. Παρακάτω η δημιουργία του **connection string** γίνεται σε 3 βήματα με την επιμέρους συνένωση των δύο πρώτων string.

```
odbc_conn_str1 = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
odbc_conn_str2="DBQ={};UID={};PWD={}".format(db_file, user, password)
conn=odbc_conn_str1+odbc_conn_str2
```

Το περιεχόμενο του **connection string conn** θα είναι:

```
DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=g:/DB-examples/university.accdb;UID=admin;PWD=
```

Εκτέλεση των παρακάτω εντολών για τη σύνδεση με τη ΒΔ.

```
syndesi = pypyodbc.connect(conn)
univer_db = syndesi.cursor()
```

Δημιουργία SQL εντολής

Η εντολή SQL θα είναι αποθηκευμένη σε ένα **string**. Οι παράμετροι που θα δίνονται σε στην εντολή όπως για παράδειγα σε περιπτώσεις αναζήτησης κάτω από κριτήρια μπορούν να ενσωματωθούν εύκολα μέσω της μεθόδου **format**.

Παράδειγμα

Χωρίς παραμέτρους

```
sql="select code,eponymo,onoma from students"
```

Με πέρασμα παραμέτρου την μεταβλητή **code_tmimatos**

```
sql="select code,eponymo,onoma from students where tmima={}".format(code_tmimatos)
```

Εκτέλεση της SQL εντολής

Η εντολή που σχηματίσθηκε στο προηγούμενο βήμα στέλνεται προς εκτέλεση στη ΒΔ μέσω της εντολής :

```
univer_db.execute(sql)
```

Η εντολή εκτελείται στη ΒΔ που σχετίζεται με τη **univer_db** δηλ την **university.accdb**

Λήψη αποτελεσμάτων της εντολής

Η λήψη των αποτελεσμάτων της εντολής SQL επιτυγχάνεται με την εντολή:

```
apotelesmata=univer_db.fetchall()
```

Τα αποτελέσματα τοποθετούνται σε μία λίστα, στη συγκεκριμένη περίπτωση **apotelesmata**. Κάθε στοιχείο της λίστας είναι μία **tuple**. Κάθε στοιχείο της **tuple** αποτελείται από μία τη σειρά των πεδίων που καθορίζονται από την εντολή **select** της SQL.

Σε περιπτώση που η αναζήτηση θα επιστρέφει μία μόνον εγγραφή Η λήψη των αποτελεσμάτων της εντολής SQL επιτυγχάνεται με την εντολή:

```
apotelesmata=univer_db.fetchone ()
```

Τα αποτελέσματα τοποθετούνται σε μία **tuple**. Κάθε στοιχείο της **tuple** αποτελείται από μία τη σειρά των πεδίων που καθορίζονται από την εντολή **select** της SQL.

Εμφάνιση αποτελεσμάτων της εντολής

Για την εμφάνιση των αποτελεσμάτων θα επεξεργαστούμε τα στοιχεία της λίστας **apotelesmata**. Όπως αναφέρθηκε κάθε στοιχείο της λίστας είναι μία **tuple**. Με μία επαναληπτική διαδικασία όπως το **for** μπορούμε να πάρουμε κάθε στοιχείο της λίστας. Από κάθε στοιχείο της λίστας είναι απλό να πάρουμε σε ξεχωριστές μεταβλητές το κάθε στοιχείο της tuple και να το εμφανίσουμε ή να το επεξεργαστούμε.

```
for pedio in apotelesmata:
    kodikos=int(pedio[0]
    eponym=pedio[1]
    onom=str(pedio[2])
    print(i,kodikos,eponym,onom)  #
```

Κλείσιμο της Βάσης Δεδομένων

Μετά το τέλος της επεξεργασίας αποδεσμεύουμε τη ΒΔ με την εντολής:

```
univer_db.close()
```

Λύση

```
import pypyodbc
# ανάγνωση ΒΔ με βάση κάποια κριτήρια

db_file = "g:/DB-examples/university.accdb"  # καθορισμός θέσης της ΒΔ
user = 'admin'                               # καθορισμός χρήστη
password = ''                                # ορισμός password
                                             # στην συγκεκριμένη περίπτωση
                                             # δεν υπάρχει password
#ορισμός του connection string  για τη σύνδεση με τη ΒΔ
#γίνεται συνένωση  των παραμέτρων της ΒΔ με τους αντίστοιχους
#drivers ODBC
#η δημιουργία δύο str γίνεται για πρακτικούς λόγους
# προκειμένου να μην είναι μεγάλη η γραμμή του connection string
odbc_conn_str1 = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
odbc_conn_str2="DBQ={};UID={};PWD={}".format(db_file, user, password)
conn=odbc_conn_str1+odbc_conn_str2
print(conn)

syndesi = pypyodbc.connect(conn) # επιτυγχάνεται η σύνδεση με τη ΒΔ
print(type(syndesi))
univer_db = syndesi.cursor()
print(type(univer_db))
# εισαγωγή του κωδικού του Τμήματος
code_tmimatos=int(input('Κωδικός Τμήματος 1-6 :'))

# SQL εντολή για λήψη της ονομασίας του Τμήματος
sql='select onomasia_tm from tmimata where code_tm={}'.format(code_tmimatos)
univer_db.execute(sql) # εκτέλεση της εντολής SQL

apotelesma=univer_db.fetchone()# λήψη των αποτελεσμάτων της εντολής SQL
                               # το αποτέλεσμα είναι μία tuple με ένα στοιχείο
το όνομα του  Τμήματος
tmima=apotelesma[0]    # λήψη του ονόματος του Τμήματος
print(tmima)

# διαμόρφωση της SQL για την εμφάνιση των φοιτητών του συγκεκριμένου Τμήματος
sql="select code,eponymo,onoma from students where
tmima={}".format(code_tmimatos)
univer_db.execute(sql) # εκτέλεση της εντολής SQL
apotelesmata=univer_db.fetchall() # λήψη των αποτελεσμάτων της εντολής SQL
                                  # τα αποτελέσματα τοποθετούνται σε μία λίστα.
                                  # Κάθε στοιχείο της λίστας είναι tuple.
                                  # κάθε στοιχείο της tuple αποτελείται
                                  # από μία τη σειρά των πεδίων
                                  # που καθορίζονται από το  select


i=0
for pedio in apotelesmata:  # για κάθε στοιχείο της tuple
pedio[0],pedio[1],pedio[2]
    i +=1                   # αύξων αριθμός
    kodikos=int(pedio[0])   # 1ο πεδίο της tuple που αντιστοιχεί στον κωδικό
    eponym=pedio[1]         # 2ο πεδίο της tuple που αντιστοιχεί στο επώνυμο
    onom=str(pedio[2])      # 3ο πεδίο της tuple που αντιστοιχεί στο όνομα
    print(i,kodikos,eponym,onom) # εκτύπωση

univer_db.close() # κλείσιμο της ΒΔ
```

21.3.2 Παράδειγμα ενημέρωσης ΒΔ

Στην ίδια Βάση Δεδομένων θέλουμε να αλλάξουμε το ονοματεπώνυμο του προέδρου ενός Τμήματος.Τα βασικά βήματα θα είναι η εξής:

- Θα δίνεται ο κωδικός του Τμήματος
- Θα αναζητάται ο κωδικός του Τμήματος στον αντίστοιχο πίνακα μέσω SQL εντολής
- Θα προβάλεται το Τμήμα και το ονοματεπώνυμο του παλαιού Προέδρου.
- Θα ζητάται το ονοματεπώνυμο του νέου Προέδρου
- Θα σχηματίζεται η νέα εντολή SQL για την ενημέρωση της εγγραφής
- Θα εκτελείται η εντολή SQL
- Ενημερώνεται και κλείνει η ΒΔ

```
import pypyodbc
# ενημέρωση  ΒΔ.
# αλλαγή του ονόματος του Προέδρου ενός Τμήματος

db_file = "g:/DB-examples/university.accdb"  # καθορισμός θέσης της ΒΔ
user = 'admin'                               # καθορισμός χρήστη
password = ''                                # ορισμός password
                                             # στην συγκεκριμένη περίπτωση
                                             # δεν υπάρχει password
#ορισμός του connection string  για τη σύνδεση με τη ΒΔ

odbc_conn_str1 = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
odbc_conn_str2="DBQ={};UID={};PWD={}".format(db_file, user, password)
conn=odbc_conn_str1+odbc_conn_str2

syndesi = pypyodbc.connect(conn) # επιτυγχάνεται η σύνδεση με τη ΒΔ
print(type(syndesi))
univer_db = syndesi.cursor()

# εισαγωγή του κωδικού του Τμήματος
code_tmimatos=int(input('Κωδικός Τμήματος 1-6 :'))

# SQL εντολή για λήψη της ονομασίας του Τμήματος
# και του υφιστάμενου Προέδρου
sql='select * from tmimata where code_tm={}'.format(code_tmimatos)
univer_db.execute(sql) # εκτέλεση της εντολής SQL
apotelesmata=univer_db.fetchone() # λήψη των αποτελεσμάτων της εντολής SQL
tmima=apotelesmata[1] # λήψη του ονόματος του Τμήματος
proedr=apotelesmata[2]# λήψη του ονόματος του Προέδρου

print("Τμήμα {} Πρόεδρος  {}".format(tmima,proedr ))

minima="Δώσε το όνομα νέου προέδρου για το Τμήμα {} :".format(tmima)

chairman=input(minima)

sql_update = "Update Tmimata set proedros='{}' where
code_tm={}".format(chairman,code_tmimatos)
univer db.execute(sql update) # εκτέλεση της εντολής SQL
syndesi.commit() # ενημέρωση της ΒΔ

univer_db.close() # κλείσιμο της ΒΔ
```

21.3.3 Παράδειγμα εισαγωγής εγγραφής σε πίνακα

Στην ίδια Βάση Δεδομένων θέλουμε να προσθέσουμε τα στοιχεία που αφορούν ένα νέο Τμήμα:

Λύση

```
import pypyodbc
# εισαγωγή μιας εγγραφής στη ΒΔ.
# εισαγωγή  ενός  νέου Τμήματος

db_file = "g:/DB-examples/university.accdb"  # καθορισμός θέσης της ΒΔ
user = 'admin'                               # καθορισμός χρήστη
password = ''                                # ορισμός password
                                             # στην συγκεκριμένη περίπτωση
                                             # δεν υπάρχει password
#ορισμός του connection string  για τη σύνδεση με τη ΒΔ

odbc_conn_str1 = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
odbc_conn_str2="DBQ={};UID={};PWD={}".format(db_file, user, password)
conn=odbc_conn_str1+odbc_conn_str2

syndesi = pypyodbc.connect(conn) # επιτυγχάνεται η σύνδεση με τη ΒΔ
print(type(syndesi))
univer_db = syndesi.cursor()
print(type(univer_db))

# εισαγωγή του κωδικού του Τμήματος
code_tmi = int(input('Κωδικός Νέου Τμήματος     :'))
onoma_tm = input("Ονομασία Νέου Τμήματος    :")
chair = input("Ονοματεπώνυμο του Προέδρου    :")
gram = input("Ονοματεπώνυμο του Γραμματέα      :")
tel = input("Τηλέφωνο της Γραμματεία          :")

# SQL εντολή για την εισαγωγή των παραπάνω στοιχείων στην ΒΔ

sql_insert= "insert into Tmimata values ( {}, '{}', '{}', '{}', '{}'
)".format(code_tmi, onoma_tm,chair, gram, tel)
univer_db.execute(sql_insert) # εκτέλεση της εντολής SQL
syndesi.commit() # ενημέρωση της ΒΔ

univer db.close() # κλείσιμο της ΒΔ
```

21.4 Παραδείγματα με SQLite3

Στην παράγραφο αυτή παρουσιάζεται η επεξεργασία της Βάσης Δεδομένων του προηγούμενου παραδείγματα με υλοποίηση σε **sqlite3**. Η μονη ουσιαστική διαφορά με το προηγούμενο πρόγραμμα για της **MSAccess** είναι η διαδικασία σύνδεσης με τη ΒΔ. Δεν απαιτείται η δημιουργία **connection string** και καθορισμός των **drivers** για τη ΒΔ.

```
import sqlite3
# ανάγνωση ΒΔ με βάση κάποια κριτήρια

conn=sqlite3.connect("g:/DB-examples/univerity.acccdb")
univer_db=conn.cursor()      # επιτυγχάνεται η σύνδεση με τη ΒΔ

# εισαγωγή του κωδικού του Τμήματος
code_tmimatos=int(input('Κωδικός Τμήματος 1-6 :'))
```

```
# SQL εντολή για λήψη της ονομασίας του Τμήματος
sql='select * from tmimata where code_tm={}'.format(code_tmimatos)
univer_db.execute(sql) # εκτέλεση της εντολής SQL
apotelesmata=univer_db.fetchall() # λήψη των αποτελεσμάτων της εντολής SQL
tmima=apotelesmata[0][1] # λήψη του ονόματος του Τμήματος
print(tmima)

# διαμόρφωση της SQL για την εμφάνιση των φοιτητών του συγκεκριμένου Τμήματος
sql="select code,eponymo,onoma from students where
tmima={}".format(code_tmimatos)
univer db.execute(sql) # εκτέλεση της εντολής SQL
apotelesmata=univer_db.fetchall() # λήψη των αποτελεσμάτων της εντολής SQL
                                  # τα αποτελέσματα τοποθετούνται σε μία λίστα.
                                  # Κάθε στοιχείο της λίστας είναι tuple.
                                  # κάθε στοιχείο της tuple αποτελείται
                                  # από μία τη σειρά των πεδίων
                                  # που καθορίζονται από το  select
i=0
for pedio in apotelesmata:
# για κάθε στοιχείο της tuple pedio[0],pedio[1],pedio[2]
    i +=1                     # αύξων αριθμός
    kodikos=int(pedio[0])     # 1ο πεδίο της tuple που αντιστοιχεί στον κωδικό
    eponym=pedio[1]           # 2ο πεδίο της tuple που αντιστοιχεί στο επώνυμο
    onom=str(pedio[2])        # 3ο πεδίο της tuple που αντιστοιχεί στο όνομα
    print(i,kodikos,eponym,onom) # εκτύπωση

univer_db.close() # κλείσιμο της ΒΔ
```

22 Λυμένες Ασκήσεις –ΠΡΟΓΡΑΜΜΑΤΑ

22.1 Εισαγωγή δεδομένων-επεξεργασία –έξοδος αποτελεσμάτων

Να υπολογισθεί το πληρωτέο ποσό ενός ωρομίσθιου εργαζόμενου, όταν είναι γνωστές οι ώρες εργασίας, η ωριαία αποζημίωση, το ποσοστό κρατήσεων για ασφάλιση και το ποσοστό κράτησης φόρου.

Ανάλυση του προβλήματος

Δεδομένα

- Ώρες Εργασίας.
- Ωριαία Αποζημίωση.
- Ποσοστό Ασφάλισης.
- Ποσοστό Φόρου.

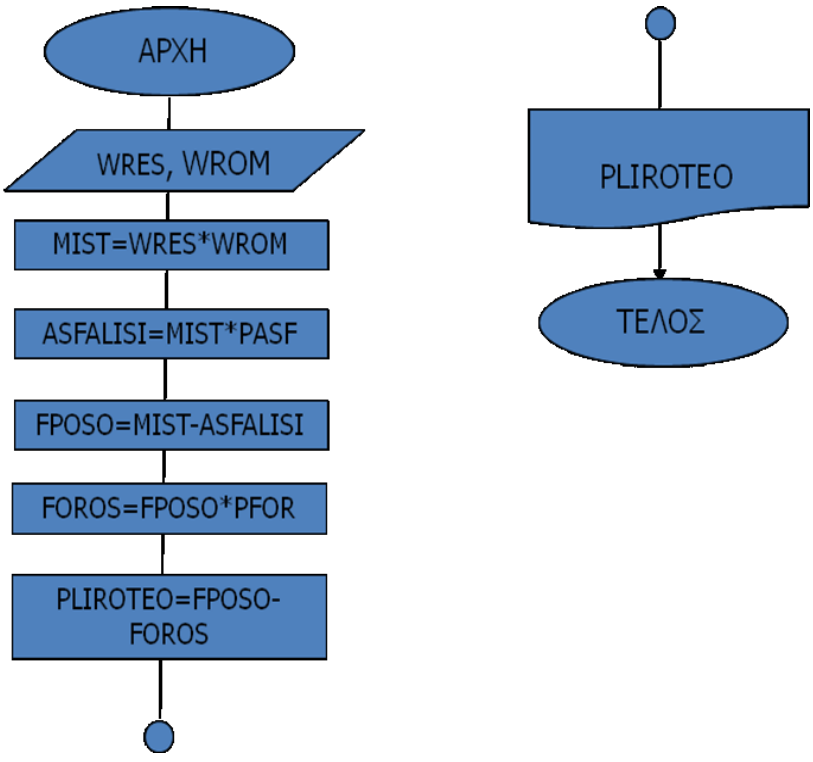
Ζητούμενα

- Πληρωτέο Ποσό

Σχέσεις που συνδέουν Δεδομένα -Ζητούμενα

- **Μισθός** =(Ώρες Εργασίας) * (Ωριαία Αποζημίωση)
- **Κρατήσεις Ασφάλισης** = (Μισθός) * (Ποσοστό Ασφάλισης)
- **Φορολογητέο ποσό** = (Μισθός) – (Κρατήσεις Ασφάλισης)
- **Φόρος** = (Φορολογητέο Ποσό) * (Ποσοστό Φόρου)
- **Πληρωτέο Ποσό** = (Φορολογητέο Ποσό) - (Φόρος)

Διάγραμμα Ροής



```
pforou=0.25  # 25% ποσοστό υπολογισμού φόρου
pasf=0.27    # 27% ποσοστό υπολογισμού ασφαλιστικής εισφοράς

wres=input("Δώσε τις ώρες εργασίας ")    # εισαγωγή ωρών εργασίας
wrom=input("Δώσε το ωρομίσθιο")          # εισαγωγή ωρομισθίου
wres=float(wres)                         # μετατροπή σε πραγματικό αριθμό
wrom=float(wrom)                         # μετατροπή σε πραγματικό αριθμό
mist = wres * wrom                       # υπολογισμό ακαθάριστου μισθού
asfalisi = mist * pasf                   # υπολογισμός ασφαλ. εισφοράς
forol = mist - asfalisi                  # υπολογισμός φορολογητέου ποσού
foros = forol * pforou                   # υπολογισμός φόρου
pliroteo = forol - foros                 # υπολογισμός πληρωτέου ποσού
print("πληρωτέο ποσό ",pliroteo)         # εκτύπωση πληρωτέου ποσού
```

22.2 Ασκήσεις με δομές ελέγχου if

22.2.1 'Ασκηση-1

Να γραφεί πρόγραμμα σε σε Python επιλύει την πρωτοβάθμια εξίσωση ax+b=0

Σύμφωνα με τα μαθηματικά για να λυθεί η εξίσωση εξετάζονται τα παρακάτω

- Αν a≠0 τότε x=-b/a
- Αν a=0 και b=0 τότε αόριστη εξίσωση
- Αν a=0 και b≠0 τότε αδύνατη εξίσωση

Παρακάτω παρουσιάζεται η λύση με τρεις τρόπους:

Λύση 1η

```
print("Λύση εξίσωσης ax+b=0")
a=float(input("Δώσε το a="))# ανάγνωση του a και μετατροπή σε πραγματικό αριθμό
b=float(input("Δώσε το b="))# ανάγνωση του b και μετατροπή σε πραγματικό αριθμό
if a!=0:
    x=-b/a
    print("x=",x)
else:
    if b==0:
        print("αόριστη εξίσωση ")
    else:
        print("αδύνατη εξίσωση ")
```

Λύση 2η

```
print("Λύση εξίσωσης ax+b=0")
a=float(input("Δώσε το a="))# ανάγνωση του a και μετατροπή σε πραγματικό αριθμό
b=float(input("Δώσε το b="))# ανάγνωση του b και μετατροπή σε πραγματικό αριθμό
if a!=0:
    x=-b/a
    print("x=",x)
elif b==0:
        print("αόριστη εξίσωση ")
else:
        print("αδύνατη εξίσωση ")
```

Λύση 3η

```
print("Λύση εξίσωσης ax+b=0")
a=float(input("Δώσε το a="))# ανάγνωση του a και μετατροπή σε πραγματικό αριθμό
```

```
b=float(input("Δώσε το b="))# ανάγνωση του b και μετατροπή σε πραγματικό αριθμό
if a!=0:
    x=-b/a
    print("x=",x)
if a==0 and  b==0:
        print("αόριστη εξίσωση ")
if a==0 and  b!=0:
        print("αδύνατη εξίσωση ")
```

22.2.2 Άσκηση-2

Να γραφεί πρόγραμμα σε Python που να διαβάζει από την οθόνη το ονοματεπώνυμο ενός φοιτητή και τους βαθμούς που έλαβε σε τρείς προόδους. Στη συνέχεια:

- Υπολογίζει τον ΜΟ
- Εμφανίζει το όνομα του φοιτητή και μήνυμα "ΕΠΙΤΥΧΩΝ" αν ο μέσος όρος είναι μεγαλύτερος ή ίσος του 5, αλλιώς εμφανίζει "ΑΠΟΤΥΧΩΝ".

Λύση

```
print("Πρόγραμμα υπολογισμού ΜΟ βαθμολογίας 3 βαθμών ")
onoma=input("Ονοματεπώνυμο ")
bathmos1=float(input("1η Βαθμολογία "))
bathmos2=float(input("2η Βαθμολογία "))
bathmos3=float(input("3η Βαθμολογία "))
mesos=(bathmos1+bathmos2+bathmos3)/3
if mesos>=5:
    print(onoma, mesos," επιτυχών ")
else:
     print(onoma, mesos," αποτυχών ")
```

22.2.3 Άσκηση-3

Να γραφεί πρόγραμμα σε python που να διαβάζει τα παρακάτω στοιχεία πώλησης ενός τυροκομείου:

- Κωδικός προϊόντος (ακέραιος, με τιμές 1= ΦΕΤΑ, 2= ΓΡΑΒΙΕΡΑ 3= ΑΝΘΟΤΥΡΟ)
- Ποσότητα προϊόντος σε κιλά (πραγματικός)

και να υπολογίζει και εμφανίζει το αντίστοιχο κέρδος από τον πώληση λαμβάνοντας υπόψη τον παρακάτω πίνακα:

Προϊόν	**Κέρδος/κιλό σε ευρώ**
φέτα	**1.5**
Γραβιέρα	**2.5**
ανθότυρο	**2.0**

Παρακάτω παρουσιάζεται η λύση με δύο τρόπους

Λύση-1

```
# χωρίς ελεγχο για το κωδικό είδους
code_eidous=int(input("δωσε το είδος 1=φετα 2= γραβιέρα 3=ανθοτυρο :"))
posotita=float(input("δώσε την ποσότητα του προϊόντος σε κιλά :"))
if code_eidous==1:
```

```
    eidos="Φέτα"
    kerdos=posotita*1.5
if code_eidous==2:
    eidos="Γραβιέρα"
    kerdos=posotita*2.5
if code_eidous==3:
    eidos="Ανθότυρο"
    kerdos=posotita*2.0
print("το κέρδος για το προϊόν ",eidos, " είναι ",  kerdos ," ευρώ")
```

Λύση-2

```
#  με  ελεγχο για το κωδικό είδους
code_eidous=int(input("δωσε το είδος 1=φετα 2= γραβιέρα 3=ανθοτυρο :"))
if code_eidous==1 or code_eidous==2 or code_eidous==3:
    posotita=float(input("δώσε την ποσότητα του προϊόντος σε κιλά :"))
    if code_eidous==1:
        eidos="Φέτα"
        kerdos=posotita*1.5
    elif code_eidous==2:
        eidos="Γραβιέρα"
        kerdos=posotita*2.5
    elif code_eidous==3:
        eidos="Ανθότυρο"
        kerdos=posotita*2.0
    print("το κέρδος για το προϊόν ",eidos, " είναι ",  kerdos ," ευρώ")
else:
    print("λάθος κωδικός είδους ")
```

22.3 Ασκήσεις με δομή επανάληψης for

22.3.1 Άσκηση-1 με for

Να γραφεί πρόγραμμα το οποίο να εμφανίζει στην οθόνη το άθροισμα των αριθμών από 1 έως και 100.

Λύση-1

```
# Να γραφεί πρόγραμμα το οποίο να εμφανίζει στην οθόνη  το άθροισμα των
αριθμών από 1 έως και 100.
sum = 0
for ar in range(100):
    sum=sum+ar+1
print(sum)
```

Λύση-2

```
# Να γραφεί πρόγραμμα το οποίο να εμφανίζει στην οθόνη  το άθροισμα των
αριθμών από 1 έως και 100.
sum = 0
for ar in range(1,101):
    print(ar)
    sum=sum+ar
print(sum)
```

22.3.2 Άσκηση-2 με for

Να γραφεί πρόγραμμα το οποίο να διαβάζει από την οθόνη 5 αριθμούς και στο τέλος να εμφανίζει τον μεγαλύτερο.

Λύση

```
for i in range(1,6):
    print(i,"ος  αριθμός ",end =' ')
    ar=float(input(" :"))
    if i==1 :
        megistos=ar
    else:
        if ar>megistos:
            megistos=ar;
print("ο μεγαλύτερος είναι ",megistos)
```

22.3.3 Άσκηση-3 με for

Να γραφεί πρόγραμμα το οποίο Διαβάζει 10 αριθμούς, υπολογίζει το άθροισμα των αριθμών αυτών.

Μετά το τέλος εισαγωγής των στοιχείων το πρόγραμμα εμφανίζει στην οθόνη

- το άθροισμα των αριθμών αυτών
- τον μεγαλύτερο από τους αριθμούς που δόθηκαν
- τον μικρότερο από τους αριθμούς που δόθηκαν

Λύση

```
sum=0
for i in range(1,11):
    print(i,"ος  αριθμός ",end =' ')
    ar=float(input(" :"))
    sum+=ar  # αυτο είναι ισοδύναμο με sum=sum+ar
    if i==1 :
        megistos=ar
        mikroteros=ar
    else:
        if ar>megistos:
            megistos=ar;
        else:
            if ar<mikroteros:
                mikroteros=ar
print("το άθροισμα των αριθμών είναι ",sum)
print("ο μεγαλύτερος είναι ",megistos)
print("ο μικρότερος  είναι ",mikroteros)
```

22.4 Ασκήσεις με λιστες

22.4.1 Άσκηση-1 λιστών

Να γραφεί πρόγραμμα που να για κάθε μία ημέρα της εβδομάδας διαβάζει τη μέση θερμοκρασία της ημέρας. Μετά το τέλος εισαγωγής των στοιχείων εμφανίζει στην οθόνη:

- Τη μέση θερμοκρασία της εβδομάδας
- Το πλήθος των ημερών με θερμοκρασία μεγαλύτερη της μέσης εβδομαδιαίας θερμοκρασίας

- Για κάθε μία ημέρα της εβδομάδας την θερμοκρασία και τη διαφορά της από τη μέση θερμοκρασία.
- Την τυπική απόκλιση της θερμοκρασίας

Η μέση τιμή ***m*** και η τυπική απόκλιση ***s*** των τιμών x_i με ***i=1..n*** δίδονται από τους τύπους:

$$m = \frac{\sum_{i=1}^{n} x_i}{n} \qquad s = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i - m)^2}$$

Λύση

```
import  math
# λιστα για την αποθήκευση των θερμοκρασιών καθε μιάς από τις 7 ημέρες της εβδομάδας
thermo=[0,0,0,0,0,0,0]
imera=["Δευτέρα","Τρίτη","Τεταρτη","Πέμπτη","Παρασκευή","Σαββάτο","Κυριακή"]
#εισαγωγή δεδομένων και καταχώριση σε λίστα
for i in range(7):
    print(" μέση θερμοκρασία  για την ",imera[i], end=" ")
    thermo[i]=float(input(" :"))
# υπολογισμός μέσης εβδομαδιαίας θερμοκρασίας
sum=0
for i in range(7):
    sum=sum+thermo[i]
mesi=sum/7
print("Μέση θερμοκρασία εβδομάδας ",format(mesi,'+.2f'))
# πλήθος ημερών με θερμοκρασία μεγαλύτερη  της μέσης
pl=0
for i in range(7):
    if thermo[i]>mesi:
        pl+=1
print("Πλήθος ημερών με θερμοκρασία μεγαλύτερη  της μέσης ",pl)

# εμφάνιση της διαφορά θερμοκρασία κάθε ημέρας από τη μέση εβδομαδιαία
for i in range(7):
    diafora=mesi-thermo[i]
    # εκτύπωση με 2 δεκαδικά ψηφία και εμφάνιση του προσήμου με χρήση format
    print(imera[i], " διαφορά από την μέση ", format(diafora, '+.2f'))
#υπολογισμός και εμφάνιση της τυπικής απόκλισης της θερμοκρασίας στη διάρκεια της εβδομάδας
sum=0
for i in range(7):
    sum=sum+(thermo[i]-mesi)**2
sum=sum/7
s=math.sqrt(sum)
print("η τυπική απόκλιση είναι :",format(s,'.4f'))
```

22.4.2 Άσκηση -2 λιστών

Να γραφεί πρόγραμμα το οποίο να διαβάζει από την οθόνη τα στοιχεία 10 φοιτητών

- Κωδικός φοιτητή
- Ονοματεπώνυμο φοιτητή

- Βαθμός-1
- Βαθμός-2
- Βαθμός-3

Και να τα αποθηκεύει σε κατάλληλες λίστες.

Στη συνέχεια να ζητά από τον χρήστη έναν αριθμό που να αντιστοιχεί σε **κωδικό φοιτητή**, να αναζητά τον φοιτητή και

άν υπάρχει να εμφανίζει στην οθόνη:

- Τον κωδικό,
- το ονοματεπώνυμο,
- τις τρεις βαθμολογίες,
- τον μέσο όρο βαθμολογίας

αν δεν υπάρχει εμφανίζει στην οθόνη κατάλληλο μήνυμα

Δομή των Λιστών

α/α	ΑΜ	Ονομ/νυμο	Β-1	Β-2	Β-3
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

```
# Δημιουργία κενών λιστών για την καταχώριση των δεδομένων
kodikos=[]
eponymo=[]
bathmos1=[]
bathmos2=[]
bathmos3=[]

#εισαγωγή στοιχείων και καταχώριση σε λίστες

for i in range(10):
    #εισαγωγή των δεδομένων
    kod= int(input("κωδικός φοιτητή "))
    epon= input("επωνυμο φοιτητή :")
    bath1=float(input("1ο βαθμός   :"))
    bath2=float(input("2ος βαθμός  :"))
    bath3=float(input("3ος βαθμός  :"))
    #καταχώριση των δεδομένων στις αντίστοιχες λίστες
    kodikos.append(kod)
    eponymo.append(epon)
    bathmos1.append(bath1)
    bathmos2.append(bath2)
```

```
    bathmos3.append(bath3)

# αναζήτηση
print(" αναζήτηση με βάση τον κωδικό φοιτητή")
kod=int(input("κωδικός φοιτητή "))
if kod not in kodikos: # έλεγχος ύπαρξης του κωδικού  kod στη λίστα kodikos
    print("Δεν υπάρχει αυτός ο κωδικός ")
else:
    thesi=kodikos.index(kod) # βρίσκει τη θέση του kod  στη λίστα kodikos
    mesos=(bathmos1[thesi]+bathmos2[thesi]+bathmos3[thesi])/3
    print(kod,eponymo[thesi],format(mesos,'.2f'))
```

22.5 Ασκήσεις με λίστες με περιεχόμενα λίστες

22.5.1 Άσκηση -1 με λίστα με περιεχόμενα λίστες

Να γραφεί πρόγραμμα που να διαβάζει από την οθόνη 21 αριθμούς και να τους αποθηκεύει μία λίστα x, 3 στοιχείων που κάθε στοιχείο είμαι μια λίστα με 7 θέσεις.

Στη συνέχεια να εμφανίζει στην οθόνη:

- τα στοιχεία της x σε μορφή πίνακα (3 γραμμές, 7στήλες)
- τον μέσο όρο όλων των αριθμών κάθε γραμμής

Λύση-1

```
x=[[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0]] # ορισμό της λίστας
aa=0  # μετρητής των στοιχείων
#εισαγωγή δεδομένων
for i in range(3): # για κάθε μία από τις 3 λίστες της λίστας x
    for j in range(7): # για κάθε στοιχείο της καθε μιας από τις 3 λίστες
        aa+=1
        # διαμόρφωση του μηνύματος στον χρήστη
        print('δώσε τον {}ο αριθμό '.format(aa), end=":")

        x[i][j]=int(input("  ")) # εισαγωγή του στοιχείου

for i in range(3):
    for j in range(7):
        print(format(x[i][j],'4d')," ",end= "")
    print(" ")

for i in range(3):
    sum=0  # αθροιστής στοιχείων στοιχείων
    for j in range(7):
        sum+=x[i][j]  #υπολογισμός αθροίσματος στοιχείων
    mesos=sum/7  # ευρεση μέσου όρου
    print('ΜΟ γραμμής', i+1,end= "   :")
    print(format(mesos, '.3f'))
for i in range(3):
    for j in range(7):
        print(format(x[i][j],'4d')," ",end= "")
    print("")
```

Λύση-2

```
x=[]# ορισμός της λίστας
```

```
# δημιουργία  της λίστα x=[[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0]]

for i in range(3):
    grammi=[]
    for j in range(7):
        grammi.append(0)
    x.append(grammi)


aa=0  # μετρητής των στοιχείων
#εισαγωγή δεδομένων
for i in range(3): # για κάθε μία από τις 3 λίστες της λίστας x
    for j in range(7): # για κάθε στοιχείο της καθε μιας από τις 3 λίστες
        aa+=1
        # διαμόρφωση του μηνύματος στον χρήστη
        print('δώσε τον {}ο αριθμό '.format(aa), end=":")
        x[i][j]=int(input("  ")) # εισαγωγή του στοιχείου

for i in range(3):
    for j in range(7):
        print(format(x[i][j],'4d')," ",end= "")
    print("")
for i in range(3):
    sum=0  # αθροιστής στοιχείων στοιχείων
    for j in range(7):
        sum+=x[i][j]  #υπολογισμός αθροίσματος στοιχείων
    mesos=sum/7  # ευρεση μέσου όρου
    print('ΜΟ γραμμής', i+1,end= "   :")
    print(format(mesos, '.3f'))
```

22.5.2 Άσκηση -2 με λίστα με περιεχόμενα λίστες

Σε ένα παρατηρητήριο τιμών της αγοράς καταγράφονται οι τιμές ενός προϊόντος στη διάρκεια της εβδομάδας σε **4** super market. Να γραφεί πρόγραμμα που:

- Να διαβάζει από την οθόνη τα ονόματα των **super market** και να τα αποθηκεύει σε μία λίστα με ονομα **sm**
- Για κάθε super market και για κάθε ημέρα της εβδομάδας, από τις **6** εργάσιμες ημέρες, διαβάζει την τιμή του προϊόντος και την αποθηκεύει σε μία λιστα με όνομα timi(**4x6)**.
- Για κάθε super market υπολογίζει την μέση τιμή του προϊόντος στο τέλος της εβδομάδας και την καταχωρεί σε μία λίστα.
- Βρίσκει και εμφανίζει το super market με την μικρότερη μέση τιμή του προϊόντος.
- Βρίσκει τη μέση τιμή πώλησης του προϊόντος στο τέλος της εβδομάδας.

Λύση

```
import random
sm=[] # λιστα με τα ονόματα των SM
timi=[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]] # ορισμό της λίστας με
τις τιμές ανά SM και προϊόν

imera=["Δευτέρα","Τρίτη","Τεταρτη","Πέμπτη","Παρασκευή","Σαββάτο"] # λίστα με τις
εργάσιμες  ημέρες της εβδομάδας
mt=[] # λίστα με τις εβδομάδιαίες μέσες τιμές ανά SM

#εισαγωγή δεδομένων
```

```
# εισαγωγή των ονομάτων των SM
for i in range(4):
    onoma=input("Δωσε το όνομα του SM :")
    sm.append(onoma)
# εισαγωγή της τιμής για κάθε SM και για κάθε ημέρα της εβδομάδας
for i in range(4): # για κάθε μία από τις 4 λίστες της λίστας timi
    for j in range(6): # για κάθε στοιχείο της καθε μιας από τις 4 λίστες

        # διαμόρφωση του μηνύματος στον χρήστη
        print('Για το SM {} δώσε την τιμή προϊόντος την {} '.format(sm[i],imera[j]),
end=":")
        timi[i][j]=random.randrange(170,200)

       #timi[i][j]=float(input("  ")) # εισαγωγή του στοιχείου
print("")
print("=============================================")
# για κάθε SM υπολογίζει την μέση τιμή του προϊόντος στο τέλος της εβδομάδας
# και καταχωρεί το αποτέλεσμα στη λίστα mt
for i in range(4):
    sum=0
    for j in range(6):
        sum+=timi[i][j]
    mesi=sum/6
    mt.append(mesi)

#για κάθε SM εμφανίζει  την μέση τιμή του προϊόντος στο τέλος της εβδομάδας
for i in range(4):
    print("SM ",sm[i],format(mt[i],'.3f'))

fthino=min(mt)# βρίσκει τη φθηνότερη  μέση τιμή
thesi=mt.index(fthino) # βρίσκει τη θέση στη λίστα που αντιστοιχεί στη
                       #φθηνότερη  μέση τιμή

print(" το φθηνότερο είναι το SM:",sm[thesi])

sum=0
for i in range (4):
    sum+=mt[i]
mesi=sum/4
print("η μέση εβδομαδιαία τιμή του προϊόντος ειναι:", format(mesi,'.3f'))
```

22.6 Ασκήσεις με τη δομή Επανάληψης while

22.6.1 Άσκηση -1

Να γραφεί πρόγραμμα με χρήση while το οποίο εμφανίζει στην οθόνη το άθροισμα των αριθμών από 1 έως και 100.

Λύση -1

```
k=0
sum=0
while k<=100:
    sum+=k
    k+=1
print("Αθροισμα ",sum)
```

Λύση-2

```
k=0
sum=0
while k<100:
    k+=1
```

```
    sum+=k

print("Άθροισμα ",sum)
```

22.6.2 Άσκηση -2

Να γραφεί πρόγραμμα το οποίο να διαβάζει από την οθόνη:

- Το επώνυμο ενός φοιτητή
- Την βαθμολογία του σε ένα μάθημα ελέγχοντας ότι ο βαθμός περιέχεται στο διάστημα [0-10].

Η διαδικασία επαναλαμβάνεται μέχρι να δοθεί ως επώνυμο η λέξη “ΤΕΛΟΣ”

Μετά το τέλος εισαγωγής στοιχείων το πρόγραμμα εμφανίζει:

- Το επώνυμο του φοιτητή με τη μεγαλύτερη βαθμολογία
- Το μέσο όρο της βαθμολογίας στο μάθημα

```
eisagogi=True
max_bathmos=-1 # δινεται ένας βαθμός εκτός ορίων προκειμένου κατά την
εισαγωγή
               # δεδομένων να μπορεί να γίνει έλεγχος για τον πρώτο φοιτητή
max_epon=""
sum=0
pl=0
while eisagogi:
    eponymo=input("επώνυμο φοιτητή ή ΤΕΛΟΣ για ολοκλήρωση εργασίας :")
    if eponymo!="ΤΕΛΟΣ":
        epan=True
        while epan:
            bathmos=float(input("βαθμός φοιτητή [0-10] "))
            if bathmos<0 or bathmos>10:
                print("λάθος βαθμολογία !")
            else:
                epan=False
        if bathmos>max_bathmos:
            max_bathmos=bathmos
            max_epon=eponymo
        sum+=bathmos
        pl+=1
    else:
        eisagogi=False
if pl>0:
    mesos=sum/pl
    print("φοιτητή με το μεγαλύτερο βαθμό ο/η ",max_epon, " με ",
max_bathmos)
    print("μέσος όρος  βαθμολογίας στο μάθημα ",format(mesos, ".2f"))
else:
    print("δεν δόθηκαν στοιχεία ")
```

22.6.3 Άσκηση -3

Να γραφεί πρόγραμμα που να διαβάζει από την οθόνη τα παρακάτω στοιχεία ελέγχου πρατηρίων καυσίμων.

- το ΑΦΜ του εμπόρου(**afm**: χαρακτήρες)

- το ονοματεπώνυμο του (**onoma**: χαρακτήρες)
- το είδος καυσίμου (**eidos**: ακέραιος με τιμές 1=βενζίνη super, 2=βενζίνη αμόλυβδη και 3=πετρέλαιο **)**
- την τιμή αγοράς ανά λίτρο (**timi_a**: πραγματικός)
- την τιμή πώλησης ανά λίτρο(**timi_p**: πραγματικός)

στην συνέχεια υπολογίζει το ποσοστό κέρδους ανά λίτρο του με το οποίο πουλάει ο έμπορος: Εάν το ποσοστό κέρδους είναι μεγαλύτερο από **12%** εμφανίζει στην οθόνη τα στοιχεία που διάβασε καθώς και το ποσοστό κέρδους.

Η εισαγωγή των δεδομένων τερματίζεται όταν ως ΑΦΜ εμπόρου, δοθεί "**999999999**"

Μετά το τέλος της εισαγωγής των δεδομένων:

- Υπολογίζει και εμφανίζει τον αριθμό των εμπόρων με υπερβολικό κέρδος.
- Την μέση τιμή πώλησης κάθε καυσίμου.

Λύση

```
kaysimo=["βενζίνη super","βενζίνη αμόλυβδη","πετρέλαιο"]
timi_pol=[0,0,0] # λιστα για τα αθροίσματα πώλησης τιμής καυσίμου
plithos=[0,0,0]  # λίστα για πλήθος πώλησης κάθε κατηγορίας καυσίμου
epan=True # αρχική τιμή για την είσοδο στην επανάληψη
pl=0
while epan:
    afm=input("ΑΦΜ εμπόρου ή 999999999  για τέλος επεξεργασίας ")
    if afm!='999999999':
        eponymo=input("Ονοματεπώνυμο εμπόρου ")
        lathos=True  #αρχική τιμή για την είσοδο στην επανάληψη
        while lathos:
            eidos=int(input("ειδος καυσίμου 1=βενζίνη super 2=βενζίνη αμόλυβδη
3=πετρέλαιο "))
            if eidos==1 or eidos==2 or eidos==3:
                lathos =False
        lathos=True
        while lathos:  #αρχική τιμή για την είσοδο στην επανάληψη
            timi_a=float(input("τιμή αγοράς "))
            if timi_a>0:
                lathos=False
        lathos=True
        while lathos: #αρχική τιμή για την είσοδο στην επανάληψη
            timi_p=float(input("τιμή πώλησης "))
            if timi_p>0:
                lathos=False
        p_kerdous=100*(timi_p-timi_a)/timi_a  # υπολογισμός ποσοστού κέρδους
        if p_kerdous>12:
            print(afm,eponymo,kaysimo[eidos-1],timi_a,timi_p)
            pl+=1
        timi_pol[eidos-1]+=timi_p# timi_pol[eidos-1]=timi_pol[eidos-1]+timi_p
        plithos[eidos-1]+=1 #  plithos[eidos-1]=plithos[eidos-1]+1

    else:
        epan=False

print("αποτελέσματα ")
print ("πλήθος με ποσοστό κέρδους >12% :", pl)
for i in range (3):
    if plithos[i]>0:
        mt=timi_pol[i]/plithos[i]
        print(kaysimo[i],mt)
```

22.7 Ασκήσεις με συναρτήσεις χρήστη

22.7.1 Ασκήση-1

Να γραφεί πρόγραμμα το οποίο να διαβάζει από την οθόνη:

- Τον ΑΜ του εργαζόμνου,
- το ονοματεπώνυμο του,
- το βασικό μισθό του (ΒΜ)
- τον αριθμό υπερωριών του.

Υπολογίζει

- το ποσό αμοιβής υπερωρίας.Το ποσό αυτό υπολογίζεται ως το πηλίκο:$yp = \frac{BM}{150}$. Το ποσό αμοιβής της υπερωρίας δεν μπορεί να υπερβαίνει τα 18€ την ώρα
- Το ποσό των υπερωριών
- τις αποδοχές του
- τις κρατήσεις για την ασφάλιση του ως 16 % των αποδοχών του
- την εισφορά του εργοδότη ως 28.06% επι τωναποδοχών του.
- Το συνολικό ποσό καταβολής στο Ασφαλιστικό Ταμείο
- Το φορολογητέο ποσό (αποδοχές –κρατήσεις ασφάλισης)
- Υπολογίζει τον φόρο.Για τον υπολογισμό το φορολογητέο ποσό ανάγεται σε ετήσιο, πολλαπλασιαζόμενο με 14 (12 μισθοί+ δώρο Χριστουγέννων+ δώρο Πάσχα+ επίδομα άδειας) και υπολογίζεται ο φόρος με βάση τον παρακάτω πίνακα.

Ετήσιο εισόδημα	**Ποσοστό φορολόγησης**
10000	0%
(10000,25000]	8%
(25000,30000]	15%
(30000,40000]	30%
>40000	40%

Στην συνέχεια ο υπολογισθείς ετήσιος φόρος ανάγεται σε μηνιαίο διαιρούμενος με 14

Μετά τους υπολογισμούς εμφανίζει στην οθόνη:

- Τον ΑΜ του εργαζόμενου
- Το ονοματεπώνυμο του
- Τις ακαθάριστες αποδοχές
- Τις κρατήσεις ασφάλισης
- Το φόρο
- Το Πληρωτέο ποσό

Η διαδικασία επαναλαμβάνεται μέχρις ότου να δοθεί ως **ΑΜ του εργαζόμενου ο αριθμός 999**

Μετά το τέλος της επεξεργασίας εμφανίζει στην οθόνη:

- Το συνολικό ποσό καταβολής στο ασφαλιστικό ταμείο
- Το συνολικό ποσό καταβολής φόρου στην Εφορία
- Το συνολικό ποσό πληρωτέο ποσό

Λύση-1

```
def yp_forou(ap):# συνάρτηση υπολογισμού φόρου
    poso=ap*14 # αναγωγή σε ετήσια βάση
    if poso<=10000:
        fo=0
    else:
        if poso<=25000:
            fo=0*10000+(poso-10000)*0.08
        else:
            if poso<=30000:
                fo=0*10000+15000*0.08+(poso-25000)*0.15
            else:
                if poso<=40000:
                    fo=0*10000+15000*0.08+5000*0.15 +(poso-30000)*0.3
                else:
                    fo=0*10000+15000*0.08+5000*0.15 +10000*0.3+(poso-40000)*0.4
    fo=fo/14 # αναγωγή σε μηνιαία βάση
    return fo

def yp_yper(mistos, wres):# συνάρτηση υπολογισμού αποζημίωση υπερωρίας
    wrom=mistos/150
    if wrom >18:
        wrom=18
    return wrom


sum_asf=0
sum_for=0
sum_pli=0
while True:
    am=int(input("Αριθμός Μητρώου :"))
    if am!=999:
        onoma=input("Ονοματεπώνυμο :")
        bm=float(input("Βασικός μισθός :"))
        yp=float(input("αριθμός υπερωριών :"))
        wr=yp_yper(bm,yp) # κλήση συνάρτησης για τον υπολογισμό αποζημίωσης υπερωρίας
        yper=wr*yp # υπολογισμός αμοιβής υπερωριών
        apodoxes= bm + yper # υπολογισμός συνόλου αποδοχών
        kra_asf= 0.16 * apodoxes  # υπολογισμός κρατήσεων ασφάλισης εργαζόμενου
        eis_asf= 0.2806 * apodoxes # υπολογισμός εισφοράς ασφάλισης εργοδότη
        asfalisi=kra_asf+eis_asf    # σύνολο κρατήσεων και εισφοράς  ασφάλισης
        forol= apodoxes - kra_asf  # φορολογητέο ποσό
        foros=yp_forou(forol)       # κλήση συνάρτησης υπολογισμού φόρου
        pliroteo=forol-foros        # πληρωτέο ποσό
        sum_asf+=asfalisi           # συνολικό ποσό προς ασφαλιστικό ταμείο
        sum_for+=foros             # συνολικό ποσό προς εφορία
        sum_pli+=pliroteo # συνολικό πληρωτέο ποσό
        print(am,onoma,  end='')
        print(format(bm," ,.2f"),format(yper,",.2f"),format(apodoxes,",.2f"),end='')
        print(format(kra_asf,",.2f"),format(foros,",.2f"),format(pliroteo,",.2f"))
    else:
        break
print("=================================================")
print("Συνολικό ποσό προς Ασφαλιστικού Ταμείου :",format(sum_asf, '10,.2f'))
print("Συνολικό ποσό προς Εφορία               :",format(sum_for, '10,.2f'))
print("Συνολικό πληρωτέο ποσό                  :",format(sum_pli, '10,.2f'))
```

Λύση -2

Με χρήση **module** και στογγυροποίηση στα δύο δεκαδικά.

Δημιουργούμε ένα προγραμμα python με όνομα **ypologismoi.py** που θα περιέχει τις συναρτήσεις υπολογισμού φόρου και υπερωριών.

Το **module ypologismoi.py**. Το **module** θα πρέπει να αποθηκευτεί στο ίδιο **folder** με το κύριο πρόγραμμα.

```
# module υπολογισμών μισθοδοσίας
def yp_forou(ap):# συνάρτηση υπολογισμού φόρου
    poso=ap*14 # αναγωγή σε ετήσια βάση
    if poso<=10000:
        fo=0
    else:
        if poso<=25000:
            fo=0*10000+(poso-10000)*0.08
        else:
            if poso<=30000:
                fo=0*10000+15000*0.08+(poso-25000)*0.15
            else:
                if poso<=40000:
                    fo=0*10000+15000*0.08+5000*0.15 +(poso-30000)*0.3
                else:
                    fo=0*10000+15000*0.08+5000*0.15 +10000*0.3+(poso-40000)*0.4
    fo=round(fo/14,2) # αναγωγή σε μηνιαία βάση & στρογγυλοποίηση σε 2 δεκαδικά
    return fo  # επιστροφή του υπολογισμένου φόρου

def yp_yper(mistos, wres):# συνάρτηση υπολογισμού αποζημίωση υπερωρίας
    wrom=round(mistos/150,2) # στρογγυλοποίηση σε 2 δεκαδικά
    if wrom >18:
        wrom=18.
    return wrom # επιστροφή του υπολογισμένου ωρομισθίου υπερωρίας

def efka(ap):
     kratiseis= 0.16 * ap  # υπολογισμός κρατήσεων ασφάλισης εργαζόμενου
     eisfora= 0.2806 * ap      # υπολογισμός εισφοράς ασφάλισης εργοδότη
     synolo=kratiseis+eisfora   # σύνολο κρατήσεων και εισφοράς  ασφάλισης
     return kratiseis,eisfora,synolo
```

Δημιουργούμε ένα προγραμμα **python** με όνομα **mistos.py** στο οποίο θα γίνεται εισαγωγή του **module ypologismoi**. Στο πρόγραμμα αυτό οι κλήσεις των συναρτήσεων υπολογισμού φορου και υπερωρίας θα γίνονται με πρόθεμα το ονομα του **module ypologismoi**. Δηλαδή η κλήση υπολογισμου του φόρου ενός συγκεκριμένου εισοδήματος και ανάθεση της τιμή του φόρου σε μία μεταβλητή με ονομα **foros** θα είναι η εξής:

foros=ypologismoi.yp_forou(eisodima)

Το κύριο πρόγραμμα **mistos.py**

```
import  ypologismoi
sum_asf=0
sum_for=0
sum_pli=0
while True:
    am=int(input("Αριθμός Μητρώου :"))
    if am!=999:
        onoma=input("Ονοματεπώνυμο :")
        bm=float(input("Βασικός μισθός :"))
```

```
        yp=float(input("αριθμός υπερωριών :"))
        wr=ypologismoi.yp_yper(bm,yp)
         # κλήση συνάρτησης για τον υπολογισμό αποζημίωσης υπερωρίας
        yper=wr*yp # υπολογισμός αμοιβής υπερωριών
        apodoxes= bm + yper # υπολογισμός συνόλου αποδοχών
        # υπολογισμός κρατήσεων ασφάλισης εργαζόμενου
        # υπολογισμός εισφοράς ασφάλισης εργοδότη
        # σύνολο κρατήσεων και εισφοράς  ασφάλισης
        kra_asf,eis_asf,asfalisi=ypologismoi.efka(apodoxes)
        forol= apodoxes - kra_asf  # φορολογητέο ποσό
        foros=ypologismoi.yp_forou(forol) # κλήση συνάρτησης υπολογισμού φόρου
        pliroteo=forol-foros       # πληρωτέο ποσό
        sum_asf+=asfalisi          # συνολικό ποσό προς ασφαλιστικό ταμείο
        sum_for+=foros            # συνολικό ποσό προς εφορία
        sum_pli+=pliroteo # συνολικό πληρωτέο ποσό
        print(am,onoma,  end='')
        print(format(bm,",.2f"),format(yper,",.2f"),format(apodoxes,",.2f"),end='')
        print(format(kra_asf,",.2f"),format(foros,",.2f"),format(pliroteo,",.2f"))
    else:
        break
print("=======================================================")
print("Συνολικό ποσό προς Ασφαλιστικού Ταμείου :",format(sum_asf, '10,.2f'))
print("Συνολικό ποσό προς Εφορία               :",format(sum_for, '10,.2f'))
print("Συνολικό πληρωτέο ποσό                  :",format(sum_pli, '10,.2f'))
```

Λύση -3

Με χρήση **class** και στογγυροποίηση στα δύο δεκαδικά.

Δημιουργούμε ένα αρχείο τύπου py με όνομα **ypologismoi.py** το οποίο περιέχει τον παρακάτω κώδικα:

```
class Ypologismos():
    def yp_forou(ap):# συνάρτηση υπολογισμού φόρου
        poso=ap*14 # αναγωγή σε ετήσια βάση
        if poso<=10000:
            fo=0
        else:
            if poso<=25000:
                fo=0*10000+(poso-10000)*0.08
            else:
                if poso<=30000:
                    fo=0*10000+15000*0.08+(poso-25000)*0.15
                else:
                    if poso<=40000:
                        fo=0*10000+15000*0.08+5000*0.15 +(poso-30000)*0.3
                    else:
                        fo=0*10000+15000*0.08+5000*0.15 +10000*0.3+(poso-40000)*0.4
        fo= round(fo/14,2) # αναγωγή σε μηνιαία βάση & στρογγυλοποίηση σε 2 δεκαδικά
        return fo

    def yp_yper(mistos, wres):# συνάρτηση υπολογισμού αποζημίωση υπερωρίας
        wrom= round(mistos/150,2) # στρογγυλοποίηση σε 2 δεκαδικά
        if wrom >18:
            wrom=18
        return wrom
```

Δημιουργούμε ένα αρχείο τύπου **py** για το κύριο πρόγραμμα επεξεργασίας το οποίο περιέχει τον παρακάτω κώδικα:

```
import Ypologismoi as Yp  # ενσωμάτωση του module Ypologismoi.py  με ψευδώνυμο Yp


sum_asf=0
sum_for=0
sum_pli=0
while True:
    am=int(input("Αριθμός Μητρώου :"))
    if am!=999:
        onoma=input("Ονοματεπώνυμο :")
```

```
            bm=float(input("Βασικός μισθός :"))
            yp=float(input("αριθμός υπερωριών :"))
            # Από το module Yp και από την κλάση ypologismos
            #γίνεται κλήση συνάρτησης για τον υπολογισμό αποζημίωσης υπερωρίας
            wr=Yp.Ypologismos.yp_yper(bm,yp)
            yper=wr*yp # υπολογισμός αμοιβής υπερωριών
            apodoxes= bm + yper # υπολογισμός συνόλου αποδοχών
            kra_asf= 0.16 * apodoxes  # υπολογισμός κρατήσεων ασφάλισης εργαζόμενου
            kra_asf=round(kra_asf,2) # στρογγυλοποίηση σε 2 δεκαδικά
            eis_asf= 0.2806 * apodoxes # υπολογισμός εισφοράς ασφάλισης εργοδότη
            eis_asf=round(eis_asf,2)
            asfalisi=kra_asf+eis_asf    # σύνολο κρατήσεων και εισφοράς  ασφάλισης
            forol= apodoxes - kra_asf  # φορολογητέο ποσό
            # Από το module Yp και από την κλάση ypologismos
            #γίνεται κλήση συνάρτησης για τον υπολογισμό φόρου
            foros=Yp.Ypologismos.yp_forou(forol)      # κλήση συνάρτησης υπολογισμού φόρου
            pliroteo=forol-foros        # πληρωτέο ποσό
            sum_asf+=asfalisi           # συνολικό ποσό προς ασφαλιστικό ταμείο
            sum_for+=foros             # συνολικό ποσό προς εφορία
            sum_pli+=pliroteo # συνολικό πληρωτέο ποσό
            print(am,onoma,  end='')
            print(format(bm," ,.2f"),format(yper,",.2f"),format(apodoxes,",.2f"),end='')
            print(format(kra_asf,",.2f"),format(foros,",.2f"),format(pliroteo,",.2f"))
        else:
            break
print("=======================================================")
print("Συνολικό ποσό προς Ασφαλιστικού Ταμείου :",format(sum_asf, '10,.2f'))
print("Συνολικό ποσό προς Εφορία               :",format(sum_for, '10,.2f'))
print("Συνολικό πληρωτέο ποσό                  :",format(sum_pli, '10,.2f'))
```

Ασκήση-2

Μια εμπορική εταιρεία χονδρικής πώλησης διαθέτει ένα δίκτυο πωλητών για την πώληση των προϊόντων της σε πελάτες εμπόρους λιανικής πώλησης. Να γραφεί πρόγραμμα το οποίο για κάθε παραγγελία διαβάζει από την οθόνη:

- Κωδικό προϊόντος (code, ακέραιος)
- Κωδικό πωλητή (politis, ακέραιος)
- Ονοματεπώνυμο πωλητής (eponymo χαρακτήρες)
- Ποσότητα παραγγελίας (posot, πραγματικός)
- Τιμή μονάδος (timi, πραγματικός)

Στη συνέχεια υπολογίζει την αξία παραγγελίας και το ποσό της προμήθειας που αναλογεί στον πωλητή και γράφει στην οθόνη τον κωδικό του, το επώνυμο του και το ποσό προμήθειας. Το ποσό της προμήθειας υπολογίζεται κλιμακωτά με βάση τον παρακάτω πίνακα.

***Αξία παραγγελίας* (σε €)**	***Ποσοστό προμήθειας***
Έως και 500	0%
Πάνω από 500 έως και 1200	3%
Πάνω από 1200	4%

Η διαδικασία επαναλαμβάνεται μέχρι να δοθεί ως **κωδικός προϊόντος ο αριθμ**ός 0 (μηδέν). Μετά το τέλος της διαδικασία, το πρόγραμμα εμφανίζει στην οθόνη:

1. Για κάθε πωλητή το συνολικό ποσό προμήθειας που θα λάβει.

2. Το συνολικό ποσό που θα δοθεί για προμήθεια σε όλους τους πωλητές.
3. Τον επώνυμο του πωλητή με τη μεγαλύτερη συνολική αξία προμήθειας καθώς και την αντίστοιχη αξία.
4. Το ποσοστό της προμήθειας που δίνεται για το **προϊόν με κωδικό 1** έναντι της συνολικής προμήθειας (όλα τα προϊόντα)

Λύση

```
promithies={}# λεξικό με συνολικό ποσό προμήθειας για κάθε πωλητή
eponyma={} # λεξικό με τα  επώνυμα των πωλητών.

def bonus(poso): # συνάρτηση υπολογισμού προμήθειας

    if poso <= 500:
        pro=0
    else:
        if poso<=1200:
            pro=500*0+(poso-500)*0.03
        else:
            pro=700*0.03+(poso-1200)*0.04
    return pro

def erotima_1(): # συνάρτηση εκτυπωσης αποτελεσμάτων 1ου ερωτήματος
    for pol in eponyma:
        axia_prom=promithies.get(pol)
        ep=eponyma.get(pol)
        print(pol,ep,axia_prom)

def synolo_prom():# συνάρτηση υπολογισμού συνολικής προμήθειας
    sum=0
    for pol in promithies:
        sum+=promithies.get(pol)
    return sum

def kalyt_pol():# συνάρτηση εύρεσης του πωλητή με την μεγαλύτερη
                # συνολική προμήθεια
    meg=-1
    for pol in promithies:
        poso=promithies.get(pol)
        if poso>meg:
            meg=poso
            epon=eponyma.get(pol)
    return meg,epon

sum_1=0
meg=-1


while True:
    code=int(input("κωδικός προϊόντος ή 0 για τέλος "))
    if code!=0:
        politis = int(input("κωδικός πωλητή   :"))
        eponymo = input("επωνυμο πωλητή   :")
        posot = float(input("ποσότητα πώλησης :"))
        timi = float(input("τιμή  μονάδος    :"))
        axia = posot*timi
        prom = bonus(axia)

        if politis in eponyma:# αν ο πωλητής έχει ήδη εισαχθεί στα λεξικά
            s_axia=promithies.get(politis)
            s_axia+=prom
            promithies[politis]=s_axia
        else: # αν ο πωλητής είναι νέος
            promithies[politis]=prom
            eponyma[politis]=eponymo

        if code==1:
```

```
            sum_1+=prom

        print(politis,eponymo,format(prom,'.2f'))
    else:
        break
print("====================================")

erotima_1() # εκτύπωση αποτελεσμάτων  1ου ερωτήματος
syn_prom=synolo_prom()
print("Συνολική προμήθεια ", format(syn_prom,".2f"))
axia,eponymo=kalyt_pol()
print("Πωλητής με τη μεγαλύτερη συνολική προμήθεια :",eponymo, "με προμήθεια ",axia)
pososto=sum_1/syn_prom
print("Το ποσοστό της προμήθειας που δίνεται για το προϊόν με κωδικό 1 ",end ="")
print("έναντι της συνολικής προμήθειας", format(pososto,".2%"))
```

22.8 Ασκήσεις χειρισμού αρχείων

22.8.1 Άσκηση -1

Να γραφεί πρόγραμμα το οποίο διαβάζει από την οθόνη τα παρακάτω στοιχεία που αφορούν στη βαθμολογία ενός φοιτητή ελέγχοντας την βαθμολογία ότι είναι στο διάστημα [0,10]:

- Κωδικός φοιτητή
- Ονοματεπώνυμο
- Βαθμός-1
- Βαθμός-2
- Βαθμός-3

Και να τα γράφει σε ένα αρχείο με όνομα **students.txt.** Η διαδικασία να επαναλαμβάνεται μέχρις ότου να δοθεί ως κωδικός ο αριθμός 0 (μηδέν).

Λύση

```
arxeio = open("g:/arxeia/students.txt", 'w')

while True:
    code=input("δώσε κωδικό φοιτητή ")
    if int(code)!=0:
        epon=input("επώνυμο φοιτητή ")
        while True:
            ba1=float(input("βαθμός 1ος :"))
            if ba1>=0 and ba1<=10:
                break
        while True:
            ba2=float(input("βαθμός 2ος :"))
            if ba2>=0 and ba2<=10:
                break
        while True:
            ba3=float(input("βαθμός 3ος :"))
            if ba3>=0 and ba3<=10:
                break
        #  σχηματισμός λογικής εγγραφής
        # record=code+","+epon+","+ba1+","+ba2+","+ba3+"\n"  # ή καλύτερα
        record="{},{},{},{},{} \n".format(code,epon,ba1,ba2,ba3)
        arxeio.write(record) # εγγραφή στο αρχείο
    else:
        break

arxeio.close()
```

22.8.2 Άσκηση -2

Σε ένα αρχείο με όνομα **students.txt** είναι αποθηκευμένα τα στοιχεία φοιτητών με τις βαθμολογίες τους σε τρεις προόδους. Το αρχείο έχει την παρακάτω δομή

- Κωδικός φοιτητή
- Ονοματεπώνυμο
- Βαθμός-1
- Βαθμός-2
- Βαθμός-3

Να γραφεί πρόγραμμα που να διαβάζει κάθε εγγραφή του αρχείου να υπολογίζει τον μέσο όρο στην βαθμολογία και να εμφανίζει στην οθόνη

- Τον κωδικό του
- Το ονοματεπώνυμο του
- Τις τρεις βαθμολογίες του
- Τον μέσο όρο βαθμολογίας
- Το μήνυμα «Επιτυχών» αν ο μέσος όρος είναι ≥5 αλλιώς το μήνυμα «Αποτυχών»

Λύση 1

```
arxeio = open("g:/arxeia/students-1.txt", 'r')

for record in arxeio:
    pedio=record.split(',')
    kodikos=pedio[0]
    eponymo=pedio[1]
    bath_1=float(pedio[2])
    bath_2=float(pedio[3])
    bath_3=float(pedio[4])
    mo=(bath_1+bath_2+bath_3)/3
    if mo>=5:
        xaraktirismos= "Επιτυχών"
    else:
        xaraktirismos= "Αποτυχών"
    print("{} {} {} {} {}
".format(kodikos,eponymo,bath_1,bath_2,bath_3),end=" ")
    print("M.O. ",format(mo," .2f"),xaraktirismos)

arxeio.close()
```

Λύση 2

```
arxeio = open("g:/arxeia/students-1.txt", 'r')

for record in arxeio:
    pedio=record.split(',')
    kodikos=pedio[0]
    eponymo=pedio[1]
    bath_1=float(pedio[2])
    bath_2=float(pedio[3])
    bath_3=float(pedio[4])
```

```
    sum=0
    for i in range(2,5):
        sum+=float(pedio[i])
    mo=sum/3
    if mo>=5:
        xaraktirismos= "Επιτυχών"
    else:
        xaraktirismos= "Αποτυχών"
    print("{} {} {} {} {}
".format(kodikos,eponymo,bath_1,bath_2,bath_3),end=" ")
    print("M.O. ",format(mo," .2f"),xaraktirismos)

arxeio.close()
```

Λύση 3

```
arxeio = open("g:/arxeia/students-1.txt", 'r')
record=arxeio.readline() # αναγνωση της πρώτης εγγραφής για αρχικοποίηση της μεταβλητής record
while record!="": # όσο η μεταβλητή record δεν είναι κενή δηλ τέλος αρχειου
    pedio=record.split(',')
    kodikos=pedio[0]
    eponymo=pedio[1]
    bath_1=float(pedio[2])
    bath_2=float(pedio[3])
    bath_3=float(pedio[4])
    sum=0
    for i in range(2,5):
        sum+=float(pedio[i])
    mo=sum/3
    if mo>=5:
        xaraktirismos= "Επιτυχών"
    else:
        xaraktirismos= "Αποτυχών"
    print("{} {} {} {} {} ".format(kodikos,eponymo,bath_1,bath_2,bath_3),end=" ")
    print("M.O. ",format(mo," .2f"),xaraktirismos)
    record=arxeio.readline() # ανάγνωσης της επόμενης εγγραφής
arxeio.close()
```

22.9 Σύνθετες Ασκήσεις

22.9.1 Άσκηση -1

Σε ένα αρχείο με όνομα **sales.txt** είναι αποθηκευμένα τα στοιχεία πωλήσεων που πραγματοποίησαν ανά εξάμηνο σε μία επιχείρηση. Το αρχείο έχει την παρακάτω δομή:

- Κωδικός πωλητή
- Ονοματεπώνυμο πωλητή
- Πωλήσεις 1ου εξαμήνου
- Πωλήσεις 2ου εξαμήνου

Να γραφεί πρόγραμμα το οποίο:

- Διαβάζει κάθε εγγραφή του αρχείου υπολογίζει το σύνολο των πωλήσεων του έτους και την προμήθεια του πωλητή η οποία υπολογίζεται κλιμακωτά με βάση τον παρακάτω πίνακα:

***Αξία ετήσιων πωλήσεων* (σε €)**	***Ποσοστό προμήθειας***
Μέχρι 30000	2%
Πάνω από 30000 έως και 50000	5%
Πάνω από 50000 έως και 12000	6.5%
Πάνω από 120000	7.5%

- Για κάθε πωλητή γράφει στην οθόνη το κωδικό του, το ονοματεπώνυμο του την συνολική αξία πωλήσεων του και την προμήθεια που δικαιούται.

Στο τέλος της διαδικασίας εμφανίζει:

- Τη μεγαλύτερη προμήθεια που δόθηκε
- Τους πωλήτες με τη μεγαλύτερη προμήθεια που εισέπραξαν.
- Το συνολικό ποσό προμήθειας που δόθηκε στους πωλητές

Λύση

```
promithies={}# λεξικό με σύνολα πωλήσεων για κάθε πωλητή
eponyma={} # λεξικό με τα  επώνυμα των πωλητών.

def bonus(poso): # συνάρτηση υπολογισμού προμήθειας
    if poso <= 30000:
        pro=0.02*poso
    else:
        if poso<=50000:
            pro=30000*0.02+(poso-30000)*0.05
        else:
            if poso<=120000:
                pro=30000*0.02+20000*0.05+(poso-50000)*0.065
            else:
                pro=30000*0.02+20000*0.05+70000*0.065+(poso-120000)*0.075
    return pro

try:
    # γίνεται έλεγχος για την υπάρξη του αρχείου
    arxeio = open("g:/arxeia/sales.txt", 'r')

    for record in arxeio:
        pedio=record.split(',')
        kodikos=int(pedio[0])
        eponymo=pedio[1]
        sales_a=float(pedio[2])
        sales_b=float(pedio[3])
        sum=sales_a+sales_b
        prom=bonus(sum)  # κλήση της συνάρτησης υπολογισμού προμήθειας
        #εφόσον υπάρχει μόνο μία εγγραφή κάθε πωλητή στο αρχείο
        #γίνεται εισαγωγή των στοιχείων στα λεξικά
        eponyma[kodikos]=eponymo
        promithies[kodikos]=prom
        print("{} {} ".format(kodikos, eponymo), end= ' ')
        print(format(sum, ","), format(prom, ","))
    arxeio.close()
    meg_kod=0
    meg_pro=0
```

```
    print ("Συγκεντρωτικά αποτελέσματα επεξεργασίας")
    for  kod in  promithies: #  σάρωση του λεξικού με τις προμήθειες
                             # για την εύρεση της μεγαλύτερης τιμής
        prom=promithies.get(kod)
        if prom>meg_pro:
            meg_pro=prom
    print("Η μεγαλύτερη προμήθεια ήταν :",meg_pro)
    print("Κατάσταση πωλητών με την μεγαλύτερη προμήθεια ")
    for kodikos in promithies:
        prom=promithies.get(kodikos)
        if prom==meg_pro:
            ep=eponyma.get(kodikos)
            print (kodikos, ep, prom)
    arxeio.close()
except Exception as IOError:
    #αν παρουσιαστεί λάθος στο άνοιγμα του αρχείου
    print("το αρχείο αυτό δεν υπάρχει ")
```

22.9.2 Άσκηση -2

Ένας κτηνοτροφικός συνεταιρισμός διαθέτει σε ένα ψηφιακό αρχείο με όνομα **PARAGOGI.TXT** τα στοιχεία παραγωγής κρέατος ανά κτηνοτροφική μονάδα. Το αρχείο περιέχει τα εξής στοιχεία σφαγής:

- - Κωδικός μονάδας (ακέραιος)
- - Επωνυμία μονάδας (χαρακτήρες).
- - Κωδικός ζώου (5 ψήφιος ακέραιος)
- - Κωδικός είδους (μονοψήφιος ακέραιος, 1= μοσχάρι, 2=αρνί, 3=κατσίκι)
- - Ημέρα (2-ψήφιος ακέραιος)
- - Μήνας σφαγής (2-ψήφιος ακέραιος).
- - Έτος σφαγής (4-ψήφιος ακέραιος)
- - Ποσότητα κρέατος σε κιλά (πραγματικός).

Να γραφεί πρόγραμμα που διαβάζει το αρχείο και:

Υπολογίζει το ποσόν της επιδότησης που θα εισπράξει ***κάθε μονάδα για κάθε σφαγή*** που έγινε το 2017. Η επιδότηση υπολογίζεται ως ποσό ανά κιλό κρέατος και είδος ζώου, ως εξής

Είδος ζώου	**Ποσό επιδότησης ανά κιλό**
• Μοσχάρι	2.30 €
• Αρνί	1.30€
• Κατσίκι	1.20€

Για κάθε εγγραφή που αφορά σε σφαγή ***μοσχαριού*** το **2017** αποθηκεύει σε ένα αρχείο εξόδου, τον κωδικό της μονάδας, την επωνυμία, την ποσότητα κρέατος και το ποσό επιδότησης.

Τέλος το κύριο πρόγραμμα υπολογίζει και εμφανίζει στην οθόνη για το έτος **2017**:

- Για κάθε μονάδα την επωνυμία της μονάδας και το συνολικό ποσό επιδότησης που έλαβε.
- Τη συνολική αξία επιδότησης που θα λάβουν όλες οι μονάδες
- Το συνολικό βάρος κρέατος ανά είδος.

Λύση

```
epid=(2.40,1.4,1.2) # tuple με ποσοστά επιδότησης
zwo=("μοσχάρι","αρνί", "κατσίκι")    # tuple με τα είδη ζώων
posotites=[0,0,0] # λίστα που θα αποθηκευτούν οι ποσότητες ανά είδος
synola={} # λεξικό που για κάθε κωδικό μονάδας θα περιέχει τη συνολική
επιδότηση
monades={} # λεξικό που για κάθε κωδικό μονάδας θα περιέχει τη επωνυμία

try:
    # γίνεται έλεγχος για την υπάρξη του αρχείου
    arxeio =open("g:/arxeia/paragogi.txt", 'r')
    apotel=open("g:/arxeia/mosxaria.txt",'w')

    for record in arxeio:
        pedio=record.split(',')
        kodikos=int(pedio[0])
        eponymia=pedio[1]
        eidos= int(pedio[3])
        im=int(pedio[4])
        minas=int(pedio[5])
        etos=int(pedio[6])
        varos=float(pedio[7])
        print(etos)
        if etos == 2017:
            epidot=varos*epid[eidos-1]# πολλαπλασιάζεται το βάρος επί
                                      # το αντίστοιχο ποσό αναλογα με το
                                       είδος ζώου
            epidot=round(epidot,2)
            posotites[eidos-1]+=varos

            if kodikos in monades:
                poso=float(synola.get(kodikos))
                poso+=epidot
                synola[kodikos]=poso
            else:
                monades[kodikos]=eponymia
                synola[kodikos]=epidot

            if eidos==1:
                #  σχηματισμός λογικής εγγραφής
                record="{},{},{},{} \n".format(kodikos,eponymia,varos,epidot)
                apotel.write(record) # εγγραφή στο αρχείο

    arxeio.close()
    apotel.close()
    print("Συγκεντρωτικά αποτελέσματα ")
    i=0
    sum=0
    for kodikos in monades:
        i+=1
        poso=round(synola.get(kodikos),2)
        sum+=poso
        print(i,kodikos, monades[kodikos],poso)
    print("Το συνολικό ποσό που θα λάβουν όλες οι μονάδες είναι ",sum)
    print("Συνολικές ποσότητες ανά είδος ")
    for j in range(3):
        print(j+1,zwo[j],posotites[j]," kgr")
except Exception as IOError:
    #αν παρουσιαστεί λάθος στο άνοιγμα του αρχείου
    print("το αρχείο αυτό δεν υπάρχει ")
```

22.9.3 Άσκηση -3

Σε ένα ψηφιακό αρχείο με όνομα ΜΕΤΕΟ.ΤΧΤ είναι αποθηκευμένες σε ημερήσια βάση, η διάρκεια της ηλιοφάνειας στην περιοχή του σταθμού καθώς και η θερμοκρασίες ανά εξάωρο. Το αρχείο περιέχει μια εγγραφή για κάθε μέρα με την εξής δομή:

- κωδικός μετεωρολογικού σταθμού (ακέραιος)
- ημέρα (ακέραιος)
- μήνας (ακέραιος)
- έτος (ακέραιος)
- ηλιοφάνεια σε λεπτά (ακέραιος)
- θερμοκρασία στις 0.0 (πραγματικός)
- θερμοκρασία στις 6.0 (πραγματικός)
- θερμοκρασία στις 12.0 (πραγματικός)
- θερμοκρασία στις 18.0 (πραγματικός)

Να γραφεί πρόγραμμα το οποίο να διαβάζει κάθε εγγραφή του αρχείου και για κάθε εγγραφή που αφορά στον Μάρτιο 2018

- Υπολογίζει την και μέση θερμοκρασία της ημέρας.
- Εμφανίζει στην οθόνη τον κωδικό του σταθμού και τη μέση θερμοκρασία της ημέρας.

Στο τέλος της διαδικασίας εμφανίζει στην οθόνη:

- Μέση ηλιοφάνεια του Μαρτίου 2018 σε ώρες.
- Τη μέση θερμοκρασία για κάθε μια από τις ωρες 0:00, 06:00,12:00,18:00 τον Μάρτιο του 2018.
- Τη μέση θερμοκρασία του μηνός Μαρτίου 2018.

Λύση

```
thermo=[0,0,0,0] # θερμοκρασία ημέρας στις 0.00, 6.00, 12.00, 18.00
ther minos=[]
plithos=[]
def create_lists():
    for i in range(32):
        ther_minos.append(0)
        plithos.append(0)
try:
    # γίνεται έλεγχος για την υπάρξη του αρχείου στο συγκεκριμένο δίσκο και
folder
    arxeio =open("g:/arxeia/meteo.txt", 'r')
    create_lists()
    pl=0
    sum_wres=0
    for record in arxeio:
        pedio=record.split(',')
        stathmos=int(pedio[0])
        im=int(pedio[1])
        minas=int(pedio[2])
        etos=int(pedio[3])
        iliof=float(pedio[4])
        if etos==2018 and minas==3:
            sum=0
```

```
            sum_wres+=iliof/60
            pl+=1
            for i in range (4):
                temp=float(pedio[5+i])
                sum+=temp
                thermo[i]+=temp
            mesi=round(sum/4,2)
            ther_minos[im-1]+=mesi
            plithos[im-1]+=1
            print("Σταθμός ",stathmos,"μέση θερμοκρασία της ",
im,"/",minas,"/",etos , " :",mesi)
    arxeio.close()
    print("Συγκεντρωτικά αποτελέσματα ")
    mesi_iliof=round(sum_wres/pl)
    sum=0
    pl=0
    for j in range(31):
        if plithos[j]>0:
            mesi=ther_minos[j]/plithos[j]
            mesi=round(mesi,2)
            print(" {}η Μαρτίου 2018 μέση θερμοκρασία: {}".format( j+1,mesi))
            sum+=mesi
            pl+=1

    mesi martiou=round(sum / pl, 2)

    print("μέση θερμοκρασία του Μαρτίου 2018 :", mesi_martiou)
    print("μέση ηλιοφάνεια του Μαρτίου 2018 ",mesi_iliof, "ώρες")

except Exception as IOError:
    #αν παρουσιαστεί λάθος στο άνοιγμα του αρχείου
    print("το αρχείο αυτό δεν υπάρχει ")
```

22.9.4 Άσκηση -4

Ένας γεωργικός συνεταιρισμός διαθέτει αρχείο, με όνομα **OLIVES.TXT,** με στοιχεία παραγωγής των ελαιοπαραγωγών του, για διάφορες χρονολογίες. Συγκεκριμένα, το αρχείο περιέχει τα εξής στοιχεία:

- - Κωδικός αγρότη (ακέραιος).
- - Ονοματεπώνυμο παραγωγού (Χαρακτήρες).
- - ΑΦΜ παραγωγού (Χαρακτήρες).
- - Έτος παραγωγής (ακέραιος).
- - Ποσότητα παραγωγής λαδιού σε λιτρα (πραγματικός).
- - Ποσότητα παραγωγής βρώσιμης ελιάς σε κιλά (πραγματικός).

Να γραφεί κύριο πρόγραμμα που διαβάζει το αρχείο και για **κάθε παραγωγό που είχε παραγωγή το 2018**:

Υπολογίζει το συνολικό ποσόν που κατέβαλε ο συνεταιρισμός στον παραγωγό για την αγορά της παραγωγής του. Για το υπολογισμό της παραγωγής λαμβάνεται υπόψη ότι οι τιμή αγοράς της του λαδιού είναι **4,2€ ανά lt** και της βρώσιμη ελιάς **2,5€ ανά kgr ενώ υπάρχει επιδότηση για το λάδι ως εξής:**

- Έως και 1000 lt επιδότηση 1.6€/lt
- Πάνω από 10 έως και 5000 lt επιδότηση 1.2€/ lt

- Πάνω από 5000 lt και άνω δεν υπάρχει επιδότηση

Εμφανίζει στην οθόνη τα στοιχεία ταυτότητος του παραγωγού και το συνολικό ποσόν που του κατέβαλε ο συνεταιρισμός .

Στο τέλος της επεξεργασίας και εμφανίζει στην οθόνη για το έτος 2018:

- Το συνολικό ποσό που αφορά στην αξία της βρώσιμης ελιάς,
- Το συνολικό ποσό που αφορά στην αξία στην αξία του λαδιού,
- Το συνολικό ποσό που αφορά στην αξία επιδότησης του λαδιού.

Λύση

```
def axia(oil,olives): # συνάρτηση υπολογισμού αξίας ελαιολάδου και ελιάς
    poso_la=oil*4.5
    poso_el=olives*2.5
    return poso_la,poso_el

def epidot(litra): # συνάρτηση αξίας επιδότησης ελαιολάδου
    if litra<=1000:
        poso=litra*1.6
    else:
        if litra<=5000:
            poso=1000*1.6+(litra-1000)*1.2
        else:
            poso=1000*1.6+4000*1.2
    return  poso



arxeio=open('g:/arxeia/olives.txt','r')
s_axia_ladi=0
s_axia_elia=0
s_epidot=0
aa=0
for eggrafi in arxeio:
    pedia=eggrafi.split(',')
    kodikos=int(pedia[0])
    eponymia=pedia[1]
    afm=pedia[2]
    etos=int(pedia[3])
    if etos==2015:
        aa+=1
        ladi=float(pedia[4])
        elies=float(pedia[5])
        ax_ladi,ax_elia=axia(ladi,elies)
        epidotisi=epidot(ladi)
        syn_poso=ax_ladi+ax_elia+epidotisi
        print(" {:2} {:4} {} {} {:5,.1f} ".format(aa,kodikos,eponymia,afm,syn_poso))
        s_axia_elia+=ax_elia
        s_axia_ladi+=ax_ladi
        s_epidot+=epidotisi

print("=================================================================")
print("Σύνολα")
print("Το συνολικό αξία της βρώσιμης ελιάς     {:10,.2f}". format(s_axia_elia ))
print("Το συνολικό αξία του ελαιολάδου         {:10,.2f}". format(s_axia_ladi ))
print("Το συνολικό αξία επιδοότητση ελαιολάδου {:10,.2f}". format(s_epidot))

arxeio.close()
```