Python: Δομές Δεδομένων

Ενότητες

- Δομές δεδομένων
- Αλφαριθμητικά
- Λίστες
- Πλειάδες
- Λεξικά
- Πίνακες

Δομές Δεδομένων

- Με τον όρο **δομή δεδομένων (data structure)** εννοούμε τον τρόπο αποθήκευσης δεδομένων, οργανωμένα και βασιζόμενα σε πιο απλούς τύπους δεδομένων
- Σκοπός είναι να επιτύχουμε όσο το δυνατόν καλύτερο τρόπο οργάνωσης και αποθήκευσής τους, ώστε αυτά να μπορούν να χρησιμοποιηθούν αποδοτικά

Δομές Δεδομένων

- Γνωστές δομές δεδομένων είναι οι **εγγραφές (records)**, οι **πίνακες (arrays)**, οι **λίστες (lists)** και τα **δέντρα (trees)**
- Επιπλέον, η Python υποστηρίζει **πλειάδες (tuples)**, **σύνολα (sets)** και **λεξικά (dictionaries)**
- Τα **αλφαριθμητικά (strings)** μπορούν επίσης να ενταχθούν στην κατηγορία αυτή

Αλφαριθμητικά

- **Αλφαριθμητικά (strings)** λέγονται ακολουθίες από χαρακτήρες, ψηφία ή άλλα σύμβολα
- Για να ξεχωρίσουν από τις μεταβλητές, τις λέξεις-κλειδιά της γλώσσας και άλλα σύμβολα, τα περικλείουμε μέσα σε εισαγωγικά
- Στην Python είναι επιτρεπτό να χρησιμοποιήσουμε τόσο τα μονά (**'...'**) όσο και τα διπλά εισαγωγικά (**"..."**) για το σκοπό αυτό

Αλφαριθμητικά

- **Αλφαριθμητικά (strings)** λέγονται ακολουθίες από χαρακτήρες, ψηφία ή άλλα σύμβολα
- Για να ξεχωρίσουν από τις άλλες μεταβλητές, τις λέξεις-κλειδιά της γλώσσας και άλλα σύμβολα, τα περικλείουμε μέσα σε εισαγωγικά
- Στην Python είναι επιτρεπτό να χρησιμοποιήσουμε τόσο τα μονά (**'...'**) όσο και τα διπλά εισαγωγικά (**"..."**) για το σκοπό αυτό

Αλφαριθμητικά

- Παράδειγμα

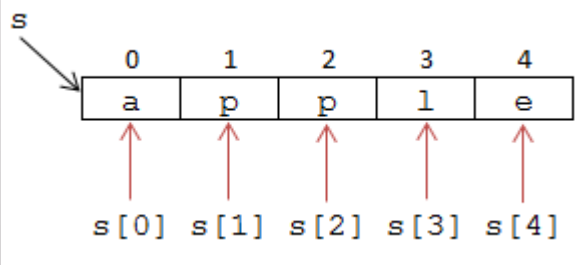
```
>>> s1 = 'Hello'
>>> s2 = "Hello"
>>> print(s1)
Hello
>>> print(s2)
Hello
>>> type('Hello')
<class 'str'>
>>> s3 = '''Hello,
World!'''
>>> print(s3)
Hello,
World!
>>>
```

Τις συμβολοσειρές μπορούμε να τις διαβάσουμε από το πληκτρολόγιο ως είσοδο, μπορούμε να τις τυπώσουμε και στην οθόνη του υπολογιστή μας

Αλφαριθμητικά

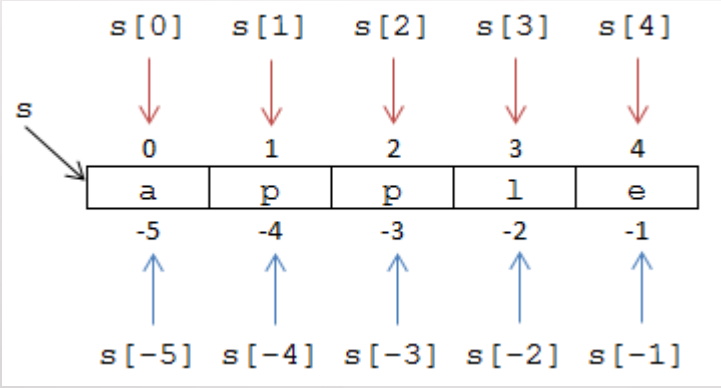
- Προσπέλαση με δείκτες
  - Ένας δείκτης (index) αναφέρεται σε μέλος ενός διατεταγμένου συνόλου, δηλαδή σε ένα μέλος του συνόλου χαρακτήρων μιας συμβολοσειράς
  - Δείκτης μπορεί να είναι κάθε ακέραιος αριθμός ή έκφραση

```
>>> s = 'apple'
>>> s[0]
'a'
>>> s[1]
'p'
>>> s[2]
'p'
>>> s[3]
'l'
>>> s[4]
'e'
>>>
```



Αλφαριθμητικά

- Προσπέλαση με δείκτες
  - Εναλλακτικά, με χρήση αρνητικών δεικτών




```
>>> s[-1]
'e'
>>> s[-2]
'l'
>>> s[-3]
'p'
>>> s[-4]
'p'
>>> s[-5]
'a'
>>>
```

Αλφαριθμητικά

- Η συνάρτηση **len** επιστρέφει το μήκος μιας συμβολοσειράς (το πλήθος των στοιχείων της)

```
>>> fruit = 'banana'
>>> print(len(fruit))
6
>>>
```

Αλφαριθμητικά

- Προσπέλαση στοιχείων μιας συμβολοσειράς με χρήση της **while**

```
fruit = 'banana'
index = 0

while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

```
>>> ================================ RESTART ===
>>>
b
a
n
a
n
a
```

Αλφαριθμητικά

- Προσπέλαση στοιχείων μιας συμβολοσειράς με χρήση της **for**

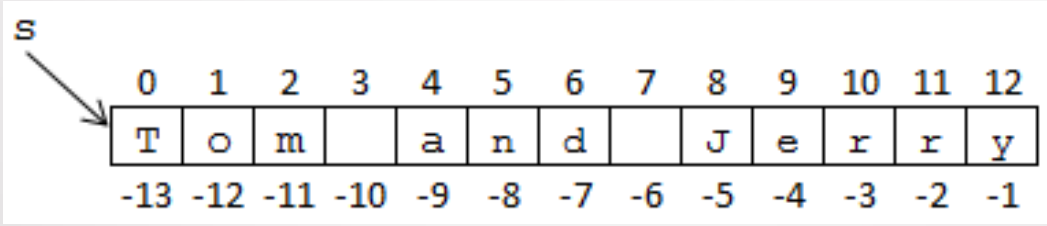
```
fruit = 'banana'

for char in fruit:
    print(char)
```

```
>>> ================================ RESTART ===
>>>
b
a
n
a
n
a
```

Αλφαριθμητικά

- Προσπέλαση μέρους των στοιχείων μιας συμβολοσειράς με χρήση του τελεστή : **([start:end])**




```
>>> s = 'Tom and Jerry'
>>> s[0:3]
'Tom'
>>> s[4:7]
'and'
>>> s[8:13]
'Jerry'
>>> s[0:1]
'T'
>>>
```

```
>>> s[:3]
'Tom'
>>> s[4:]
'and Jerry'
>>>
```

```
>>> s[-13:-10]
'Tom'
>>>
```

Αλφαριθμητικά

- Σύγκριση

```
>>> 'a' < 'b'
True
>>> 'aa' < 'ab'
True
>>> 'aa' == 'aa'
True
>>> 'apple' < 'banana'
True
>>> 'Apple' > 'apple'
False
>>> 'Αντώνης' < 'Εμμανουέλα'
True
```

Τα κεφαλαία προηγούνται των πεζών γραμμάτων και τα Λατινικά προηγούνται των Ελληνικών γραμμάτων. Μια σημαντική χρήση της σύγκρισης συμβολοσειρών είναι η αλφαβητική ταξινόμηση ονομάτων

Αλφαριθμητικά

- Είναι αμετάβλητα!

```
>>> greeting = 'Hello, World!'
>>> greeting[0] = 'J'
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    greeting[0] = 'J'
TypeError: 'str' object does not support item assignment
>>>
```

```
>>> greeting = 'Hello, World!'
>>> new_greeting = 'J' + greeting[1:]
>>> print(new_greeting)
Jello, World!
>>>
```

Αλφαριθμητικά

- Ο τελεστής **in** στα αλφαριθμητικά
  - Ο τελεστής in είναι ένας λογικός τελεστής που εφαρμόζεται σε δύο συμβολοσειρές και ελέγχει αν η μία εμφανίζεται μέσα στην άλλη

```
>>> 'a' in 'banana'
True
>>> 'an' in 'banana'
True
>>> 'c' in 'banana'
False
>>>
```

Αλφαριθμητικά

- Μέθοδοι ελέγχου συμβολοσειράς
  - **s.isalpha()** Η s περιέχει μόνο γράμματα
  - **s.isdigit()** Η s περιέχει μόνο ψηφία
  - **s.islower()** Η s περιέχει μόνο πεζά γράμματα
  - **s.isupper()** Η s περιέχει μόνο κεφαλαία γράμματα
  - **s.isprintable()** Η s περιέχει μόνο εκτυπώσιμους χαρακτήρες
  - **s.isspace()** Η s περιέχει μόνο whitespaces (κενά, tab, αλλαγές γραμμής)
  - **s.startswith(t)** Η s ξεκινάει με την t
  - **s.endswith(t)** Η s τελειώνει με την t

Αλφαριθμητικά

- Παράδειγμα

```
>>> s = 'hello'
>>> s.isalpha()
True
>>> s = '1209'
>>> s.isdigit()
True
>>> s = 'python'
>>> s.islower()
True
>>> s = 'PYTHON'
>>> s.isupper()
True
>>> s.isprintable()
True
>>> s = '\t \n'
>>> s.isspace()
True
>>> s = 'Tom and Jerry'
>>> s.startswith('Tom')
True
>>> s.endswith('Jerry')
True
>>>
```

Αλφαριθμητικά

- Μέθοδοι αναζήτησης σε συμβολοσειρές
  - **s.find(t)** Επιστρέφει τον πρώτο δείκτη, στον οποίο ξεκινάει η t μέσα στην s, αλλιώς επιστρέφει -1
  - **s.rfind(t)** Όμοια με τη find, αλλά αναζητάει από τα δεξιά προς τα αριστερά
- Μέθοδοι απαρίθμησης σε συμβολοσειρές
  - **s1.count(s2)** απαριθμεί την εμφάνιση μιας συμβολοσειράς μέσα σε μία άλλη

Αλφαριθμητικά

- Παράδειγμα

```
>>> s = 'apple'
>>> s.find('le')
3
>>> s.find('w')
-1
>>> s.find('p')
1
>>> s.rfind('p')
2
>>>
```

```
>>> fruit = 'banana'
>>> fruit.count('a')
3
>>> fruit.count('na')
2
>>>
```

Αλφαριθμητικά

- Μέθοδοι μετατροπής σε συμβολοσειρές
  - **s.capitalize()** Το s[0] μετατρέπεται σε κεφαλαίο.
  - **s.lower()** Όλα τα γράμματα της s μετατρέπονται σε πεζά.
  - **s.upper()** Όλα τα γράμματα της s μετατρέπονται σε κεφαλαία.
  - **s.swapcase()** Τα πεζά μετατρέπονται σε κεφαλαία και τα κεφαλαία σε πεζά.

Αλφαριθμητικά

- Παράδειγμα

```
>>> s = 'hello'
>>> s.capitalize()
'Hello'
>>> s.lower()
'hello'
>>> s.upper()
'HELLO'
>>> s = 'hElLo'
>>> s.swapcase()
'HeLlO'
>>>
```

Αλφαριθμητικά

- Μέθοδοι αφαίρεσης ανεπιθύμητων χαρακτήρων σε συμβολοσειρές
  - **s.strip(ch)** Αφαιρούνται όλοι οι χαρακτήρες ch που εμφανίζονται στην αρχή ή το τέλος της s
  - **s.lstrip(ch)** Αφαιρούνται όλοι οι χαρακτήρες ch που εμφανίζονται στην αρχή (αριστερά) της s
  - **s.rtrip(ch)** Αφαιρούνται όλοι οι χαρακτήρες ch που εμφανίζονται στο τέλος (δεξιά) της s

Αλφαριθμητικά

- Παράδειγμα

```
>>> fruit = '--banana--'
>>> fruit.strip('-')
'banana'
>>> fruit.lstrip('-')
'banana--'
>>> fruit.rstrip('-')
'--banana'
>>>
```

```
>>> fruit = ' banana  '
>>> fruit.strip()
'banana'
>>>
```

Αλφαριθμητικά

- Μορφοποίηση συμβολοσειράς με χρήση της μεθόδου **format**

```
>>> name = 'Αντώνης'
>>> age = 13
>>> print('Ο {0} είναι {1} ετών.'.format(name, age))
Ο Αντώνης είναι 13 ετών.
>>>
```

Αλφαριθμητικά

- Ζητήστε αναλυτική βοήθεια από τον διερμηνευτή πληκτρολογώντας την εντολή **help(str)**

Ασκήσεις Α

1. Γράψτε ένα πρόγραμμα που θα δέχεται ως είσοδο μια λέξη και ένα ακέραιο αριθμό n και θα εκτυπώνει τη λέξη αφού έχει αφαιρέσει τους χαρακτήρες μέχρι τον αριθμό n
2. Γράψτε ένα πρόγραμμα που θα δέχεται το όνομα του χρήστη και θα εκτυπώνει τους χαρακτήρες αυτού που βρίσκονται σε θέση με ζυγό αριθμό θέσης

Ασκήσεις Β

1. Γράψτε ένα πρόγραμμα που θα επιστρέφει πόσες φορές συναντάται ένα string μέσα σε ένα άλλο
2. Γράψτε ένα πρόγραμμα που θα δέχεται ένα string το οποίο θα περιέχει πεζά και κεφαλαία γράμματα και θα εκτυπώνει πρώτα τα πεζά και μετά τα κεφαλαία

Λίστες

- Μία **λίστα** (**list**) είναι μια διατεταγμένη συλλογή τιμών, οι οποίες αντιστοιχίζονται σε δείκτες
- Οι τιμές που είναι μέλη μιας λίστας ονομάζονται **στοιχεία** (**elements**)
- Τα στοιχεία μιας λίστας δεν χρειάζεται να είναι ίδιου τύπου και ένα στοιχείο σε μία λίστα μπορεί να υπάρχει περισσότερες από μία φορές
- Μία λίστα μέσα σε μία άλλη λίστα ονομάζεται **εμφωλευμένη λίστα** (**nested list**)

Λίστες

- Τα στοιχεία μιας λίστας διαχωρίζονται με κόμμα και περικλείονται σε τετράγωνες αγκύλες ([και])

```
mylist1 = [6, 2, -4, 2]
```

- Μία λίστα που δεν περιέχει στοιχεία ονομάζεται **άδεια λίστα** και συμβολίζεται με []

```
mylist2 = []
```

Λίστες

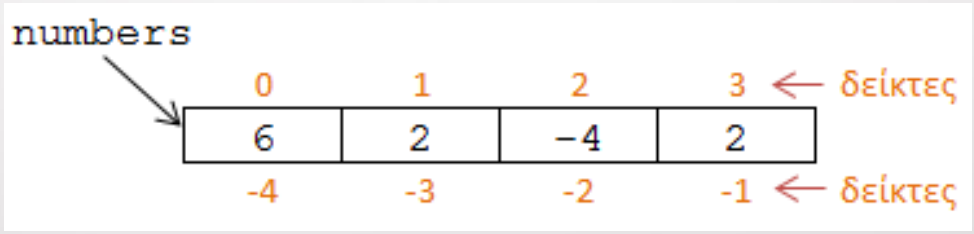
- Παράδειγμα

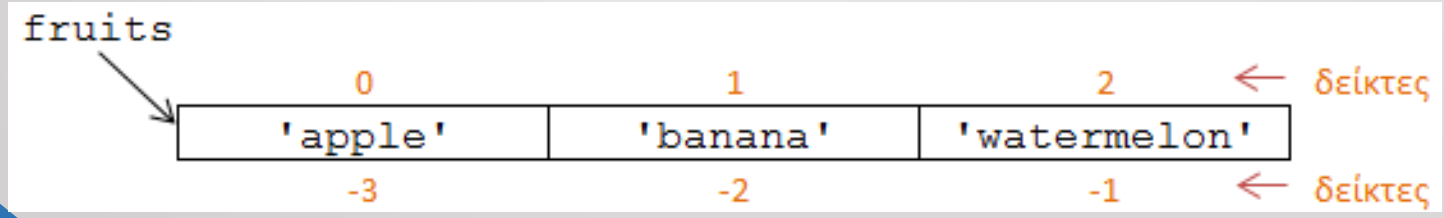
```
>>> numbers = [6, 2, -4, 2]
>>> fruits = ['apple', 'banana', 'watermelon']
>>> lst = [1, 6.5, 'cherry', [5, 7, 9]]
>>> empty_list = []
>>> print(numbers, '\n', fruits, '\n', lst, '\n', empty_list)
[6, 2, -4, 2]
 ['apple', 'banana', 'watermelon']
 [1, 6.5, 'cherry', [5, 7, 9]]
 []
>>>
```

Λίστες

- Παράδειγμα

```
>>> numbers = [6, 2, -4, 2]
>>> fruits = ['apple', 'banana', 'watermelon']
```





Λίστες

- Οι λίστες που περιέχουν συνεχόμενους ακέραιους αριθμούς είναι συνηθισμένες, και έτσι η Python μάς προσφέρει έναν εύκολο τρόπο, για να τις δημιουργούμε με τη συνάρτηση **range(start,end,step)**

```
>>> numbers = list(range(1,10))
>>> print(numbers)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

Λίστες

- Παράδειγμα

```
>>> numbers1 = list(range(10))
>>> print(numbers1)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> numbers2 = list(range(1,10,2))
>>> print(numbers2)
[1, 3, 5, 7, 9]
>>>
```

Λίστες

- Προσπέλαση στοιχείων μιας λίστας με δείκτες με θετική τιμή

```
>>> numbers = [2, 5, 9, 15]
>>> print(numbers[0])
2
>>> print(numbers[1])
5
>>> print(numbers[3])
15
>>> print(numbers[3-1])
9
>>> print(numbers[5])
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    print(numbers[5])
IndexError: list index out of range
>>>
```

Λίστες

- Προσπέλαση στοιχείων μιας λίστας με δείκτες με αρνητική τιμή

```
>>> numbers = [2, 5, 9, 15]
>>> print(numbers[-1])
15
>>> print(numbers[-2])
9
>>> print(numbers[-3])
5
>>> print(numbers[-4])
2
>>>
```

Λίστες

- Προσπέλαση στοιχείων μιας λίστας με χρήση της **while**

```
fruits = ['apple', 'cherry', 'banana', 'mango']
i = 0
while i < len(fruits):
    print(fruits[i])
    i = i + 1
```

```
>>> ================================ RESTART ===
>>>
apple
cherry
banana
mango
```

Λίστες

- Η συνάρτηση **len** επιστρέφει το μήκος μιας λίστας (το πλήθος των στοιχείων της)
- Αν μία λίστα περιέχει άλλη λίστα ως στοιχείο, τότε η εμφωλευμένη λίστα μετράει ως απλό στοιχείο

```
>>> mylist = [2, 5, [8, 12, 20]]
>>> print(len(mylist))
3
>>>
```

Λίστες

- Προσπέλαση στοιχείων μιας λίστας με χρήση της **for**

```
fruits = ['apple', 'cherry', 'banana', 'mango']
for fruit in fruits:
    print(fruit)
```

```
>>> ================================ RESTART ===
>>>
apple
cherry
banana
mango
```

Λίστες

- Με τον λογικό τελεστή **in** (**not in**) μπορούμε να ελέγξουμε αν μια τιμή **ανήκει** (**δεν ανήκει**) σε μια λίστα

```
>>> fruits = ['apple', 'cherry', 'banana']
>>> 'zebra' in fruits
False
>>> 'zebra' not in fruits
True
>>>
```

```
for fruit in ['apple', 'banana', 'mango']:
    print('I like to eat ' + fruit + 's.')
```

```
>>> ================================ RESTART ===
>>>
I like to eat apples.
I like to eat bananas.
I like to eat mangos.
```

Λίστες

- Παράδειγμα

```
for fruit in ['apple', 'banana', 'mango']:
    print('I like to eat ' + fruit + 's.')
```

```
>>> ================================ RESTART ===
>>>
I like to eat apples.
I like to eat bananas.
I like to eat mangos.
```

Λίστες

- Πράξεις με λίστες
  - Με τον τελεστή **+** μπορούμε να συνενώσουμε λίστες
  - Με τον τελεστή * να επαναλάβουμε μια λίστα

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5]
>>> 5 * [1]
[1, 1, 1, 1, 1]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>>
```

Λίστες

- Ο τελεστής :

```
>>> mylist = ['a', 'b', 'c', 'd', 'e', 'f']
>>> mylist[1:3]
['b', 'c']
>>> mylist[:4]
['a', 'b', 'c', 'd']
>>> mylist[3:]
['d', 'e', 'f']
>>> mylist[:]
['a', 'b', 'c', 'd', 'e', 'f']
>>>
```

Λίστες

- Μεταβολή στοιχείων λίστας

```
>>> fruits = ['apple', 'cherry', 'banana']
>>> fruits[1] = 'orange'
>>> fruits[-1] = 'melon'
>>> print(fruits)
['apple', 'orange', 'melon']
>>>
```

```
>>> mylist = ['a', 'b', 'c', 'd', 'e', 'f']
>>> mylist[1:3] = ['x', 'y']
>>> print(mylist)
['a', 'x', 'y', 'd', 'e', 'f']
>>>
```

```
>>> mylist = ['a', 'b', 'c', 'd', 'e', 'f']
>>> mylist[1:3] = []
>>> print(mylist)
['a', 'd', 'e', 'f']
>>>
```

Λίστες

- Διαγραφή στοιχείων λίστας με τη μέθοδο **del**

```
>>> mylist = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del mylist[5]
>>> print(mylist)
['a', 'b', 'c', 'd', 'e']
>>> del mylist[-1]
>>> print(mylist)
['a', 'b', 'c', 'd']
>>> del mylist[1:4]
>>> print(mylist)
['a']
>>>
```

Λίστες

- Κλωνοποίηση λίστας με τον τελεστή :

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> print(b)
[1, 2, 3]
>>> b[0] = 3
>>> print(b)
[3, 2, 3]
>>> print(a)
[1, 2, 3]
>>>
```

Λίστες

- Λίστες και Συμβολοσειρές
  - Με τη μέθοδο **split** μπορούμε να δημιουργήσουμε λίστα από συμβολοσειρά. Οι χαρακτήρες whitespace θεωρούνται όρια λέξεων. Προαιρετικά μπορεί να χρησιμοποιηθεί όρισμα που θα ορίζει τους χαρακτήρες που οριοθετούν λέξεις

```
>>> phrase = 'What a wonderful world'
>>> phrasewords = phrase.split()
>>> print(phrasewords)
['What', 'a', 'wonderful', 'world']
>>> phrasewords1 = phrase.split('wo')
>>> print(phrasewords1)
['What a ', 'nderful ', 'rld']
>>>
```

Λίστες

- Λίστες και Συμβολοσειρές
  - Η συνάρτηση **list** «σπάει» μια συμβολοσειρά σε χαρακτήρες δημιουργώντας μία λίστα

```
>>> s = 'spam'
>>> chlist = list(s)
>>> print(chlist)
['s', 'p', 'a', 'm']
>>>
```

Λίστες

- Λίστες και Συμβολοσειρές
  - Η μέθοδος **join** δέχεται μία λίστα από συμβολοσειρές και συνενώνει τα στοιχεία της

```
>>> phrasewords = ['What', 'a', 'wonderful', 'world']
>>> delimiter1 = ' '
>>> phrase1 = delimiter1.join(phrasewords)
>>> print(phrase1)
What a wonderful world
>>> delimiter2 = '-'
>>> phrase2 = delimiter2.join(phrasewords)
>>> print(phrase2)
What-a-wonderful-world
>>>
```

Λίστες

- Άλλες μέθοδοι για Λίστες (*help(list)*)
  - **l.count(x)** Μετράει πόσες φορές το στοιχείο x εμφανίζεται στη λίστα l
  - **l.index(x) Ε**πιστρέφει τη θέση του πρώτου στοιχείου που εμφανίζεται στη λίστα l και είναι ίσο με το x
  - **l.append(x)** Προσθέτει το στοιχείο x στο τέλος της λίστας l
  - **l.insert(i,x)** Προσθέτει το στοιχείο x στη θέση i της λίστας l
  - **l.extend(lst)** Προσθέτει κάθε στοιχείο της lst στο τέλος της l
  - **l.pop(i)** Αφαιρεί και επιστρέφει το στοιχείο που βρίσκεται στο δείκτη i
  - **l.remove(x)** Αφαιρεί την πρώτη εμφάνιση από αριστερά του στοιχείου x
  - **l.sort()** Ταξινομεί τη λίστα l κατά αύξουσα σειρά
  - **l.sort()** Αντιστρέφει τη σειρά των στοιχείων της λίστας l

Λίστες

- Παράδειγμα

```
>>> x=[1,2,3,'ok',(2,3),[1,2,3],2]
>>> del x[2]
>>> x
[1, 2, 'ok', (2, 3), [1, 2, 3], 2]
>>> x.append('new')
>>> x
[1, 2, 'ok', (2, 3), [1, 2, 3], 2, 'new']
>>> x.count(2)
2
>>> x.count((2,3))
1
>>> x.index([1,2,3])
4
>>> x.index(2)
1
>>> x.insert(3,'inserted')
>>> x
[1, 2, 'ok', 'inserted', (2, 3), [1, 2, 3], 2, 'new']
>>>
```

?

Λίστες

- Παράδειγμα

```
>>> mylist = ['a', 'b', 'c']
>>> mylist.append('d')
>>> mylist.append('e')
>>> print(mylist)
['a', 'b', 'c', 'd', 'e']
>>>
>>> mylist1 = ['c', 'a', 'e', 'b', 'd']
>>> mylist1.sort()
>>> print(mylist1)
['a', 'b', 'c', 'd', 'e']
>>>
```

?

Λίστες

- Παράδειγμα

```
>>> l = ['a', 'b']
>>> lst = ['c', 'd', 'e']
>>> l.extend(lst)
>>> print(l)
['a', 'b', 'c', 'd', 'e']
>>> letter = l.pop(0)
>>> print(letter)
a
>>> print(l)
['b', 'c', 'd', 'e']
>>> l.remove('e')
>>> print(l)
['b', 'c', 'd']
>>>
```

?

Λίστες

- Παράδειγμα

?

```
>>> x
[1, 2, 'ok', 'inserted', (2, 3), [1, 2, 3], 2, 'new']
>>> x.pop()
'new'
>>> x
[1, 2, 'ok', 'inserted', (2, 3), [1, 2, 3], 2]
>>> x.pop(3)
'inserted'
>>> x
[1, 2, 'ok', (2, 3), [1, 2, 3], 2]
>>> x.remove(2)
>>> x
[1, 'ok', (2, 3), [1, 2, 3], 2]
>>> x.reverse()
>>> x
[2, [1, 2, 3], (2, 3), 'ok', 1]
>>> y=['a','b','c','d']
>>> x.extend(y)
>>> x
[2, [1, 2, 3], (2, 3), 'ok', 1, 'a', 'b', 'c', 'd']
>>> y
['a', 'b', 'c', 'd']
```

Ασκήσεις

1. Γράψτε ένα πρόγραμμα που θα δέχεται μια λίστα ακεραίων αριθμών και θα επιστρέφει TRUE αν το πρώτο στοιχείο της λίστα είναι ίσο με το τελευταίο
2. Γράψτε ένα πρόγραμμα που θα δέχεται μια λίστα ακεραίων αριθμών και θα εκτυπώνει μόνον αυτούς που διαιρούνται με το 5 και το υπόλοιπο της διαίρεσης είναι μηδέν

Πλειάδες

- Μία **πλειάδα** (**tuple**) είναι μια διατεταγμένη ακολουθία τιμών, οι οποίες αντιστοιχίζονται σε δείκτες
- Οι τιμές που είναι μέλη μιας πλειάδας ονομάζονται **στοιχεία** (**elements**) και μπορεί να είναι οποιουδήποτε τύπου (αριθμοί, συμβολοσειρές, λίστες, πλειάδες)
- Συντακτικά, μια πλειάδα είναι μια λίστα τιμών που χωρίζονται με κόμμα

Πλειάδες

- Οι πλειάδες μοιάζουν με τις λίστες στη χρήση δεικτών, στον τρόπο με τον οποίο διατρέχονται και στη χρήση του τελεστή :
- Όμως οι πλειάδες, όπως και οι συμβολοσειρές, είναι **αμετάβλητες**
- Οι πλειάδες αξιοποιούνται συνήθως στις περιπτώσεις όπου πρόκειται να χρησιμοποιηθεί μια ακολουθία τιμών (πλειάδα) που δεν πρόκειται να αλλάξει

Πλειάδες

- Παραδείγματα

```
>>> t1 = 'a', 'b', 'c', 'd'
>>> print(t1)
('a', 'b', 'c', 'd')
>>> t2 = (1, 3, 5, 7, 9)
>>> print(t2)
(1, 3, 5, 7, 9)
>>>
```

```
>>> t3 = ()
>>> print(t3)
()
>>> t4 = ('a',)
>>> print(t4)
('a',)
>>> t5 = ('a')
>>> print(t5)
a
>>> type(t5)
<class 'str'>
>>>
```

Πλειάδες

- Παράδειγμα: Ένας μαθητής θα μπορούσε να αναπαρασταθεί με πλειάδα. Αποτελείται από τον κωδικό, το ονοματεπώνυμο, τη διεύθυνση, την τάξη και τον μέσο όρο της βαθμολογίας

```
>>> x=333,'Γιώργος Μανής','Νεοκαισάρεια',4,3.3
>>> x
(333, 'Γιώργος Μανής', 'Νεοκαισάρεια', 4, 3.3)
>>> x[1]
'Γιώργος Μανής'
```

Πλειάδες

- Οι πράξεις πάνω σε πλειάδες είναι παρόμοιες με τις πράξεις πάνω σε λίστες
- Ο τελεστής **[ ]** επιλέγει ένα στοιχείο από μια πλειάδα
- Ο τελεστής **:** επιλέγει διάστημα τιμών, όπως ακριβώς και στις λίστες
- Ο τελεστής **in** ελέγχει εάν μια τιμή ανήκει σε μια πλειάδα
- Η συνάρτηση **len** επιστρέφει το μήκος μιας πλειάδας (τον αριθμό των στοιχείων που περιέχει)

Πλειάδες

- Παραδείγματα

```
>>> t1 = ('a', 'b', 'c')
>>> t2 = ('d', 'e')
>>> t = t1 + t2
>>> print(t)
('a', 'b', 'c', 'd', 'e')
>>> print(t[1])
b
>>> print(t[2:4])
('c', 'd')
>>> 'a' in t
True
>>> 'f' in t
False
>>> print(len(t))
5
>>> print(len((1, 2, 'a', [3, 4], (9, 12))))
5
>>>
```

Πλειάδες

- Για να διατρέξουμε μια πλειάδα (πέρασμα πλειάδας), κάνουμε ακριβώς ό,τι κάναμε και στις λίστες

```
fruits = ('apple', 'orange', 'banana', 'mellon')
i = 0
while i < len(fruits):
    print(fruits[i])
    i = i + 1
```

```
fruits = ('apple', 'orange', 'banana', 'mellon')
for fruit in fruits:
    print(fruit)
```

Πλειάδες

- Οι πλειάδες είναι αμετάβλητες
- Αν προσπαθήσουμε να αλλάξουμε ένα από τα στοιχεία μιας πλειάδας, θα εμφανιστεί μήνυμα λάθους
- Μπορούμε όμως να αντικαταστήσουμε μια πλειάδα με μία άλλη

```
>>> t = ('a', 'b', 'c')
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c')
>>>
```

Πλειάδες

- Η συνάρτηση **tuple** δέχεται ως όρισμα μια συμβολοσειρά ή μια λίστα και επιστρέφει μια πλειάδα
- Με κενό όρισμα επιστρέφει μια άδεια πλειάδα

```
>>> t1 = tuple()
>>> print(t1)
()
>>> t2 = tuple('spam')
>>> print(t2)
('s', 'p', 'a', 'm')
>>> t3 = tuple([1, 2, 3])
>>> print(t3)
(1, 2, 3)
>>>
```

Πλειάδες

- Η μέθοδος **count** επιστρέφει τον αριθμό των εμφανίσεων μιας τιμής σε μια πλειάδα
- Η μέθοδος **index** επιστρέφει τον πρώτο δείκτη στον οποίο αντιστοιχεί μια τιμή. Αν δεν υπάρχει τιμή, εμφανίζεται μήνυμα λάθους

```
>>> t = ('a', 'b', 'c', 'a')
>>> t.count('a')
2
>>> t.index('a')
0
>>>
```

Ασκήσεις

1. Γράψτε ένα πρόγραμμα που θα δέχεται μια πλειάδα ως είσοδο και θα βρίσκει τη θέση της πρώτης εμφάνισης ενός στοιχείου στην πλειάδα

Λεξικά

- Το **λεξικό** (**dictionary**) είναι μια δομή δεδομένων για αποθήκευση ζευγαριών τιμών της μορφής

  κλειδί:τιμή (*key:value*)

- Πρόκειται για έναν σύνθετο τύπο
- Κάθε κλειδί αντιστοιχίζεται σε μια τιμή και είναι μοναδικό σε ένα λεξικό

Λεξικά

- Για κλειδιά ενός λεξικού χρησιμοποιούμε μόνο αμετάβλητα αντικείμενα (όπως ακέραιους αριθμούς, συμβολοσειρές, πλειάδες)
- Γ ια τις τιμές μπορούμε να έχουμε είτε αμετάβλητα ή μετατρέψιμα αντικείμενα
- Τα λεξικά είναι μετατρέψιμα, και μπορούμε εύκολα να προσθέσουμε και να διαγράψουμε στοιχεία
- Επίσης, ένα λεξικό αποτελεί μια μη διατεταγμένη συλλογή από ζεύγη κλειδιών-τιμών, τα οποία δεν ταξινομούνται με κανένα τρόπο (απροσδιόριστη σειρά)
- Δεν υπάρχει η έννοια της θέσης δείκτη

Λεξικά

- Παράδειγμα

```
>>> eng2sp = {}
>>> eng2sp['one'] = 'uno'
>>> eng2sp['two'] = 'dos'
>>> print(eng2sp)
{'two': 'dos', 'one': 'uno'}
>>>
```

```
>>> eng2sp = {'one':'uno', 'two':'dos', 'three':'tres'}
>>> print(eng2sp)
{'two': 'dos', 'one': 'uno', 'three': 'tres'}
>>> print(eng2sp['one'])
uno
>>> print(eng2sp['two'])
dos
>>> print(eng2sp['three'])
tres
>>>
```

Λεξικά

- Παράδειγμα. Δημιουργία λεξικού με την ενσωματωμένη συνάρτηση **dict**

```
>>> d1 = dict()
>>> print(d1)
{}
>>> d2 = dict(red = 1, green = 2, blue = 3)
>>> print(d2)
{'green': 2, 'blue': 3, 'red': 1}
>>>
```

Λεξικά

- Ο τελεστής **del**

```
>>> inventory = {'apples':400, 'oranges':200, 'bananas':500}
>>> print(inventory)
{'oranges': 200, 'apples': 400, 'bananas': 500}
>>> del inventory['oranges']
>>> print(inventory)
{'apples': 400, 'bananas': 500}
>>>
```

Λεξικά

- Η συνάρτηση **len**

```
>>> inventory = {'apples':400, 'oranges':200, 'bananas':500}
>>> print(len(inventory))
3
>>> 'apples' in inventory
True
>>> 'mangos' in inventory
False
>>>
```

Λεξικά

- Οι μέθοδοι **keys**, **values**, **items**

```
>>> inventory = {'apples':400, 'oranges':200, 'bananas':500}
>>> keys_view = inventory.keys()
>>> print(keys_view)
dict_keys(['oranges', 'apples', 'bananas'])
>>> values_view = inventory.values()
>>> print(values_view)
dict_values([200, 400, 500])
>>> items_view = inventory.items()
>>> print(items_view)
dict_items([('oranges', 200), ('apples', 400), ('bananas', 500)])
>>>
```

Λεξικά

- Προσπέλαση του λεξικού με την **for**

```
eng2sp = {'one':'uno', 'two':'dos', 'three':'tres'}
keys_view = eng2sp.keys()
values_view = eng2sp.values()
items_view = eng2sp.items()

for key in keys_view:
    print(key)

for value in values_view:
    print(value)

for key,value in items_view:
    print(key,value)
```

```
>>> ================================ RESTART ===
>>>
three
two
one
tres
dos
uno
three tres
two dos
one uno
```

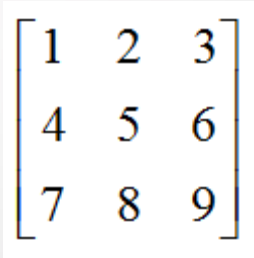
Λεξικά

- Παράδειγμα

```
>>> tel={'giorgos':'26542','petros':'25421','andreas':'56254'}
>>> tel['giorgos']
'26542'
>>> tel.update({'sylvia':'56541'})
>>> tel
{'andreas': '56254', 'sylvia': '56541', 'petros': '25421',
                                        'giorgos': '26542'}
>>> tel['sylvia']='unknown'
>>> tel
{'andreas': '56254', 'sylvia': 'unknown', 'petros': '25421',
                                         'giorgos': '26542'}
>>> x=tel.pop('giorgos')
>>> tel
{'andreas': '56254', 'sylvia': 'unknown', 'petros': '25421'}
>>> x
'26542'
```

Πίνακες

- Με εμφωλευμένες λίστες μπορούμε να αναπαραστήσουμε **πίνακες**



```
>>> matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> print('\n', matrix[0], '\n', matrix[1], '\n', matrix[2])

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

Πίνακες

- Άσκηση

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

Τι θα τυπώσει στο παραπάνω παράδειγμα η εντολή
print(matrix[2][1])

Τι θα τυπώσει στο παραπάνω παράδειγμα η εντολή
print(matrix[3][3])

Βιβλιογραφία

- Μανής, Γ., 2015. Εισαγωγή στον Προγραμματισμό με αρωγό τη γλώσσα Python. [ηλεκτρ. βιβλ.] Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών. Διαθέσιμο στο: http://hdl.handle.net/11419/2745
- Αγγελιδάκης, Ν., 2015. Εισαγωγή στον προγραμματισμό με την Python, Ηράκλειο. [ηλεκτρ. βιβλ.] Διαθέσιμο στο: http://aggelid.mysch.gr/ pythonbook/INTRODUCTION_TO_COMPUTER_PROGRAMMING_WITH_PYTHON.pdf