Μάθε Python[1]

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

Περιεχόμενα

1 1
2 Τα βασικά 1
3 Προετοιμασία 1
3.1 Εναλλακτικά περιβάλλοντα ανάπτυξης 2
4 Μεταβλητές και ανάθεση 3
5 Τελεστές 3
6 Συμβολοσειρές 3
7 Εντολή επιλογής `if` 4
8 Εντολές επανάληψης `while` και `for` 4
9 Συναρτήσεις 4
10 Αντικείμενα 5
11 Συλλογές 5
12 Αρχεία 6
13 Άλλα 6

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 το `;` στη Java)
4. Τα μπλοκ εντολών υποδεικνύονται μέσω οδόντωσης και όχι με αγκύλες `{}`
5. Διακρίνει κεφαλαία και πεζά
6. Υποστηρίζει συναρτησιακό και αντικειμενοστραφή προγραμματισμό

```
x = 'γεια'  # συμβολοσειρά
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 (σφάλμα) αρ.5
```

Ικανός λόγος για να τη συμπαθήσεις είναι να διαβάσεις το Zen of Python δίνοντας την εντολή `import this`

Πολύ χρήσιμη λειτουργία της Python είναι η συγγραφή Jupyter notebooks που προβάλλουν περιεχόμενο και τρέχουν κώδικα ζωντανά.

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.

```
>>> print(5+6)
11
```

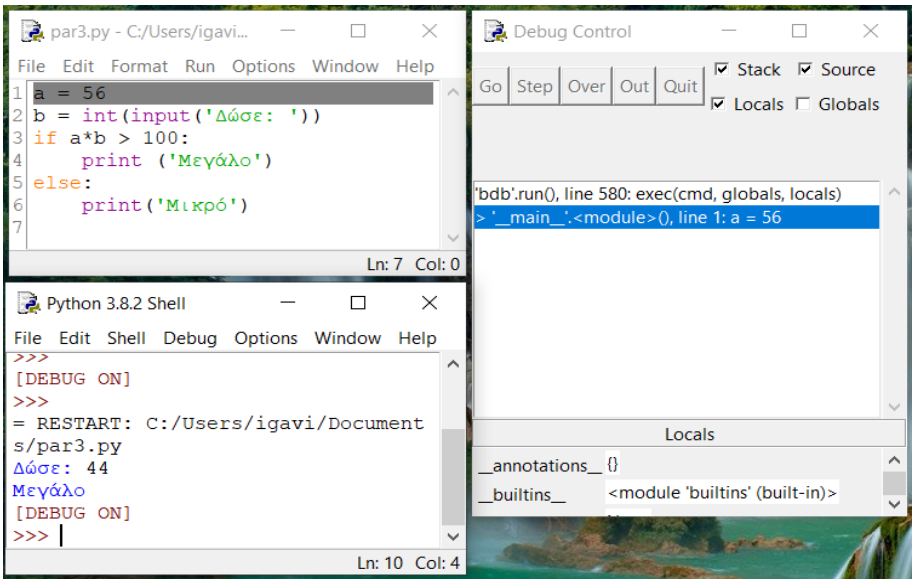
[1] Απευθύνεται σε ανθρώπους που ξέρουν προγραμματισμό – δεν είναι εισαγωγική παρουσίαση προς αρχαρίους.

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.

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 περιβάλλον ανάπτυξης (IDE), 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,

```
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:])  # πολλαπλασιασμός και κατάτμηση

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
```

αυτό είναι. Ως IDE (=Integrated Development Environment), νομίζω χτυπάει το sweet spot.

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

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

3 Μεταβλητές και ανάθεση

Οι μεταβλητές δημιουργούνται με την εντολή ανάθεσης και λαμβάνουν τον τύπο της τιμής που τους ανατίθεται. 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  # ψευδώνυμα (aliases)
```

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`. Η εμβέλεια (scope) των ονομάτων είναι το μπλοκ στο οποίο ορίζονται και η ζωή τους διαρκεί μέχρι την έξοδο της ροής εκτέλεσης από αυτό.

4 Τελεστές

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 **
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 --

```
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'
```

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

5 Συμβολοσειρές

Οι συμβολοσειρές (strings) στην Python είναι πίνακες χαρακτήρων με δείκτη που ξεκινά από το 0.

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

```
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
```

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

6 Εντολή επιλογής `if`

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.

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

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.

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 έκφραση – θυμίζει ανάθεση στην Pascal.

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

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.

Μπορείς να ορίσεις και δικές σου συναρτήσεις ονοματίζοντάς τις με την `def` και επιστρέφοντας τιμή με τη `return`.

```
from math import sqrt

def ypoteinousa(a, b):
    return sqrt(a * a + b * b)

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

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

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

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, μπορείς να ονοματίσεις τις τιμές που περνάς κατά την κλήση της συνάρτησης.

```
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
```

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

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

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.

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 γρήγορα.

```
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.

Ο τύπος δεδομένων `list` μοιάζει με τους κλασικούς πίνακες (arrays) αφού διατηρεί τη σειρά εισαγωγής και τα περιεχόμενα στοιχεία μπορούν να αλλάζουν (mutable). Επιπρόσθετα η λίστα μπορεί να αλλάζει δυναμικά μέγεθος σαν ακορντεόν. Εγκλείει τις τιμές σε τετράγωνες παρενθέσεις [ ]. Ο δείκτης των στοιχείων ξεκινάει δυστυχώς από το 0 ως `len()`-1 κατά την παράδοση της C και όχι όπως της Pascal. Οι λειτουργίες πάνω σε συλλογές είναι μέθοδοι.

```
fruits = ['apple', 'banana', 'pear', 'melon']
print(fruits[2])    # δίνει pear
print(fruits[1:3])  # δίνει ['banana', 'pear']
print(fruits[2:])   # δίνει ['pear', 'melon']
fruits[1] = 'mango' # αλλάζει τη banana σε mango
for f in fruits:      # iterator
    print(f)
if 'pear' in fruits:  # αναζήτηση
    print('found')
fruits.insert(2, 'orange')  # μακραίνει τη λίστα
fruits.remove('pear')       # κονταίνει τη λίστα
```

[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!

Κοντινή παραλλαγή της `list` είναι η `tuple` που εγκλείει τις τιμές σε παρενθέσεις `()` και χαρακτηριστικό της είναι ότι δεν αλλάζει (immutable).

Το `set` εγκλείει τις τιμές των στοιχείων του σε `{}`. Όπως και στα σύνολα των μαθηματικών, στα `set` της Python δεν επιτρέπονται διπλότυπες τιμές και η σειρά των στοιχείων δεν έχει σημασία αφού δεν είναι δεικτοδοτημένα.

Τέλος το `dictionary` λειτουργεί όπως και η `list`, αλλά περιέχει ζευγάρια του στιλ *κλειδί* : *τιμή*. Λειτουργεί όπως ένα ευρετήριο αναζήτησης με το κλειδί.

Αν η τιμή μιας μεταβλητής μείνει ορφανή, δηλαδή δεν την 'δείχνει' κάποιο όνομα, ο σκουπιδιάρης αναλαμβάνει την αποκομιδή της (garbage collection).

```
tilKat = {'νίκος': '2281012345',
          'άννα': '2101234567',
          'στέλλα': '2310123456'
          }
print(tilKat['άννα'])             # δίνει 2101234567
tilKat['νίκος'] = '2107654321'    # αλλάζει τιμή
tilKat['αντώνης'] = '2251077777' # προσθέτει εγγραφή
```

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

```
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()
```

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.