Μαθαίνοντας Προγραμματισμό με Python

Μαθαίνοντας Προγραμματισμό με python

https://sites.google.com/site/pythonlessons/Home

Η σειρά αυτών των μαθημάτων αναπτύσσεται από τον Κ. Σκιαδόπουλο, που εργάζεται ως καθηγητής πληροφορικής στο Γυμνάσιο Καστελλάνων Κέρκυρας. Δεν διεκδικεί δάφνες πρωτοτυπίας, απλά προσπαθεί να καλύψει την ζήτηση φίλων, γνωστών και άγνωστων και ίσως και κάποιων μαθητών με σχετικές ανησυχίες.

Για την συγγραφή αυτής της σειράς χρησιμοποιήθηκαν ιδέες από τις πάρα πολλές σχετικές σελίδες που υπάρχουν στο διαδίκτυο. Όπου χρησιμοποιηθούν αυτούσια παραδείγματα ή τμήματα άλλων εργασιών θα αναφέρεται ρητά η πηγή τους.

Η παρούσα σελίδα δεν φιλοδοξεί να αποτελέσει έναν πλήρη οδηγό εκμάθησης της γλώσσας python, υπάρχουν άλλωστε αρκετοί που καλύπτουν κάθε απαίτηση. Η προσπάθεια που γίνεται εδώ επικεντρώνεται στο να δώσει μια πρώτη προσέγγιση σε όσους δεν έχουν ιδιαίτερη ή και καθόλου γνώση προγραμματισμού.

Εισαγωγή

Η python είναι μια εύκολη στην εκμάθηση και πανίσχυρη γλώσσα προγραμματισμού. Δίνει τη δυνατότητα για εύκολη και γρήγορη παραγωγή μικρών προγραμμάτων (scripts) από τη γραμμή εντολών και για αποτελεσματική προσέγγιση στον αντικειμενοστραφή προγραμματισμό (object-oriented programming).

Μερικά από τα πλεονεκτήματά της :

- Απλή

Η python είναι μια απλή και συμπαγής γλώσσα προγραμματισμού. Όταν διαβάζει κανείς ένα πρόγραμμα python είναι σαν να διαβάζει ένα κείμενο σε απλά Αγγλικά. Έτσι ο προγραμματιστής μπορεί να επικεντρωθεί στην επίλυση του προβλήματος και όχι με την ίδια τη γλώσσα.

- Εύκολη στην εκμάθηση

Η python έχει απλούστατο συντακτικό και είναι πολύ εύκολο να ξεκινήσει κανείς, όπως θα διαπιστώσετε παρακάτω.

- Ελεύθερο και ανοικτό λογισμικό

Η python είναι προϊόν ελεύθερου και ανοικτού λογισμικού και αναπτύσσεται από την κοινότητά της, η οποία σαν μόνη επιδίωξη έχει την συνεχή βελτίωσή της, πέρα από οικονομικές και εμπορικές πολιτικές.

- Ανεξαρτησία λειτουργικού συστήματος

Επειδή η γλώσσα είναι προϊόν ελεύθερου και ανοικτού λειτουργικού έχει μεταφερθεί στα περισσότερα λειτουργικά συστήματα, οπότε μπορεί κανείς να μεταφέρει τα προγράμματά του και να τα λειτουργήσει σε οποιοδήποτε από τα παρακάτω λειτουργικά συστήματα : Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE ακόμη και PocketPC!

- Interpreted (Διερμηνευόμενη)

Αυτό απαιτεί κάποια εξήγηση. Ένα πρόγραμμα που έχει γραφεί σε μια compiled γλώσσα, όπως η C ή η C++, μετατρέπεται από την γλώσσα αυτή στη γλώσσα που "καταλαβαίνει" ο υπολογιστής (μια σειρά από 0 και 1) με τη χρήση ενός προγράμματος μεταγλώττισης (compiler). Η python αντίθετα δεν χρειάζεται μετατροπή. Εκτελεί το πρόγραμμα απευθείας. Μ' αυτόν τον τρόπο αντιγράφοντας το πρόγραμμα σ' ένα άλλο λειτουργικό σύστημα μπορεί κανείς να το εκτελέσει απευθείας.

- Εκτεταμένες βιβλιοθήκες

Η Python Standard Library είναι τεράστια. Περιλαμβάνει threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, κρυπτογραφία, GUI (γραφικό περιβάλλον εργασίας), Tk και πάρα πολύ άλλο υλικό σε διάφορους τομείς. Όλα αυτά είναι διαθέσιμα οπουδήποτε είναι εγκατεστημένη η python. Αυτό αποκαλείται η 'Batteries Included' φιλοσοφία της python.

Εγκατάσταση

Για να δούμε αν είναι εγκατεστημένη η python στο σύστημά μας ανοίγουμε ένα τερματικό ή πηγαίνουμε στην γραμμή εντολών των windows (C:\>) και γράφουμε **python**. Αν πάρουμε κάτι σαν το παρακάτω :

```
~$ python
Python 2.5.2 (r252:60911, Jul 31 2008, 17:28:52)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

η python είναι ήδη εγκατεστημένη στο σύστημά μας. Αν όχι πηγαίνουμε στην σελίδα του οργανισμού python http://www.python.org και την εγκαθιστούμε στο σύστημά μας.

Το Πρόγραμμα HelloWorld

Παραδοσιακά το πρώτο πρόγραμμα που γράφει κανείς σε μια γλώσσα προγραμματισμού είναι το **hello world**. Ένα πρόγραμμα που εμφανίζει στην οθόνη το μήνυμα hello world (δηλαδή "γεια σου κόσμε"). Όσοι έχουν δοκιμάσει κάποια άλλη γλώσσα προγραμματισμού θα εκτιμήσουν την απλότητα της python.

Πηγαίνουμε στην γραμμή εντολών (ανοίγοντας ένα τερματικό ή για τα windows το C:\>) και γράφουμε python. Στην προτροπή (prompt) >>> της pyhton γράφουμε :

```
>>> print('hello world - γεια σου κόσμε')
hello world - γεια σου κόσμε
>>>
```

Τόσο απλό. Στην προτροπή >>> γράφουμε την εντολή μας και μόλις πατήσουμε enter η εντολή εκτελείται και στην παρακάτω γραμμή εμφανίζεται το αποτέλεσμα. Παρακάτω εμφανίζεται πάλι η προτροπή >>> και ο υπολογιστής περιμένει την επόμενη εντολή μας.

Οι Αριθμοί στην Python

Πηγαίνουμε στη γραμμή εντολών (ανοίγοντας ένα τερματικό ή για τα windows το C:\>) και γράφουμε python. Μετά γράφουμε τα παρακάτω :

```
>>> 1+2
3
```

Γράφοντας στην προτροπή (prompt) της python μία αριθμητική πράξη και πατώντας enter έχουμε στην παρακάτω γραμμή το αποτέλεσμα. Παρακάτω παραθέτω έναν πίνακα με τις βασικές εντολές των πράξεων.

Εντολή	**Όνομα**	**Παράδειγμα**	**Αποτέλεσμα**
+	πρόσθεση	1+2	3
-	αφαίρεση	5-4	1
*	πολλαπλασιασμός	7*3	21
/	διαίρεση	10/2	5
%	υπόλοιπο διαίρεσης	9%2	1
**	ύψωση σε δύναμη	2**3	8

Η python όταν γράφουμε κάποιους αριθμούς, όπως το 7 ή το 2, τους καταχωρεί σαν ακέραιους και το αποτέλεσμα των μεταξύ τους πράξεων μας το δίνει κι αυτό σαν ακέραιο :

```
>>> 7 / 2
3
```

Μπορούμε όμως να δώσουμε τους αριθμούς σε μορφή δεκαδικών αν θέλουμε να πάρουμε αποτέλεσμα με δεκαδικούς :

```
>>> 7.0 / 2.0
3.5
```

Παρ' όλο που μόλις αρχίσαμε να μαθαίνουμε την python μπορούμε ήδη να την χρησιμοποιήσουμε σαν κομπιουτεράκι. Ώρα για εξάσκηση ... Και κάτι ακόμα. Οι παρενθέσεις. Αν θέλουμε να κάνουμε πιο πολύπλοκους υπολογισμούς μπορούμε να χρησιμοποιήσουμε τις **παρενθέσεις**.

Η python πρώτα εκτελεί τις πράξεις μέσα στις παρενθέσεις και μετά τις υπόλοιπες. Μάλιστα αν υπάρχουν πολλές πράξεις στη σειρά η python θα τις εκτελέσει με την παρακάτω σειρά :

1. παρενθέσεις ()
2. ύψωση σε δύναμη **
3. πολλαπλασιασμός *, διαίρεση / και υπόλοιπο διαίρεσης %
4. πρόσθεση + και αφαίρεση -

```
>>> 1 + 2 * 3
7
>>> (1 + 2) * 3
9
```

Στο πρώτο παράδειγμα πρώτα εκτελείται ο πολλαπλασιασμός 2 * 3 και μετά στο αποτέλεσμα προστίθεται το 1. Στο δεύτερο παράδειγμα πρώτα εκτελείται η πράξη μέσα στην παρένθεση (1 + 2) και μετά το αποτέλεσμα πολλαπλασιάζεται επί 3. Προσπάθησε να καταλάβεις τι συμβαίνει στα παρακάτω παραδείγματα :

```
>>> 3 – 30 – 2
-29
>>> 3 – (30 – 2)
-25
```

Οι Μεταβλητές (Variables)

Οι μεταβλητές είναι αποθηκευτικοί χώροι. Σ' αυτές μπορείς να αποθηκεύσεις ο,τιδήποτε θα χρειαστείς. Κάθε μεταβλητή αντιστοιχεί σε μια θεση της μνήμης του υπολογιστή και έχει ένα όνομα. Η μεταβλητή μπορεί να κρατήσει μόνο μία τιμή κάθε φορά, μπορείς όμως να αλλάζεις αυτήν την τιμή όποτε χρειάζεται.

Όπως θα προχωράς στην python θα πρέπει να μπορείς να διαβάζεις την τεράστια βιβλιογραφία που υπάρχει σχετικά μ' αυτήν. Το μόνο που χρειάζεται είναι να κατέχεις την ορολογία. Γι' αυτό όπου είναι απαραίτητο, θα δίνω τους όρους και στα Αγγλικά μέσα σε παρένθεση όταν τους παρουσιάζω για πρώτη φορά, όπως μεταβλητές (variable).

Το όνομα μιας μεταβλητής μπορεί να περιέχει γράμματα του λατινικού αλφάβητου και αριθμούς, πρέπει όμως υποχρεωτικά να αρχίζει από γράμμα :

```
>>> a = 5
>>> b = 4
>>> a + b
9
```

Εδώ στην πρώτη γραμμή δημιουργήσαμε μια μεταβλητή με όνομα a και της δώσαμε να αποθηκεύσει τον αριθμό 5. Στην επόμενη γραμμή δημιουργήσαμε την b και της δώσαμε την τιμή 4. Στην τρίτη γραμμή ζητήσαμε να προστεθούν οι τιμές της a και της b και στην τέταρτη έχουμε το αποτέλεσμα.

Σ' αυτό το σημείο πρέπει να τονίσω ότι το σύμβολο = δεν είναι αυτό που ξέρουμε από τα μαθηματικά. Εδώ το = είναι μια εντολή για την python, στα αριστερά έχει το όνομα μιας μεταβλητής και στα δεξιά την τιμή που θα αποθηκευτεί σ' αυτήν. Έχουμε τη δυνατότητα να δώσουμε σε πολλές μεταβλητές τιμή ταυτόχρονα, όπως :

```
>>> a = b = c = 10
>>> a + b + c
30
```

Εδώ στην πρώτη γραμμή δώσαμε στις μεταβλητές a, b, c την ίδια τιμή 10, στην δεύτερη ζητήσαμε να προστεθούν οι τιμές τους και στην τρίτη έχουμε το αποτέλεσμα. Στη συνέχεια θα δώσω ένα παράδειγμα με ένα λαθος, για να εξοικειωθείς με τον τρόπο που αντιδρά η python στα λάθη.

```
>>> 2a = 5
File "<stdin>", line 1
2a = 5
^
SyntaxError: invalid syntax
```

Εδώ το λάθος είναι ότι έδωσα όνομα μεταβλητής που αρχίζει από αριθμό 2a. Στην δεύτερη γραμμή αναγράφεται πού βρέθηκε το λάθος File "<stdin>", line 1 και στην τρίτη γραμμή τονίζει το σημείο και αναγράφει από κάτω το είδος του λάθους (SyntaxError: invalid syntax).

Σε μεγάλα προγράμματα θα χρησιμοποιείς πάρα πολλές μεταβλητές. Αυτό εγκυμωνεί λάθη και μπερδέματα, Για να αποφύγεις παρόμοια προβλήματα πρέπει να αποκτήσεις τη συνήθεια να δίνεις στις μεταβλητές ονόματα που να έχουν νόημα. Όπως :

- length
- myLimit
- finalScore

ή αν προτιμάς σε greeklish (ελληνικά με λατινικά γράμματα) όπως :

- mikos
- protoKerdos
- telikoSkore

Να σημειώσω εδώ ότι και τα κεφαλαία γράμματα επιτρέπονται στα ονόματα των μεταβλητών και η python τα ξεχωρίζει από τα μικρά (case sensitive). Παράδειγμα :

```
>>> ax = 5
>>> Ax = 3
>>> ax + Ax
8
```

Μία μεταβλητή μπορεί να αποθηκεύσει και γράμματα ή λέξεις ή φράσεις εκτός από αριθμούς.

Οι Συμβολοσειρές (Strings)

String ονομάζουμε μια σειρά χαρακτήρων. Παράδειγμα :

```
>>> a = 'good'
>>> a
'good'
```

Στην πρώτη γραμμά δημιουργώ μια μεταβλητή με όνομα a και αποθηκεύω σ' αυτήν τη λέξη good. Στην δεύτερη γραμμή ζητάω να δω το περιεχόμενό της και στην τρίτη το βλέπω. Να σημειώσουμε εδώ ότι ένα string πρέπει να περιβάλλεται από εισαγωγικά μονά ή διπλά. Σε κάποιο παρακάτω σημείο θα αναφέρω τι συμβαίνει με τα μονά ή διπλά εισαγωγικά για τα οποία είναι πολύ

υπερήφανοι οι κατασκευαστές της python. Ας δούμε στη συνέχεια και ένα παράδειγμα με ελληνικά.

```
>>> a = 'καλός'
>>> a
'\xce\xba\xce\xb1\xce\xbb\xcf\x8c\xcf\x82'
```

Εδώ βλέπουμε ότι ζητάμε το περιεχόμενο της μεταβλητής a και παίρνουμε αυτήν την περίεργη σειρά συμβόλων. Η python στην πραγματικότητα δεν έχει κανένα πρόβλημα με τα ελληνικά (όπως και με κάθε άλλη γλώσσα) και για να πεισθείς παραθέτω το παρακάτω :

```
>>> a = 'καλός'
>>> print (a)
καλός
```

Εδώ για να εμφανίσω το περιεχόμενο της μεταβλητής χρησιμοποίησα την εντολή *print* που κάνει αυτή τη δουλειά. Τα σύμβολα που εμφάνισε στο προηγούμενο παράδειγμα οφείλονται στο σύστημα απεικόνισης των χαρακτήρων (Unicode) και είναι ένα θέμα που σε κάποια επόμενη έκδοση της γλώσσας θα λυθεί. Μπορώ όμως να εγγυηθώ ότι δεν πρόκειται να αντιμετωπίσεις κανένα πρόβλημα με τη χρήση των ελληνικών στην python.

```
>>> a = ' Ένα string μπορεί να είναι αρκετά μεγάλο ώστε να καταλαμβάνει χώρο αρκετών γραμμών, αν όμως θέλω μπορώ να τοποθετήσω σε κάποιο σημείο του το σύμβολο αλλαγής γραμμής \n που υποχρεώνει την εντολή print να αλλάξει γραμμή'.

>>> print a

Ένα string μπορεί να είναι αρκετά μεγάλο ώστε να καταλαμβάνει χώρο αρκετών γραμμών, αν όμως θέλω μπορώ να τοποθετήσω σε κάποιο σημείο του το σύμβολο αλλαγής γραμμής \n που υποχρεώνει την εντολή print να αλλάξει γραμμή.
```

Μπορώ να συνενώσω (concatenate) δύο strings :

```
>>> a = 'Καλή '
>>> b = 'μέρα'
>>> c = a + b
>>> print (c)
Καλή μέρα
```

Μπορώ να πολλαπλασιάσω ένα string με έναν αριθμό :

```
>>> a = ' Ηχώ '
>>> b = a * 5
>>> print (b)
Ηχώ  Ηχώ  Ηχώ  Ηχώ  Ηχώ
```

Μπορώ να πάρω έναν χαρακτήρα ενός string :

```
>>> a = 'abcdefghi'
>>> a[0]
'a'
>>> a[1]
'b'
>>> a[7]
'h'
```

Μπορώ να πάρω ένα τμήμα ενός string :

```
>>> a = 'abcdefghi'
>>> a[:4]
'abcd'
>>> a[4:]
'efghi'
>>> a[4:7]
'efg'
```

Σύνθεση Προγράμματος

Στα μέχρι τώρα μαθήματα δίναμε τις εντολές στην προτροπή (prompt) της python. Τώρα μπορούμε να ξεκινήσουμε να φτιάχνουμε προγράμματα και να τα αποθηκεύουμε για μελλοντική χρήση.

- *Με ποιο πρόγραμμα θα γράφουμε τα προγράμματά μας;*

Χρειαζόμαστε έναν editor, μια εφαρμογή δηλαδή που να παράγει απλό κείμενο χωρίς τη διαμόρφωση που κάνουν οι επεξεργαστές κειμένου. Υπάρχουν πολλοί, κάποιοι απ' αυτούς είναι ειδικευμένοι στην δημιουργία προγραμμάτων και προσφέρουν πολλές διευκολύνσεις. Στην παρακάτω διεύθυνση του οργανισμού python υπάρχουν αρκετές δεκάδες editor για κάθε λειτουργικό σύστημα, διάλεξε όποιον πιστεύεις ότι σου ταιριάζει καλύτερα :

http://wiki.python.org/moin/PythonEditors

- *Πού θα αποθηκεύω τα προγράμματά μου;*

Είναι πολύ εύκολο η παραγωγή προγραμμάτων να εξελιχθεί σε μια χαοτική κατάσταση, αν δεν τηρηθούν κάποιοι κανόνες. Πρώτα αποφάσισε σε ποιο σημείο του δίσκου σου θα αναρτήσεις έναν καινούργιο φάκελο με όνομα

python. Μέσα σ' αυτόν να αποθηκεύεις όλα σου τα προγράμματα και μάλιστα όχι έτσι χύμα. Οργάνωσέ τα σε φακέλους. Δίνε στα προγράμματά σου ονόματα περιγραφικά και ακολούθησε τη λογική των εκδόσεων, δηλαδή όταν τροποποιείς ένα υπάρχον πρόγραμμα να το αποθηκεύεις χωρίς να διαγράφεις αυτό που έχεις κάνει μέχρι τώρα. Οπότε ακολούθησε κάτι σαν το εξής :

fractal_1_0.py
fractal_1_1.py
fractal_2_0.py
.....

Όπως θα διαπιστώσεις τα παραγόμενα προγράμματα είναι πάρα πολύ μικρά και δεν συντρέχει λόγος να κάνεις οικονομία στον αποθηκευτικό σου χώρο.

Άνοιξε λοιπόν τον editor επέλεξες και γράψε το παρακάτω πρόγραμμα :

```
a = 3
b = 7
print (a, b, a**b)
```

Αποθήκευσέ το στον φάκελο που δημιούργησες με όνομα **first.py**. Στη συνέχεια άνοιξε ένα τερματικό (ή πήγαινε στο C\: στα windows) και πήγαινε στον φάκελο που έχει αποθηκευμένο το πρόγραμμά σου. Στη συνέχεια γράψε :

python proto.py

και πάτησε enter. Το αποτέλεσμα μπορεί να μη σε εντυπωσιάσει αλλά αν είναι το :

3 7 2187

το πρώτο σου αποθηκευμένο πρόγραμμα δούλεψε σωστά και έμαθες ότι $3^7 = 2187$. Αυτό που συμβαίνει είναι ότι η εντολή **python proto.py** εκτελεί όλες τις εντολές που βρίσκει στο αρχείο proto.py. Τώρα που κατάφερες να γράψεις και να αποθηκεύσεις ένα πρόγραμμα, ο κόσμος της python ανοίγεται μπροστά σου. Από το επόμενο κεφάλαιο θα χρησιμοποιούμε αυτόν τον τρόπο για την δημιουργία προγραμμάτων και θα μπορέσουμε να ξεκινήσουμε την ενασχόληση με πιο ενδιαφέροντα θέματα.

Εντολές Εισόδου και Εξόδου

Σύμφωνα με τα καθιερωμένα ένα πρόγραμμα εισάγει στοιχεία (input data), τα επεξεργάζεται και εξάγει (output) τα αποτελέσματα. Εντολές εισόδου και εξόδου υπάρχουν πολλές, εδώ θα ξεκινήσουμε με την εντολή input για είσοδο και την εντολή print για έξοδο.

- **input()**

Χρήση : μεταβλητή = input(μήνυμα)
Παράδειγμα : name = input ('Δώσε το όνομά σου : ')

Η εντολή αυτή όταν εκτελείται σταματάει την εκτέλεση του προγράμματος, αφού εμφανίσει το μήνυμα που υπάρχει μέσα στην παρένθεση, και περιμένει να γράψουμε κάτι και να πατήσουμε enter. Μετά καταχωρεί αυτό που γράψαμε στη μεταβλητή.

- **print**

Χρήση : print(μεταβλητή ή μήνυμα)
Παράδειγμα : print(a) ή print('Καλημέρα')

Η εντολή print εμφανίζει στην οθόνη ένα string. Εάν ζητήσουμε να εμφανίσει το περιεχόμενο μιας μεταβλητής που δεν είναι string, το μετατρέπει πρώτα σε string και μετά το εμφανίζει. Μπορούμε να ζητήσουμε την εμφάνιση σύνθετων αριθμητικών παραστάσεων με μεταβλητές ή και με αριθμούς, όπως :

print (a(b+3*c))*

Σ' αυτήν την περίπτωση πρώτα εκτελεί τους υπολογισμούς και στην συνέχεια μετατρέπει το αποτέλεσμα σε string και το εμφανίζει στην οθόνη. Μπορούμε να δώσουμε πολλές μεταβλητές και μηνύματα χωρισμένα με κόμματα, όπως :

print ('Το κεφάλαιο είναι : ' , k , ' και ο τόκος είναι : ' , t)

Ας χρησιμοποιήσουμε αυτές τις εντολές για την δημιουργία ενός προγράμματος, που θα υπολογίζει πόσα χρήματα θα έχουμε στον τραπεζικό λογαριασμό μας μετά από ένα χρόνο, αν κάνουμε σήμερα μια κατάθεση. Πριν αρχίσουμε, να πούμε για τα σχόλια ένα απαραίτητο συστατικό ενός καλού προγράμματος.

Τα **σχόλια (comments)** δεν έχουν καμία επίδραση στην εκτέλεση του προγράμματος. Είναι σχόλια που γράφει ο προγραμματιστής και παρεμβάλλονται μέσα στον κώδικα του προγράμματος και σκοπό έχουν να εξηγήσουν και να σχολιάσουν τις εντολές του προγράμματος. Είναι απαραίτητα για να μπορεί ένας τρίτος σχετικά εύκολα να καταλάβει το πρόγραμμά μας αλλά κι εμείς οι ίδιοι αν χρειαστεί να το ξαναδούμε μετά από λίγο καιρό.

Τα σχόλια αρχίζουν με το σύμβολο **#** και συνεχίζουν μέχρι το τέλος της γραμμής.

Ανοίγουμε λοιπόν τον editor που επιλέξαμε να χρησιμοποιούμε και γράφουμε το παρακάτω πρόγραμμα, το οποίο αποθηκεύουμε με το όνομα **interest1.py** :

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
# Αυτό το πρόγραμμα υπολογίζει το ποσό που θα έχουμε σε έναν χρόνο #
αν κάνουμε σήμερα μια κατάθεση.
# Το αρχικό κεφάλαιο και το επιτόκιο εισάγονται από τον χειριστή του
προγράμματος.
# Το ποσό που θα έχουμε μετά από ένα χρόνο αποτελεί την έξοδο του
προγράμματος.
# Το επιτόκιο πρέπει να εισάγεται με τη μορφή δεκαδικού και όχι σαν
ποσοστό (π.χ. 0,05 και όχι 5%)
print ('* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *')
kefalaio = input('Δώσε το αρχικό κεφάλαιο : ')
epitokio = input('Δώσε το επιτόκιο : ')
tokos = int(kefalaio) * int(epitokio)
kefalaio = kefalaio + tokos
print ('Το ποσό που θα έχουμε μετά από ένα χρόνο θα είναι : ' , kefalaio ,
'ευρώ')
print ('* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *')
```

Ας εξηγήσουμε τώρα τι ακριβώς κάνει το παραπάνω πρόγραμμα.

Τα δύο πρώτα σχόλια είναι τα αποκαλούμενα και **μαγικά σχόλια** που αναλαμβάνουν να εξηγήσουν στην python ότι γράφουμε σε γλώσσα που δεν χρησιμοποιεί λατινικούς χαρακτήρες. Αναζήτησε σε κάποια συσκευή αναζήτησης (π.χ. Google) πληροφορίες σχετικά με τους χαρακτήρες unicode για να πάρεις περισσότερες πληροφορίες σχετικά με το θέμα. Αυτές τις δύο πρώτες γραμμές θα πρέπει να τις βάζεις στην αρχή κάθε προγράμματος που θα δημιουργείς.

Στη συνέχεια έχουμε τέσσερις γραμμές με σχόλια που περιγράφουν τι κάνει το πρόγραμμα. Η πρώτη εντολή του προγράμματος (print) εμφανίζει στην οθόνη μια σειρά από * για να ξεχωρίσει ο χώρος που χρησιμοποιεί το πρόγραμμα. Οι δύο επόμενες εντολές είναι εντολές εισόδου. Ζητάνε από τον χρήστη του προγράμματος να δώσει τιμές για το αρχικό κεφάλαιο και το επιτόκιο και αποθηκεύουν τις αντίστοιχες τιμές στις μεταβλητές kefalaio και epitokio.

Οι εντολές tokos = ... και kefalaio = ... εκτελούν τους υπολογισμούς. Η επόμενη εντολή εμφανίζει στην οθόνη το αποτέλεσμα και η τελευταία εντολή print δηλώνει με τα * ότι τελείωσε το πρόγραμμα.

Λίστες (Lists)

Η python έχει έναν τύπο μεταβλητών ο ρόλος του οποίου είναι να ομαδοποιεί άλλα δεδομένα. Ένας τέτοιος τύπος είναι οι **lists**, που στην ουσία είναι κατάλογοι (λίστες) τιμών που διαχωρίζονται με κόμμα και όλες μαζί βρίσκονται μέσα σε τετράγωνες παρενθέσεις. Δεν είναι υποχρεωτικό όλες οι τιμές να είναι του ίδιου τύπου.

```
>>> a =  ['london', 'rome', 1452, 9]
>>> a
['london', 'rome', 1452, 9]
```

Κάθε μέλος της λίστας έχει για όνομα, το όνομα της λίστας και την θέση (index) του σ' αυτήν. Η θέση του πρώτου μέλους μιας λίστας είναι η θέση 0.

```
>>> a[0]
'london'
>>> a[2]
1452
```

Τα μέλη μιας λίστας μπορεί να είναι διαφορετικού τύπου, η python όμως τα αντιμετωπίζει το καθένα ανάλογα με τον τύπο του. Στο προηγούμενο παράδειγμα η λίστα a έχει τα δύο πρώτα μέλη της strings (london, rome) και τα δύο επόμενα ακέραιους (1452, 9). Ας δούμε πώς τα αντιμετωπίζει :

```
>>> a[0] + a[1]
'londonrome'
>>> a[2] + a[3]
1461
```

Στην πρώτη περίπτωση συνένωσε (concatenate) τα δύο strings και στη δεύτερη πρόσθεσε τους δύο ακέραιους. Έχετε υπόψη ότι τέτοια ευελιξία στη διαχείριση των δεδομένων δεν έχουν οι περισσότερες γλώσσες προγραμματισμού. Σ' αυτές ο τύπος των δεδομένων πρέπει να δηλώνεται πριν την οποιαδήποτε χρήση του. Για να δούμε, όμως, τι θα συμβεί αν ζητήσουμε να προστεθεί το μέλος 0 (london) με το μέλος 2 (1492).

```
>>> a[0] + a[2]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Εδώ βγάζει μήνυμα λάθους. Δεν μπορεί να συνενώσει ένα αντικείμενο (object) string (str) με ένα αντικείμενο τύπου ακέραιου (int).

Ο αριθμός που προσδιορίζει τη θέση ενός μέλους μπορεί να είναι αρνητικός. Σ' αυτή την περίπτωση ξεκινάμε να υπολογίζουμε τη θέση μετρώντας από το τέλος προς την αρχή. Η θέση του τελευταίου μέλους μπορεί να προσδιοριστεί και με το -1.

```
>>> a[3]
9
>>> a[-1]
9
>>> a[-3]
'rome'
```

Μπορούμε να πάρουμε ένα κομμάτι από μια λίστα :

```
>>> a[1:3]
['rome', 1452]
```

Το a[1:3] σημαίνει ότι από την λίστα a παίρνω το μέλος a[1] και όλα τα επόμενα μέχρι το a[3] χωρίς να συμπεριλαμβάνεται το a[3]. Θα μπορούσαμε να πούμε ότι ο πρώτος αριθμός δηλώνει τη θέση (index) του μέλους με το οποίο θα αρχίζει το κομμάτι μου και ο δεύτερος τη θέση του μέλους από το οποίο και μετά δεν θέλω να συμπεριλαμβάνονται στο κομμάτι.

Μπορώ να έχω και αρνητικούς αριθμούς χωρίς κανένα πρόβλημα :

```
>>> a[1:-1]
['rome', 1452]
```

Αν λείπει ο πρώτος αριθμός σημαίνει ότι εννοώ την αρχή της λίστας, δηλαδή τον αριθμό 0.

```
>>> a[:3]
['london', 'rome', 1452]
```

Αν λείπει ο δεύτερος αριθμός σημαίνει ότι εννοώ να συμπεριληφθεί στο κομμάτι μου μέχρι και το τελευταίο μέλος της λίστας.

```
>>> a[2:]
[1452, 9]
>>> a[2:4]
[1452, 9]
```

Στο a[2:4] το 4 (δεν υπάρχει μέλος στη λίστα a με θέση 4) σημαίνει ότι το τελευταίο μέλος της λίστας που θα έχει το κομμάτι μου είναι αυτό με τη θέση 3, το τελευταίο δηλαδή της συγκεκριμμένης λίστας. Αν λείπουν και οι δύο αριθμοί σημαίνει ότι παίρνω όλα τα μέλη της λίστας στο κομμάτι μου.

```
>>> a[:]
['london', 'rome', 1452, 9]
```

Αυτό δεν είναι το ίδιο με το όνομα της λίστας. Το a[:] είναι μια νέα λίστα που τυχαίνει να έχει τα ίδια μέλη με την λίστα a. Αυτός είναι ένας τρόπος να δημιουργήσουμε ένα αντίγραφο μιας λίστας.

Οι λίστες στην python είναι αντικείμενα (objects). Θα ασχοληθούμε με τα αντικείμενα αργότερα, εδώ θα αναφέρω μόνο ότι τα αντικείμενα θα μπορούσαμε να τα παρουσιάσουμε σαν δοχεία που περιέχουν δεδομένα και μεθόδους επεξεργασίας αυτών των δεδομένων. Αυτές οι μέθοδοι επεξεργασίας των δεδομένων, που στην ουσία είναι κάποιες εντολές της γλώσσας, ονομάζονται συναρτήσεις (functions).

Μπορούμε όταν εργαζόμαστε με κάποιο αντικείμενο και να χρησιμοποιήσουμε όσες συναρτήσεις διαθέτει, χωρίς να χρειάζεται να ασχοληθούμε με τις εντολές που τις απαρτίζουν. Οι λίστες είναι εφοδιασμένες με μια σειρά συναρτήσεων που έγραψαν οι προγραμματιστές της python.

Μ’ αυτές μπορούμε :

- Να προσθέσουμε μέλη σε μια λίστα :
  - **append()**

    Η συνάρτηση αυτή προσθέτει ένα νέο μέλος στο τέλος της λίστας

    ```
    >>> a.append(12)
    >>> a
    ['london', 'rome', 1452, 9, 12]
    ```

  - **insert()**

    Η συνάρτηση αυτή προσθέτει ένα νέο μέλος σε μια συγκεκριμένη θέση της λίστας :

    ```
    >>> a.insert(2, 'paris')
    >>> a
    ['london', 'rome', 'paris', 1452, 9, 12]
    ```

Όπως βλέπουμε, ο αριθμός στη συνάρτηση insert καθορίζει ποιο μέλος θα παραχωρήσει τη θέση του στο νέο προχωρώντας μια θέση μπροστά. Στη συγκεκριμένη περίπτωση είναι το μέλος 1492 (μην ξεχνάς ότι η αρίθμηση αρχίζει από το 0) που παραχωρεί τη θέση του στο νέο μέλος paris και προχωράει μια θέση μπροστά παρασύροντας και όλα τα υπόλοιπα μέλη.

- **extend()**

  Η συνάρτηση αυτή προσθέτει στο τέλος της λίστας όσα μέλη υπάρχουν μέσα στην παρένθεση :

  >>> a.extend(['milano', 1812])
  >>> a
  ['london', 'rome', 'paris', 1452, 9, 12, 'milano', 1812]

- Να αφαιρέσουμε μέλη από μια λίστα
  - **remove()**

    Η συνάρτηση αυτή αφαιρεί από μια λίστα την πρώτη εμφάνιση μιας συγκεκριμένης τιμής :

    >>> a = ['a', 1, 'b', 3, 'd', 1, 'a', 7]
    >>> a.remove(1)
    >>> a
    ['a', 'b', 3, 'd', 1, 'a', 7]

Εδώ με την συνάρτηση a.remove(1) αφαιρούμε από τη λίστα a το δεύτερο μέλος που είναι το 1. Το έκτο μέλος που είναι επίσης 1 μένει ανεπηρρέαστο. Αν ζητήσουμε να αφαιρεθεί ένα μέλος της λίστας που δεν υπάρχει θα πάρουμε μήνυμα λάθους.

>>> a.remove(9)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list

- **pop()**

  Η συνάρτηση αυτή αφαιρεί από τη λίστα το τελευταίο μέλος της και το επιστρέφει :

  >>> c = a.pop()
  >>> c
  7
  >>> a
  ['a', 'b', 3, 'd', 1, 'a']

Στο παραπάνω παράδειγμα ορίζω μια μεταβλητή με όνομα c και αποθηκεύω σ' αυτήν το αποτέλεσμα της συνάρτησης pop στη λίστα a, δηλαδή το 7.

- Να ψάξουμε μια λίστα :
  - **index()**

    >>> a
    ['a', 'b', 3, 'd', 1, 'a']
    >>> a.index(3)
    2

Η συνάρτηση index() βρίσκει το πρώτο μέλος της λίστας που είναι ίδιο μ' αυτό που έχουμε βάλλει μέσα στην παρένθεση και επιστρέφει τη θέση του μέσα στη λίστα. Στο παραπάνω παράδειγμα η θέση του 3 είναι η 2 (μην ξεχνάμε ότι η πρώτη θέση είναι η 0). Αν στη συνάρτηση index δώσουμε μια τιμή που δεν είναι μέλος της λίστας τότε παίρνουμε μήνυμα λάθους.

```
>>> a.index(8)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: list.index(x): x not in list
```

Να πάρω τον αριθμό των μελών μιας λίστας :

```
>>> len(a)
6
```

Εδώ πρέπει να παρατηρήσουμε ότι η συνάρτηση **len()** δεν είναι μια συνάρτηση που ανήκει στις λίστες αλλά μια συνάρτηση όλης της python. Οι συναρτήσεις που ανήκουν στις λίστες καλούνται με το όνομα της συγκεκριμμένης λίστας ακολουθούμενο από μια τελεία και το όνομα της συνάρτησης, όπως π.χ. a.pop(). Η περίπτωση της len είναι διαφορετική καθώς αυτή δεν έχει εφαρμογή μόνο στις λίστες αλλά σ' όλα τα αντικείμενα της python.

```
>>> k = 'asdfghjkl'
>>> len(k)
9
```

- Να πάρω το άθροισμα των μελών μιας λίστας (αν είναι αριθμοί) :

```
>>> a = [1, 2, 3, 4, 5]
>>> sum(a)
15
```

Άλλη μια πολύ χρήσιμη συνάρτηση της python είναι η **range**(). Η συνάρτηση αυτή επιστρέφει μια λίστα αριθμών, π.χ. :

```
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
```

Ο αριθμός μέσα στην παρένθεση σηματοδοτεί την τελευταία τιμή η οποία δεν συμπεριλαμβάνεται στη λίστα.

```
>>> range(5,11)
[5, 6, 7, 8, 9, 10]
```

Αν δώσω δύο αριθμούς μέσα στην παρένθεση, ο πρώτος δηλώνει τον πρώτο αριθμό της λίστας και ο δεύτερος τον πρώτο αριθμό που δεν θα περιλαμβάνει η λίστα.

```
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
```

Αν δώσω τρεις αριθμούς, οι δύο πρώτοι σηματοδοτούν την αρχή και το τέλος της λίστας, όπως στα προηγούμενα παραδείγματα, και ο τρίτος την απόσταση (βήμα) ανάμεσα σε δύο αριθμούς της λίστας.

```
>>> range(100, 0, -10)
[100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Όπως φαίνεται στο παραπάνω παράδειγμα, η συνάρτηση range() μπορεί να δουλέψει και κατά φθίνουσα σειρά αν δώσουμε τον τρίτο αριθμό αρνητικό.

Δομές Ελέγχου

Ένα πρόγραμμα είναι μια σειρά εντολών. Κατά την συνηθισμένη εκτέλεση του προγράμματος ο υπολογιστής εκτελεί τις εντολές με τη σειρά με την οποία εμφανίζονται, τη μια μετά την άλλη. Αυτό βέβαια οδηγεί σύντομα στην εξάντληση των εντολών και στο τέλος του προγράμματος.

Υπάρχουν όμως τρόποι που επιτρέπουν την αλλαγή στην ροή των εκτελούμενων εντολών (flow control), αυτοί οι τρόποι είναι οι δομές ελέγχου (control structures). Υπάρχουν δύο είδη δομών ελέγχου, οι *δομές επανάληψης (loops)* και οι *δομές επιλογής* ή *διακλάδωσης (branches)*.

Οι δομές επανάληψης επιτρέπουν σε μια ομάδα εντολών να επαναλαμβάνονται ξανά και ξανά, ενώ οι δομές επιλογής επιτρέπουν στον υπολογιστή να επιλέξει ανάμεσα σε δύο ή περισσότερες ομάδες εντολών εξετάζοντας τις συνθήκες που προκύπτουν κατά την εκτέλεση του προγράμματος.

Δομές Επανάληψης

Οι δομές επανάληψης χρησιμοποιούνται όταν κάποιες εργασίες πρέπει γίνουν περισσότερες από μία φορές. Στις περισσότερες περιπτώσεις αυτό σημαίνει να σαρώσουμε μια σειρά δεδομένων και να τα επεξεργαστούμε με τον ίδιο τρόπο ή να επαναλαμβάνονται μια σειρά εντολών ώσπου να να ικανοποιηθεί κάποιο κριτήριο. Στην πρώτη περίπτωση χρησιμοποιούμε την εντολή **for** και στη δεύτερη την εντολή **while**.

- **Εντολή for**

Με την εντολή for μπορώ να σαρώσω μια λίστα και να πάρω διαδοχικά ένα-ένα τα μέλη της.

```
>>> li = [1, 3, 5, 7, 9]
>>> for mel in li:
...     print(mel)
...
1
3
5
7
9
```

Εδώ ορίζω μια λίστα με όνομα li και μέλη τους αριθμούς 1, 3, 5, 7, 9. Στην επόμενη γραμμή δίνω την εντολή for, το όνομα μιας νέας μεταβλητής της mel η οποία θα τροφοδοτηθεί διαδοχικά με τα μέλη της λίστας li .Στο τέλος της γραμμής που περιέχει την εντολή for βάζω πάντα το :.

Η επόμενη γραμμή αρχίζει ένα tab πιο μέσα και όσες γραμμές εντολών περιέχονται μέσα στην επανάληψη for πρέπει κι αυτές να είναι στοιχισμένες κάτω απ' αυτήν. Η εντολή αυτή εκτελείται τόσες φορές όσα είναι τα μέλη της λίστας και κάθε φορά η μεταβλητή mel παίρνει την τιμή ενός από αυτά με τη σειρά. Στο παρακάτω παράδειγμα χρησιμοποιώ και τη συνάρτηση index για να πάρω και τη θέση του κάθε μέλους μέσα στη λίστα.

```
>>> for mel in li :
        ...     print(mel, li.index(mel))
        ...
        1 0
        3 1
        5 2
        7 3
        9 4
```

Άλλο ένα παράδειγμα στο οποίο θα χρησιμοποιήσω την συνάρτηση len() είναι για να μετρήσω τον αριθμό των γραμμάτων μιας σειράς ονομάτων που είναι μέλη μιας λίστας.

```
>>> onoma = ['John', 'Roger', 'Natalie', 'Tamara']
        >>> for a in onoma :
        ...     print(a, len(a))
        ...
        John 4
        Roger 5
        Natalie 7
        Tamara 6
```

- **Εντολή while**

Με την εντολή αυτή επαναλαμβάνεται μία ομάδα εντολών για όσο διάστημα είναι αληθής μια σχέση. Ας το δούμε όμως καλύτερα στην πράξη με ένα πρόγραμμα που υπολογίζει τους όρους της σειράς Fibonacci που είναι μικρότεροι από 200.

```
>>> # Σειρά Fibonacci
... # Το άθροισμα δύο διαδοχικών όρων της σειράς μάς δίνει τον επόμενο.
... # Οι δύο πρώτοι είναι οι 0, 1
... a, b = 0 , 1
>>> while b < 200 :
 ...     print (b)
...     a, b = b, a+b
 ...
1
1
2
3
5
8
13
21
34
55
89
144
```

Στην πρώτη γραμμή του προγράμματος (a, b = 0 , 1) ορίζω δύο μεταβλητές τις a και b και τους δίνω τις τιμές 0 και 1 αντίστοιχα. Η επόμενη εντολή είναι η while b < 200 : Αυτή ελέγχει αν η μεταβλητή b είναι μικρότερη από 200 και αν είναι εκτελεί τις εντολές που βρίσκονται από κάτω και είναι δεξιότερα κατά μια εσοχή και στη συνέχεια ελέγχει πάλι την μεταβλητή b.

Οι εντολές που εκτελεί είναι η print (b) και η a, b = b , a+b. Η πρώτη εμφανίζει στην οθόνη τον νέο όρο της σειράς και η δεύτερη δίνει στην μεταβλητή a την τιμή που έχει η b και στην b το άθροισμα της τιμής που είχε συν

την τιμή της a. Η διαδικασία αυτή συνεχίζεται μέχρι το περιεχόμενο της b να πάψει να είναι μικρότερο από 200.

Προσοχή με την εντολή while μπορούμε να βάλουμε τον υπολογιστή σε μια ατέρμονη διαδικασία, αν δώσουμε μια συνθήκη η οποία είναι πάντοτε αληθής. Για παράδειγμα, αν δώσουμε την εντολή **while b > 0 :** ο υπολογιστής θα υπολογίζει συνεχώς νέους όρους της σειράς Fibonacci και πώς θα σταματήσει;

Ένας τρόπος είναι με τον συνδυασμό από πλήκτρα **Ctrl + C**, αυτά έχουν σαν αποτέλεσμα να σταματάει άμεσα η εκτέλεση του προγράμματος. Το αποτέλεσμα θα είναι κάπως έτσι :

```
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>>
```

Δομές Επιλογής

- **Εντολή if**

Η εντολή **if** θα μπορούσαμε να πούμε ότι εισάγει μια διακλάδωση στη ροή εκτέλεσης των εντολών ενός προγράμματος. Ας το δούμε με ένα παράδειγμα :

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
x = input("Παρακαλώ δώστε έναν αριθμό : ")
if x < 0 :
        print('Αρνητικός αριθμός ')
elif x == 0 :
        print('Μηδέν ')
else :
        print('Θετικός')
```

Οι δύο πρώτες εντολές όπως έχουμε ξαναπεί επιτρέπουν την χρήση των ελληνικών. Η επόμενη εντολή δημιουργεί την μεταβλητή x, εμφανίζει στην οθόνη το μήνυμα *"Παρακαλώ δώστε έναν αριθμό : "*, και περιμένει να εισάγουμε έναν αριθμό. Η επόμενη εντολή ελέγχει αν το περιεχόμενο της μεταβλητής x που έχουμε εισάγει είναι μικρότερο από το 0.

Αν είναι αληθής η συνθήκη x < 0, θα εκτελεστούν οι εντολές που βρίσκονται από κάτω και έχουν μετακινηθεί κατά ένα Tab δεξιότερα, αν δεν είναι αληθής η εκτέλεση του προγράμματος θα συνεχισθεί με την πρώτη εντολή που θα βρεθεί από κάτω και δεν θα έχει μετακινηθεί δεξιότερα με το Tab.

Βλέπουμε εδώ όπως και στις προηγούμενες εντολές πόσο σημαντική είναι η κατακόρυφη στοίχιση για την python. Ένα κενό διάστημα (καλύτερα να βάζουμε ένα Tab για να είναι περισσότερο εμφανές) στην αρχή μιας σειράς εντολών σημαίνει για την python ό,τι οι παρενθέσεις για τις άλλες γλώσσες προγραμματισμού.

Η εντολή ***elif*** σημαίνει **else if**, δηλαδη αν δεν είναι αληθής η συνθήκη της εντολής **if** εξετάζεται η συνθήκη που υπάρχει στην **elif** και αν αυτή είναι αληθής εκτελούνται οι παρακάτω εντολές με την εσοχή, αν όχι η εκτέλεση του προγράμματος πηγαίνει στην επόμενη εντολή elif και αν δεν υπάρχει άλλη πηγαίνει στην εντολή ***else*** και εκτελεί τις εντολές που θα βρεί από κάτω αλλά πάντοτε με την εσοχή.

Το παραπάνω πρόγραμμα το ονόμασα *if_para.py* και παρακάτω ακολουθούν μερικά παραδείγματα εκτέλεσής του.

~/programming/python$ python if_para.py
Παρακαλώ δώστε έναν αριθμό : -25
Αρνητικός αριθμός

~/programming/python$ python if_para.py
Παρακαλώ δώστε έναν αριθμό : 0
Μηδέν

~/programming/python$ python if_para.py
Παρακαλώ δώστε έναν αριθμό : 84
Θετικός

Αλγόριθμοι

Ο προγραμματισμός είναι δύσκολος όπως οι περισσότερες χρήσιμες και αξιόλογες δραστηριότητες και όπως οι περισσότερες από αυτές τις δραστηριότητες προσφέρει μεγάλη ευχαρίστηση και ικανοποίηση. Όταν γράφουμε ένα πρόγραμμα είμαστε υποχρεωμένοι να περιγράφουμε στον υπολογιστή με κάθε λεπτομέρεια το τι πρέπει να κάνει.

Πρέπει όλες αυτές οι περιγραφές να είναι ακριβέστατες, γιατί ο υπολογιστής θα τις ακολουθήσει τυφλά όπως ακριβώς είναι γραμμένες. Οπότε πώς τα καταφέρνουν οι άνθρωποι να γράφουν πολύπλοκα προγράμματα; Στην ουσία δεν είναι κανένα μυστήριο, είναι ζήτημα να αποκτήσει κανείς τον κατάλληλο τρόπο σκέψης.

Ένα πρόγραμμα είναι στην ουσία η έκφραση μιας ιδέας. Ο προγραμματιστής ξεκινά με μια γενική ιδέα για μια εργασία που θέλει να εκτελέσει ο υπολογιστής. Αυτή τη γενική ιδέα πρέπει να την μετατρέψει σε μια σειρά σαφών

βήμα προς βήμα οδηγιών που θα έχουν σαν αποτέλεσμα την ολοκλήρωση της εργασίας. Αυτό είναι ένας αλγόριθμος.

Ο αλγόριθμος δεν είναι το ίδιο με το πρόγραμμα. Το πρόγραμμα είναι γραμμένο σε μια γλώσσα προγραμματισμού, την python στην περίπτωσή μας. Ένας αλγόριθμος είναι περισσότερο η ιδέα που βρίσκεται πίσω από ένα πρόγραμμα. Μια ιδέα που είναι εκφρασμένη σε αναλυτικές οδηγίες που πρέπει να εκφραστούν από το πρόγραμμα για να ολοκληρωθεί η εργασία.

Ο αλγόριθμος μπορεί να εκφραστεί σε οποιαδήποτε ανθρώπινη γλώσσα. Για να μετατραπεί ένας αλγόριθμος σε πρόγραμμα πρέπει να περιέχει όλες τις απαραίτητες λεπτομέρειες που απαιτούνται. Ας τα δούμε όλα αυτά στην πράξη. Έστω ότι η εργασία που θέλουμε να εκτελεστεί είναι ο υπολογισμός του υπολοίπου ενός τραπεζικού λογαριασμού με σταθερό επιτόκιο για 5 συνεχόμενα χρόνια. Ας το γράψουμε πιο αναλυτικά.

Υπολόγισε και εμφάνισε το υπόλοιπο ενός τραπεζικού λογαριασμού για κάθε ένα από τα επόμενα πέντε χρόνια, όπου το αρχικό κεφάλαιο και το επιτόκιο θα δωθούν από τον χειριστή του προγράμματος.

Θα μπορούσε η παραπάνω φράση να αναπτυχθεί :

Πάρε τα στοιχεία από τον χειριστή.
Υπολόγισε το υπόλοιπο μετά από 1 χρόνο.
Εμφάνισε το υπόλοιπο.
Υπολόγισε το υπόλοιπο μετά από 2 χρόνια.
Εμφάνισε το υπόλοιπο.
Υπολόγισε το υπόλοιπο μετά από 3 χρόνια.
Εμφάνισε το υπόλοιπο.
Υπολόγισε το υπόλοιπο μετά από 4 χρόνια.
Εμφάνισε το υπόλοιπο.
Υπολόγισε το υπόλοιπο μετά από 5 χρόνια.
Εμφάνισε το υπόλοιπο.

Το παραπάνω είναι σωστό αλλά έχει μάλλον περιττές επαναλήψεις. Θα μπορούσε να απλοποιηθεί με τη χρήση κάποιας δομής επανάληψης. Μια τέτοια δομή θα έκανε την εργασία μας πιο γενική γιατί η ίδια δομή επανάληψης θα μπορούσε να δουλεύει ανεξάρτητα από των αριθμό των χρόνων που θα υπολογίζει.

Πάρε τα στοιχεία από τον χειριστή.
Όσο υπάρχουν ακόμα χρόνια για να υπολογίσεις :
    Υπολόγισε το υπόλοιπο μετά ένα χρόνο
    Εμφάνισε το υπόλοιπο

Αν ακολουθήσει κανείς τον παραπάνω αλγόριθμο σίγουρα θα μπορέσει να λύσει το πρόβλημα, ο υπολογιστής όμως χρειάζεται αναλυτικότερες οδηγίες. Θα πρέπει να αναλύσουμε περισσότερο τις φράσεις : Πάρε τα στοιχεία από τον χειριστή Όσο υπάρχουν ακόμα χρόνια για να υπολογίσεις Υπολόγισε το υπόλοιπο μετά ένα χρόνο. Θα ξεκινήσουμε με τη φράση : Πάρε τα στοιχεία από τον χειριστή.

Ζήτα από τον χειριστή το αρχικό κεφάλαιο
Διάβασε την απάντηση του χειριστή
Ζήτα από τον χειριστή το επιτόκιο
Διάβασε την απάντηση του χειριστή

Για να μπορέσουμε να πούμε στον υπολογιστή *Υπολόγισε το υπόλοιπο μετά ένα χρόνο* θα πρέπει δυστυχώς να ξέρουμε εμείς τον τρόπο του υπολογισμού. Δεν χρειάζεται να είναι κανείς διευθύνων σύμβουλος σε τράπεζα για να βρει ότι ο τρόπος υπολογισμού είναι ο πολλαπλασιασμός του κεφαλαίου με το επιτόκιο για να βρεθεί ο τόκος και στη συνέχεια η πρόσθεση του τόκου στο κεφάλαιο. Οπότε ο αλγόριθμός μας γίνεται :

Ζήτα από τον χειριστή το αρχικό κεφάλαιο
Διάβασε την απάντηση του χειριστή
Ζήτα από τον χειριστή το επιτόκιο
Διάβασε την απάντηση του χειριστή
Όσο υπάρχουν ακόμα χρόνια για να υπολογίσεις :
    *Υπολόγησε τον τόκο = κεφάλαιο * επιτόκιο*
    Πρόσθεσε τον τόκο στο κεφάλαιο
    Εμφάνισε το κεφάλαιο

Τώρα μένει μόνο να εξετάσουμε τον τρόπο με τον οποίο η δομή επανάληψης θα κάνει τους υπολογισμούς μόνο για τα 5 ζητούμενα χρόνια. Εδώ θα χρησιμοποιήσουμε μια τεχνική που θα συναντήσουμε σε πάρα πολλά προγράμματα. Θα δημιουργήσουμε μια μεταβλητή που θα πάρει αρχική τιμή το 0 και κάθε φορά που θα εκτελούνται οι υπολογισμοί θα αυξάνει κατά 1. Ας την ονομάσουμε years.

years =0
Όσο years < 5 :
    years = years +1
    *Υπολόγησε τον τόκο = κεφάλαιο * επιτόκιο*
    Πρόσθεσε τον τόκο στο κεφάλαιο
    Εμφάνισε το κεφάλαιο

Έτσι φτάσαμε στο σημείο όπου μπορούμε να μετρέψουμε τον αλγόριθμο που δημιουργήσαμε σε πρόγραμμα. Μένει ακόμα να δώσουμε ονόματα στις μεταβλητές που θα χρησιμοποιήσουμε και να συγκεκριμενοποιήσουμε τα μυ-

νήματα που θα εμφανίσει ο υπολογιστής. Μια μετατροπή σε πρόγραμμα python θα ήταν όπως παρακάτω :

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-
kefalaio = input('Δώσε το αρχικό κεφάλαιο : ')
epitokio = input('Δώσε το επιτόκιο σε μορφή δεκαδικού : ')
years = 0
while years < 5 :
        years = years + 1
        tokos = kefalaio * epitokio
        kefalaio = kefalaio + tokos
        print kefalaio
```

Συναρτήσεις (Functions)

Μια από τις βασικές ιδέες για την επίλυση ενός πολύπλοκου προβλήματος, είναι να το διασπάσουμε σε σειρά απλών προβλημάτων. Επιλύουμε το καθένα από αυτά και οδηγούμαστε στη λύση του αρχικού. Στον προγραμματισμό εκτός από την παραπάνω ιδέα είναι πολύ χρήσιμο το κάθε απλό πρόβλημα που επιλύουμε να μπορούμε να το χρησιμοποιήσουμε ξανά, για να λύσουμε ένα άλλο σύνθετο πρόβλημα.

Ακόμα καλύτερα θα είναι να δημιουργήσουμε βιβλιοθήκες από έτοιμα προγράμματα που επιλύουν βασικά προβλήματα και να τις έχουμε διαθέσιμες όποτε τις χρειαστούμε. Τέλος η ιδανική λύση είναι αυτές οι βιβλιοθήκες να εμπλουτίζονται από όλους όσους γράφουν προγράμματα σε μια γλώσσα και να είναι διαθέσιμες σε όσους χρησιμοποιούν αυτή τη γλώσσα. Αυτή είναι σε γενικές γραμμές η ιδέα πίσω από τις συναρτήσεις.

Μια συνάρτηση μπορούμε να την φανταστούμε σαν ένα μαύρο κουτί το οποίο εκτελεί κάποια λειτουργία όταν την καλέσουμε από το πρόγραμμά μας. Σε ορισμένες περιπτώσεις πρέπει να την τροφοδοτήσουμε με κάποια δεδομένα (data), σε άλλες όχι, σε ορισμένες περιπτώσεις μάς επιστρέφει κάποιες τιμές όταν την καλέσουμε ενώ σε άλλες όχι.

Ας δούμε ένα παράδειγμα κατασκευής και χρήσης μιας συνάρτησης. Θέλω να φτιάξω ένα πρόγραμμα που θα μου ζητάει έναν αριθμό και θα με πληροφορεί αν ο αριθμός αυτός είναι μονός ή ζυγός. Σκέφτομαι να φτιάξω μια συνάρτηση που θα της δίνω έναν ακέραιο (int) και θα μου επιστρέφει ένα string ('μονός' ή 'ζυγός' ανάλογα με την περίπτωση). Θα δώσω σ' αυτή τη συνάρτηση το όνομα even_or_odd().

Γράφω το υπόλοιπο πρόγραμμα :

```
a = int (input('Δώσε έναν αριθμό : '))
print('Ο αριθμός ' + str( a ) + ' είναι ' + even_or_odd(a))
```

Τώρα δεν μένει παρά να γράψω τον κώδικα για την συνάρτηση :

```
def even_or_odd(x) :
        if x % 2 == 0 :
                return 'ζυγός'
        else:
                return 'μονός'
```

Τοποθέτησε την συνάρτηση στο ίδιο αρχείο με το πρόγραμμα και έλεγξε αν δουλεύει σωστά.

Να δούμε τώρα πώς είναι γραμμένη η συνάρτηση. Αρχίζει με τα γράμματα **def** και στη συνέχεια έχει το όνομα της συνάρτησης. Το def (βγαίνει από το **define**) σημαίνει για την python ότι ακολουθει ο κώδικας μιας συνάρτησης το όνομα της οποίας είναι αμέσως μετά. Μετά το όνομα έχει μια παρένθεση. Μέσα στην παρέθεση βάζουμε τις παραμέτρους (αν υπάρχουν) που θέλουμε να δώσουμε στη συνάρτηση.

Μετά την παρένθεση βάζουμε **:** και από κάτω αρχίζουμε να γράφουμε τον κώδικα με μια εσοχή προς τα δεξιά. Στο συγκεκριμμένο παράδειγμα ελέγχουμε αν το υπόλοιπο της παραμέτρου που λαμβάνει η συνάρτηση (σ' αυτήν την περίπτωση x) δια δύο είναι μηδέν. Αν είναι 0 η συνάρτηση επιστρέφει (return) την τιμή 'ζυγός' ενώ αν όχι την τιμή 'μονός'.

Η συνάρτηση αυτή δεν είναι πλήρης γιατί δεν εξετάζει αν η παράμετρος που παίρνει είναι όντως ακέραιος αλλά ας την αφήσουμε έτσι για λόγους απλότητας. Αυτή τη συνάρτηση θα μπορούσα να την αποθηκεύσω σ' ένα ξεχωριστό αρχείο μαζί με άλλες συναρτήσεις και αυτό το αρχείο να αποτελεί ένα είδος προσωπικής βιβλιοθήκης συναρτήσεων.

Μ' αυτόν τον τρόπο αν μου χρειαστεί σε κάποιο άλλο πρόγραμμα που θα αναπτύξω στο μέλλον θα μπορώ να την ξαναχρησιμοποιήσω. Η python διαθέτει μια τεράστια συλλογή από έτοιμες συναρτήσεις σε αχανείς βιβλιοθήκες, όλες διαθέσιμες σ' όποιον τις χρειαστεί και είναι σε θέση να τις βρεί. Για να μπορέσω να χρησιμοποιήσω έτοιμες συναρτήσεις θα πρέπει να πω στην python πού θα τις βρει. Αυτό γίνεται με τον παρακάτω τρόπο.

```
import math
a = int(input ('Δώσε έναν αριθμό '))
print('1 x 2 x ... x ' + str(a) + ' = ' + str(math.factorial(a)))
```

Στο παραπάνω πρόγραμμα στην πρώτη γραμμή ζητήσαμε από την python να ενσωματώσει στο πρόγραμμά μας τις συναρτήσεις που βρίσκονται στην **βιβλιοθήκη** της python με όνομα **math**. Αυτή όπως λέει και το όνομά της περιέχει συναρτήσεις σχετικές με τα μαθηματικά. Εμείς χρησιμοποιήσαμε την συνάρτηση **factorial()** που δέχεται έναν ακέραιο και επιστρέφει το γινόμενο 1 x 2 x 3 x ... μέχρι τον αριθμό που δώσαμε. Δεν ξέρουμε πώς κάνει τον υπολογισμό και δεν χρειάζεται να δούμε τον κώδικά της απλά την χρησιμοποιούμε.

Αντικειμενοστραφής Προγραμματισμός (Object Oriented Programming) στην Python

Ο τίτλος ακούγεται βαρύς όμως τα πράγματα είναι σχετικά απλά. Αντικείμενα (objects) είναι κομμάτια κώδικα μαζί με δεδομένα (data). Τα κομμάτια του κώδικα είναι αυτά που μπορούν να διαχειριστούν τα δεδομένα. Χρησιμοποιούμε αντικείμενα στον προγραμματισμό γιατί απλοποιούν σημαντικά την επίλυση κάθε είδους προβλημάτων.

Πώς δημιουργούνται τα αντικείμενα ;

Έστω ότι γράφουμε ένα πρόγραμμα το οποίο θα διαχειρίζεται πληροφορίες σχετικές με αυτοκίνητα. Οι πληροφορίες που θα πρέπει να έχουμε για κάθε αυτοκίνητο είναι ίδιες : *Αριθμός κυκλοφορίας, Κατασκευαστής, Έτος κυκλοφορίας, Κυβισμός, Αριθμός επιβατών*.

Στον αντικειμενοστραφή προγραμματισμό δημιουργούμε ένα τμήμα κώδικα, το οποίο καλούμε κάθε φορά που θέλουμε να δημιουργήσουμε ένα νέο αντικείμενο αυτοκίνητο και του δίνουμε τα στοιχεία (data) που χαρακτηρίζουν αυτό το αντικείμενο. Το τμήμα αυτό του κώδικα λειτουργεί σαν καλούπι που παράγει κάθε φορά που θα του ζητηθεί παρόμοια αντικείμενα και ονομάζεται **class**. Ας υλοποιήσουμε το παραπάνω παράδειγμα σε python.

```
class car:
def __init__(self, arky, kata, etky, kybi, arep):
        self.ak = arky
        self.ka = kata
        self.ek = etky
        self.ky = kybi
        self.ae = arep
```

Για να δούμε τι ακριβώς κάνει αυτός ο κώδικας. Στην πρώτη γραμμή ορίζεται το όνομα της class που στην συγκεκριμμένη περίπτωση είναι car. Στην επόμενη γραμμή το def σημαίνει ότι ακολουθεί μια συνάρτηση η οποία θα μπορεί να κάνει κάποιες ενέργειες με τα δεδομένα αυτού του αντικειμένου. Το όνομα της συγκεκριμμένης συνάρτησης είναι __init__ .

Αυτή η συνάρτηση έχει την ιδιομορφία να εκτελείται αυτόματα κάθε φορά που η class δημιουργεί ένα νέο αντικείμενο. Μέσα στην παρένθεση έ-

χουμε τα δεδομένα που πρέπει να πάρει η συνάρτηση για να μπορέσει να εκτελέσει τη λειτουργία της. Το πρώτο από αυτά το self είναι το όνομα του αντικειμένου στο οποίο αναφέρεται η συνάρτηση και πρέπει να μπαίνει στα δεδομένα κάθε συνάρτησης ενός αντικειμένου.

Το πρόβλημα είναι ότι κάθε αντικείμενο που θα δημιουργεί η class θα έχει διαφορετικό όνομα, γι' αυτό επιλέχθηκε το self. Στη συνέχεια έχουμε 5 μεταβλητές που μεταφέρουν τα δεδομένα του συγκεκριμμένου αυτοκινήτου που θα δημιουργηθεί. Στις παρακάτω γραμμές τα δεδομένα αυτά τροφοδοτούν τις μεταβλητές του αυτοκινήτου, που για τον λόγο που ανέφερα παραπάνω έχουν μπροστά το self.

Ας δούμε τώρα πώς μπορούμε να δημιουργήσουμε ένα αυτοκίνητο. Στον παραπάνω κώδικα προσθέτουμε παρακάτω :

```
c1 = car('AXA5674', 'Ford', 2003, 1600, 5)
```

Όταν εκτελεστεί αυτή η εντολή δημιουργείται στη μνήμη του υπολογιστή χώρος για το νέο αντικείμενο c1. Εκτελείται η συνάρτηση __init__ , παίρνοντας τα δεδομένα που χρειάζεται. Προσοχή στη γραμμή def __init__ μέσα στην παρένθεση βλέπουμε ότι συνάρτηση δέχεται 6 τιμές ενώ εμείς τις στέλνουμε 5 ('AXA5674', 'Ford', 2003, 1600, 5) , η έκτη που δεν στείλαμε είναι η self που παίρνει την τιμή c1. Έτσι στις παρακάτω γραμμές παίρνουν τιμές οι μεταβλητές του αντικειμένου c1 δηλαδή οι c1.ak , c1.ka, c1.ek, c1.ky, c1.ae. Για να δούμε τι έχουμε κάνει ας δώσουμε την παρακάτω εντολή :

```
print ('Το αυτοκίνητο με αριθμό κυκλοφορίας : ' + c1.ak + ' είναι
        κατασκευασμένο από την ' + c1.ka)
```

Ας προσθέσουμε άλλη μια συνάρτηση στην class που θα επιστρέφει ένα string με τα φορολογικα στοιχεία δηλαδή *Αριθμός κυκλοφορίας, Έτος κυκλοφορίας, Κυβισμός* χωρισμένα με ένα κόμμα.

```
class car:
    def __init__(self, arky, kata, etky, kybi, arep):
        self.ak = arky
        self.ka = kata
        self.ek = etky
        self.ky = kybi
        self.ae = arep
    def forologika(self):
        a = self.ak + str(self.ek) + str(self.ky)
        return a
    c1 = car('AXA5674', 'Ford', 2003, 1600, 5)
    print (c1.forologika())
```

Εδώ βλέπουμε ότι καλούμε την συνάρτηση του αυτοκινήτου c1 με όνομα forologika χωρίς να τις δώσουμε καμμία τιμή c1.forologika(). Ας φτιάξουμε άλλο ένα αντικείμενο με όνομα c2 :

c2 = car('INA9481', 'FIAT', 2001, 1100, 4)

Και ας δούμε πόσα άτομα μπορούν να μεταφέρουν τα δύο αυτά αυτοκίνητα :

print ('Μεταφορική ικανότητα των ' + c1.ak + ' + ' + c2.ak + ' ' +
str(c1.ae + c2.ae) + ' ατομα ')

Με το παραπάνω παράδειγμα δεν έγιναν φανερές οι δυνατότητες του αντικειμενοστραφούς προγραμματισμού, έγινε μόνο μια παρουσίαση του τρόπου δημιουργίας των αντικειμένων.