Προγραμματισμός Η/Υ

Περιεχόμενα

Τμήμα: Μηχανικών Χωροταξίας, Πολεοδομίας και Περιφερειακής Ανάπτυξης

Τίτλος Επιστημονικού Πεδίου: Πληροφορική και Ανάλυση Δεδομένων

Κωδικός Μαθήματος: ΜΕ0200

Τίτλος Μαθήματος: Προγραμματισμός Η/Υ

Κατηγορία Μαθήματος: Επιλογής

Εξάμηνο: Εαρινό

Περίγραμμα του μαθήματος

Το μάθημα εισάγει τους φοιτητές στις βασικές έννοιες και αρχές του προγραμματισμού και είναι προσαρμοσμένο στις ανάγκες και στο υπόβαθρο των χωροτακτών. Στόχος του μαθήματος είναι να αποκτήσουν οι φοιτητές τις αναγκαίες γνώσεις για την επεξεργασία και ανάλυση δεδομένων, την αυτοματοποίηση διαδικασιών και την σύνταξη σεναρίων (scripts) για την αναπαραγωγισιμότητα της ερευνάς τους. Πέρα από τις βασικές αρχές προγραμματισμού η διδασκαλία επεκτείνεται σε εξειδικευμένες θεματικές ενότητες που αφορούν τα γεωχωρικά δεδομένα και την ανάλυσή τους μέσω προγραμματισμού. Η διδασκαλία θα στηριχθεί στην Python, μια σύγχρονη, ευρέως διαδεδομένη και υψηλού επιπέδου γλώσσα προγραμματισμού. Επιπλέον, όπου κριθεί αναγκαίο, θα επιδειχτούν συμπληρωματικά διαδικασίες με την γλώσσα προγραμματισμού R.

Το πρόγραμμα των διαλέξεων καθώς και η προτεινόμενη βιβλιογραφία παρατίθενται στην συνέχεια.

1. Εισαγωγή στον προγραμματισμό

Ενότητες του μαθήματος

Το μάθημα χωρίζεται στις ακόλουθες ενότητες:

1. Εισαγωγή στον προγραμματισμό

Κατά την διάρκεια της διάλεξης διευκρινίζεται ο σκοπός του μαθήματος και περιγράφονται συνοπτικά οι ενότητες που θα διδαχθούν οι φοιτητές κατά την διάρκεια του εξαμήνου. Διατυπώνονται συγκεκριμένοι ορισμοί που αφορούν τον προγραμματισμό Η/Υ και αναπτύσσονται έννοιες για την επιστήμη των υπολογιστών. Στην συνέχεια εγκαθίσταται στους υπολογιστές των φοιτητών η γλώσσα προγραμματισμού Python μαζί με το απαραίτητο λογισμικό για την συγγραφή και αποσφαλμάτωση του κώδικα. Ακολουθεί εξοικείωση με το περιβάλλον εργασίας.

2. Τιμές, τύποι και μεταβλητές

Περιγραφή της έννοιας των μεταβλητών, των σταθερών, τύποι δεδομένων, εκχώρηση τιμών στις μεταβλητές, κανόνες ονοματοδοσίας των μεταβλητών.

3. Εκφράσεις, τελεστές

Ορισμός εκφράσεων, τι είναι τελεστές, ποια είναι η προτεραιότητα των τελεστών, πως εισάγουμε σχόλια στον κώδικα και γιατί είναι σημαντική πρακτική.

4. Έλεγχος ροής εκτέλεσης

Η λογική Boolean, Εκτέλεση υπό συνθήκη, αλυσιδωτές και εμφωλευμένες συνθήκες, βρόχος και οι εντολές επανάληψης for και while.

5. Συναρτήσεις

Ορισμός και κλήση συνάρτησης, παράμετροι συναρτήσεων, εμβέλεια μεταβλητών, αναδρομή.

6. Συμβολοσειρές/Δομές Δεδομένων

Προσπέλαση συμβολοσειρών, χαρακτήρες διαφυγής, υποσύνολα συμβολοσειράς, συγκρίσεις και ιδιότητες, μέθοδοι συμβολοσειρών. Λίστες, Πλειάδες, Λεξικά.

7. Ανάγνωση & εγγραφή αρχείων, φάκελοι

Ανάγνωση και εγγραφή σε αρχείο, σειριοποίηση (serialization) αντικειμένου, διαχείριση φακέλων και αρχείων.

8. Πίνακες και διαγράμματα

Ανάγνωση αρχείων csv ή excel, pandas dataframes

9. Πίνακες και διαγράμματα

Πίνακες στην βιβλιοθήκη numpy, διαγράμματα με την βιβλιοθήκη seaborn.

10. Γεωεπεξεργασία διανυσματικών δεδομένων

Ανάγνωση και εγγραφή διανυσματικών δεδομένων, μετα-δεδομένα, φιλτράρισμα, αλλαγή προβολικού συστήματος.

11. Ανάλυση διανυσματικών δεδομένων

Χωρικές σχέσεις, στατιστικά ομαδοποιήσεων, οπτικοποίηση διανυσματικών δεδομένων.

12. Γεωεπεξεργασία ψηφιδωτών δεδομένων

Ανάγνωση και εγγραφή διανυσματικών δεδομένων, μετα-δεδομένα, ορισμός μάσκας/αποκοπή περιοχής, αλλαγή τιμών, επαναταξινόμηση, αλλαγή προβολικού συστήματος.

13. Ανάλυση ψηφιδωτών δεδομένων

Άλγεβρα ψηφιδωτών αρχείων, στατιστικά ζωνών, ιστόγραμμα συχνοτήτων.

Ορισμοί

> **Definition 1**
>
> «*Αλγόριθμο*» ονομάζουμε κάθε πεπερασμένη και αυστηρά καθορισμένη σειρά βημάτων (οδηγιών) για την επίλυση ενός προβλήματος. [Αγγελιδάκης, 2015]
>
> Ένας αλγόριθμος είναι μια αυστηρά καθορισμένη διαδικασία που λαμβάνει μια τιμή ή ένα σύνολο τιμών εισόδου και αποδίδει μια ή περισσότερες τιμές εξόδου. Είναι κατά συνέπεια μια ακολουθία υπολογιστικών βημάτων που μετατρέπει την είσοδο δεδομένων σε έξοδο αποτελεσμάτων [Cormen, 2009].

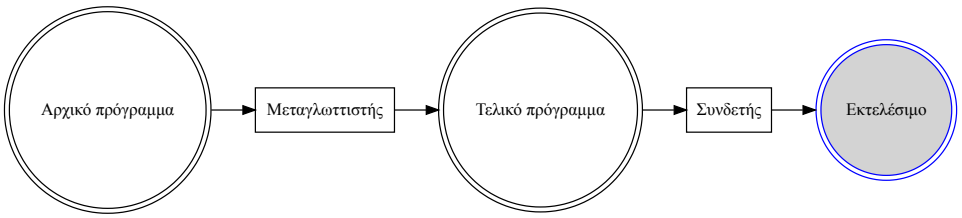
Για παράδειγμα η αύξουσα (ή φθίνουσα) ταξινόμηση μιας λίστας αριθμών είναι ένα χαρακτηριστικό παράδειγμα αλγορίθμου. Οπότε με τιμές εισόδου {31, 41, 59, 26, 41, 58}, ο αλγόριθμος ταξινόμησης επιστρέφει ως τιμές εξόδου {26, 31, 41, 41, 58, 59}.

> **Definition 2**
>
> Ως «*Πρόγραμμα*» ορίζεται ένας αλγόριθμος γραμμένο σε γλώσσα κατανοητή για τον υπολογιστή και περιέχει εντολές (οδηγίες) που κατευθύνουν με κάθε λεπτομέρεια τον υπολογιστή, για να εκτελέσει μια συγκεκριμένη εργασία και να επιλύσει ένα πρόβλημα [Αγγελιδάκης, 2015]. Ένα Πρόγραμμα αναγνώσιμο από τον άνθρωπο ονομάζεται «*πηγαίος κώδικας*».

> **Definition 3**
>
> «*Προγραμματισμός*» ονομάζεται η διαδικασία συγγραφής προγραμμάτων και περιλαμβάνει τη διατύπωση των κατάλληλων εντολών προς τον υπολογιστή με τη χρήση τεχνητών γλωσσών, των γλωσσών προγραμματισμού [Αγγελιδάκης, 2015].

Ο προγραμματισμός είναι μια διαδικασία που απαιτεί μια σειρά εργαλείων και διαδικασιών τα οποία συνήθως ενσωματώνονται όλα μαζί σε ένα ενοποιημένο περιβάλλον που αποκαλείται «ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών» (Integrated Development Environment, IDE).

Η διαδικασία μετατροπής του πηγαίου κώδικα σε εκτελέσιμο αρχείο περιγράφεται στο παρακάτω διάγραμμα:



Εικ. 1 Η ροή εκτέλεσης του κώδικα σε εκτελέσιμο.

Τα εργαλεία προγραμματισμού τα οποία κάνουν την μεταγλωττιστή του πηγαίου προγράμματος σε εκτελέσιμο πρόγραμμα είναι τα εξής:

- ο *επεξεργαστής κειμένου* με την βοήθεια οποίου συντάσσεται ο πηγαίο κώδικάς του προγράμματος.
- ο *μεταγλωττιστής* ή *διερμηνευτής* οι οποίοι χρησιμοποιούνται για την μετατροπή του πηγαίου κώδικα σε γλώσσα μηχανής η οποία είναι απαραίτητη για την αναγνώριση και εκτέλεση των εντολών από τον Η/Υ. Τα παραγόμενο πρόγραμμα από την μεταγλώττιση ονομάζεται *αντικείμενο πρόγραμμα* (object). Ο διερμηνευτής διαβάζει διαδοχικά τις εντολές και για κάθε εντολή που διαβάζει, εκτελεί αμέσως μια ισοδύναμη ακολουθία εντολών μηχανής. Από την άλλη, ο μταγλωττιστής δέχεται στην είσοδο ένα πρόγραμμα γραμμένο σε γλώσσα υψηλού επιπέδου (πηγαίο κώδικα) και παράγει ισοδύναμο πρόγραμμα σε γλώσσα μηχανής (αντικείμενο).
- ο *συνδέτης - φορτωτής* (linker- loader) ο οποίος συνδέει το *αντικείμενο πρόγραμμα* με άλλα τμήματα του προγράμματος ή απαραίτητες βιβλιοθήκες που διατίθεται από την γλώσσα προγραμματισμού. Το τελικό πρόγραμμα που προκύπτει από την μεταγλώτισση και την σύνδεση των τμημάτων του προγράμματος είναι το *εκτελέσιμο πρόγραμμα* (executable) το οποίο μπορεί να διαβάσει και να εκτελέσει ο υπολογιστής.
- τα *εργαλεία αποσφαλμάτωσης* με την βοήθεια των οποίων δοκιμάζεται η εκτέλεση και η ορθότητα του πηγαίου κώδικα και εντοπίζονται λάθη σε αυτόν.

Τα λάθη στον κώδικα συνοψίζονται σε τρείς βασικές κατηγορίες:

1. *σφάλμα μεταγλώττισης* τα οποία προκύπτουν κατά την λανθασμένη συγγραφή του πηγαίου κώδικα. Ο μεταγλωττιστής δεν επιτρέπει την μετάφραση του πηγαίου κώδικα σε γλώσσα μηχανής αν προηγουμένος δεν έχει διορθωθεί το συντακτικό λάθος. Συντακτικά λάθη συμβαίνουν συνήθως όταν δεν ακολοθούνται οι κανόνες σύνταξης μια γλώσσας (π.χ. μια παρένθεση που δεν έχει κλείσει, ένα ξεχασμένο εισαγωγικό ή κόμμα κτλ.).

Το παρακάτω είναι ένα παράδειγμα συντακτικού σφάλματος και το μήνυμα που επιστρέφει ο μεταγλωττιστής της Python. Η αιτία του σφάλματος είναι η ξεχασμένη παρένθεση στην συνάρτηση (function) *print*

```
print("an example"
```

```
  Input In [2]
    print("an example"
          ^
SyntaxError: '(' was never closed
```

1. *σφάλμα εκτέλεσης* (run-time errors) τα οποία συμβαίνουν κατά την εκτέλεση του προγράμματος παρότι δεν υπάρχουν σφάλματα σύνταξης. Χαρακτηριστικά παραδείγματα τέτοιων λαθών είναι η διαίρεση με το μηδέν, η πρόσβαση σε ένα στοιχείο μιας λίστας εκτός του εύρους της, η ανάγνωση ενός αρχείου το οποίο δεν υπάρχει, ή η πρόσβαση σε ένα ανύπαρκτο object. Τα σφάλματα εκτέλεσης έχουν επικρατήσει να αναφέρονται και ως «bugs» [1]. Παρακάτω δίνεται ένα σφάλμα που προκύπτει από την διαίρεση ενός ακέραιου με το μηδέν.

```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 1/0

ZeroDivisionError: division by zero
```

1. *σφάλμα λογικής*, κατά το οποίο το πρόγραμμα εκτελείται κανονικά χωρίς σφάλματα αλλά δεν συμπεριφέρεται όπως έχει σχεδιαστεί να συμπεριφέρεται. Αυτά τα σφάλματα δεν σταματούν την εκτέλεση του προγράμματος αλλά το αποτέλεσμα της εκτέλεσης δεν ειναι το αναμενόμενο.

```
x = 6
y = 4

z = x+y/2
print('Ο μέσος όρος των δύο αριθμών είναι:',z)
```

```
Ο μέσος όρος των δύο αριθμών είναι: 8.0
```

Το παραπάνω είναι σφάλμα λογικής γιατί έπρεπε να γραφτεί ως εξής (δώστε προσοχή στις παρενθέσεις που δίνουν προτεραιότητα στις πράξεις):

```
x = 6
y = 4

z = (x+y)/2
print('Ο μέσος όρος των δύο αριθμών είναι:',z)
```

```
Ο μέσος όρος των δύο αριθμών είναι: 5.0
```

Όλες οι παραπάνω μορφές σφαλμάτων εντοπίζονται μέσω της *αποσφαλμάτωσης*, της συστηματικής δηλαδή διαδικασίας εντοπισμού και επιδιόρθωσης σφαλμάτων. Η αποσφαλμάτωση συνοψίζεται στα εξής βήματα:

- Επανάληψη του προβλήματος
- Απομόνωση του σημείου που εμφανίζεται το σφάλμα
- Αναγνώριση της αιτίας που το προκαλεί
- Διόρθωση του σφάλματος
- Επιβεβαίωση της διόρθωσης

Οι εντολές των προγραμμάτων γράφονται από τους προγραμματιστές σε τεχνητές γλώσσες που ονομάζονται «*γλώσσες προγραμματισμού*». Μια γλώσσα προγραμματισμού θα πρέπει να έχει αυστηρά ορισμένη σύνταξη και σημασιολογία. Η σύνταξη καθορίζει αν μια σειρά από σύμβολα αποτελούν «νόμιμες» εντολές ενός προγράμματος γραμμένου σε μια συγκεκριμένη γλώσσα προγραμματισμού και η σημασιολογία καθορίζει τη σημασία του προγράμματος, δηλαδή τις υπολογιστικές διαδικασίες που υλοποιεί. [Αγγελιδάκης, 2015].

Η γλώσσα προγραμματισμού Python

Η Python είναι μια ευρέως διαδομένη, αντικειμενοστραφής, υψηλού επιπέδου γλώσσα προγραμματισμού γενικής χρήσης. Η Python είναι μια γλώσσα που εκτελεί τις εντολές στον διερμηνέα, που όπως αναφέρθηκε,διαβάζει τον πηγαίο κώδικα γραμμή προς γραμμή και το μετατρέπει σε γλώσσα μηχανής. Αυτός ο τρόπος λειτουργίας της Python την καθιστά πιο αργή σε σύγκριση με άλλες γλώσσες μεταγλωττιστού όπως η C. Η Python είναι διαδραστική υπό την έννοια ότι ο χρήστης εκτελεί εντολές μέσω της γραμμή εντολών της Python, εκτελείται άμεσα και λαμβάνει το αποτέλεσμα εξόδου.

Δημιουργήθηκε από τον Guido van Rossum και πρωτοκυκλοφόρησε στις 20 Φεβρουαρίου του 1991. Το όνομά της, αν και παρεπέμπει, δεν έχει σχέση με το φίδι Πύθωνα αλλά προέρχεται από την γνωστή κωμικη σειρά του BBC, Monty Python's Flying Circus. Αν και αρχικά αναπτύχθηκε σαν μεμονωμένη ατομική προσπάθεια στην συνέχεια υποστηρίχθηκε από μια παγκόσμια κοινότητα προγραμματιστών και χρηστών. Στις 6 Μαρτίου 2001 ιδρύθηκε το αμερικάνικο μη κερδοσκοπικό ίδρυμα *Python Software Foundation (PSF)*, το οποίο στόχο έχει την διάδοση και υποστήριξη της Python μέσω της

διοργάνωσης συνεδρίων, την ανάπτυξη κοινοτήτων χρηστών, την υποστήριξη προσπαθειών μέσω υποτροφιών και την διασφάλιση οικονομικών πόρων για την ανάπτυξη της γλώσσας. Το ίδρυμα κατέχει τα πνευματικά δικαιώματα της γλώσσας και διασφαλίζει ότι αυτή θα διατίθεται με όρους ελεύθερου λογισμικού προς το ευρύτερο κοινό.

Οι βασικοί στόχοι που έθεσε ο δημιουργός κατά την ανάπτυξή της είναι να είναι εύκολη και κατανοητή με ισχυρές δυνατότητες εφάμιλλες των ανταγωνιστικών γλωσσών. Ταυτόχρονα έθεσε το όρο να είναι ανοιχτού κώδικα (open source) για να μπορεί εύκολα να αναπτύσσεται απο τους ενδιαφερόμενους προγραμματιστες και να έχει πρακτική αξία σε καθημερινές εργασίες ρουτίνας. Την τρέχουσα περίοδο (03/2022) κατατάσσεται ως η κορυφαία γλώσσα προγραμματισμού σύμφωνα με την κοινότητα προγραμματιστών TIOBE αλλά και τον δείκτη PopularitY of Programming Language Index (PYPL).

Mar 2022	Mar 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	14.26%	+3.95%
2	1	↓	C	13.06%	-2.27%
3	2	↓	Java	11.19%	+0.74%
4	4		C++	8.66%	+2.14%
5	5		C#	5.92%	+0.95%
6	6		Visual Basic	5.77%	+0.91%
7	7		JavaScript	2.09%	-0.03%
8	8		PHP	1.92%	-0.15%
9	9		Assembly language	1.90%	-0.07%
10	10		SQL	1.85%	-0.02%
11	13	↑	R	1.37%	+0.12%
12	14	↑	Delphi/Object Pascal	1.12%	-0.07%
13	11	↓	Go	0.98%	-0.33%
14	19	⇈	Swift	0.90%	-0.05%
15	18	↑	MATLAB	0.80%	-0.23%
16	16		Ruby	0.66%	-0.52%
17	12	⇊	Classic Visual Basic	0.60%	-0.66%
18	20	↑	Objective-C	0.59%	-0.31%
19	17	↓	Perl	0.57%	-0.58%
20	38	⇈	Lua	0.56%	+0.23%

Εικ. 2 Η κατάταξη σύμφωνα με την κοινότητα TIOBE (Μάρτιος 2022)

Η Python πλέον είναι μια ώριμη γλώσσα προγραμματισμού με εφαρμογές στην ανάπτυξη διαδικτυακών εφαρμογών και υπηρεσιών, την εκπαίδευση, την ανάλυση δεδομένων, την τηλεπισκόπηση και τα ΣΓΠ, την δημιουργία γραφικών, την διαχείριση συστημάτων, τα παιχνίδια, το εμπόριο και την επιχειρηματικότητα, τους μικροελεγκτές και το Internet of Things (IOT).

Η φιλοσοφία της Python ως προς την μεθοδολογία ανάπτυξης και προγραμματισμού συνοψίζεται σε 20 αρχές, οι οποίες εκτυπώνονται μέσω της γλώσσας με την παρακάτω εντολή:

```
import this
```

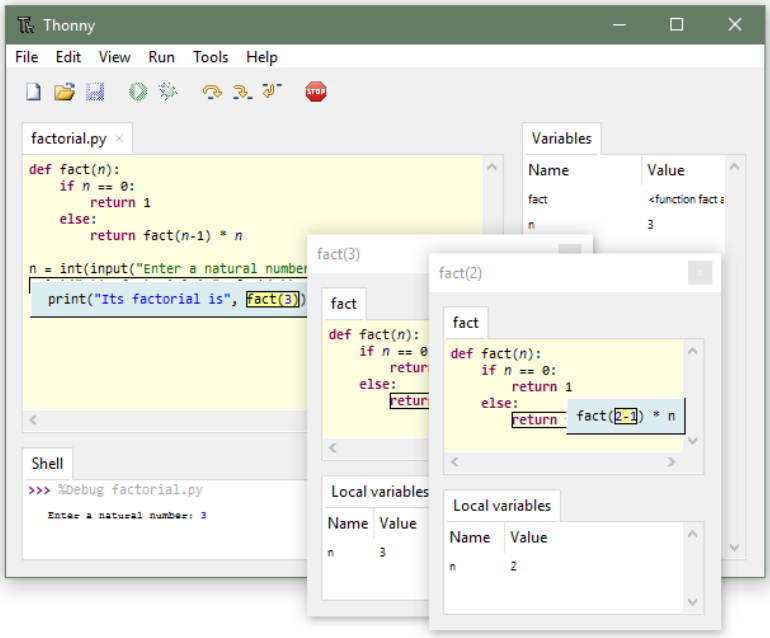
```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Το ολοκληρωμένο περιβάλλον ανάπτυξης thonny

Η συγγραφή κώδικα θα γίνει στο ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment, IDE) thonny. Δεν απαιτείται η προεγκατάσταση της Python καθώς το thonny έρχεται με ενσωματωμένη την γλώσσα προγραμματισμού Python 3.7 και διατίθεται για Windows, Mac και Linux. Το thonny αποτελεί εκπαιδευτικό περιβάλλον για την συγγραφή και αποσφαλμάτωση κώδικα Python. Για τον λόγο αυτό διαθέτει πολύ συγκεκριμένες αλλά ζωτικές λειτουργίες και δεν είναι επιφορτισμένο με δυνατότητες που απαιτούνται από προχωρημένους προγραμματιστές. Το Γραφικό Περιβάλλον Χρήστη (GUI, Graphical User Interface) είναι λιτό ώστε να μην αποπροσανατολίζει τον αρχάριο χρήστη. Το thonny παρέχει βοηθητικές λειτουργίες για τον χρήστη όπως είναι η σήμανση συντακτικών λαθών, η αυτόματη συμπλήρωση κώδικα και η ευκολία στην επέκταση των λειτουργίων της Python με την εγκατάσταση συμπληρωματικών πακέτων.

Ενναλλακτικά, στους χρήστες παρέχεται online περιβάλλον ανάπτυξης που βασίζεται στο Jupyter Lab. Η χρήση του δεν απαιτεί την εγκατάσταση λογισμικού παρά μόνον έναν απλό φυλλομετρητή (προτείνεται Chrome, Safari ή Firefox). Το online περιβάλλον Jupyter είναι προσβάσιμο από εδώ: https://kokkytos.github.io/programming



Εικ. 3 Το περιβάλλον εργασία thonny.

Εκτέλεση εντολών στο περιβάλλον thonny

Στην παρακάτω ενότητα παρουσιάζονται μερικά εισαγωγικά παραδείγματα από εντολές της Python. Δεν θα επικεντρωθούμε σε λεπτομέρειες ούτε θα αναλύσουμε τις εντολές που διατυπώνονται στα αρχεία. Παρουσιάζονται σαν μια μορφή συνοπτικής επίδειξης των δυνατοτήτων που διαθέτει η γλώσσα και θα περιγράψουμε σε επόμενα μαθήματα.

Δοκιμάστε να τρέξετε την παρακάτω εντολή στην γραμμή εντολών της Python στο thonny:

```
25+30
```

```
55
```

Τι παρατηρείτε; Η Python λειτούργησε σαν μια απλή αριθμομηχανή.

Στην συνέχεια δοκιμάστε να τρέξετε την παρακάτω εντολή γραμμή προς γραμμή:

```
number1 = 25
number2 = 30
number3 = number1+number2
number3
```

```
55
```

Το αποτέλεσμα είναι το ίδιο με το προηγούμενο.

Δώστε την παρακάτω εντολή. Αντικαταστήστε την συμβολοσειρά *AnyName* με το ονομά σας.

```
name="AnyName"

for i in range(10):
    print("Εκτύπωση", i,":",  name)
```

```
Εκτύπωση 0 : AnyName
Εκτύπωση 1 : AnyName
Εκτύπωση 2 : AnyName
Εκτύπωση 3 : AnyName
Εκτύπωση 4 : AnyName
Εκτύπωση 5 : AnyName
Εκτύπωση 6 : AnyName
Εκτύπωση 7 : AnyName
Εκτύπωση 8 : AnyName
Εκτύπωση 9 : AnyName
```

Όπως βλέπετε η Python επανέλαβε την εκτύπωση του ονόματός σας 10 φορές. Από που ξεκινά όμως η αρίθμηση της πρώτης εκτύπωσης;

Στο επόμενο παράδειγμα η Python θα σας ενημερώσει αν τρέχετε γρήγορα ή αργά ή αν είστε ακίνητος:

```
speed=70

if speed>50:
    if speed>=100:
        print("Τρέχεις πολύ γρήγορα")
    else:
        print("Τρέχεις γρήγορα")
else:
    if speed==0:
        print("Είσαι ακίνητος".upper())
    if speed>0:
        print("Τρέχεις αργά")
```

```
Τρέχεις γρήγορα
```

Έστω ότι κινείσθε σε αυτοκινητόδρομο με 120 km/h. Αν ορίσετε την ταχύτητα (speed) στον κώδικα, τι θα σας απαντήσει η Python; Αν κινείστε με μηδενική ταχύτητα (speed=0) τι μήνυμα θα λάβετε; Υπάρχουν περιπτώσεις που η Python, και δικαιολογημένα, αγνοεί να απαντήσει με μήνυμα στην ταχύτητα που ορίζεται. Μπορείτε να εντοπίσετε σε ποιές περιπτώσεις;

---

[1] Η πρώτη περίπτωση *bug* σε υπολογιστή καταγράφεται το 1947 από τον Grace Murray Hopper και πρόκειται για την κυριολεκτική έννοια του όρου. Στο ημερολόγιό του καταγράφει προβλήματα στην λειτουργία του υπολογιστή του Harvard, Mark II, από την ύπαρξη ενός εντόμου στο εσωτερικό του κύκλωμα.

2. Τιμές, τύποι και μεταβλητές. Συμβολοσειρές

Σταθερές (Constants)

Η Python δεν διαθέτει προκαθορισμένες *σταθερές* όπως άλλες γλώσσες προγραμματισμού. Όμως κατά σύμβαση και όχι κατά κανόνα έχει συμφωνηθεί οι *σταθερές* να ονοματίζονται με κεφαλαίους χαρακτήρες. Η αδυναμία της Python στην περίπτωση της δήλωσης *σταθερών* είναι ότι επιτρέπεται η αλλαγή των τιμών τους Παρακάτω παρατίθεται ένα παράδειγμα δήλωσης *σταθερών*.

```
RATIO_FEET_TO_METERS = 3.281
RATIO_LB_TO_KG = 2.205
PI = 3.14
```

Κυριολεκτικές σταθερές (literal constants)

Η κυριολεκτική *σταθερά* ή τιμή είναι ένας αριθμός, ή χαρακτήρας ή μιά συμβολοσειρά. Για παράδειγμα τα παρακάτω αποτελούν τιμές: *3.25* (στην python η υποδιαστολή ορίζεται με . και όχι ,), *«ένα τυχαίο κείμενο»*, *5.25e-1*. Αυτές οι τιμές δεν μεταβάλλονται κατά τη διάρκεια εκτέλεσης του προγράμματος γι" αυτό και λέγονται σταθερές. Μπορούν να εκχωρηθούν σε μεταβλητές και να χρησιμοποιηθούν σαν τελεστέοι σε λογικές εκφράσεις ή σαν παραμέτροι σε συναρτήσεις.

Τύποι δεδομένων

Οι τιμές ανήκουν σε τρεις τύπους δεδομένων (data types) ή κλάσσεις (class):

- τους ακέραιους αριθμούς (integer) π.χ. το 15
- τους αριθμούς κινητής υποδιαστολής (floating point) π.χ. το 201.25)
- τις συμβολοσειρές (string) π.χ. το «Time is money»

Με την εντολή `type` ο διερμηνευτής μας απαντάει με τον τύπο της τιμής, όπως παρακάτω:

```
type("No news, good news.")
```

```
str
```

Η Python είναι *Dynamic typing* δηλαδή δεν ο τύπος των μεταβλητών δεν προκαθορίζεται κατά την συγγραφή αλλά κατά την εκτέλεση.

Κανόνες ονοματοδοσίας μεταβλητών

Τα ονόματα των μεταβλητών στην Python υπακούουν στους παρακάτω κανόνες:

- Το όνομα μίας μεταβλητής μπορεί να ξεκινά από ένα γράμμα ή από κάτω πάυλα.
- Το όνομα μίας μεταβλητής δεν μπορεί με αριθμό.
- Το όνομα μίας μεταβλητής μπορεί να περιέχει μόνο αλφαριθμητικούς χαρακτήρες.
- Στα ονόματα των μεταβλητών γίνεται διάκριση ανάμεσα σε πεζά και κεφαλαία (case sensitive).
- Οι δεσμευμένες λέξεις της Python (keywords) δεν μπορούν να χρησιμοποιηθούν σε ονόματα μεταβλητών.

Συμβολοσειρές (Strings)

Μια συμβολοσειρά είναι μια ακολουθία από χαρακτήρες όπως το `"Το πεπρωμένον φυγείν αδύνατον."`. Μπορεί να είναι σε κάθε γλώσσα που υποστηρίζεται από το πρώτυπου Unicode. Οι συμβολοσειρές περικλείονται σε μονά, διπλά ή τριπλά εισαγωγικά. Με τριπλά εισαγωγικά μπορούν να ενσωματωθούν με ευκολία συμβολοσειρές σε πολλές γραμμές και πολλαπλά εισαγωγικά εντός αυτόν. Ακολουθούν παραδείγματα συμβολοσειρά.

Χαρακτήρες διαφυγής,κενά, νέες γραμμές

Μπορούμε να σπάσουμε μια συμβολοσειρά κατά την συγγραφή σε νέα γραμμή με τον χαρακτήρα `\` και κατά την εκτέλεση με τον χαρακτήρα `\n` π.χ.

```
message = 'There is no smoke \
without fire'

print(message)
```

```
There is no smoke without fire
```

```
message = 'There is no smoke \nwithout fire'

print(message)
```

```
There is no smoke
without fire
```

Ή να ορίσουμε κενά με το `\t`

```
message = 'There is no smoke \twithout fire'
print(message)
```

```
There is no smoke       without fire
```

Ο χαρακτήρας `\` είναι χαρακτήρας διαφυγής που απενεργοποιεί την ειδική λειτουργία των παραπάνω ή την παράθεση εισαγωγικών μεσα σε εισαγωγικά.

```
print('There is no smoke \\n without fire')
```

```
There is no smoke \n without fire
```

```
print('Where there\'s a will, there\'s a way')
```

```
Where there's a will, there's a way
```

Ανεπεξέργαστες συμβολοσειρές (Raw Strings)

Παρόμοιο αποτέλεσμα με τα παραπάνω πετυχαίνουμε τις ανεπεξέργαστες συμβολοσειρές οι οποίες ορίζονται με ένα r σαν πρόθεμα

```
print(r"It was made by \n συνέχεια")
```

```
It was made by \n συνέχεια
```

Αφαίρεση κενών

Σε αρκετές περιπτώσεις οι συμβολοσειρές περιέχουν κενά είτε στην αρχή είτε στο τέλος. Για παράδειγμα οι παρακάτω συμβολοσειρές δεν είναι το ίδιες για την Python. Και επιβεβαιώνεται σε μέσω ελέγχου ισότητας.

```
departmentA='ΤΜΧΠΑ'
departmentB = ' ΤΜΧΠΑ '
print(departmentA == departmentB) #not equal
```

```
False
```

Για την αφαίρεση των κένων αριστερά, δεξιά ή ταυτόχρονα και στις δύο πλευρές της συμβολοσειρας χρησιμοποιούμε την μέθοδο `strip` και τις παραλλαγές της `rstrip` και `lstrip`

```
print(departmentB.rstrip())
print(departmentB.lstrip())
print(departmentB.strip())
```

```
 ΤΜΧΠΑ
ΤΜΧΠΑ
ΤΜΧΠΑ
```

Συνένωση (Concatenation) συμβολοσειρών

Η απλή παράθεση συμβολοσειρών οδηγεί στην συνενωσή τους δηλ.

```
message = "Curiosity " "killed " 'the ' '''cat'''
print(message)
```

```
Curiosity killed the cat
```

Συνένωση συμβολοσειρών και μεταβλητών

Η συνένωση μεταβλητών και συμβολοσειρών γίνεται με τον τελεστη `+`.

```
city='Βόλος'
perifereia='Θεσσαλία'

print('Ο '+city+' είναι πόλη της Ελλάδα στην ' +perifereia)
```

```
Ο Βόλος είναι πόλη της Ελλάδα στην Θεσσαλία
```

Η μέθοδος format

Άλλη μια πιο πρακτική μέθοδος κατά την συννένωση μεταβλητών και συμβολοσειρών είναι η μέθοδος format.

```
print('Ο {0} έχει υψόμετρο {1} μέτρα'.format("Όλυμπος", 2918))
print('Ο {} έχει υψόμετρο {} μέτρα'.format("Όλυμπος", 2918))
print('Ο {name} έχει υψόμετρο {height} μέτρα'.format(name="Σμόλικας", height= 2637
))
```

```
Ο Όλυμπος έχει υψόμετρο 2918 μέτρα
Ο Όλυμπος έχει υψόμετρο 2918 μέτρα
Ο Σμόλικας έχει υψόμετρο 2637 μέτρα
```

Δεσμευμένες λέξεις (reserved words)

Ορισμένες λέξεις έχουν ιδιαίτερη σημασία για την python και δεν μπορούν να χρησιμοποιηθούν σαν ονόματα μεταβλητών. Τα παρακάτω κομμάτια κώδικα θα εκδηλώσουν σφάλμα μεταγλώττισης.

Πρόκειται για 33 λέξεις στην τρέχουσα έκδοση της Python. Μπορούμε να δούμε ποιές είναι αυτές οι δεσμεύνες λέξεις με την παρακάτω εντολή:

```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

Η εντολή help

Γενικά με την εντολή `help` καλούμε για βοήθεια και πληροφορίες την Python:

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
help(abs)
```

```
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

```
help(max)
```

```
Help on built-in function max in module builtins:

max(...)
    max(iterable, *[, default=obj, key=func]) -> value
    max(arg1, arg2, *args, *[, key=func]) -> value

    With a single iterable argument, return its biggest item. The
    default keyword-only argument specifies an object to return if
    the provided iterable is empty.
    With two or more arguments, return the largest argument.
```

Αλλαγή Πεζών Κεφαλαίων (Convert case)

Μπορούμε να κάνουμε αλλαγή ανάμεσα σε κεφαλαία και πεζά με τις παρακάτω μεθόδους συμβολοσειρών:`upper()`, `title()`, `lower()`. Αξίζει να σημειώσουμε ότι οι μέθοδοι αυτές δεν έχουν επίδραση στην μεταβλητή που τις καλούμε αλλά πρέπει να επαναεκχωρήσουμε το αποτέλεσμα της μεθόδου στην μεταβλητή με το ίδιο όνομα.

```
agios="άγιος νικόλαος"

print(agios.upper())
print(agios) # ο agios παραμένει "άγιος νικόλαος"

print(agios.title())
print('ΑΓΊΑ ΕΛΈΝΗ'.lower())

agios = agios.upper()
print(agios) # ο agios μετά την εκχώρηση στην ίδια μεταβλητή γινεται ΆΓΙΟΣ
ΝΙΚΌΛΑΟΣ
```

```
ΆΓΙΟΣ ΝΙΚΌΛΑΟΣ
άγιος νικόλαος
Άγιος Νικόλαος
αγία ελένη
ΆΓΙΟΣ ΝΙΚΌΛΑΟΣ
```

Οι συμβολοσειρές είναι μη μεταβαλλόμενη δομή δεδομένων

Οι συμβολοσειρές αποτελούνται από ακολουθίες χαρακτήρων με σταθερό μέγεθος και μη μεταβαλλόμενα περιεχόμενα. Αυτό σημαίνει ότι δεν είναι δυνατόν να προστίθενται ή να αφαιρούνται χαρακτήρες, ούτε να τροποποιούνται τα περιεχόμενα του αλφαριθμητικού. Πρόκειται για μια μη μεταβαλλόμενη (immutable) δομή της Python. Η αρίθμηση των χαρακτήρων σε ένα αλφαριθμητικό ξεκινάει από το 0.

Έτσι στην συμβολοσειρά `country = Ελλάδα` έχουμε:

`country[0]` → Ε (η αρίθμηση ξεκινά από το 0)

`country[1]` → λ

`country[2]` → λ

`country[3]` → ά

`country[4]` → δ

`country[5]` → α

Η παραπάνω συμβολοσειρά έχει μήκος 6 χαρακτήρες.

Μήκος συμβολοσειράς

Μέσω της συνάρτησης `len` η Python μας επιστρέφει το μήκος συμβολοσειράς δηλαδή το πλήθος των χαρακτήρων (μαζί με τα κενά) από τους οποιούς αποτελείται.

```
message = 'Ή τώρα ή ποτέ.'
len(message)
```

```
14
```

Η μέθοδος find

Η μέθοδος `find` μας επιτρέπει να αναζητήσουμε μια συμβολοσειρά μέσα σε μια άλλη συμβολοσειρά. Η μέθοδος μας επιστρέφει την τοποθεσία από την ξεκινάει η αναζητούμενη συμβολοσειρά δηλαδή τον δείκτη (index) στην οποία εντοπίζεται ο πρώτος χαρακτηρας της αναζητούμενης συμβολοσειράς μέσα στα περιεχόμενα της αρχικής συμβολοσειράς. Στην παρακάτω συμβολοσειρά θα αναζητήσουμε την λέξη `ποτέ`.

```
stixos = 'Η Ελλάδα ποτέ δεν πεθαίνει'
index = stixos.find('ποτέ')
```

Κανονικά αν πάμε στον χαρακτήρα με ευρετηρίο (index) 9 πρέπει να εντοπίσουμε τον πρώτο χαρακτήρα της συμβολοσειράς που είναι το `π`. Πράγματι:

```
stixos[index]
```

```
'π'
```

Αν δεν εντοπιστεί η λέξη που αναζητούμε στην συμβολοσειρά η Python θα επιστρέψει: `-1`

```
stixos.find('πάντα')
```

```
-1
```

Η αναζήτηση είναι case sensitive δηλαδή γίνεται διάκριση ανάμεσα σε πεζά και κεφαλαία.

```
stixos.find('Ελλάδα') # επιστρέφει τον δείκτη 2 γιατί εντοπίστηκε η λέξη κλειδί
```

```
2
```

```
stixos.find('ΕΛΛΆΔΑ') # επιστρέφει -1 γιατί δεν εντοπίστηκε η λέξη κλειδί
```

```
-1
```

Μια άλλη σημαντική μέθοδος των συμβολοσειρών είναι η μέθοδος `replace` κατά την οποία μπορούμε να αντικαταστήσουμε τα περιεχόμενα μιας συμβολοσειράς. Στην πρώτη παράμετρο ορίζουμε την συμβολοσειρά που θέλουμε να αντικαταστήσουμε με την δεύτερη παράμετρο.

```
stixos.replace('ποτέ', 'πάντα')
```

```
'Η Ελλάδα πάντα δεν πεθαίνει'
```

3. Λίστες. Εκφράσεις, τελεστές

Στην τρέχουσα ενότητα γίνεται αναφορά σε μια από τις πιο δυνατές και χρήσιμες δομές της Python, τις λίστες. Οι λίστες μας επιτρέπουν να αποθηκεύομαι σύνολα πληροφορίας σε ένα μέρος είτε πρόκειται για ένα είτε για εκατομμύρια στοιχεία. Οι λίστες αποτελούνται από μία σειρά από στοιχεία, καθένα από τα οποία μπορεί να ανήκει σε διαφορετικό τύπο δεδομένων.

> **Definition 4**
>
> Μία **λίστα** (list) είναι μια διατεταγμένη συλλογή τιμών, οι οποίες αντιστοιχίζονται σε δείκτες. Οι τιμές που είναι μέλη μιας λίστας ονομάζονται **στοιχεία** (elements). Τα στοιχεία μιας λίστας δεν χρειάζεται να είναι ίδιου τύπου και ένα στοιχείο σε μία λίστα μπορεί να υπάρχει περισσότερες από μία φορές. Μία λίστα μέσα σε μία άλλη λίστα ονομάζεται **εμφωλευμένη λίστα** (nested list). Επιπρόσθετα, τόσο οι λίστες όσο και οι συμβολοσειρές, που συμπεριφέρονται ως διατεταγμένες συλλογές τιμών, ονομάζονται **ακολουθίες** (sequences). Τα στοιχεία μιας λίστας διαχωρίζονται με κόμμα και περικλείονται σε τετράγωνες αγκύλες ([και]). Μία λίστα που δεν περιέχει στοιχεία ονομάζεται άδεια λίστα και συμβολίζεται με [] [Αγγελιδάκης, 2015].

Παρακάτω δίνονται μερικά παραδείγματα από λίστες

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
colors = ["red", "green", "black", "blue"]
scores = [10, 8, 9, -2, 9]
myList = ["one", 2, 3.0]
languages=[['English'],['Gujarati'],['Hindi'],'Romanian','Spanish'] # εμφωλευμένη λίστα (nested list)
list_A = [] # άδεια λίστα
```

Οι λίστες είναι ταξινομημένες συλλογές δεδομένων. Η πρόσβαση στα στοιχεία της λίστας γίνεται μέσω του δείκτη ή της θέσης του κάθε στοιχείου. Το σημαντικό που πρέπει να συγκρατηθεί είναι ότι η αρίθμηση των στοιχείων σε μια λίστα ξεκινάει από το μηδεν. Το πρώτο στοιχείο έχει δείκτη 0, το δεύτερο 1 κ.ο.κ. Το τελευταίο στοιχείο στην λίστα έχει τον δείκτη -1, το δεύτερο στοιχείο από το τέλος τον δείκτη -2 κ.ο.κ. Δείτε στο παρακάτω παράδειγμα τι εκτυπώνεται με βάση την θέση που δηλώνουμε στην λιστα.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
print(bicycles[1])
print(bicycles[-1])
print(bicycles[1:3])
print(bicycles[1:3])
```

```
trek
cannondale
specialized
['cannondale', 'redline']
['cannondale', 'redline']
```

Παράδειγμα με εμφωλευμένη λίστα (nested list):

```
two_by_two = [[1, 2], [3, 4]]

print(two_by_two[0][1])
print(two_by_two[1][1])
```

```
2
4
```

Οι λίστες που περιέχουν συνεχόμενους ακέραιους αριθμούς μπορούν εύκολα να δημιουργηθούν ως εξής:

```
mylist = list(range(1,20))
print(mylist)

mylist1 = list(range(10))
print(mylist1)

mylist2 = list(range(1,20,4))
print(mylist2)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 5, 9, 13, 17]
```

Η μέθοδο index() μιας λίστας επιστρέφει το ευρετήριο (index) μιας τιμής:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles.index('redline'))
```

```
2
```

Σε αντίθεση με τις συμβολοσειρές (strings) οι λίστες είναι μεταβαλλόμενες δομές δεδομένων. Αυτό σημαίνει ότι μπορούμε να τροποποιήσουμε τα στοιχεία της λίστα, να προσθέσουμε νέα ή να αφαιρέσουμε.

Για παράδειγμα μπορούμε να τροποποιήσουμε τα στοιχεία μιας λίστας ως εξής:

```
colors=['caramel','gold','silver','occur']
colors[3]='bronze'
print(colors)
```

```
['caramel', 'gold', 'silver', 'bronze']
```

```
colors=['caramel','gold','silver','occur']
colors[2:]=['bronze','silver']
print(colors)
```

```
['caramel', 'gold', 'bronze', 'silver']
```

```
colors=['caramel','gold','silver','occur']
colors[2:3]=['bronze','silver']
print(colors)
```

```
['caramel', 'gold', 'bronze', 'silver', 'occur']
```

Μπορούμε να προσθέσουμε **ένα** στοιχείο σε μια λίστα με την μέθοδο *append* π.χ.

```
gods = ['Δίας', 'Ερμής', 'Ποσειδώνας', 'Ήφαιστος']
print("{} θεοί".format(len(gods)))
print(gods)


gods.append("Απόλωνας")
gods.append("Άρης")
print("{} θεοί".format(len(gods)))
print(gods)
```

```
4 θεοί
['Δίας', 'Ερμής', 'Ποσειδώνας', 'Ήφαιστος']
6 θεοί
['Δίας', 'Ερμής', 'Ποσειδώνας', 'Ήφαιστος', 'Απόλωνας', 'Άρης']
```

Αν επιθυμούμε να προσθέσουμε πολλά στοιχεία τότε χρησιμοποιείται η μέθοδος *extend*

```
months=["Ιανουάριος", "Φεβρουάριος", "Μάρτιος"]
months.extend(["Απρίλιος", "Μαϊος"]) # προσοχή το όρισμα στην μέθοδο extend είναι
λίστα (ή κάποιο iterable object)
```

Μπορούμε να αφαιρέσουμε στοιχεία από μία λίστα με διάφορους τρόπους.

Με την πρόταση *del*:

Όταν γνωρίζουμε την θέση του στοιχείου που θέλουμε να αφαιρεθεί από μια λίστα χρησιμοποιούμαι την πρόταση *del*. Για παράδειγμα

```
gods = ['Δίας', 'Ερμής', 'Ποσειδώνας', 'Ήφαιστος']
del gods[2]
print(gods)

# Το μήκος άλλαξε και μερικές από τις αντιστοιχίες στην λίστα. Πλέον ο θεός στην
θέση 2 είναι ο:
print(gods[2])
```

```
['Δίας', 'Ερμής', 'Ήφαιστος']
Ήφαιστος
```

Με την μέθοδο *pop*:

Μία άλλη χρήσιμη μέθοδο για την αφαίρεση στοιχείων από μια λίστα είναι η μέθοδος *pop*. Μέσω της μεθόδου αυτής όχι απλά αφαιρείται το στοιχείο από την λίστα αλλά επιστρέφεται και ως τιμή διαθέσιμη να την εκμεταλλευτεί ο προγραμματιστής π.χ. σε μία νέα μεταβλητή. Δείτε το εξής παράδειγμα:

```
cars=["Alfa Romeo", "Renault", "BMW", "Renault", "Porsche"]
speed_car=cars.pop()
print(cars)
print(speed_car)

speed_car2=cars.pop(0)
print(speed_car2)
```

```
['Alfa Romeo', 'Renault', 'BMW', 'Renault']
Porsche
Alfa Romeo
```

Όπως φαίνεται από το παράδειγμα αν χρησιμοποιηθεί η μέθοδος pop() χωρίς δείκτη τότε αφαιρείται το τελευταίο στοιχείο της λίστας.

Με την μέθοδο *remove*:

Επιπλέον μπορούμε να αφαιρέσουμε στοιχεία από μία λίστα με την χρήση μιας τιμής και όχι με βάση τον δείκτη. Ωστόσο η παραπάνω τεχνική θα αφαιρέσει το πρώτο στοιχείο που θα εντοπιστεί με την τιμή αυτή.

```
gods = ['Δίας', 'Ερμής', 'Ποσειδώνας', 'Ερμής', 'Ήφαιστος']

gods.remove("Ερμής")
print(gods)
```

```
['Δίας', 'Ποσειδώνας', 'Ερμής', 'Ήφαιστος']
```

Χρήσιμες μέθοδοι (methods) και συναρτήσεις (functions)

Μία από τις πλέον χρήσιμες μεθόδους της κλάσσης list είναι η ταξινόμηση (*sort*).

```
cars=[ "Porsche", "Alfa Romeo", "Renault", "BMW", "Audi" ]
cars.sort()
print(cars)
```

```
['Alfa Romeo', 'Audi', 'BMW', 'Porsche', 'Renault']
```

όπως φαίνεται η ταξινόμηση των στοιχείων της λίστας είναι μόνιμη. Ωστόσο αν θέλουμε προσωρινή ταξινόμηση τότε χρησιμοποιείται η function *sorted*.

```
cars=[ "Porsche", "Alfa Romeo", "Renault", "BMW", "Audi" ]
print(sorted(cars))
print(cars)
```

```
['Alfa Romeo', 'Audi', 'BMW', 'Porsche', 'Renault']
['Porsche', 'Alfa Romeo', 'Renault', 'BMW', 'Audi']
```

Όπως βλέπετε κατά την τελευταία εκτύπωση η λίστα διατηρεί την αρχική ταξινόμηση.

Επιπλέον με την μέθοδο *reverse* μπορούμε να αντιστρέψουμε την διάταξη των στοιχείων της λίστας. Και εδώ το αποτέλεσμα είναι μόνιμο.

```
cars=[ "Porsche", "Alfa Romeo", "Renault", "BMW", "Audi" ]

cars.reverse()
print(cars)
```

```
['Audi', 'BMW', 'Renault', 'Alfa Romeo', 'Porsche']
```

Ακόμα μέσω της συνάρτησης *len* επιστρέφεται το πλήθος των στοιχείων της λίστας.

```
languages=['English','Gujarati', 'Hindi','Romanian','Spanish']
print(len(languages))
```

```
5
```

Εδώ πρέπει να δοθεί προσοχή. Γιατί ενώ η αρίθμηση των δεικτών ξεκινά από το 0 το μήκος της λίστας ξεκινά από το 1 για λίστα με ένα στοιχείο. Οπότε στο παρακάτω παράδειγμα θα λάβουμε σφάλμα εκτέλεσης αν πάμε να πάρουμε το 5ο και τελευταίο στοιχείο της λίστας. Γιατί αυτό ορίζεται με τον δείκτη 4 και όχι 5.

```
cars=[ "Porsche", "Alfa Romeo", "Renault", "BMW", "Audi" ]
print("Το μήκος της λίστας (πλήθος στοιχείων) είναι: ", len(cars))
print(cars[5])
```

```
Το μήκος της λίστας (πλήθος στοιχείων) είναι:  5
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [18], in <cell line: 3>()
      1 cars=[ "Porsche", "Alfa Romeo", "Renault", "BMW", "Audi" ]
      2 print("Το μήκος της λίστας (πλήθος στοιχείων) είναι: ", len(cars))
----> 3 print(cars[5])

IndexError: list index out of range
```

Σημείωση

Συχνά χρησιμοποιούμε έναν βρόγχο (loop) όπως το *for* για να προσπελάσουμε ένα προς ένα τα στοιχεία μιας λίστας. Δείτε το επόμενο παράδειγμα.

```
list = ['physics', 'chemistry', 1997, 2000]
for item in list:
    print(item)
```

```
physics
chemistry
1997
2000
```

Σημαντικό

Ιδιαίτερη προσοχή πρέπει να δίνεται στον τρόπο που αντιγράφουμε λίστες. Δείτε γιατί στο επόμενο παράδειγμα.

```
nisia = ["Μήλος", "Κρήτη", "Λέσβος"]
greek_islands = nisia

greek_islands.append("Κέρκυρα")

print(nisia)
```

```
['Μήλος', 'Κρήτη', 'Λέσβος', 'Κέρκυρα']
```

Όπως φαίνεται οι μεταβλητές nisia και greek_islands αντιστοιχούν στο ίδιο υποκείμενο object και αν μεταβάλλοντας τα στοιχεία της μίας μεταβλητής η αλλαγή αντικατοπτρίζεται και στα στοιχεία της δεύτερης. Ο ενδεδειγμένος τρόπος για να αντιγράψουμε μια λίστα είναι να αντιγράψουμε όλα τα στοιχεία της ώστε να έχουμε δύο ανεξάρτητα objects (κλωνοποίηση).

```
nisia = ["Μήλος", "Κρήτη", "Λέσβος"]
greek_islands = nisia[:]

greek_islands.append("Κέρκυρα")

print(nisia)

print(greek_islands)
```

```
['Μήλος', 'Κρήτη', 'Λέσβος']
['Μήλος', 'Κρήτη', 'Λέσβος', 'Κέρκυρα']
```

Εκφράσεις και τελεστές

Μία **έκφραση (expression)** είναι ένας συνδυασμός από *τιμές*, *μεταβλητές*, *τελεστές* και *κλήσεις σε συναρτήσεις*. Οι **τελεστές (operators)** είναι λειτουργίες που κάνουν κάτι και μπορούν να αναπαρασταθούν με σύμβολα όπως το + ή με λέξεις κλειδιά όπως το and. Η αποτίμηση μιας έκφρασης παράγει μία τιμή και αυτός είναι και ο λόγος που μία έκφραση μπορεί να βρίσκεται στο δεξί μέρος μια εντολής εκχώρησης. Όταν μία μεταβλητή εμφανίζεται σε έκφραση, αντικαθίσταται από την τιμή της, προτού αποτιμηθεί η έκφραση [Αγγελιδάκης, 2015]. Δεν απαιτείται μία έκφραση να περιέχει ταυτόχρονα και τιμές και μεταβλητές και τελεστές. Μία τιμή, όπως και μία μεταβλητή, από μόνες τους είναι επίσης εκφράσεις.

Οι τελεστές χρησιμοποιούνται με αριθμητικές τιμές για την εκτέλεση μαθηματικών πράξεων. Ορισμένοι τελεστές έχουν εφαρμογή κα σε συμβολοσειρές. Πιο συγκεκριμένα διατίθενται οι παρακάτω τελεστές:

Αριθμητικοί τελεστές

Τελεστής	Όνομα	Παράδειγμα
+	Πρόσθεση αριθμών ή συνένωση συμβολοσειρών	x + y
-	Αφαίρεση	x - y
*	Πολλαπλασίασμός ή επανάληψη συμβολοσειράς	x * y
/	Διαίρεση	x / y
%	Υπόλοιπο διαίρεσης δύο αριθμών	x % y
**	Ύψωση αριθμού σε δύναμη	x ** y
//	Διαίρεση δύο αριθμών στρογγυλοποιημένη προς τα κάτω	x // y

Τελεστές εκχώρησης

Χρησιμοποιούνται για να αποδώσουν τιμές σε μεταβλητές.

Τελεστής	Παράδειγμα	Αντίστοιχο με
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Τελεστές σύγκρισης

Χρησιμοποιούνται για την σύγκριση 2 τιμών

Τελεστής	Σύγκριση	Παράδειγμα
==	Ίσον	x == y
!=	Διαφορετικό	x != y
>	Μεγαλύτερο από	x > y
<	Μικρότερο από	x < y
>=	Μεγαλύτερο ή ίσον από	x >= y
<=	Μικρότερο ή ίσον από	x <= y

Λογικοί τελεστές

Τελεστής	Περιγραφή	Παράδειγμα
and	Επιστρέφει Αληθές (True) αν και οι δύο προτάσεις είναι αληθείς	x < 5 and x < 10
or	Επιστρέφει Αληθές (True) αν έστω μία από τις προτάσεις είναι αληθή	x < 5 or x < 4
not	Αντιστροφή απότελέσματος, επιστρέφει Μη Αληθές όταν το αποτέλεσμα είναι αληθές	not(x < 5 and x < 10)

Εκτός από τους παραπάνω τελεστές υπάρχουν πιο εξειδικεύμενοι τελεστές που δεν θα αναφερθούμε (Bitwise, Membership, Identity operators).

Οι τελεστές στη Python τηρούν την αντίστοιχη προτεραιότητα που χρησιμοποιείται και στα μαθηματικά.

Οι παρενθέσεις έχουν τη μεγαλύτερη προτεραιότητα. Πολλαπλασιασμός και διαίρεση έχουν υψηλότερα προτεταιότητα από την πρόσθεση και αφαίρεση.

Τελεστές με την ίδια προτεραιότητα αποτιμώνται από τα αριστερά προς τα δεξιά.

Αξιολόγηση

Η αξιολόγηση γίνεται:

- 50% από ατομικές ασκήσεις στην διάρκεια του εξαμήνου.
- 50% από τελικές εξετάσεις.

Με Λεωνίδας Λιάκος