

Γλώσσες Προγραμματισμού Ι

Παραδόσεις και εργαστήριο 10/5/2018 και 17/5/2018, Νίκος Παπασπύρου.

Εισαγωγή στις γλώσσες σεναρίων (scripting languages)

Βλ. π.χ. Κεφάλαιο 13 του βιβλίου *Πραγματολογία των Γλωσσών Προγραμματισμού* του Michael L. Scott.

- Κοινά χαρακτηριστικά
 - Χρήση μαζική (batch) αλλά και διαδραστική (interactive)
 - Οικονομία έκφρασης
 - Απουσία δηλώσεων, απλοί κανόνες εμβέλειας
 - Ευέλικτο, δυναμικό σύστημα τύπων
 - Εύκολη πρόσβαση σε άλλα προγράμματα
 - Πολύπλοκο ταίριασμα προτύπων (pattern matching) και επεξεργασία συμβολοσειρών
 - Τύποι δεδομένων υψηλού επιπέδου
- Περιοχές εφαρμογών
 - Γλώσσες εντολών φλοιού (shell command languages)
 - * {c, tc, k, ba, z}?sh — δηλαδή sh, csh, tcsh, ksh, bash, zsh
 - Επεξεργασία κειμένου και παραγωγή εκθέσεων
 - * sed, awk, Perl
 - Μαθηματικά και στατιστική
 - * “3M” (Maple, Mathematica, Matlab), S, R
 - Γλώσσες “συγκόλλησης” (glue) και σενάρια γενικού σκοπού
 - * Tcl, Python, Ruby
 - Γλώσσες επέκτασης (extension languages)
 - * JavaScript, Visual Basic, AppleScript, Emacs Lisp
 - Σενάρια στον παγκόσμιο ιστό
 - * CGI (Perl, ...)
 - * Server-side embedded (PHP, Visual Basic, ...)
 - * Client-side (JavaScript, ...)
 - * Applets (Java, ...)
 - * διάφορες άλλες τεχνολογίες (XSLT, XML, ...)

Εισαγωγή στην Python 3

1. Γεια σου κόσμο!

```
print("Hello world!")
```

2. Μαθαίνω να μετρώ

- Από 0 έως και 9

```
for i in range(10):  
    print(i)
```

- Από 17 έως και 41

```
for i in range(17, 42):  
    print(i)
```

- Από 17 έως και 41 αλλά μόνο τους περιττούς:

```
for i in range(17, 42):
    if i % 2 == 1:
        print(i)
```

- Από 17 έως και 41 με βήμα 2 (πάλι μόνο τους περιττούς)

```
for i in range(17, 42, 2):
    print(i)
```

3. Η στοίχιση (indentation) είναι σημαντική!

```
for i in range(17, 42):
    if i % 2 == 1:
        print(i)
        print("just printed an odd number...")
```

```
for i in range(17, 42):
    if i % 2 == 1:
        print(i)
    print("still counting numbers...")
```

```
for i in range(17, 42):
    if i % 2 == 1:
        print(i)
print("done counting...")
```

4. Συμβολοσειρές

```
print("In double quotes")
print('or in single quotes')
print("""Multiline strings
      You must try this!""")
print('''One of the simplest programs in Python 3 is:
      print("Hello world!")''')
```

5. Σχόλια

- Μίας γραμμής:

```
# this is a comment
n = 42 # and so is this
```

- Πολλών γραμμών δεν υπάρχουν, αλλά οι συμβολοσειρές δουλεύουν και ως σχόλια:

```
"""This string serves the purpose of
a multiline comment in the program that follows"""
print("Hello world!")
```

6. “Εκτελέσιμα” προγράμματα Python

```
#!/usr/bin/env python3

print("Hello world!")
```

και φροντίστε να είναι εκτελέσιμο το αρχείο όπου θα αποθηκεύσετε το script (chmod +x σε Unix/Linux).

Μεταβλητές

1. Untyped (με δυναμικούς τύπους) και χωρίς δηλώσεις μεταβλητών:

```
a = 42
print(a+1)
a = True
print(a+1)
```

2. Πολλαπλή ανάθεση όπως στη C/C++:

```
a = b = c = 0
print(a, b, c)
```

3. Πολλαπλή ανάθεση με περισσότερες μεταβλητές στο αριστερό και στο δεξιό μέλος:

```
a, b, c = 1, 2, 3
print(a, b, c)
```

4. Αντιμετάθεση (swap) στην Python

```
a, b = 17, 42
a, b = b, a      # this indeed works!
print(a, b)
```

Συναρτήσεις

1. Υπολογισμός του n-οστού όρου της ακολουθίας Fibonacci (1, 1, 2, 3, 5, ...)

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        c = a+b
        a, b = b, c
    return b

print(fib(5))          # prints 8

print(fib(1000))      # prints a huge number with 209 digits
print(len(str(fib(100000)))) # prints 20899, the number of digits
                           # of the 100000th Fibonacci number
```

2. Υπολογισμός του μικρότερου αριθμού Fibonacci που είναι μεγαλύτερος του n:

```
def fib(n):
    a, b = 0, 1
    while b <= n:
        c = a+b
        a, b = b, c
    return b

print(fib(42))
```

3. Οι παράμετροι έχουν δυναμικούς τύπους. Αν μια συνάρτηση θέλει να ελέγξει τους τύπους των παραμέτρων της, μπορεί να το κάνει π.χ. με τη συνάρτηση `type`. Οι συναρτήσεις `int` και `str` μετατρέπουν σε ακέραιο και `string` αντίστοιχα.

```

def fib(n):
    convert = False
    if type(n) == str:
        n = int(n)
        convert = True
    a, b = 0, 1
    while b <= n:
        c = a+b
        a, b = b, c
    if convert:
        return str(b)
    else:
        return b

print(fib(42))    # the result is the number 55
print(fib("42")) # the result is the string "55"

```

4. Το τελευταίο if στο παραπάνω πρόγραμμα μπορεί να γραφεί:

```
return str(b) if convert else b
```

που είναι αντίστοιχο του τελεστή a ? b : c στη C/C++. Στην Python θα το γράφαμε b if a else c.

5. Default τιμές παραμέτρων

```

def fib(n, start_a=0, start_b=1):
    a, b = start_a, start_b
    while b <= n:
        c = a+b
        a, b = b, c
    return b

print(fib(42))
print(fib(1000, 17, 42))
print(fib(1000, start_b=42, start_a=17))

```

6. Εμβέλεια μεταβλητών

```

a = 42          # this a is global

def fib(n):
    a, b = 0, 1 # this a is local
    while b <= n:
        c = a+b
        a, b = b, c
    return b

print(fib(a))
print(a)      # prints 42

```

7. Εμβέλεια με global μεταβλητές

```

a = 42          # this a is global

def reasonable():
    print(a)

def what():
    a = 17
    print(a)

```

```

reasonable()      # prints 42
print(a)          # prints 42
what()            # prints 17
print(a)          # prints 42, what???
```

Η διαφορά βρίσκεται στο ότι η συνάρτηση `reasonable` δεν αναθέτει στη μεταβλητή `a`, ενώ η `what` αναθέτει. Η ανάθεση κάνει την Python να πιστεύει ότι η μεταβλητή πρέπει να είναι `local` στη `what`. Αυτό μπορεί να διορθωθεί με χρήση του `global`:

```

a = 42            # this a is global

def what():
    global a
    a = 17
    print(a)

what()            # prints 17
print(a)          # prints 17
```

Συμβολοσειρές

1. Αποτελούνται από χαρακτήρες

```

s = "hello world"
print(s[4])        # prints 'o'
```

2. Τα περιεχόμενά τους δεν αλλάζουν (strings are immutable)

```

s[4] = 'x'         # ERROR!
```

αλλά αυτό είναι OK (φτιάχνει νέο string):

```

s = s[:4] + "x" + s[5:]
print(s)
```

3. Το παραπάνω λάθος (εξαίρεση, μπορεί κανείς να το “πιάσει”):

```

try:
    s[4] = 'x'
    print(s)
except TypeError:
    print("strings are immutable!")
```

4. “Φέτες” (slices)

```

print(s[:4])      # "hell"      --- from start until 4
print(s[4:])      # "o world"   --- from 4 until end
print(s[4:8])     # "o wo"     --- from 4 until 8
print(s[::2])     # "hlowrd"   --- every second char (step = 2)
print(s[::-1])   # "dlrow olleh" --- in reverse (step = -1)

def palindrome(s):
    return s == s[::-1] # that was quick...
```

Πλειάδες (tuples)

Αντιπαράβαλε με τα παραπάνω (strings):

1. Αποτελούνται από οτιδήποτε

```
s = (1, 2, "what", 3)
print(s[1])           # prints 2

s = tuple("hello world") # ('h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd')
print(s[4])           # prints 'o'
```

2. Τα περιεχόμενά τους δεν αλλάζουν (tuples are immutable)

```
s[4] = 'x'           # ERROR!
```

αλλά αυτό είναι OK (φτιάχνει νέο tuple):

```
s = s[:4] + ("x",) + s[5:]
print(s)
```

Προσέξτε το κόμμα στο ("x",) που ορίζει ένα tuple με μήκος 1.

3. Το παραπάνω λάθος (εξαιρέση, μπορεί κανείς να το “πιάσει”):

```
try:
    s[4] = 'x'
    print(s)
except TypeError:
    print("tuples are immutable!")
```

4. “Φέτες” (slices)

```
print(s[:4])         # ('h', 'e', 'l', 'l')
print(s[4:])         # ('o', ' ', 'w', 'o', 'r', 'l', 'd')
print(s[4:8])        # ('o', ' ', 'w', 'o')
print(s[::2])        # ('h', 'l', 'o', 'w', 'r', 'd')
print(s[::-1])       # ('d', 'l', 'r', 'o', 'w', ' ', 'o', 'l', 'l', 'e', 'h')
```

Λίστες (lists)

Αντιπαράβαλε με τα παραπάνω (strings και tuples):

1. Αποτελούνται από χαρακτήρες

```
s = [1, 2, "what", 3]
print(s[1])           # prints 2

s = list("hello world") # ['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
print(s[4])           # prints 'o'
```

2. Τα περιεχόμενά τους αλλάζουν (lists are mutable)

```
s[4] = 'x'           # OK!
print(s)
```

και αυτό είναι OK αλλά φτιάχνει νέα λίστα:

```
s = s[:4] + ["x"] + s[5:]
print(s)
```

3. “Φέτες” (slices)

```
print(s[:4])      # ['h', 'e', 'l', 'l']
print(s[4:])     # ['o', ' ', 'w', 'o', 'r', 'l', 'd']
print(s[4:8])    # ['o', ' ', 'w', 'o']
print(s[::2])    # ['h', 'l', 'o', 'w', 'r', 'd']
print(s[::-1])  # ['d', 'l', 'r', 'o', 'w', ' ', 'o', 'l', 'l', 'e', 'h']
```

4. Μετατροπή λίστας σε string (συνένωση πολλών strings)

```
print("".join(s)) # "hello world"
print("-".join(s)) # "h-e-l-l-o- -w-o-r-l-d"
```

Διάβασμα από το standard input

1. Συμβολοσειρά από μία γραμμή

```
name = input()
print("Your name is:", name)
```

2. Ακέραιο από μία γραμμή

```
n = int(input())
print("Your number was:", n)
```

3. Δύο λέξεις από μία γραμμή

```
first, last = input().split()
print("Your first name is", first, "and your last name is", last)
```

Γεννήτριες (generators)

1. List comprehensions

```
b = [i*i for i in range(10)]
print(b) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
b = [i*i for i in range(10) if i*i%2==1]
print(b) # [1, 9, 25, 49, 81]
```

2. Διάβασμα δύο αριθμών από μία γραμμή, ξανά

```
[n, m] = [int(word) for word in input().split()]
```

ισοδύναμα

```
n, m = [int(word) for word in input().split()]
```

ή καλύτερα — η `map` είναι σαν την `map` της ML

```
n, m = map(int, input().split())
```

3. Η έκφραση `i*i for i in range(10)` παραπάνω είναι μία γεννήτρια (generator), όπως και το ίδιο το `range(10)`. Γεννάει μία ακολουθία από τιμές, τις οποίες μπορούμε να τοποθετήσουμε σε μία λίστα ή άλλη δομή δεδομένων, να τις διατρέξουμε με ένα `for loop`, κ.λπ. Η γεννήτρια αυτή αντιστοιχεί στο μαθηματικό σύνολο $\{i^2 \mid i \in \{0..9\}\}$.

4. Μπορεί κανείς να φτιάξει μία δική του γεννήτρια, ορίζοντας μία συνάρτηση που αντί για return κάνει yield τις τιμές που γεννάει:

```
def fib(n):
    a, b = 0, 1
    yield a
    while b <= n:
        yield b
        c = a+b
        a, b = b, c

for i in fib(42):
    print(i)          # prints 0, 1, 1, 2, 3, 5, 8, 13, 21, 34
```

5. Οι τιμές μίας γεννήτριας δεν αποθηκεύονται κάπου. Καταναλώνονται όπως γεννιούνται. Μπορούμε επομένως να ορίσουμε άπειρες γεννήτριες:

```
def fib():
    a, b = 0, 1
    yield a
    while True:
        yield b
        c = a+b
        a, b = b, c

for i in fib():
    print(i)
    if i > 10**6: break    # prints 0, 1, 1, 2, ... 832040, 1346269
```

Λεξικά (dictionaries)

1. Γνωστά και ως συσχετιστικοί πίνακες (associative arrays). Είναι ουσιαστικά arrays που τα στοιχεία τους προσπελάζονται μέσω κλειδιών (keys) οποιουδήποτε τύπου — όχι μόνο με ακέραιους όπως οι συμβολοσειρές, οι πλειάδες και οι λίστες.

```
d = {}
d["yes"] = "ναί"
d["no"] = "όχι"

print(d["yes"])
```

Το παρακάτω προκαλεί εξαίρεση τύπου KeyError:

```
print(d["maybe"])
```

2. Μπορούμε να πιάσουμε την εξαίρεση και να ορίσουμε ένα λεξικό με default τιμές

```
def get(d, key):
    try:
        return d[key]
    except KeyError:
        return 0

d = {}

d[1, 2, 3] = 42          # notice how keys are tuples here!
print(get(d, (1, 2, 3)))
print(get(d, (4, 5, 6)))
```


3. Οι τύποι των κλειδιών μπορούν να είναι (σχεδόν) οτιδήποτε:

```
d = {
    "maybe": "ίσως",
    "why not?": "γιατί όχι;"
}
d[1, 2, 3] = "ναι"
d[4, 5, 6, 7] = "όχι"
```

```
print(d[4, 5, 6, 7])
```

Πρέπει όμως είτε να είναι immutable (δηλαδή όχι λίστες ή άλλα λεξικά) είτε να ορίζουν τις μεθόδους `__hash__` και `__eq__`.

4. Μπορούμε να διατρέξουμε όλα τα κλειδιά ενός λεξικού:

```
for key in d:
    print(key)
```

και να τυπώσουμε και τις τιμές:

```
for key in d:
    print(key, d[key])
```

ή καλύτερα:

```
for key, value in d.items():
    print(key, value)
```

Σύνολα

1. Αναπαριστούν μαθηματικά σύνολα με στοιχεία (σχεδόν) οποιουδήποτε τύπου (πρέπει να πληρούν τις ίδιες προϋποθέσεις με τα κλειδιά ενός λεξικού).

```
s = set()
s.add(1)
s.add(2)
s.add(3)
print(2 in s)    # prints True
print(4 in s)    # prints False
s.remove(2)
print(2 in s)    # prints False now
```

2. Πράξεις συνόλων:

```
s1 = set([1, 2, 3])
s2 = {3, 4, 5}      # equivalent to set([3, 4, 5])

print(s1 | s2)      # union: {1, 2, 3, 4, 5}
print(s1 & s2)      # intersection: {3}
print(s1 - s2)      # difference: {1, 2}
```

3. Set comprehensions:

```
s = set(i*i for i in range(10))    # {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}
```

ή ισοδύναμα:

```
s = {i*i for i in range(10)}
```

4. Τα σύνολα (set) είναι mutable. Υπάρχει και η immutable εκδοχή τους (frozenset) που μπορούν να χρησιμοποιηθούν ως κλειδιά σε λεξικά ή ως στοιχεία σε άλλα σύνολα.

```
s1 = frozenset([1, 2, 3])    # two immutable sets
s2 = frozenset([4, 5, 6])
s = {s1, s2}                 # and a (mutable) set containing them
```

Αντικείμενα και κλάσεις

Η Python μπορεί να χρησιμοποιηθεί ως αντικειμενοστρεφής γλώσσα προγραμματισμού. Δείτε περισσότερα στη λύση της άσκησης “pistes”, παρακάτω, όπου οι καταστάσεις αναπαρίστανται ως αντικείμενα.

Η βιβλιοθήκη της Python

1. Είναι οργανωμένη σε modules. Τα χρησιμοποιούμε με εντολές `import`.
2. Documentation: <https://docs.python.org/3/library/>
3. Π.χ. το module `itertools` έχει πολλές χρήσιμες συναρτήσεις για να κατασκευάζουμε γεννήτριες:

```
import itertools

# all (6) permutations of elements 1, 2, 3
for l in itertools.permutations([1, 2, 3]):
    print(l)

# all (6) pairs of Daltons
for l in itertools.combinations(["joe", "jack", "william", "averel"], 2):
    print(l)
```

4. Μπορεί κανείς να ορίζει τα δικά του modules.

Η λύση της άσκησης “pistes”

Εκφώνηση: <https://courses.softlab.ntua.gr/pl1/2018a/Exercises/exerc18-2.pdf>

Η λύση περιέχει σχόλια χρήσιμα για την κατανόησή της. Επίσης, περιέχει αρκετές “πυθωνιές” που, αν σας εξάπτουν την περιέργεια, ίσως θα θέλατε να ψάξετε περισσότερο (όπως για παράδειγμα την ανάθεση `k, r, s, *keys = ...`).

```
#!/usr/bin/env python3

# Δύο χρήσιμοι τύποι δεδομένων από τη βιβλιοθήκη της Python:
# Counter: Συμπεριφέρεται ως λεξικό, μετρώντας πόσες φορές έχει
#          προστεθεί κάποιο στοιχείο. Ιδανικό για υλοποίηση multisets.
# deque:   Λίστα με αποδοτική προσθήκη και αφαίρεση στοιχείων και από
#          τα δύο άκρα.
from collections import Counter, deque

def solve(f):
    """
    Επιλύει το πρόβλημα "pistes", διαβάζοντας την είσοδο από το αρχείο f.
    Εκτυπώνει την απάντηση στο standard output.
    """

    # Διάβασε την είσοδο.
    N = int(f.readline())
    pista = []
    for i in range(N+1):
        k, r, s, *keys = map(int, f.readline().split())
        inkeys = Counter(keys[:k])
        outkeys = Counter(keys[k:])
        # Οι τιμές του πίνακα pista είναι πεντάδες της παρακάτω μορφής:
        pista.append((k, r, s, inkeys, outkeys))

    class state:
        """
        Κλάση που υλοποιεί μία κατάσταση, στο χώρο καταστάσεων του προβλήματος.
        """
```

Μια κατάσταση προσδιορίζεται από:

- την πίστα όπου βρισκόμαστε (id), ακέραιος
 - τα κλειδιά που έχουμε (keys), Counter με ακέραιους
 - τις πίστες που έχουμε ήδη επισκεφθεί (visited), frozenset με ακέραιους
- και επίσης περιέχει:

- τα αστέρια που έχουμε πάρει (stars), ακέραιος
- την τιμή κατακερματισμού (hash), χρησιμοποιείται προκειμένου να διατηρείται το σύνολο των καταστάσεων που έχουμε εξερευνήσει.

.....

```
def __init__(self, id, keys, stars, visited):
    """
    Κατασκευαστής.
    Αντιγράφει τις παραμέτρους και υπολογίζει την τιμή κατακερματισμού.
    """
    self.id = id
    self.keys = keys
    self.stars = stars
    self.visited = visited
    # Στον υπολογισμό της τιμής κατακερματισμού συμμετέχουν μόνο τα
    # πεδία id, visited και keys της κατάστασης.
    self.hash = hash((id, visited) + tuple(sorted(keys.items())))

def next(self):
    """
    Επιστρέφει τη λίστα των επόμενων καταστάσεων.
    """
    result = []
    for i in range(N+1):
        # Απόκλεισε τις πίστες που έχουμε ήδη επισκεφθεί.
        if i in self.visited: continue
        # Απόκλεισε τις πίστες που δεν μπορούμε να ανοίξουμε γιατί
        # μας λείπουν κάποια κλειδιά.
        if len(pista[i][3] - self.keys) > 0: continue
        # Υπολόγισε την επόμενη κατάσταση.
        keys = self.keys - pista[i][3] + pista[i][4]
        stars = self.stars + pista[i][2]
        visited = self.visited | frozenset([i])
        # Και πρόσθεσέ τη στο αποτέλεσμα.
        result.append(state(i, keys, stars, visited))
    return result

def __hash__(self):
    return self.hash

def __eq__(self, other):
    # Στον υπολογισμό ισότητας καταστάσεων συμμετέχουν μόνο τα
    # πεδία id, visited και keys. Το πλήθος των αστεριών σε
    # καταστάσεις που είναι ίσες ως προς αυτά δεν μπορεί να διαφέρει.
    return self.id == other.id and self.keys == other.keys \
        and self.visited == other.visited

# Η αρχική κατάσταση: είμαστε στην πίστα 0, έχουμε πάρει τα κλειδιά
# και τα αστέρια που δίνει, και είναι η μοναδική πίστα που έχουμε
# επισκεφθεί μέχρι τώρα.
init = state(0, pista[0][4], pista[0][2], frozenset([0]))
# Πρόσθεσέ τη στην ουρά και στο σύνολο των καταστάσεων που έχουμε εξερευνήσει.
Q = deque([init])
seen = set([init])

best = 0 # το μέγιστο ως τώρα πλήθος αστεριών
```

```

# Όσο η ουρά έχει καταστάσεις προς εξερεύνηση.
while Q:
    # Αφαίρεσε την πρώτη από αυτές και ενημέρωσε το best.
    s = Q.popleft()
    best = max(best, s.stars)
    # Για όλες τις επόμενες από αυτήν καταστάσεις...
    for t in s.next():
        # ... αν δεν τις έχεις εξερευνήσει, πρόσθεσέ τις στην ουρά και
        # σημείωσε ότι τώρα τις έχεις εξερευνήσει.
        if t not in seen:
            Q.append(t)
            seen.add(t)
# Τύπωσε το αποτέλεσμα.
print(best)

# Ψάξτε και βρείτε τι κάνει το παρακάτω if...
if __name__ == "__main__":
    import sys
    # Αν έχει δοθεί command line argument...
    if len(sys.argv) > 1:
        # ... διάβασε από αυτό το αρχείο την είσοδο.
        with open(sys.argv[1], "rt") as f:
            solve(f)
    else:
        # διαφορετικά διάβασε από το standard input.
        solve(sys.stdin)

```