Εισαγωγή στον Προγ/μό Υπολογιστών

Διάλεξη 7

Ακολουθίες (*sequences*)

Περιεχόμενα

1. Ακολουθίες
2. Λίστα (`list`)
3. `range`
4. Επαναληπτικοί υπολογισμοί με ακολουθίες
5. Επεξεργασία ακολουθιών
6. Τελεστές ακολουθιών
7. Συμβολοσειρές (`str`)

Ακολουθίες

Ακολουθίες

- Οι ***ακολουθίες*** *(sequences*) δεν είναι ένας συγκεκριμένος *τύπος δεδομένων*, αλλά σύνολο χαρακτηριστικών που μοιράζονται πολλοί τύποι

- Κάθε τύπος που έχει αυτά τα χαρακτηριστικά θεωρείται *ακολουθία*:
  - **Μέγεθος:** Κάθε ακολουθία έχει πεπερασμένο μέγεθος (πλήθος στοιχείων). Μια κενή ακολουθία έχει μέγεθος 0
  - **Επιλογή στοιχείου**: κάθε θετικός ακέραιος μικρότερος του μεγέθους αντιστοιχεί σε στοιχείο της ακολουθίας, με το `0` να αντιστοιχεί στο πρώτο στοιχείο

- Ενσωματωμένοι τύποι της Python που είναι ακολουθίες: `list, range, tuple, str, …`

Λίστα (`list`)

Λίστα

- Παραδείγματα χρήσης `list`

```
>>> ls = [1, 3, 5, 3, 9]
>>> ls
[1, 3, 5, 3, 9]
>>> len(ls)
5
>>> ls[4]
9
>>> ls + 2*[-10, 23]
[1, 3, 5, 3, 9, -10, 23, -10, 23]
```

Λίστα

- Γενική μορφή επιλογέα στοιχείου ακολουθίας: `<ακολουθία>[i]`
  - Η τιμή του `i`-οστού στοιχείου της `<ακολουθία>`
  - 1ο στοιχείο: `<ακολουθία>[0]`
  - Τελευταίο στοιχείο: `<ακολουθία>[ len(<ακολουθία>) - 1 ]`
  - Το `i` λέγεται ***δείκτης*** (*index*) και αντιπροσωπεύει απόσταση από την πρώτη θέση
  - Ισοδύναμο με την κλήση `getitem(<ακολουθία>, i)`

- Αρνητικοί δείκτες αντιπροσωπεύουν απόσταση από το τέλος, δηλαδή από την τιμή δείκτη `len(<ακολουθία>)` :

  `<ακολουθία>[-i] == <ακολουθία>[len(<ακολουθία>)-i]` για `i>0`

```
>>> x = [1, 3, 5, 9, 19]
>>> x[4]
19
>>> x[-1]
19
>>> x[-5]
1
```

Λίστα

- Τα στοιχεία μιας λίστας μπορεί να ανήκουν σε διαφορετικούς τύπους

```
>>> ls2 = [1, 4.5, 'Hello', print, [10, 20]]
>>> ls2[2]
'Hello'
>>> ls2[3]
<built-in function print>
>>> ls2[3]('world!!!')
world!!!
>>> ls2[4]
[10, 20]
>>> ls2[4][0]
10
```

Λίστα

- Αναπαράσταση αλγεβρικών πινάκων με φωλιασμένες λίστες
- Πχ, πίνακας $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, A_{23} = 6$

```
>>> a = [[1, 2, 3], [4, 5, 6]]
>>> a[1][2]
6
```

Ακολουθία `range`

Ακολουθία `range`

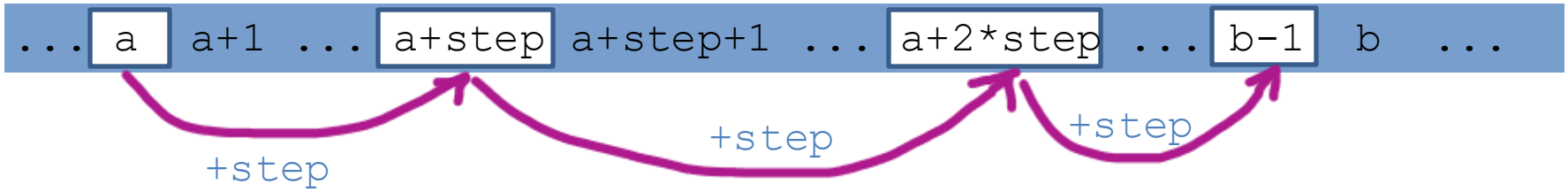
```
>>> digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> digits = range(0, 10) # ή range(10) ισοδύναμα
>>> digits[3]
3
>>> digits
range(0, 10)
>>> list(digits)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> even_digits = range(0, 10, 2)
>>> list(even_digits)
[0, 2, 4, 6, 8]
```

Ακολουθία `range`

- `range(a, b)`

- `range(b)`

- `range(a, b, step)`



(2ο στοιχείο: `a+step`, 3ο στοιχείο: `a+2*step, ...`)

Ακολουθία `range`

- ***Lazy evaluation***: η `range` σε αντίθεση με τη `list`, δεν κατασκευάζει τα στοιχεία της πριν αποτιμηθούν αυτά πχ, με `r[i]` αργότερα

```
>>> r = range(1000000000000000000)
>>> r[0]
0
>>> r[-1]
999999999999999999
>>> len(r)
1000000000000000000
>>> list(r)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
MemoryError
```

Επαναληπτικοί υπολογισμοί με ακολουθίες

Επαναληπτικοί υπολογισμοί με ακολουθίες

```
>>> def sum_list0(ls):
        i, total = 0, 0
        while i < len(ls):
            total += ls[i]
            i = i + 1
        return total

>>> def sum_list(ls):
        total = 0
        for x in ls:
            total += x
        return total

>>> s = [1, 5, -3, 20]
>>> sum_list0(s)
23
>>> sum_list(s)
23
>>> sum(s)
23
```

Επαναληπτικοί υπολογισμοί με ακολουθίες

- Γενική μορφή εντολής επανάληψης `for`:

```
for <όνομα> in <έκφραση>:
    <μπλόκ εντολών>
```

- Τιμή τύπου *iterable*, δηλαδή μπορεί να το «διατρέξει» η εντολή `for`, πχ, ακολουθίες – άλλα παραδείγματα παρακάτω

Εκτέλεση εντολής `for`:

1. *Στο τρέχον πλαίσιο το* `<όνομα>` *συνδέεται στο 1ο στοιχείο του iterable που αποτιμάται η* `<έκφραση>`*. Εάν η ακολουθία δεν έχει στοιχεία τελειώνει η εκτέλεση της* `for`
2. *Εκτελούνται όλες οι εντολές του μπλοκ*
3. *Το* `<όνομα>` *συνδέεται στο επόμενο στοιχείο του iterable και επαναλαμβάνεται το βήμα 2 έως ότου δεν υπάρχει άλλο επόμενο στοιχείο*

Επαναληπτικοί υπολογισμοί με ακολουθίες

- Όπως στη `while`, στο μπλοκ εντολών της `for` μπορούν να χρησιμοποιηθούν οι εντολές `break, continue`

- Εντολή `break`: διακόπτεται η εκτέλεση εντολών του μπλοκ και συνεχίζει η εκτέλεση από την εντολή που ακολουθεί τη `for`
  - Παράδειγμα:

```
>>> for i in range(100)
        print(i)
        if i == 2:
            break
        print('Just printed', i)
```

```
0
Just printed 0
1
Just printed 1
2
```

Επαναληπτικοί υπολογισμοί με ακολουθίες

- Εντολή `continue`: δεν εκτελούνται οι παρακάτω εντολές στο μπλοκ και η εκτέλεση συνεχίζεται με το επόμενο στοιχείο του iterable (εάν υπάρχει, αλλιώς τελειώνει η εκτέλεση της `for`)
  - Παράδειγμα:

```
>>> for i in range(5)
        if i == 2:
            continue
        print(i)

0
1
3
4
```

Επαναληπτικοί υπολογισμοί με ακολουθίες

- *Unpacking*: ανάκτηση τιμών που βρίσκονται μέσα σε φωλιασμένες λίστες

```
>>> ls = [[1, 2], [3, 4], [5, 6]]

>>> for x, y in ls:
        print(x + y)

3
7
11
```

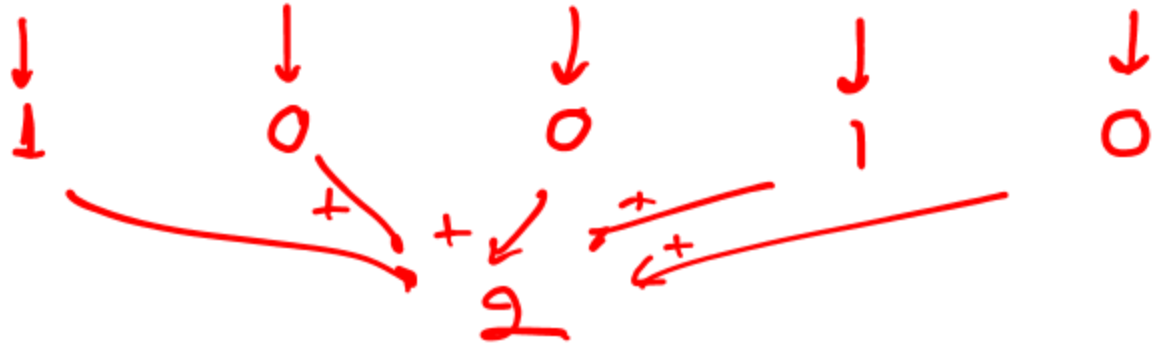
Επεξεργασία ακολουθιών (*sequence processing*)

Επεξεργασία ακολουθιών

- **Επεξεργασία ακολουθιών (*sequence processing*)**: τρόπος υπολογισμού όπου εφαρμόζονται διαδοχικά στάδια επεξεργασίας σε μια ακολουθία

- Παράδειγμα: πλήθος `'apple'` στο καλάθι:

```
basket = ['apple', 'orange', 'grapes', 'apple', 'banana']
```



Τύπος επεξεργασίας 1:
Απεικόνιση (*mapping*): κάθε στοιχείο της ακολουθίας αντιστοιχίζεται σε τιμή μέσω κάποιας συνάρτησης

Τύπος επεξεργασίας 2:
Συγκέντρωση (*reduction*): εφαρμογή ενός τελεστή, πχ. `+`, `*`, `max`, σε όλα τα στοιχεία της ακολουθίας

Επεξεργασία ακολουθιών

- **Επεξεργασία ακολουθιών (sequence processing)**: τρόπος υπολογισμού όπου εφαρμόζονται διαδοχικά στάδια επεξεργασίας σε μια ακολουθία

- Παράδειγμα: εύρεση όλων των *πρώτων αριθμών* μικρότερων του 20

  … με "κόσκινο του Ερατοσθένη":

[2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, 17, ~~18~~, 19, ~~20~~]

[2, 3, 5, 7, ~~9~~, 11, 13, ~~15~~, 17, 19]

[2, 3, 5, 7, 11, 13, 17, 19]

[2, 3, 5, 7, 11, 13, 17, 19]

...

[2, 3, 5, 7, 11, 13, 17, 19]

Τύπος επεξεργασίας 3:
Φιλτράρισμα (*filtering*): κατασκευή νέας ακολουθίας με μόνο τα στοιχεία που ικανοποιούν κάποια συνθήκη

Επεξεργασία ακολουθιών

- Η Python διαθέτει δύο τρόπους για επεξεργασία ακολουθιών:

  1. ***List comprehensions*:** *εκφράσεις* που αποτιμώνται σε λίστα. Δίδονται με ειδικό συντακτικό που μοιάζει με αυτό των εντολών `for` και `if` και υλοποιούν *απεικόνιση* και *φιλτράρισμα*

  2. Με τις ενσωματωμένες *συναρτήσεις υψηλότερου επιπέδου* `map`, `reduce`, `filter`

Επεξεργασία ακολουθιών

- Απεικόνιση στοιχείων (*mapping*) με list comprehension

```
>>> numbers = [1, -43, 8, 3, 4]
>>> increased_numbers = [x + 1 for x in numbers]
>>> increased_numbers
[2, -42, 9, 4, 5]

>>> [pow(x, 2) for x in increased_numbers]
[4, 1764, 81, 16, 25]
```

Επεξεργασία ακολουθιών

- Φιλτράρισμα στοιχείων (*filtering*) με list comprehension

```
>>> numbers
[1, -43, 8, 3, 4]
>>> [x+1 for x in numbers if x >= 0]
[2, 9, 4, 5]

>>> is_odd = lambda x: x % 2 != 0
>>> [x for x in numbers if is_odd(x)]
[1, -43, 3]
```

Επεξεργασία ακολουθιών

- Γενική μορφή ***list comprehension***:

```
[ <έκφραση> for <όνομα> in <iterable> if <έκφραση_if> ]
```

- Έκφραση με τιμή τύπου `list`

- Ισοδύναμη με την τιμή του `result` μετά τις εντολές:

```
result = []
for <όνομα> in <iterable>:
    if <έκφραση_if>:
        result = result + [ <έκφραση> ]
```

Επεξεργασία ακολουθιών

- Στις *επαναληπτικές εντολές* (`for`, `while`) περιγράφεται η σειρά πραγματοποίησης των υπολογισμών

- Στα *comprehensions* δεν καθορίζεται η σειρά
  - Ο διερμηνευτής θα μπορούσε να εκτελέσει το comprehension παράλληλα σε πολλούς επεξεργαστές για μεγαλύτερη ταχύτητα!

Επεξεργασία ακολουθιών

- Συγκέντρωση (*reduction*): το αποτέλεσμα εφαρμογής ενός τελεστή σε όλα τα στοιχεία μιας ακολουθίας

- Ενσωματωμένες συναρτήσεις συγκέντρωσης `sum, max, min`

```
>>> sum([1, 4, -6, 9, -10])
-2
>>> max([1, 4, -6, 9, -10])
9
>>> min([1, 4, -6, 9, -10])
-10
```

Επεξεργασία ακολουθιών

- Μέτρηση φορών που εμφανίζεται η τιμή `'apple'` στο `basket`:

```
>>> basket = ['apple','orange','grapes','apple','banana']

>>> isapple = lambda x: x == 'apple'
>>> sum([1 for x in ls if isapple(x)])
2
>>> len([x for x in ls if isapple(x)])
2
>>> sum([1 if isapple(x) else 0 for x in ls])
2
>>> sum([isapple(x) for x in ls])
2
```

Επεξεργασία ακολουθιών

- Παράδειγμα: κατασκευή λίστας *τέλειων αριθμών* μικρότερων του 1000
- *Τέλειος αριθμός*: είναι ίσος με το άθροισμα των διαιρετών του
  - Πχ, 6 = 1 + 2 + 3, 28 = 1 + 2 + 4 + 7 + 14

```
>>> def divisors(n):
        return [x for x in range(1, n) if n % x == 0]

>>> [n for n in range(1, 1000) if n == sum(divisors(n))]
[6, 28, 496]
```

Επεξεργασία ακολουθιών

- Φωλιασμένα `for` σε *list comprehensions*

```
>>> fruits = ['apple', 'orange', 'peach']
>>> cream = ['vanilla', 'chocolate']
>>> combos = [x+' '+y for x in fruits for y in cream]
>>> combos
['apple vanilla', 'apple chocolate', 'orange vanilla',
'orange chocolate', 'peach vanilla', 'peach chocolate']

>>> [x+' '+y for x in fruits if x != 'peach' \
        for y in cream if [x,y] != ['apple', 'chocolate']]
['apple vanilla', 'orange vanilla', 'orange chocolate']
```

Επεξεργασία ακολουθιών

```
[ <έκφραση> for <όνομα1> in <iterable1> if <έκφραση_if1> \
            for <όνομα2> in <iterable2> if <έκφραση_if2> \
                        ...
              for <όνομαN> in <iterableN> if <έκφραση_ifN> ]
```

- Ισοδύναμο με την τιμή της `newlist` μετά τις εντολές:

```
newlist = []
for <όνομα1> in <iterable1>:
    if <έκφραση_if1>:
        for <όνομα2> in <iterable2>:
            if <έκφραση_if2>:
                ...
                 for <όνομαN> in <iterableN>:
                     if <έκφραση_ifN>:
                          newlist = newlist + [<έκφραση>]
```

- Το `if` που ακολουθεί το `for` είναι προαιρετικό

Επεξεργασία ακολουθιών

- Επεξεργασία ακολουθιών με τις *συναρτήσεις υψηλότερου επιπέδου*:

  `map``(func, iterable)`: *απεικόνιση* στοιχείων `iterable` με τη συνάρτηση `func`

  `filter``(func, iterable)`: *φιλτράρισμα* στοιχείων `x` του `iterable` όπου `func(x)` είναι `True`

  `reduce``(func, iterable)`: υπολογισμός τιμής εφαρμογής της συνάρτησης `func` μεταξύ όλων των στοιχείων του `iterable`
    - Απαιτεί `from functools import reduce`

Επεξεργασία ακολουθιών

- Απεικόνιση με τη συνάρτηση `map(<έκφραση>, <iterable>)`
  - Η `<έκφραση>` έχει ως τιμή μια συνάρτηση μιας παραμέτρου

- Παράδειγμα:

```
>>> numbers = [1, -43, 8, 3, 4]
>>> map(lambda x: x+1, numbers)
<map object at 0x10c7dd9b0>
>>> list(map(lambda x: x+1, numbers))
[2, -42, 9, 4, 5]
```

- Ισοδύναμο με

```
>>> [x+1 for x in numbers]
[2, -42, 9, 4, 5]
```

Επεξεργασία ακολουθιών

- Η απεικόνιση κάθε στοιχείου γίνεται με ***lazy evaluation***: η απεικόνιση υπολογίζεται όταν το στοιχείο αποτιμηθεί

- Παράδειγμα:

```
>>> numbers = [1, -43, 8, 3, 4]
>>> def print_and_add_one(x):
        print('Increasing', x)
        return x + 1
>>> increased_numbers = map(print_and_add_one, numbers)
>>> list(increased_numbers)
Increasing 1
Increasing -43
Increasing 8
Increasing 3
Increasing 4
[2, -42, 9, 4, 5]
```

Επεξεργασία ακολουθιών

- Φιλτράρισμα με τη συνάρτηση `filter(<έκφραση>, <iterable>)`
  - Η `<έκφραση>` έχει ως τιμή μια συνάρτηση μιας παραμέτρου

- Παράδειγμα:

```
>>> numbers = [2, -42, 9, 4, 5]
>>> def is_positive(x):
        return x >= 0
>>> filter(is_positive, numbers)
<filter object at 0x10c7dd908>
>>> list(filter(is_positive, numbers))
[2, 9, 4, 5]
```

- Ισοδύναμο με

```
>>> [x for x in numbers if is_positive(x)]
[2, 9, 4, 5]
```

Επεξεργασία ακολουθιών

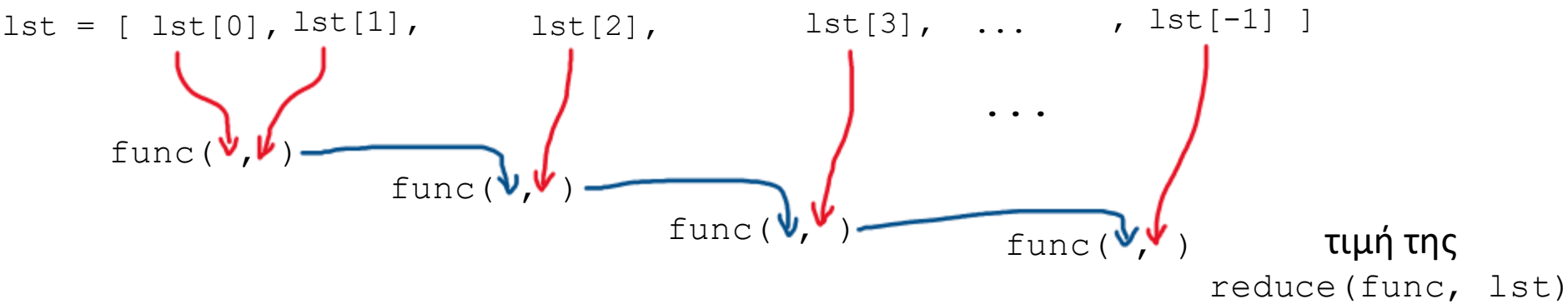
- Παράδειγμα: κατασκευή λίστας *τέλειων αριθμών* μικρότερων του 1000

```
>>> def divisors(n):
        return filter(lambda x: n % x == 0, range(1, n))

>>> list(filter(lambda n: n == sum(divisors(n)),
                range(1, 1001)))
[6, 28, 496]
```

Επεξεργασία ακολουθιών

- Συγκέντρωση με τη συνάρτηση `reduce(<έκφραση>, <iterable>)` που βρίσκεται στο module functools
  - Η `<έκφραση>` έχει ως τιμή μια συνάρτηση *δύο* παραμέτρων

- Αποτίμηση της `reduce(func, lst)`:



- Δηλ., η `reduce(func, lst)` είναι ισοδύναμη με `func(…func(func(lst[0], lst[1]), lst[2]),…lst[-1])`

Επεξεργασία ακολουθιών

- Παράδειγμα συγκέντρωσης με `reduce`

```
>>> numbers = [2, -42, 9, 4, 5]
>>> from functools import reduce
>>> reduce(max, numbers)
9
>>> reduce(lambda x, y: x + y, numbers)
-22
```

- Ισοδύναμα με

```
>>> max(numbers)
9
>>> sum(numbers)
-22
```

Τελεστές ακολουθιών

Τελεστές ακολουθιών

- Έλεγχος ιδιότητας μέλους με τελεστές `in` και `not in`:

```
>>> 5 in [2, -1, 'a', 5, 10]
True

>>> 11 not in [2, -1, 'a', 5, 10]
True
>>> not 11 in [2, -1, 'a', 5, 10]
True
```

- Γενική μορφή: `<έκφραση> in <έκφραση_συνόλου>`
  `<έκφραση> not in <έκφραση_συνόλου>`

  – Τιμή τελεστή `in` (`not in`) = `True` εάν η τιμή της `<έκφρασης>` (δεν) είναι μέλος του συνόλου (*container*) που αποτιμάται η `<έκφραση_συνόλου>`
  – *container*: όλοι οι τύποι που περιέχουν στοιχεία, πχ, ακολουθίες

Τελεστές ακολουθιών

- Τεμαχισμός (slicing)

```
>>> ls = [1, 3, 5, -4, 6, 10]
>>> ls[1:4]
[3, 5, -4]
>>> ls[1:4:2]
[3, -4]
>>> ls[:4]
[1, 3, 5, -4]
>>> ls[3:]
[-4, 6, 10]
>>> ls[:]
[1, 3, 5, -4, 6, 10]
>>> ls[3:-1]
[-4, 6]
>>> ls[3:-2]
[-4]
```

`<ακολουθία>[a:b:step]`

Μια νέα ακολουθία που αποτελείται από τα στοιχεία της `<ακολουθία>` *με δείκτες*
`a, a+step, a+2*step, ...,` **όχι πέρα του** `b-1`

προκαθορισμένες τιμές εάν παραληφθούν ή `step > 0`*:*
`a = 0, b = len(<ακολουθία>), step = 1`

Τελεστές ακολουθιών

- Αντιστροφή φοράς

```
>>> ls = [1, 3, 5, -4, 6, 10]
>>> ls[4:1]
[]
>>> ls[4:1:-1]
[6, -4, 5]
>>> ls[4:0:-1]
[6, -4, 5, 3]
>>> ls[4:-7:-1]
[6, -4, 5, 3, 1]
>>> ls[4::-1]
[6, -4, 5, 3, 1]
>>> ls[::-1]
[10, 6, -4, 5, 3, 1]
```

`<ακολουθία>[a:b:step]`

Μια νέα ακολουθία που αποτελείται από τα στοιχεία της `<ακολουθία>` *με δείκτες*
`a, a+step, a+2*step, ...,` **όχι πέρα του** `b-1`

προκαθορισμένες τιμές εάν παραληφθούν και `step < 0`*:*
`a = -1, b = -len(<ακολουθία>)-1`

Συμβολοσειρές (`str`)

Συμβολοσειρές

- Οι *συμβολοσειρές* είναι *ακολουθίες*

```
>>> s = 'hello'
>>> len(s)
5
>>> s[1]
'e'
>>> s[1:]
'ello'
>>> s[::-1]
'olleh'
>>> s = '' # κενό string
>>> len(s)
0
```

Συμβολοσειρές

- Οι τελεστές `in`, `not in` ελέγχουν για ταίριασμα τμήματος της συμβολοσειράς

```
>>> 'ello' in 'hello'
True
>>> 'ello' in ['h', 'e', 'l', 'l', 'o']
False
```

- Κατασκευαστής αλφαριθμητικού: `str`

```
>>> "It's "+str(1==1)+':'+str(25+2)+' days until Xmas!'
"It's True:8 days until Xmas!"
```

- Κατασκευή `list` από `str`

```
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```