

Προγραμματιστικά Εργαλεία και Τεχνολογίες για Επιστήμη Δεδομένων

Παράδοση 22/9/2020, 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 εμφάνισης
 - 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 σεναρία γενικού σκοπού
 - * Tcl, Python, Ruby
 - Γλώσσες επέκτασης (extension languages)
 - * JavaScript, Visual Basic, AppleScript, Emacs Lisp
 - Σεναρία 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

0. Documentation και Anaconda

1. Γεια σου κόσμε!

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

2. “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).

Αριθμοί, συμβολοσειρές και πράξεις

1. Ακέραιοι αριθμοί (int)

```
42
```

```
42 - 17
```

```
-1
```

```
6 * 7
```

```
42 // 17 # πηλίκο ακέραιας διαίρεσης (= 2)
```

```
42 % 17 # υπόλοιπο ακέραιας διαίρεσης (= 8)
```

```
2 ** 78 # ύψωση σε δύναμη: ακέραιοι απεριόριστου μεγέθους!
```

2. Πραγματικοί αριθμοί (float)

```

42.0      # διαφορετικό από το 42, float αντί int
3.14
-5.3
6.0 * 7   # = 42.0, όχι 42
42 / 17   # πραγματική διαίρεση (= 2.4705882352941178)
42.0 / 17.0 # το ίδιο
12.2 // 3.1 # = 3.0 (πόσες φορές χωράει το 3.1 στο 12.3)
12.2 % 3.1 # = 2.9 (και τι περισσεύει)

```

Στη πραγματικότητα, το τελευταίο δεν έχει ως αποτέλεσμα 2.9 αλλά 2.8999999999999999. Αυτό οφείλεται σε περιορισμούς στην ακρίβεια των πράξεων και αριθμητικά σφάλματα: οι υλοποιήσεις της Python δεν φημίζονται για την ακρίβεια στους αριθμητικούς υπολογισμούς μεταξύ πραγματικών αριθμών.

3. Λογικές τιμές (bool)

```

True
False
not True      # λογική άρνηση (= False)
True and False # λογική σύζευξη (= False)
False or True # λογική διάζευξη (= True)

```

Στην πραγματικότητα είναι True = 1 και False = 0.

```

1 + False     # = 1
5 * True      # = 5

```

4. Συγκρίσεις

```

42 == 17      # ισότητα (= False)
42 == 42.0    # = True (!) αλλά μη χρησιμοποιείτε == για πραγματικούς
42 != 17      # διάφορο (= True)
17 < 42       # μικρότερο (= True)
19 <= 18      # μικρότερο ή ίσο (= False)
17 > 42       # μεγαλύτερο (= False)
19 >= 18      # μεγαλύτερο ή ίσο (= True)

```

5. Συμβολοσειρές

```

"In double quotes"
'or in single quotes'

"""Multiline strings
  You must try this!"""

'''One of the simplest programs in Python 3 is:
  print("Hello world!")'''

```

6. 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!")

```

7. Μετατροπές

```

int(3.14)     # αποκοπή δεκαδικού μέρους (= 3)
int(2.718)    # = 2
float(42)     # 42.0 (!)
round(3.14)   # στρογγυλοποίηση (= 3)
round(2.718)  # = 3
str(3.14)     # = "3.14"
int("42")    # = 42

```

Υπάρχουν επίσης πολλές **built-in συναρτήσεις**, όπως οι `abs`, `len`, `max`, `min`, κ.λπ., καθώς και συναρτήσεις στο `module math`, όπως οι `floor`, `ceil`, `gcd`, κ.λπ.

Ροή ελέγχου

1. Οι εντολές κανονικά εκτελούνται με τη σειρά

```
print("First do this.")
print("Then do that.")
```

2. Η εντολή `if`

```
a = 3
if a < 10:
    print("It is a small number")
```

3. Η εντολή `print` σε περισσότερο βάθος

```
a = 3
if a < 10:
    print("The number", a, "is a small number")
```

Για να μην εκτυπώνονται κενά μεταξύ τους:

```
print(4, 2, sep="")          # prints "42"
```

ή για να εκτυπώνεται οτιδήποτε άλλο:

```
print(4, 2, sep=" and then ") # prints "4 and then 2"
```

Για να μην αλλάζει γραμμή στο τέλος:

```
print(4, end="")
print(2)          # these two print "42"
```

4. Η εντολή `for`

```
for a in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]:
    if a < 10:
        print("The number", a, "is a small number")
```

5. Η εντολή `if` με `else`

```
for a in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]:
    if a < 10:
        print("The number", a, "is a small number")
    else:
        print("But", a, "is a big number")
```

6. Μαθαίνω να μετρώ: `for` και `range`

- 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)
```

- Από 42 έως και 18 αντίστροφα με βήμα 2

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

7. 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...")
```

Μεταβλητές

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

```
a = 42
print(a+1)    # prints "43"
```

```
a = "hello"   # η ίδια μεταβλητή παίρνει τώρα τιμή άλλου τύπου!
print(len(a)) # prints "5" (the length of "hello" in characters)
```

2. Πολλαπλή ανάθεση όπως στη C/C++ και στη Java:

```
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) χωρίς βοηθητική μεταβλητή

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

Συναρτήσεις

0. Μια απλή συνάρτηση που επιστρέφει τον επόμενο αριθμό

```
def f(x):
    return x+1
```

1. Παράδειγμα: αριθμοί Fibonacci (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...)

2. Αριθμοί Fibonacci με αναδρομή: κακή ιδέα! – $O(2^n)$

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

```
print(fib(7))    # prints "13"
print(fib(37))   # prints "24157817" (but after ~20 sec.)
print(fib(100))  # will not print anything for a VERY long time!
```

3. Υπολογισμός του n-οστού όρου της ακολουθίας Fibonacci

```

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

print(fib(7))           # prints "13"
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

```

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

```

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

print(fib_gt(42))     # prints "55"

```