Μάθε Python[1]

Ιωάννης Γαβιώτης
igaviotis@gmail.com
Μηχανικός Ηλεκτρονικών Υπολογιστών & Πληροφορικής

Περιεχόμενα

1 Τα βασικά

Η Python είναι μια εύκολη στη χρήση και απλή στη σύνταξη γλώσσα γενικής χρήσης. Είναι διερμηνευόμενη (interpreted), γρήγορη στη συγγραφή κώδικα, εύκολη στην κατανόησή του και πιο αργή σε ταχύτητα εκτέλεσης. Εκτός από τις δικές της εκτεταμένες βιβλιοθήκες που καλύπτουν πολλά πεδία εφαρμογών, μπορεί να καλεί τμήματα κώδικα άλλων γλωσσών. Είναι ιδανική για τη συγγραφή σεναρίων (scripts), όχι τόσο για την ανάπτυξη εφαρμογών. Επίσης χρησιμοποιείται ως server-side γλώσσα σε web programming.

Δημιουργήθηκε από τον Ολλανδό Guido van Rossum το 1990.

Βασικά χαρακτηριστικά:

1. Δεν δηλώνει τύπους μεταβλητών (dynamically typed)
2. Δεν κάνει αυτόματα μετατροπές τύπων (strongly typed)
3. Δεν χρειάζεται τερματισμός των εντολών (όπως στη Basic και όχι όπως με το `;` στη Java)
4. Τα μπλοκ εντολών υποδεικνύονται μέσω οδόντωσης και όχι με αγκύλες `{}`
5. Διακρίνει κεφαλαίους και πεζούς χαρακτήρες
6. Υποστηρίζει συναρτησιακό και αντικειμενοστραφή προγραμματισμό (κουτσά-στραβά)

```
x = 'γεια' # μεταβλητή συμβολοσειράς
x = 13 # τώρα έγινε ακέραιος αρ.1
print(x + 'abc') # χτυπάει σφάλμα αρ.2
if x < 0:
    x = -x # μπλοκ του if αρ.4
Print(X) # σφάλμα αρ.5
```

Ικανός λόγος για να τη συμπαθήσεις είναι να διαβάσεις το Zen of Python δίνοντας την εντολή `import this` από το Python shell. Σταχυολογώ:

```
Beautiful is better than ugly.
Simple is better than complex.
Readability counts.
```

Πολύ χρήσιμη λειτουργία της Python είναι τα Jupyter notebooks, 'ζωντανά' έγγραφα που ενσωματώνουν, εκτός από αναγνώσιμο περιεχόμενο, και πηγαίο κώδικα που εκτελείται αλληλεπιδραστικά, προβάλλοντας το αποτέλεσμα του υπολογισμού.

[1] Απευθύνεται σε ανθρώπους που ξέρουν προγραμματισμό – δεν είναι εισαγωγική παρουσίαση προς αρχαρίους.

Τέλος, μια ευπρόσδεκτη χρήση της Python είναι σε web frameworks, όπως το Django και το Flask, που αποσκοπούν στην ταχεία ανάπτυξη web εφαρμογών, παρακάμπτοντας τη JavaScript.

2 Προετοιμασία

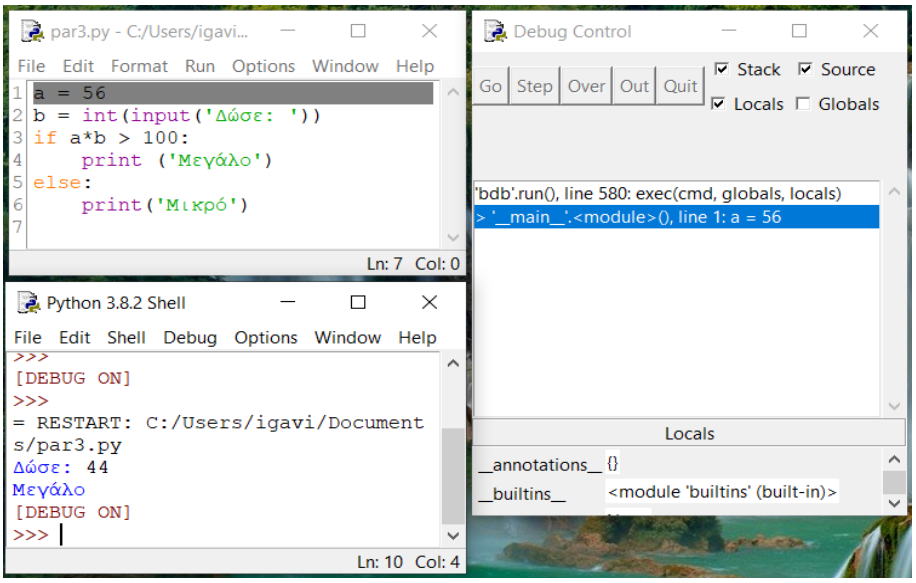
Πηγαίνεις στο https://www.python.org/downloads/, κατεβάζεις και εγκαθιστάς την έκδοση που σου προτείνεται (τυπικά 3.X).

Μετά εκτελείς την εφαρμογή IDLE που σου ανοίγει ένα κέλυφος (shell) για να εισάγεις και να εκτελείς εντολές Python άμεσα βλέποντας το αποτέλεσμα της εκτέλεσής τους.

Για να ενεργοποιήσεις τις δυνατότητες αποσφαλμάτωσης, από το μενού Debug | Debugger ανοίγεις το παράθυρο Debug Control.

Για να γράψεις ένα πρόγραμμα, από το μενού File | New File ανοίγεις editor. Το πρόγραμμα τρέχει με Run μέσα στο shell.

Έτσι έχουμε τρία παράθυρα ανοικτά: editor, debugger και shell.



Αυτή είναι η πιο απλή εκδοχή για **out-of-the-box** Python.

2.1 Εναλλακτικά περιβάλλοντα ανάπτυξης

Με δεδομένο ότι έχει εγκατασταθεί η Python, για να γράψεις προγράμματα μπορείς να χρησιμοποιήσεις το **Notepad++** ως editor. Μόλις αποθηκεύσεις το αρχείο με κατάληξη .py, το αναγνωρίζει ως Python και το 'χρωματίζει'. Από το μενού Run | Run εισάγεις

```
"C:\Program Files (x86)\Python38-32\python.exe" -i "$(FULL_CURRENT_PATH)"
```

ή όπου έχεις εγκαταστήσει την Python. Κάνεις Save για να το θυμάται και το τρέχεις με F5. Ελαφριά και απολύτως βασική λύση για να τρέξεις ένα Python script.

Το **Visual Studio Code** μπορεί να το έχεις στον υπολογιστή σου από άλλες προγραμματιστικές περιπέτειες και είναι μια χαρά περιβάλλον και για Python. Μόλις γράψεις και αποθηκεύσεις ένα `.py` αρχείο θα σου προτείνει την εγκατάσταση των πρόσθετων extensions `python`, `pylint` άντε και `autopep8` για beautify code και είσαι έτοιμος για απογείωση. Παρότι τυπικά είναι editor και όχι πλήρες περιβάλλον ανάπτυξης (IDE, Integrated Development Environment), στις πρόσφατες εκδόσεις του υποστηρίζει Python ικανοποιητικά, όπως και jupyter notebooks.

Αν έχεις ήδη εγκατεστημένο στο PC σου το **Visual Studio** Community Edition και το έχεις συνηθίσει, είναι προτιμότερο. Στο Create a new project, επιλέγεις πχ Python, Windows, Console και ξεκινάς. Προσφέρεται syntax-directed editing, completion, debugging. Πλήρες, βαρβάτο περιβάλλον ανάπτυξης, αλλά βαρύ κι ασήκωτο.

Το **PyCharm** Community είναι γλύκα, όπως λέει και το όνομά του. Out of the box κάνει τα πάντα για Python. Αν σκοπεύεις να ασχοληθείς με Python, αυτό είναι. Ως περιβάλλον ανάπτυξης, νομίζω χτυπάει το sweet spot.

Ανακεφαλαιώνοντας τα IDE με σειρά προτίμησης: PyCharm, Visual Studio Code, Visual Studio, Python IDLE, Notepad++

Για να πάρεις μια μυρωδιά από κώδικα Python, ρίξε μια ματιά στο ταμπλό και μετά θα τα δούμε ένα-ένα.

```
print('hello')
a = 4 + 3 - 2 * 5 // 6  # int
b = 3.4 ** 2 / 10  # float
z = 3 + 4j  # complex
y = -1 - 1.5j
print(z + y)

s = 'Abcd ' + 'Efg '  # συνένωση συμβολοσειρών
print(s + ' has length ' + str(len(s)))  # μετατροπή
print(4 * 'S' + s[2:4] + s[-2:-1])  # ‘πολλαπλασιασμός’ και κατάτμηση
# τυπώνει ‘SSSScdg’
primes = [1, 3, 5, 7, 13]  # list, μπορεί να φωλιαστεί
p1 = primes[:4] + [11] + primes[4:]  # κατάτμηση λίστας
p1.append(17)  # οι λίστες αλλάζουν - οι συμβολοσειρές όχι

def myControl():  # ορισμός συνάρτησης
    m, n = 0, 1  # πολλαπλή ανάθεση
    while n < 10:
        print('fib ' + str(n))  # οδόντωση δηλώνει φώλιασμα
        m, n = n, m + n

    k = int(input('Δώσε ακέραιο: '))  # είσοδος από κονσόλα
    if k == 0:  # εντολή επιλογής
        print('μηδέν')
    else:
        print('όχι μηδέν')

    for i in range(-5, 5, 3):  # επανάληψη από -5 ως 4 με βήμα 3
        print(i, end='#')
    print()

myControl() # κλήση συνάρτησης
```

3 Μεταβλητές και ανάθεση

Οι μεταβλητές δημιουργούνται με την εντολή ανάθεσης και λαμβάνουν τον τύπο της τιμής που τους ανατίθεται δυναμικά κατά την εκτέλεση, δίχως να έχει προηγηθεί δήλωση. Οι βασικοί τύποι δεδομένων είναι `int`, `float`, `bool`, `str` (ακέραιοι, πραγματικοί, δυαδικοί με τιμές `True` ή `False` και συμβολοσειρές). Έτσι για να το ξέρεις, υποστηρίζονται εγγενώς μιγαδικοί αριθμοί (`complex`).

```
count = 77
goldRatio = 1.618
isValid = False
myName = 'Γιάννης'
print(type(myName))
# δίνει <class 'str'>
```

```
a, b, c = 1, 2, 3 # πολλαπλή ανάθεση
a, b = b, a       # αντιμετάθεση τιμών
r = s = t = 3.14  # συνεχόμενη ανάθεση
```

Μερικές ενδιαφέρουσες ιδιαιτερότητες της Python στη σύνταξη της ανάθεσης.

Τα σχόλια εισάγονται με # και διαρκούν μέχρι το τέλος της γραμμής.

Σε εφαρμογές γραμμής διαταγών (command line) μπορείς να δώσεις τιμή σε μεταβλητή με τη συνάρτηση `input()` που επιστρέφει συμβολοσειρά, γι' αυτό χρησιμοποιείς την κατάλληλη συνάρτηση μετατροπής. BTW, όλες οι μεταβλητές στην Python είναι αντικείμενα, όπως αποκαλύπτει η συνάρτηση `type()`.

```
n = int(input('Δώσε ακέραιο: '))  # δίνει πχ -43
x = float(input('Δώσε αριθμό: '))  # δίνει πχ 2.1
print(type(n))  # δίνει <class 'int'>
```

Τα ονόματα των μεταβλητών είθισται να ακολουθούν camelCase, π.χ. `windowBackColor`, σαν τη ράχη μιας καμήλας με καμπούρες. Η εμβέλεια (scope) των ονομάτων είναι το μπλοκ εντολών στο οποίο ορίζονται και η ζωή τους διαρκεί μέχρι την έξοδο της ροής εκτέλεσης από αυτό.

4 Τελεστές και παραστάσεις

```
a, b = 4, 8
if a == b:
    print('ίσοι')
a += 2            # δίνει 6
dinami = 2 ** 3   # δίνει 8
c = 3 / 2         # δίνει 1.5
phliko = 7 // 5   # δίνει 1
ypoloipo = 7 % 5 # δίνει 2
s = 'γεια' + ' ' + 'χαρά'
```

Ο έλεγχος για ισότητα γίνεται με == και για ανισότητα με `!=`
Είναι νόμιμο να γράψεις `0 <= a <= 10`

Οι λογικές πράξεις είναι τα ανθρώπινα `and`, `or`, `not` και όχι τίποτα κρυπτικά &&, ||, !

Η ύψωση σε δύναμη γίνεται με **. Δοκίμασε πράξεις σε μεγάλους αριθμούς, πχ. 2**65. Ευχάριστη έκπληξη να μην σε περιορίζουν σε γελοία μεγέθη των 32 ή 64bit.

Με / γίνεται κανονική διαίρεση ακόμη και μεταξύ ακεραίων. Το πηλίκο ακέραιας διαίρεσης είναι // και το υπόλοιπο ακέραιας διαίρεσης είναι %
Υποστηρίζονται οι τελεστές προσαύξησης `+=` και απομείωσης `-=`, αλλά όχι τα ++ και --, κρίμα.

Εφαρμοζόμενο σε συμβολοσειρές, το + κάνει συνένωση (υπερφορτωμένος τελεστής).

5 Συμβολοσειρές

```
s = 'Καλή σου μέρα'
print(s[1])     # δίνει α
print(s[5:8])   # δίνει σου
# αρνητικοί δείκτες μετράνε από το τέλος
print(s[-4:])   # δίνει μέρα
print(len(s))   # δίνει 13
```

Οι συμβολοσειρές (strings) στην Python είναι πίνακες χαρακτήρων με δείκτη που ξεκινά από το 0. Ενδιαφέρουσα η χρήση αρνητικών δεικτών.

Προσφέρονται όλες οι συνηθισμένες συναρτήσεις[2] για χειρισμό συμβολοσειρών, πχ. `strip`, `lower`, `upper`, `capitalize`, `index`, `replace`.

6 Εντολή επιλογής `if`

```
x = 3
if x < 0:
    print('αρνητικός')
    x = -x
elif x == 0:
    print('μηδέν')
else:
    print('θετικός')
print('απόλυτη τιμή', x)
```

Να προσέξεις το : μετά τη συνθήκη του `if` και την οδόντωση που είναι υποχρεωτική, καθώς δηλώνει ποιες εντολές απαρτίζουν το μπλοκ. Ο κώδικας γίνεται πιο ευανάγνωστος και καταλαμβάνει λιγότερες γραμμές.

Αν έχεις πολλές αλληλοαποκλειόμενες επιλογές, υπάρχει η `elif`. Δεν υπάρχει switch / case που έχει διαφορετική σημασιολογία σε διάφορες γλώσσες και είναι πηγή σφαλμάτων, παρότι κομψή στις λίγες περιπτώσεις όπου ταιριάζει.

Εννοείται πως φωλιασμένες εντολές επιλογής οδοντώνονται ένα επίπεδο παραμέσα.

```
x = -101
if (y := abs(x)) > 50:
    print(y, 'εκτός ορίων')
```

Πρόσφατα μπήκε στη γλώσσα μια βολική σύνταξη := που λέγεται walrus, δηλ. θαλάσσιος ίππος, επειδή μοιάζει με τα δόντια του, και επιτρέπει την ανάθεση τιμών σε μεταβλητές από μια παράσταση – θυμίζει ανάθεση της Pascal.

[2] Για την ακρίβεια είναι μέθοδοι, αλλά αγνόησέ το μέχρι την Ενότητα 9.

7 Εντολές επανάληψης `while` και `for`

Υπάρχει η κλασική εντολή επανάληψης `while`, όπως και οι άσχημες `break` για να βγεις από βρόγχο πριν την ώρα σου και `continue` για να προχωρήσεις αμέσως στην επόμενη επανάληψη του βρόγχου προσπερνώντας ενδιάμεσες εντολές. Κατά τα γνωστά, οι εντολές του σώματος της επανάληψης είναι οδοντωμένες προς τα μέσα.

```
count, tries = 0, 5
while count < tries:
    name = input('Δώσε όνομα:')
    count = count + 1
    if name == 'Γιάννης':
        break;
print('Προσπάθειες:', count)
```

Εναλλακτικά, προσφέρεται η γνωστή `for`. Για να προσδιορίσεις το εύρος των τιμών που θα λάβει η μεταβλητή της επανάληψης χρησιμοποιείς τη συνάρτηση `range(`*αρχή*`,` *τέλος*`,` *βήμα*`)` που παράγει ακεραίους μόνον! Παρότι πιο ευανάγνωστη και καθαρή από την αντίστοιχη `while`, αυτή η χρήση της `for` είναι τραβηγμένη από τα μαλλιά. Η εντολή `for` ταιριάζει κυρίως για να διατρέχεις συλλογές (ενότητα 9).

```
for i in range(1, 10, 3):
    print(i) # δίνει 1 4 7
```

8 Συναρτήσεις

Ήδη έχεις καλέσει έτοιμες συναρτήσεις της Python με το γνωστό τρόπο τροφοδοτώντας τις με παραμέτρους και παίρνοντας την επιστρεφόμενη τιμή.

Η Python έχει πλούσιες βιβλιοθήκες. Για να χρησιμοποιήσεις συναρτήσεις από βιβλιοθήκες, τις κάνεις `import`. Όρεξη να ‘χεις να ψάχνεις στην τεκμηρίωση της γλώσσας και θα βρεις βιβλιοθήκες για τα πάντα!

```
import random
# τυχαίος αριθμός μεταξύ 1 και 9!
print(random.randrange(1,10))
```

```
from math import sqrt

def ypoteinousa(a, b):
""" υπολογίζει υποτείνουσα ορθογωνίου
τριγώνου με κάθετες πλευρές a, b
"""
    return sqrt(a * a + b * b)

print(ypoteinousa(3.0, 4.0)) # δίνει 5.0

def naFtiakso():
    pass
```

Μπορείς να ορίσεις και δικές σου συναρτήσεις ονοματίζοντάς τις με την `def` και επιστρέφοντας τιμή με τη `return`. Αφού ορίσεις μια συνάρτηση, μπορείς να την καλέσεις με το όνομά της και τις απαραίτητες τιμές παραμέτρων.

Κάτω από την επικεφαλίδα της μπορείς να γράψεις σε τριπλά εισαγωγικά τεκμηρίωση. Πιο άσχημο δεν γίνεται – σύγκρινε με Javadoc!

Ονόματα μέσα σε μια συνάρτηση είναι τοπικά (local) και ισχύουν μόνο στην έκταση του σώματός της και ζουν μόνο κατά την εκτέλεσή της. Ονόματα γραμμένα χύμα, εκτός κάποιας συνάρτησης είναι καθολικά ή σφαιρικά ή παγκόσμια (global) – να αποφεύγονται.

Το σώμα μιας συνάρτησης δεν μπορεί να είναι άδειο για λόγους σύνταξης. Αν θέλεις να δηλώσεις μια συνάρτηση και δεν έχεις (ακόμη) γράψει τις εντολές της, πρέπει να βάλεις την ‘κενή’ εντολή `pass` στο σώμα της. Μετά τη ‘γεμίζεις.

Ναι, η Python υποστηρίζει αναδρομικές συναρτήσεις που καλούν τον εαυτό τους.

Κατά τον ορισμό της συνάρτησης, αν σε μια παράμετρο ορίσεις προεπιλεγμένη τιμή, μπορείς να καλέσεις τη συνάρτηση περνώντας προαιρετικά τιμή στην παράμετρο. Αν έχεις πολλές προαιρετικές παραμέτρους ή για λόγους αναγνωσιμότητας και για σιγουριά, κατά την κλήση της συνάρτησης μπορείς να ονοματίσεις τις τιμές που περνάς.

```
def fullFilename(fn, fld='./'):
    return fld + fn

print(fullFilename('a.txt', '~/mine/'))  # δίνει ~/mine/a.txt
print(fullFilename('b.txt'))             # δίνει ./b.txt
print(fullFilename(fn='c.txt', fld='../'))  # δίνει ../c.txt
```

Στην Python μια συνάρτηση μπορεί να επιστρέφει πολλές τιμές. Πολύ βολικό, μακάρι να το είχαν και άλλες γλώσσες και να μη χρειάζεται να 'πακετάρουμε' τις επιστρεφόμενες τιμές, ή -χειρότερα- να χρησιμοποιούμε return-by-reference.

Αυτό το χαρακτηριστικό κουμπώνει με την πολλαπλή ανάθεση που είδαμε στην ενότητα 3.

```
def breakNum(n):
    if n < 0:
        prosimo = '-'
    else:
        prosimo = '+'
    return prosimo, abs(n)

p, val = breakNum(-9) # δίνει p = '-' και val = 9
```

Όπως στις συναρτησιακές (functional) γλώσσες, όπως η Lisp, έτσι και στην Python, μπορείς να χειρίζεσαι μια συνάρτηση ως αντικείμενο που το αναθέτεις σε μεταβλητή, το περνάς ως παράμετρο, κ.ά.

Μπορείς ακόμη να δημιουργείς ανώνυμες συναρτήσεις με την εντολή `lambda`.

```
def double(x):
    return 2 * x

def myPrint(f, *args): # η παράμετρος είναι συνάρτηση
    print(f(*args))    # τυπική κλήση

myPrint(double, 3)           # πραγματική κλήση
myPrint(lambda x: x / 2, 9)  # ανώνυμη συνάρτηση
```

9 Αντικείμενα

Θα είμαι ειλικρινής: η Python sucks στην αντικειμενοστραφή πλευρά της[3]. Ωστόσο, δυο κουβέντες χρειάζονται γιατί ακόμη κι αν δεν ορίσεις δικές σου κλάσεις, αναγκαστικά θα χρησιμοποιήσεις τις κλάσεις που έρχονται πακεταρισμένες με τη γλώσσα, τα αντικείμενα και τις μεθόδους τους.

Είδαμε προηγουμένως τις συναρτήσεις που, αφού τις εισάγουμε, τις καλούμε τροφοδοτώντας τες με παραμέτρους. Αντίθετα, οι μέθοδοι παρότι μοιάζουν σε λειτουργία με τις συναρτήσεις εφαρμόζονται πάνω σε αντικείμενα και καλούνται με διαφορετική σύνταξη, ήτοι *αντικείμενο.μέθοδος(παράμετροι)*.

Το γεγονός ότι στην Python όλα είναι αντικείμενα (όλοι οι τύποι μεταβλητών, ακόμα και οι συναρτήσεις) περιπλέκει την κατανόηση και υποβαθμίζει την αρχιτεκτονική αισθητική της, αν και τη δουλειά σου την κάνεις μια χαρά.

```
from math import factorial
print(factorial(12345)) # συνάρτηση
# δίνει ένα μακρινάρι με 45151 ψηφία!

s = 'καλημερα'
print(len(s)) # συνάρτηση
# δίνει 8
# η upper είναι μέθοδος
print(s.upper()) # όχι upper(s)
# δίνει ΚΑΛΗΜΕΡΑ
```

10 Συλλογές

Πέρα από τους απλούς τύπους δεδομένων που διατηρούν μια τιμή, π.χ. `bool`, `int`, χρειάζονται και μεταβλητές που αποθηκεύουν πολλές τιμές.

10.1 Λίστες

Ο τύπος δεδομένων `list` μοιάζει με τους κλασικούς πίνακες (arrays) αφού διατηρεί τη σειρά εισαγωγής και τα περιεχόμενα στοιχεία μπορούν να αλλάζουν (mutable)[4]. Διαφέρει από τους κλασικούς πίνακες, καθώς τα στοιχεία μιας λίστας μπορεί να είναι ετερογενή, δηλ. ανακατεμένοι αριθμοί, συμβολοσειρές, κά. Επιπρόσθετα, η λίστα μπορεί να αλλάζει δυναμικά μέγεθος σαν ακορντεόν. Εγκλείει τις τιμές σε τετράγωνες παρενθέσεις `[]`. Υποχρεωτικά, ο δείκτης των στοιχείων ξεκινάει δυστυχώς από το 0 ως `len()`-1 κατά την παράδοση της C και όχι όπως στην Pascal.

[3] Δεν είναι μόνο που έχει απαίσια ονόματα κατασκευαστών `__init__`, που δηλώνει τις στατικές μεθόδους με εξωγλωσσική σύνταξη decorator `@staticmethod`, αλλά κυρίως ότι δεν υποστηρίζει διαβαθμισμένη πρόσβαση στις μεταβλητές αντικειμένου, πχ. `protected`, `private`, αναιρώντας ένα βασικό χαρακτηριστικό του Ο-Ο προγραμματισμού, την απόκρυψη πληροφορίας (information hiding). Ντροπή!

[4] Μια Python list υλοποιείται με δυναμικούς πίνακες αναφορών – όχι ως συνδεδεμένες λίστες.

Με τις βασικές λειτουργίες σε μια λίστα μπορώ να ανακαλέσω ένα στοιχείο της που βρίσκεται σε συγκεκριμένη θέση, ή να πάρω ένα τμήμα της που αποτελείται από στοιχεία σε ένα εύρος θέσεων.

Σε λίστες μπορώ να εφαρμόσω τον τελεστή + για συνένωση και τον * για πολλαπλασιασμό!

Επειδή η λίστα είναι απαριθμητής (iterable) μπορώ να τη διατρέξω με μια εντολή `for` και να αναζητήσω ένα στοιχείο της.

Υπάρχουν μέθοδοι που προσθέτουν, διαγράφουν, ταξινομούν, τροποποιούν, αντιστρέφουν τα περιεχόμενα μιας λίστας.

Μερικές μέθοδοι είναι μυγιάγγιχτες, πχ. αν ζητήσω το δείκτη στοιχείου που δεν υπάρχει στη λίστα, ή αν αναφερθώ σε ανύπαρκτη θέση στη λίστα, το πρόγραμμα 'χτυπάει'. Άρα επιβάλλεται πρωτύτερος έλεγχος ή να χρησιμοποιήσω εξαιρέσεις, όρα ενότητα 12.

```
fruits = ['apple', 'banana', 'pear', 'melon'] # δημιουργία
print(fruits[2])      # δίνει pear
print(fruits[1:3])    # δίνει ['banana', 'pear']
print(fruits[2:])     # δίνει ['pear', 'melon']
print(fruits[:2])     # δίνει ['apple', 'banana']
fruits[1] = 'mango'   # αλλάζει τη banana σε mango

gelio = ['χε'] * 3    # δίνει ['χε', 'χε', 'χε']
if 'pear' in fruits:  # αναζήτηση
    print('found')    # το βρίσκει
for f in fruits:  # iterator
    print(f)  # ['apple', 'mango', 'pear', 'melon']

fruits.append('grape')  # προσθέτει στο τέλος της λίστας
fruits.insert(2, 'orange')  # εισάγει στοιχείο στη θέση
fruits.extend(['watermelon', 'kiwi'])  # προσθέτει στοιχεία
# ['apple', 'mango', 'orange', 'pear', 'melon', 'grape']

del fruits[6]           # διαγράφει στοιχείο με θέση
fruits.remove('mango')  # διαγράφει το πρώτο που βρίσκει
# προσοχή: πρέπει να είναι μέσα στη λίστα, αλλιώς σφάλμα
p = fruits.index('pear') # δείκτης στοιχείου, αλλιώς σφάλμα
c = fruits.count('pear') # πόσα αχλάδια έχουν τα φρούτα;
fruits.reverse()        # αντιστρέφει
fruits[2] = 'mandarine' # αντικαθιστά

fruits.sort() # ταξινομεί κατά αύξουσα σειρά

myFruits = fruits.copy()  # αντιγράφει
```

Προφανώς τα περιεχόμενα μιας λίστας μπορεί να είναι άλλες λίστες, που δημιουργεί φωλιασμένες λίστες που μπορούν να χρησιμοποιηθούν για την αναπαράσταση ιεραρχικών δομών, πχ. δέντρων.

Λίγη προσοχή όταν αναθέτω μια λίστα σε ένα άλλο όνομα, τότε αυτό λειτουργεί ως ψευδώνυμο (alias). Για να δημιουργήσω ένα αντίγραφο της αρχικής λίστας, χρειάζεται `copy()`.

Άχρηστη πληροφορία της ημέρας: Κοντινή παραλλαγή της `list` είναι η `tuple` (πλειάδα) που εγκλείει τις τιμές σε παρενθέσεις `()` και χαρακτηριστικό της είναι ότι δεν αλλάζει (immutable), δηλαδή τα περιεχόμενά της τίθενται κατά τη δημιουργία της και μετά δεν αλλάζουν. Οι πλειάδες είναι πιο γρήγορες σε ταχύτητα και πιο οικονομικές σε χώρο από τις λίστες, αλλά αν σε ένοιαζε η ταχύτητα, θα έγραφες C και όχι Python.

Για περισσότερες πληροφορίες και παραδείγματα, Python Lists (w3schools.com), ή Python - Lists (tutorialspoint.com).

10.2 Σύνολα

Το set εγκλείει τις τιμές των στοιχείων του σε {}.

Όπως και στα σύνολα των μαθηματικών, στα σύνολα της Python δεν επιτρέπονται διπλότυπες τιμές και η σειρά των στοιχείων δεν έχει σημασία αφού δεν είναι δεικτοδοτημένα, σαν τις λίστες.

```
fruits = {"apple", "pear", "pear"} # {'apple', 'pear'} χωρίς διπλότυπα
emptySet = set() # κενό σύνολο δίχως στοιχεία. Όχι emptySet = {}
print("banana" in fruits) ### False
fruits.add("grape") # {'grape', 'apple', 'pear'} προσθήκη στοιχείου
fruits.discard("apple") # {'grape', 'pear'} αφαίρεση στοιχείου
len(fruits) # 4 πληθάριθμος συνόλου

print({'a', 'b'}.issubset({'a', 'c'})) ### False υποσύνολο
print({'a', 'b'}.union({'a', 'c'})) ### {'a', 'b', 'c'} ένωση συνόλων
print({'a', 'b'}.intersection({'a', 'c'})) ### {'a'} τομή συνόλων
print({'a', 'b'}.difference({'a', 'c'})) ### {'b'} διαφορά συνόλων
```

Υποστηρίζονται όλες οι πράξεις των συνόλων που μάθαμε στο σχολείο. Για το καρτεσιανό γινόμενο συνόλων δανειζόμαστε τη συνάρτηση product από τη βιβλιοθήκη itertools.

```
import itertools
kerma = {'H', 'T'} # Heads or Tails
dyoKermata = set(itertools.product(kerma, kerma))
# δίνει {('H', 'T'), ('T', 'T'), ('H', 'H'), ('T', 'H')}
```

10.3 Ευρετήρια

Το dictionary έχει ως στοιχεία του ζευγάρια του στιλ *κλειδί* : *τιμή*. Λειτουργεί όπως ένα ευρετήριο αναζήτησης με το κλειδί: χρησιμοποιούμε ένα ευρετήριο όταν θέλουμε να ανασύρουμε μια τιμή που σχετίζεται με ένα κλειδί. Τα στοιχεία των ευρετηρίων είναι διατεταγμένα όπως στις λίστες[5], αλλά δεν επιτρέπονται διπλότυπα κλειδιά, όπως στα σύνολα.

```
tilKat = {'νίκος': '2281012345',
          'άννα': '2101234567',
          'στέλλα': '2310123456'
         }
print(tilKat['άννα'])             # δίνει 2101234567
tilKat['νίκος'] = '2107654321'    # αλλάζει τιμή
tilKat['αντώνης'] = '2251077777' # προσθέτει κλειδί
del(tilKat['άννα'])               # σβήνει κλειδί
emptyDict = {} # κατασκευάζει άδειο ευρετήριο
```

Για περισσότερες πληροφορίες και παραδείγματα, https://www.tutorialspoint.com/python/python_dictionary.htm#.

Για όλες τις δομές δεδομένων που είδαμε πρωτύτερα, μην ανησυχείς για το χώρο που καταλαμβάνουν. Γενικά στην Python, όταν η τιμή μιας μεταβλητής μείνει ορφανή, δηλαδή δεν την 'δείχνει' κάποιο όνομα, ο σκουπιδιάρης αναλαμβάνει την αποκομιδή της (garbage collection).

11 Αρχεία

Τα αρχεία αποθηκεύουν πληροφορία εκτός του χρόνου εκτέλεσης του προγράμματος. Στην απλούστερη εκδοχή μπορείς να αποθηκεύεις τιμές μεταβλητών ως χύμα κείμενο για να είναι αναγνώσιμο. Κατά την κλασική διαδικασία, ανοίγεις αρχείο για

```
import datetime
f = open('a.txt', 'w')    # άνοιξε για write (εγγραφή)
# f = open('a.txt', 'a')  # άνοιξε για append (προσθήκη)
f.write(str(datetime.datetime.now()) + ' Θυμήσου τα ψώνια')
f.close()

g = open('a.txt', 'r')    # άνοιξε για read (ανάγνωση)
message = g.read()  # διαβάζει όλο το αρχείο στη μεταβλητή
g.close()
```

[5] Τα ευρετήρια επιτρέπουν γρήγορες αναζητήσεις με βάση το κλειδί – γι' αυτό φτιάχτηκαν άλλωστε. Η ταχύτητα της αναζήτησης δεν εξαρτάται από τη σειρά του κλειδιού στο ευρετήριο, διότι υλοποιούνται με πίνακες κατακερματισμού (hash tables).

διάβασμα / εγγραφή, εκτελείς τη λειτουργία και το κλείνεις διακόπτοντας τη χρήση του.

Εναλλακτικά, μπορείς να μετατρέψεις τα δεδομένα σε μορφή JSON για να τα ανταλλάξεις με κάποιο άλλο πρόγραμμα (machine readable).

12 Άλλα

Παρότι δεν περιγράφονται εδώ, να ξέρεις ότι η Python υποστηρίζει:

Ημερομηνίες Παρότι δεν είναι ενσωματωμένος στη γλώσσα τύπος δεδομένων, οι ημερομηνίες είναι πρωτοκλασάτα αντικείμενα (`import datetime`)

Κανονικές εκφράσεις (regular expressions) για ταίριασμα προτύπων (patterns) με συμβολοσειρές (`import re`)

Κλάσεις Όλο το σετάκι του αντικειμενοστραφούς προγραμματισμού με κατασκευαστές / καταστροφείς, ιδιότητες, κληρονομικότητα, κ.τ.λ

Αρθρώματα (modules) Μπορείς να φτιάξεις τμήματα κώδικα και να τα κάνεις import, οπότε θα λειτουργούν όπως οι διανεμόμενες βιβλιοθήκες της Python.

Εξαιρέσεις (exceptions) Τίποτα το ιδιαίτερο εδώ `try` … `except` … `finally`

Αν χρειαστείς τίποτα από αυτά, απλώς αναζήτησέ το.