

Γραμματοσειρές - Strings

Δεδομένα τύπου String

Ένα αντικείμενο τύπου string είναι μια σειρά από χαρακτήρες

Για να δημιουργήσουμε ένα αντικείμενο τύπου string θα πρέπει να το βάλουμε τη σειρά των χαρακτήρων σε " "

Ο τελεστής **+** δηλώνει ένωση δύο αντικειμένων τύπου string (**concatenation**).

Όταν το αντικείμενο τύπου string περιέχει νούμερα μέσα σε " " εξακολουθεί να είναι string .

Μπορούμε να μετατρέψουμε νούμερα που περιέχονται σε strings σε νούμερα, χρησιμοποιώντας κατάλληλες συναρτήσεις [int(), float()].

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

Ανάγνωση και μετατροπή από string σε νούμερα

Προτιμούμε να διαβάζουμε strings και κατόπιν να μετατρέπουμε στην κατάλληλη αριθμητική μορφή (int, float)

Η μεθοδολογία αυτή μας δίνει περισσότερο έλεγχο σε λάθος εισαγωγή δεδομένων

Η εισαγωγή αριθμητικών δεδομένων γίνεται επομένως μέσω strings

```
>>> name = input('Enter:')  
Enter:Fotis  
>>> print(name)  
Fotis  
>>> apple = input('Enter:')  
Enter:100  
>>> x = apple - 10  
Traceback (most recent call  
last):  File "<stdin>", line  
1, in <module>  
TypeError: unsupported  
operand type(s) for -: 'str'  
and 'int'  
>>> x = int(apple) - 10  
>>> print(x)  
90
```

Έλεγχος χαρακτήρων μέσα στο string

Ένα αντικείμενο τύπου string είναι ένα tuple (immutable object) με μήκος ίσο με τον αριθμό των χαρακτήρων που περιέχει

Μπορούμε να έχουμε πρόσβαση στον χαρακτήρα που βρίσκεται σε κάποια συγκεκριμένη θέση ακριβώς με τον ίδιο τρόπο όπου έχουμε πρόσβαση στο στοιχείο ενός tuple ή μιας λίστας.

Θα πρέπει να δώσουμε το όνομα του αντικειμένου string ακολουθούμενο από ένα νούμερο που περικλείεται σε [] (i.e. fruit[2] που αντιστοιχεί στον χαρακτήρα n

Το νούμερο μέσα στην τετραγωνική αγκύλη θα πρέπει να είναι ένας ακέραιος και η αρχική του τιμή θα είναι 0 και η μεγαλύτερη το μήκος της γραμματοσειράς - 1.

Μπορεί να είναι μια αριθμητική έκφραση που δίνει αποτέλεσμα ακέραιο

Η συνάρτηση len() επιστρέφει το μήκος της γραμματοσειράς

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

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

Επαναληπτική διαδικασία με strings

Με τη χρήση της δομής while, μιας μεταβλητής επανάληψης (index) και της συνάρτησης len() μπορούμε να εξετάσουμε όλους τους χαρακτήρες ενός αντικειμένου τύπου string

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

0 b
1 a
2 n
3 a
4 n
5 a

Η χρήση ενός for loop είναι πολύ καλύτερη

- Η μεταβλητή επανάληψης περιέχεται μέσα στην δομή του for loop

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

b
a
n
a
n
a

Θα μπορούσαμε ενώ κινούμαστε από τη μια θέση στην επόμενη ενός string να εξετάζουμε πόσες φορές εμφανίζεται κάποιος χαρακτήρας

```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

Εξέταση της επαναληπτικής διαδικασίας for loop

Η μεταβλητή επανάληψης (**letter**) εκτελεί μια επαναληπτική διαδικασία σύμφωνα με την ορισμένη και ταξινομημένη ακολουθία (**banana**)

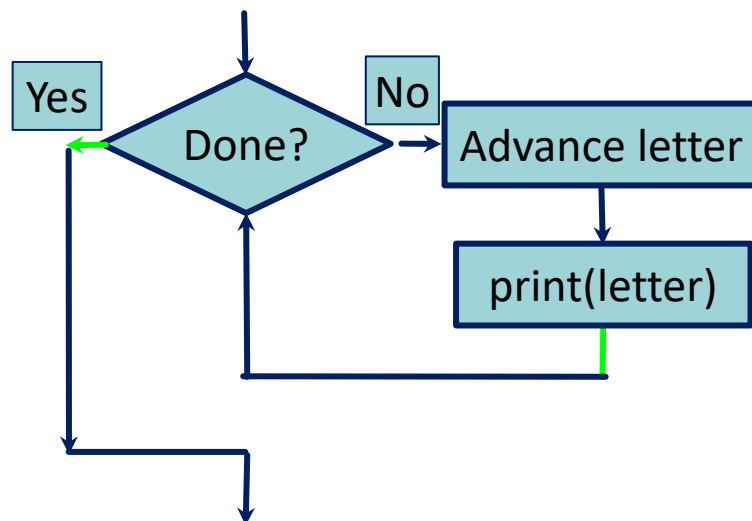
Το κύριο σώμα του loop (οι εντολές του loop) εκτελούνται μια φορά για κάθε τιμή της ακολουθίας

Η μεταβλητή επανάληψης παίρνει όλες τις τιμές που ορίζονται μέσα (**in**) στην ακολουθία

Μεταβλητή
επανάληψης

string 6-χαρακτήρων

```
for letter in 'banana':  
    print(letter)
```



Διαχείριση αντικειμένων τύπου strings

Χρήση τμήματος string - Slicing

Μπορούμε να εξετάσουμε ένα οποιοδήποτε συνεχές διάστημα ενός αντικειμένου τύπου string χρησιμοποιώντας τον τελεστή :

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Δίνουμε το όνομα του αντικειμένου ακολουθούμενο από [n1 : n2]

Το 1^ο νούμερο, n1, δίνει τη θέση από την οποία αρχίζουμε και το 2^ο νούμερο, n2, δηλώνει τη θέση στην οποία σταματούμε (χωρίς να προσμετράται η θέση αυτή). Είναι επομένως διάστημα τιμών θέσεων κλειστό στο κάτω όριο και ανοικτό στο πάνω

Αν το 2^ο νούμερο είναι μεγαλύτερο από το συνολικό μήκος του string σταματά στο τέλος του string

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```


Slicing strings

Αν παραλείψουμε την τιμή του είτε του κάτω ορίου ή την τιμή Του πάνω ορίου τότε θεωρείται αυτόματα η τιμή της αρχικής θέσης ή της τελικής θέσης του string αντίστοιχα

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

Strings και η διαχείρισή τους

Δεν μπορούμε να έχουμε πρόσθεση αριθμού σε αντικείμενο τύπου string.

`S= "This is a string" + 2` δεν επιτρέπεται

Ο τελεστής `+` δηλώνει ένωση (**concatenation**) δύο αντικειμένων τύπου string

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

Χρήση των συναρτήσεων `int()` και `float()` μπορεί να μετατρέπει strings σε integers ή float μεταβλητές

`S= "11235"`

`S= "1.1235"`

`a= int(S)`

`b= float(S)`

To a είναι 11235

To b είναι 1.1235

```
s = '123'
pie = '3.141592653589'
x = int(s) + 1
y = float(pie) + 1
z_bad = int(s) + int(pie)
z_good = int(s) + int(float(pie))
```

Η Python δεν ξέρει πως να διαχειριστεί την μετατροπή ενός αντικείμενου string που περιέχει υποδιαστολή σε αντικείμενο τύπου int.

Χρήση της χαρακτηριστικής λέξης **in** ως λογικός τελεστής


Η χαρακτηριστική λέξη **in** που εμφανίζεται στα **for loops** μπορεί να χρησιμοποιηθεί για να ελέγξουμε αν ένα αντικείμενο τύπου **string** περιέχεται σε κάποιο άλλο αντικείμενο τύπου **string**

Η έκφραση που περιέχει το **in**, αποτελεί μια λογική έκφραση και επομένως το αποτέλεσμα θα είναι είτε **True** ή **False** και μπορεί να χρησιμοποιηθεί σε ένα **if** δομή

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

Strings και η διαχείρισή τους

Η Python μπορεί να προσθέσει δύο strings με +

```
>>>  
>>> s="2" + "3"  
>>> s  
'23'  
>>> 
```

Αρκετές συναρτήσεις χρειάζονται ορίσματα που είναι αντικείμενα τύπου string

Αρκετές φορές strings θέλουμε να περιέχουν κάποια νούμερα, τα οποία όμως θα πρέπει να τα μετατρέψουμε σε strings και να προσθέσουμε στο υπόλοιπο string

π.χ. S = "The average number is " + **str**(5)

Η συνάρτηση **str(N)** μετατρέπει το όρισμα N σε string

Μπορούμε να καθορίσουμε τον αριθμό των ψηφίων που θα εμφανιστούν σε μια μεταβλητή string χρησιμοποιώντας το **%** σύμβολο μορφοποίησης ή την μέθοδο **format**

Η μέθοδος **format**

```
"The value of pi is approximately " + str(np.pi)
"The value of {} is approximately {:.5f}".format('pi', np.pi)
s = "{1:d} plus {0:d} is {2:d}"
s.format(2, 4, 2 + 4)
"Every {2} has its {3}.".format('dog', 'day', 'rose', 'thorn')
"The third element of the list is {0[2]:g}.".format(np.arange(10))
```

```
>>>
>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
>>>
>>> "The value of {} is approximately {:.5f}".format('pi', np.pi)
'The value of pi is approximately 3.14159'
>>>
>>> s="{1:d} plus {0:d} is {2:d}"
>>> s.format(2,4,2+4)
'4 plus 2 is 6'
>>>
>>> "every {2} has its {3}.".format("dog","day","rose","thorn")
'every rose has its thorn.'
>>> "The third element of the list is {0[2]:g}.".format(np.arange(10))
'The third element of the list is 2.'
>>> □
```

Όταν η Python εξετάζει το `format` ενός αντικειμένου τύπου `string`, ερμηνεύει ένα ζευγάρι `{}` ως τη θέση για να εισαγάγει κάποιους αριθμούς.

Το όρισμα της `format` είναι μια σειρά από εκφράσεις οι τιμές των οποίων θα εισαχθούν σε συγκεκριμένες θέσεις. Μπορεί να είναι `strings`, `αριθμοί`, `lists` ή κάτι πιο πολύπλοκο

Η μέθοδος **format**

- ❑ Μια άδεια {} ερμηνεύεται ως την εισαγωγή του επόμενου στοιχείου μιας λίστας. Το στοιχείο μπορεί να έχει συγκεκριμένο τρόπο γραφής (formatted) εάν στα άγκιστρα περιέχεται ο τελεστής **:** ακολουθούμενος από μια εντολή μορφοποίησης

Για παράδειγμα:

```
>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
```

{:.5f} Σημαίνει ότι μορφοποίησε το τρέχον στοιχείο της λίστας ως έναν αριθμό float με 5 ψηφία μετά την υποδιαστολή

- ❑ Μπορούμε να αναφέρουμε στοιχεία της λίστας δίνοντας τη θέση τους στη λίστα που περιέχεται στην εντολή format.

Για παράδειγμα:

```
>>> s="{1:d} plus {0:d} is {2:d}"
>>> s.format(2,4,2+4)
'4 plus 2 is 6'
>>>
```

S ορίζεται σαν string με 3 θέσεις κρατημένες και οι οποίες θα γεμίσουν αργότερα

Θα πάρει το δεύτερο στοιχείο της λίστας **{1:d}**, το πρώτο **{0:d}** και το τρίτο **{2:d}** και θα τα εισαγάγει με αυτή τη σειρά στην string s.

Το σύμβολο **d** σημαίνει αναπαράσταση του αριθμού σε βάση δεκαδικού συστήματος ως **ακέραιος**.

Η Python μπορεί να αναπαραστήσει ακραίους σε δυαδικό (**binary**) με **{:b}** οκταδικό (**octal**) **{:o}**, δεκαεξαδικό (**hexadecimal**) **{:h}**

Η μέθοδος **format**

- ❑ Δεν είναι απαραίτητο να χρησιμοποιήσουμε όλα τα στοιχεία της λίστας

```
>>> "every {2} has its {3}.".format("dog", "day", "rose", "thorn")  
'every rose has its thorn.'
```

- ❑ Μπορούμε να αναφερθούμε σε συγκεκριμένο στοιχείο μιας λίστας που εισάγεται στην εντολή **format**

```
>>> "The third element of the list is {0[2]:g}.".format(np.arange(10))  
'The third element of the list is 2.'  
>>> 
```

Το πεδίο αντικατάστασης **0[2]** αναφέρεται στο 3^ο στοιχείο του πρώτου ορίσματος ενώ ο μορφοποιητικός κωδικός **:g** δίνει την οδηγία στην Python να απεικονίσει τον αριθμό με τον λιγότερο αριθμό ψηφίων (**g**eneral format)

- ❑ Η εντολή **dir(str)** δίνει πληροφορίες για τις διαθέσιμες μεθόδους διαχείρισης strings

Η μέθοδος %

```
"The value of pi is approximately " + str(np.pi)
"The value of %s is approximately %.5f" % ('pi', np.pi)
s = "%d plus %d is %d"
s % (2, 4, 2 + 4)
```

```
>>> "The value of pi is approximately " + str(np.pi)
'The value of pi is approximately 3.141592653589793'
>>>
>>>
>>> "The value of %s is approximately %.5f" % ('pi', np.pi)
'The value of pi is approximately 3.14159'
```

- ❑ Αντικείμενο τύπου string το οποίο ακολουθείται από τον **τελεστή %** αναμένεται να ακολουθείται από έναν τρόπο μορφοποίησης και τιμές που θα εισαχθούν στο string
- ❑ Τα επιθυμητά σημεία εισαγωγής των τιμών σηματοδοτούνται με την ύπαρξη του χαρακτήρα % μέσα στο string. Αυτό ακολουθείτε με την περιγραφή του τρόπου αναπαράστασης της τιμής που θα εισαχθεί.
- ❑ **%s** σημαίνει εισαγωγή στοιχείου και μορφοποίηση ως string
- %.5f** σημαίνει εισαγωγή στοιχείου ως floating αριθμός με 5 δεκαδικά ψηφία
- %d** σημαίνει εισαγωγή στοιχείου ως integer με βάση το 10

Συναρτήσεις της string βιβλιοθήκης

- ❑ Η Python περιέχει μια σειρά από συναρτήσεις που χρησιμοποιούνται για διαχείριση των αντικειμένων τύπου string. Οι συναρτήσεις αυτές βρίσκονται στη string βιβλιοθήκη
- ❑ Για να χρησιμοποιήσουμε τις συναρτήσεις αυτές χρησιμοποιούμε το όνομα του αντικειμένου string, ακολουθούμενο από μία τελεία . και κατόπιν το όνομα της συνάρτησης
- ❑ Οι συναρτήσεις αυτές δεν αλλάζουν την τιμή του αντικειμένου string, αλλά απλά επιστρέφουν ένα νέο αντικείμενο τύπου string το οποίο είναι διαφοροποιημένο σε σχέση με το αρχικό.
- ❑ Η εντολή **dir(str)** δίνει όλες τις συναρτήσεις που περιέχονται στην βιβλιοθήκη string.

['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'identifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
- ❑ Η εντολή **help(str.function-name)** δίνει πληροφορίες για όλες τις συναρτήσεις

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

Παραδείγματα χρήσης συναρτήσεων string

- ❑ Μπορούμε να χρησιμοποιήσουμε την συνάρτηση **find()** για να εξετάσουμε αν ένα αντικείμενο τύπου string περιέχεται σε κάποιο άλλο αντικείμενο τύπου string
- ❑ Η συνάρτηση **find()** βρίσκει την πρώτη εμφάνιση της υπο-string που διερευνάται
- ❑ Αν δεν βρεθεί η υπό αναζήτηση string, τότε η συνάρτηση επιστρέφει -1
- Υπενθύμιση ότι η αρίθμηση χαρακτήρων σε string ξεκινά από το 0

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

- ❑ Μετατροπή ενός string σε μικρούς ή κεφαλαίους χαρακτήρες
- ❑ Πολλές φορές ψάχνουμε σε string είναι προτιμητέο να μετατρέψουμε το string σε μικρούς χαρακτήρες
- ❑ Η συνάρτηση **replace()** ψάχνει και αντικαθιστά όλες τις εμφανίσεις της υπο αναζήτηση string με την string που δίνεται

```
>>> greet = 'Hello Fotis'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO FOTIS
>>> www = greet.lower()
>>> print(www)
hello fotis
>>>
```

Παραδείγματα χρήσης συναρτήσεων string

- ❑ Η συνάρτηση **replace()** ψάχνει και αντικαθιστά όλες τις εμφανίσεις της υπο αναζήτηση string με την string που δίνεται
- ❑ Μερικές φορές θέλουμε να πάρουμε ένα string και να αφαιρέσουμε τα κενά (**whitespaces**) που βρίσκονται στην αρχή και στο τέλος του string
- ❑ Οι συναρτήσεις **lstrip()** και **rstrip()** αφαιρούν τα κενά στα αριστερά και δεξιά αντίστοιχα του string
- ❑ Η συνάρτηση **strip()** αφαιρεί τα κενά και από τις δύο πλευρές του string
- ❑ Η συνάρτηση **startswith()** δίνει την αρχή μιας string

```
>>> greet = 'Hello Fotis'
>>> nstr = greet.replace('Fotis','Maria')
>>> print(nstr)
Hello Maria
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HellX FXtis
>>>
```

```
>>> greet = ' Hello Fotis '
>>> greet.lstrip()
'Hello Fotis '
>>> greet.rstrip()
' Hello Fotis'
>>> greet.strip()
'Hello Fotis'
>>>
```

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

Παράδειγμα: εύρεση και εξαγωγή substring από string

18



28



From fotis.ptochos@ucy.ac.cy Sat Jan 29 09:20:45 2021

```
>>> data = 'From fotis.ptochos@ucy.ac.cy Sat Jan 29 09:20:45 2021'
>>> atpos = data.find('@')
>>> print(atpos)
18
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
28
>>> host = data[atpos+1 : sppos]
>>> print(host)
ucy.ac.cy
```

Η μέθοδος `split()`

- ❑ Η μέθοδος **`split()`** επιστρέφει μια λίστα των λέξεων σε ένα αντικείμενο `string` που ξεχωρίζουν μεταξύ τους με ένα `string`. Ιδιαίτερα χρήσιμη σε ανάγνωση δεδομένων από αρχείο
- ❑ Η σύνταξη είναι: **`str.split(sep[maxplits])`**
 - `str`**: η `string` μεταβλητή που θέλουμε να διαχωρίσουμε
 - `sep`**: η `string` μεταβλητή που χρησιμοποιείται για τον διαχωρισμό. Αυτόματα θεωρείται το `space`. Προαιρετική μεταβλητή
 - `maxplits`**: αριθμός διαχωρισμών που θα εκτελεστούν. Αυτόματα θεωρείται το `-1` που σημαίνει όλοι οι δυνατοί διαχωρισμοί. Προαιρετική μεταβλητή

```
>>> st = '1 <> 3 <> 4'
>>> st.split('<>')
['1 ', ' 3 ', ' 4']
```

Οι λέξεις χωρίζονται με `<>`

Διαχωρισμός τους με το `string <>`

Επιστροφή μιας `list` με τις επιμέρους λέξεις

```
>>> 'a b c'.split()
['a', 'b', 'c']
>>> 'a b c'.split(None)
['a', 'b', 'c']
>>> 'a b c'.split(' ', 1)
['', 'a b c']
>>> 'a b c'.split(' ', 2)
['', 'a', 'b c']
>>> 'a b c'.split(' ', 3)
['', 'a', 'b', 'c']
>>> 'a b c'.split(' ', 4)
['', 'a', 'b', 'c', '']
>>> 'a b c'.split(' ', 5)
['', 'a', 'b', 'c', '']
```

Αυτόματος διαχωρισμός με βάση το `space` (κενό) – Αγνοεί επιπλέον `spaces`

Διαχωρισμός με βάση το `space` αλλά μόνο 1 φορά – Λαμβάνει υπόψη επιπλέον κενά

Διαχωρισμός με βάση το `space` αλλά 2 φορές

Διαχωρισμός με βάση το `space` αλλά 3 φορές

Διαχωρισμός με βάση το `space` αλλά 4 φορές

Η μέθοδος split() – παραδείγματα

```
>>> '-a-b-c-'.split('-')
['', 'a', 'b', 'c', '']
>>> '-a-b-c-'.split('-', 1)
['', 'a-b-c-']
>>> '-a-b-c-'.split('-', 2)
['', 'a', 'b-c-']
>>> '-a-b-c-'.split('-', 3)
['', 'a', 'b', 'c-']
>>> '-a-b-c-'.split('-', 4)
['', 'a', 'b', 'c', '']
>>> '-a-b-c-'.split('-', 5)
['', 'a', 'b', 'c', '']

>>> '----a---b--c-'.split('-')
['', '', '', '', 'a', '', '', 'b', '', 'c', '']
>>> '----a---b--c-'.split('-', 1)
['', '----a---b--c-']
>>> '----a---b--c-'.split('-', 2)
['', '', '--a---b--c-']
>>> '----a---b--c-'.split('-', 3)
['', '', '', '-a---b--c-']
>>> '----a---b--c-'.split('-', 4)
['', '', '', '', 'a---b--c-']
>>> '----a---b--c-'.split('-', 5)
['', '', '', '', 'a', '--b--c-']
>>> '----a---b--c-'.split('-', 6)
['', '', '', '', 'a', '', '-b--c-']
```

Αρχεία

Ανάγνωση/αποθήκευση δεδομένων με χρήση αρχείων

- ❑ Οι περισσότερες πράξεις και αναλύσεις γίνονται εισάγοντας δεδομένα που είναι αποθηκευμένα σε αρχεία. Η Python θα πρέπει επομένως να είναι σε θέση να εισαγάγει τα δεδομένα αυτά από τα αρχεία.
- ❑ Χρησιμοποιούνται διάφορες μορφές αρχείων ανάμεσα στις οποίες:
 - **Comma-Separated-Value (.csv)** files. Κάθε γραμμή του αρχείου αντιπροσωπεύει μία γραμμή ενός array όπου τα στοιχεία της γραμμής διαχωρίζονται με comma ,
 - **Tab-Separated-Value (.tsv)** files. Κάθε γραμμή του αρχείου αντιπροσωπεύει μία γραμμή ενός array όπου τα στοιχεία της γραμμής διαχωρίζονται με tabs ή κενούς χαρακτήρες (spaces) ή συνδυασμό των δύο. Τα κενά δεν είναι απαραίτητα ομοιόμορφα,
 - Αρχεία με ακρωνύμια **.dat** ή **.txt** χρησιμοποιούνται επίσης για ονόματα αρχείων με κόμμα ή κενά να διαχωρίζουν τις τιμές
- ❑ Η Python έχει συναρτήσεις για ανάγνωση και γραφή αρχείων.

Πρόσβαση σε file

Για να έχουμε πρόσβαση σε file για ανάγνωση ή αποθήκευση δεδομένων χρησιμοποιούμε την εντολή open: **file_handler** = open(**filename**, **mode**)

file_handler: το αντικείμενο με το οποίο επικοινωνούμε με το file

filename: το όνομα του file σε " " συμπεριλαμβανομένου διαδρομή για directory
π.χ. "test.dat" ή "/home/fotis/data/test.dat"

mode: μεταβλητή που προσδιορίζει τι θα κάνουμε με το file

"r": read (default τιμή). Αν το file δεν υπάρχει θα δώσει error

"w": write. Αν το file δεν υπάρχει, το δημιουργεί

"a": append. Ανοίγει το file για αποθήκευση περισσότερων δεδομένων.
Αν το file δεν υπάρχει το δημιουργεί

"x": δημιουργία file. Αν το file υπάρχει επιστρέφει error

Πρόσβαση στα δεδομένα του file

Χρήση ενός **for loop** για ανάγνωση κάθε γραμμής του file →

Μέτρηση των γραμμών και εκτύπωση του αριθμού →

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)
```

```
$ python3 open.py
Line Count: 132045
```

Θα μπορούσαμε να διαβάσουμε ολόκληρο το file σε ένα και μόνο string →

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From fotis.ptohos
```

Ανάγνωση δεδομένων από αρχείο

```
my_file = open("HIVseries.csv") Δημιουργία ένα αντικείμενου (my_file) που μπορεί  
temp_data = [] να διαβάσει το file.  
for line in my_file:  
    print(line)  
    x, y = line.split(',')  
    temp_data += [ (float(x), float(y)) ]  
my_file.close()
```

Η 2^η γραμμή δημιουργεί μια άδεια λίστα για να αποθηκευτούν τα δεδομένα

Παρατηρήστε ότι η χρήση λίστας διευκολύνει γιατί δεν ξέρουμε πόσα δεδομένα έχει το αρχείο (πόσες γραμμές δηλαδή)

Σε κάθε επανάληψη του **for** loop, ανατίθεται στη μεταβλητή **line** που είναι τύπου string η επόμενη γραμμή του file που περιέχει τα δεδομένα

Μέσα στο loop υπάρχει η εντολή **line.split(',')** που αποτελεί μία μέθοδο των strings, για να διαχωρίσει τα δεδομένα της γραμμής, θεωρώντας ότι τα δεδομένα ξεχωρίζουν με ",". Επομένως **split(",")** χρησιμοποιεί το "," ως μέθοδο διαχωρισμού.

Για παράδειγμα έστω: **line = 123, 67**.

line.split(",") θα χωρίσει τη γραμμή στα επιμέρους δεδομένα '123' και '67' (strings)

Η αμέσως επόμενη γραμμή, μετατρέπει τα strings σε νούμερα και τα αποθηκεύει στο τέλος της υπάρχουσας λίστας.

Το πρόγραμμα συνεχίζεται έως ότου εξαντληθούν οι γραμμές του αρχείου

Μετά το τέλος του loop, κλείνουμε το αρχείο και μετατρέπουμε τη list σε numpy array

Αποθήκευση μεγάλου όγκου δεδομένων

Για να αποθηκεύσουμε πληθώρα δεδομένων θέλουμε να αποθηκεύσουμε τα δεδομένα σε κάποιο αρχείο

Για το σκοπό αυτό χρησιμοποιούμε την συνάρτηση **open** για να γράψουμε απευθείας στο file

Παράδειγμα: Αποθήκευση των πρώτων 10 δυνάμεων του 2 και 3

```
my_file = open('power.txt', 'w')
print( " N \t\t2**N\t\t3**N" )           # Print labels for columns.
print( "---\t\t---\t\t---" )           # Print separator.
my_file.write( " N \t\t2**N\t\t3**N\n" ) # Write labels to file.
my_file.write( "---\t\t---\t\t---\n" ) # Write separator to file.
### Loop over integers from 0 to 10 and print/write results.
for N in range(11):
    print( "{:d}\t\t{:d}\t\t{:d}".format(N, pow(2,N), pow(3,N)) )
    my_file.write( "{:d}\t\t{:d}\t\t{:d}\n".format(N, pow(2,N), pow(3,N)) )
my_file.close()
```

Ανοίγουμε ένα file για να γράψουμε όπως δηλώνει το πεδίο **'w'** στη συνάρτηση open.

Αν το αρχείο προ-υπάρχει τότε θα το σβήσει

Θα πρέπει να πούμε στην Python, που θα θέλουμε να ξεκινά και που να τελειώνει μια γραμμή. Ο χαρακτήρας **\n** εισάγει μια νέα γραμμή όπου και αν εμφανίζεται σε ένα string. Ο χαρακτήρας **\t** εισάγει ένα tab

Όταν δεν χρειάζεται πλέον πρόσβαση σε file είναι καλό να κλείνει χρησιμοποιώντας την εντολή **close()**