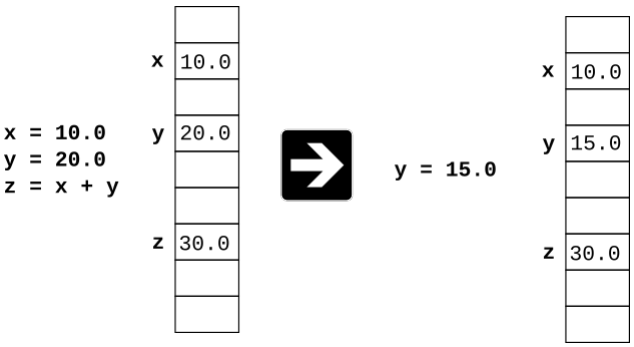
Συμβολοσειρές - Μέρος 1ο

Μιχάλης Δρακόπουλος

Μάρτιος 2021

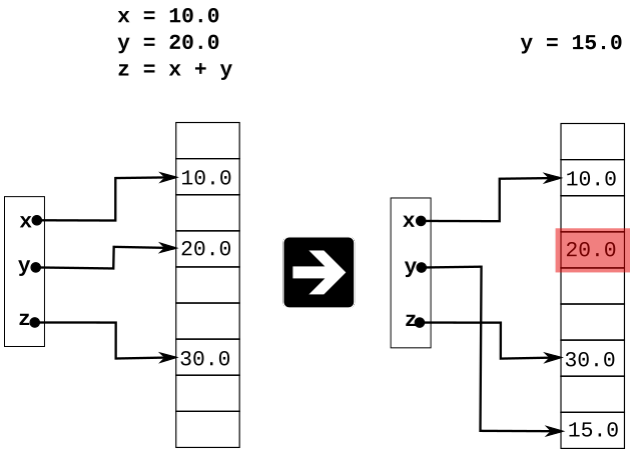
Τα αντικείμενα της Python

Σε κάποιες γλώσσες προγραμματισμού οι μεταβλητές είναι θέσης μνήμης. Κάθε μεταβλητή συνδέεται με μια συγκεκριμένη θέση μνήμης στην οποία αποθηκεύονται δεδομένα *συγκεκριμένου* τύπου



Στην Python τα δεδομένα των βασικών τύπων της, όπως είναι οι συμβολοσειρές αλλά και οι αριθμοί (ακέραιοι, πραγματικοί κ.α.) είναι **αντικείμενα**.

Στην Python οι μεταβλητές είναι **αναφορές** σε θέσεις μνήμης που αποθηκεύουν **αντικείμενα**. Δημιουργούνται όταν εκχωρηθεί κάποια τιμή (= αντικείμενο) σε αυτές



Αντικείμενα στη μνήμη, στα οποία δεν αναφέρεται κάποια μεταβλητή διαγράφονται (η μνήμη που καταλαμβάνουν ελευθερώνεται) από τον **συλλέκτη σκουπιδιών (garbage collector)**.

Η Python διαθέτει και άλλους βασικούς (= ενσωματωμένους) τύπους αντικειμένων, όπως *λίστες*, *πλειάδες*, *λεξικά*, *σύνολα* κ.π.α. Γενικά τα πάντα στην Python είναι **αντικείμενα**.

Συμβολοσειρές

Είναι βασικός τύπος της Python και παριστάνει **αλληλουχίες χαρακτήρων**.

Δημιουργούνται *άμεσα* περικλείονται *χαρακτήρες* είτε σε μονά (`'...'`) είτε σε διπλά (`"..."`) εισαγωγικά, χωρίς διαφορά ανάμεσα στις 2 αναπαραστάσεις.

[1]:
```
new = 'Hello Python'
old = "Goodbye MATLAB"
```

Μπορούν επίσης να δημιουργηθούν με τριπλά (`'''...'''`), (`"""..."""`). Στην περίπτωση αυτή μπορούν να εκτείνοντα σε παραπάνω από μια γραμμές:

[2]:
```
menu = '''
Options
=======
1) Run
2) Quit
'''
```

Εσωτερικά περιέχει και τους χαρακτήρες *νέας γραμμής* `\n`

[3]:
```
menu
```

[3]:
```
'\nOptions\n=======\n1) Run\n2) Quit\n'
```

Οι νέες γραμμές ερμηνεύονται κατά την έξοδο:

[4]:
```
print(menu)
```

```
Options
=======
1) Run
2) Quit
```

Το όνομα του τύπου για συμβολοσειρές στην Python είναι `str`

[5]:
```
type(new)
```

[5]:
```
str
```

όπως και η συνάρτηση *μετατροπής* άλλων τύπων σε συμβολοσειρές:

[6]:
```
d = str(10)
f = str(-12.53)
d, f
```

[6]:
```
('10', '-12.53')
```

Πράξεις με συμβολοσειρές

Δημιουργούν *νέα* συμβολοσειρά:

1. Συνένωση με τον τελεστή `+`

[7]:
```
'Monty' + 'Python'
```

[7]:
```
'MontyPython'
```

ΠΡΟΣΟΧΗ: και οι 2 όροι πρέπει να είναι συμβολοσειρές!

[8]:
```
'Computer Science' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-6f136489c2e8> in <module>
----> 1 'Computer Science' + 2

TypeError: can only concatenate str (not "int") to str
```

[9]:
```
'Computer Science' + str(2)
```

[9]:
```
'Computer Science2'
```

2. Επανάληψη με τον τελεστή `*`

Ο ένας όρος πρέπει να είναι τύπου `int`.

[10]:
```
'spam!' * 5
```

[10]:
```
'spam!spam!spam!spam!spam!'
```

Διαχείριση στοιχείων με δείκτες

Πλήθος στοιχείων (=χαρακτήρων) - μήκος συμβολοσειράς: η συνάρτηση `len`

[11]:
```
s = 'PYTHON'
len(s)
```

[11]:
```
6
```

Το πρώτο στοιχείο βρίσκεται στη θέση `0`

[12]:
```
s[0]
```

[12]:
```
'P'
```

Το τελευταίο στοιχείο βρίσκεται στη θέση `len(s) - 1`

[13]:
```
s[5]
```

[13]:
```
'N'
```

Οι δείκτες μπορεί να είναι και *αρνητικοί* (μετράνε από το τέλος προς την αρχή)

[14]:
```
s[-1], s[-3]
```

[14]:
```
('N', 'H')
```

Ενας αρνητικός δείκτης στην ουσία *προστίθεται* στο μήκος της λίστας και προκύπτει ο αντίστοιχος θετικός δείκτης

[15]:
```
s[-2], s[len(s)-2]
```

[15]:
```
('O', 'O')
```

Δείκτες εκτός των ορίων `0` και `len(s) - 1` δίνουν σφάλμα

[16]:
```
s[20]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-16-da1b392e7478> in <module>
----> 1 s[20]

IndexError: string index out of range
```

ΠΡΟΣΟΧΗ!!! Οι συμβολοσειρές είναι **αμετάβλητοι** τύποι! Τα στοιχεία τους δεν μπορούν να αλλάξουν!

[17]:
```
s[0] = 'J'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-d44fa6f31dd2> in <module>
----> 1 s[0] = 'J'

TypeError: 'str' object does not support item assignment
```

Τεμαχισμός συμβολοσειράς

Με τον τελεστή `:` μπορούμε να αποκόψουμε *υπο-συμβολοσειρές* μιας λίστας επιλέγοντας κατάλληλα τους δείκτες (βολεύει να θεωρήσουμε ότι δείχνουν *ανάμεσα* στους χαρακτήρες της συμβολοσειράς).

```
+---+---+---+---+---+---+
| P | Y | T | H | O | N |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
```

```
-6  -5  -4  -3  -2  -1
```

[18]:
```
s = 'PYTHON'
```

- `s[I:J]` υπο-συμβολοσειρά από τον δείκτη `I` έως **και** τον δείκτη `J-1` (η θέση `J` δεν συμπεριλαμβάνεται στην υπο-συμβολοσειρά)

[19]:
```
s[1:5]
```

[19]:
```
'YTHO'
```

[20]:
```
s[2:-1]
```

[20]:
```
'THO'
```

- `s[I:J:K]` υπο-συμβολοσειρά από `I` έως `J-1` με βήμα `K`

[21]:
```
s[1:5:2]
```

[21]:
```
'YH'
```

[22]:
```
s[4:1:-1]
```

[22]:
```
'OHT'
```

```
 +---+---+---+---+---+---+
 | P | Y | T | H | O | N |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

- Παράλειψη του `I` ισοδυναμεί με `I = 0`

[23]:
```
s[:4]
```

[23]:
```
'PYTH'
```

- Παράλειψη του `J` ισοδυναμεί με `J = len(s)`

[24]:
```
s[2:]
```

[24]:
```
'THON'
```

[25]:
```
s[1::2]
```

[25]:
```
'YHN'
```

- Παράλειψη `I` *και* `J` είναι ολόκληρη η συμβολοσειρά

[26]:
```
s[::]
```

[26]:
```
'PYTHON'
```

[28]:
```
s[:]
```

[28]:
```
'PYTHON'
```

- Αντιστροφή όλων των στοιχείων με `K < 0`

[29]:
```
s[::-1]
```

[29]:
```
'NOHTYP'
```

```
 +---+---+---+---+---+---+
 | P | Y | T | H | O | N |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Αν `I` **ή** `J` εκτός ορίων, **δεν** είναι σφάλμα!!! Παίρνουμε τη μέγιστη δυνατή υπο-συμβολοσειρά που προκύπτει!

[30]:
```
s[2:100]
```

[30]:
```
'THON'
```

[31]:
```
s[-20:4], s[-10:30]
```

[31]:
```
('PYTH', 'PYTHON')
```

- Αν τα `I`, `J`, `K` δεν επιτρέπουν τον σχηματισμό υπο-συμβολοσειράς παίρνουμε την *κενή* συμβολοσειρά

[32]:
```
s[3:3], s[5:1], s[2:5:-1]
```

[32]:
```
('', '', '')
```

Εντολές επανάληψης με συμβολοσειρές

Με την εντολή `for`, μπορούμε να διατρέξουμε τα στοιχεία μιας συμβολοσειράς με 2 τρόπους

1. Με χρήση δεικτών

[33]:
```
for i in range(len(s)):
    c = s[i]
    if c == 'Y' or c == 'O':
        c = '*'
    print(c, end='')
print()
```

```
P*TH*N
```

2. Χωρίς χρήση δεικτών

[34]:
```
for c in s:
    if c == 'Y' or c == 'O':
        c = '@'
```

```
    print(c, end='')
print()
```

```
P@TH@N
```

Προτιμητέος ο τρόπος 2, όταν δεν χρειαζόμαστε τους δείκτες για τους υπολογισμούς μας (που είναι και η συνηθέστερη περίπτωση).

Το πρωτόκολλο επανάληψης

Οι συμβολοσειρές (όπως και αρκετά άλλα αντικείμενα της Python) ακολουθούν το **πρωτόκολλο επανάληψης**. Αυτό πρακτικά σημαίνει ότι τα αντικείμενα αυτά μπορούν στα πλαίσια μιας επαναληπτικής διαδικασίας να επιστρέφουν **ένα-ένα** τα στοιχεία τους όταν τους ζητηθούν.

Αυτός είναι ο λόγος που "δουλεύει" ο τρόπος 2 παραπάνω.

Αλλά και η συνάρτηση `range()` επίσης κατασκευάζει και επιστρέφει ένα αντικείμενο που ακολουθεί και αυτό το πρωτόκολλο επανάληψης!

Σύγκριση συμβολοσειρών

Η Python ακολουθεί την *κωδικοποίηση* Unicode για την αναπαράσταση χαρακτήρων. Κάθε χαρακτήρας αντιστοιχεί σε έναν ακέραιο `i` (`0 <= i <= 1114111`).

- Χαρακτήρας σε ακέραιο κώδικα: η συνάρτηση `ord()`

[35]:
```
ord('A'), ord('B'), ord('7'), ord('8'), ord('$'), ord('\n')
```

[35]:
```
(65, 66, 55, 56, 36, 10)
```

- Ακέραιος κώδικας σε χαρακτήρα: η συνάρτηση `chr()`

[36]:
```
chr(67), chr(54), chr(37)
```

[36]:
```
('C', '6', '%')
```

Οι συνηθέστεροι *ορατοί* χαρακτήρες είναι ανάμεσα στο 32 και το 128 (ASCII $\subset$ UNICODE)

[37]:
```
print(end='  ')
for i in range(32, 129):
    print(chr(i), end=' ')
    if i%10 == 0:
        print()
```

```
    ! " # $ % & ' (
) * + , - . / 0 1 2
3 4 5 6 7 8 9 : ; <
= > ? @ A B C D E F
G H I J K L M N O P
Q R S T U V W X Y Z
[ \ ] ^ _ ` a b c d
e f g h i j k l m n
```

```
o p q r s t u v w x
y z { | } ~
```

Οι συμβολοσειρές συγκρίνονται *λεξικογραφικά* χαρακτήρα-προς-χαρακτήρα σύμφωνα με τον κώδικα των στοιχείων τους.

Η σύγκριση σταματάει στην πρώτη θέση που διαφοροποιούνται οι αντίστοιχοι χαρακτήρες των 2 συμβολοσειρών, ή στο τέλος της "κοντύτερης" από αυτές.

Χρησιμοποιούνται οι γνωστοί τελεστές σύγκρισης (==, !=, <, <=, >, >=)

[38]:
```
name='Python'
name=='Python', name=='python',name<'python', name>='PYthon', name<'Python!'
```

[38]:
```
(True, False, True, True, True)
```

Παράδειγμα Ελεγχος αν μια λέξη είναι παλινδρομική

[39]:
```
word = input('Give a word: ')
if word == word[::-1]:
    print(word, 'is palindrome')
else:
    print(word, 'is NOT palindrome')
```

```
Give a word:  AnnA
```

```
AnnA is palindrome
```