Περισσότερα σχετικά με συναρτήσεις

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.

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

- 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', 'isidentifier', '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

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

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

- 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

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

- Η συνάρτηση **startswith()** δίνει την αρχή μιας string

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

Παράδειγμα: εύρεση και εξαγωγή substring από string


```
>>> 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[maxsplits])**

  str: η string μεταβλητή που θέλουμε να διαχωρίσουμε

  sep: η string μεταβλητή που χρησιμοποιείται για τον διαχωρισμό. Αυτόματα θεωρείται το space. Προαιρετική μεταβλητή

  maxsplits: 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:

  - **C**omma-**S**eparated-**V**alue (**.csv**) files. Κάθε γραμμή του αρχείου αντιπροσωπεύει μία γραμμή ενός array όπου τα στοιχεία της γραμμής διαχωρίζονται με comma ,

  - **T**ab-**S**eparated-**V**alue (**.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 →

Μέτρημα 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")
temp_data = []
for line in my_file:
    print(line)
    x, y = line.split(',')
    temp_data += [ (float(x), float(y)) ]
my_file.close()
```

Δημιουργία ένα αντικείμενου (**my_file**) που μπορεί να διαβάζει το file.

Η 2η 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

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 κάποιο string. Ο χαρακτήρας **\t** εισάγει ένα tab

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

Χρήσιμες δομές

List Comprehension

Έστω ότι θέλουμε να εξαγάγουμε κάθε χαρακτήρα από ένα αντικείμενο τύπου string και να τον εισαγάγουμε σε μια λίστα

Ένας τρόπος είναι να χρησιμοποιήσουμε ένα *for loop*

```
LetList = []
word = 'banana'
for letter in word:
   LetList.append(letter)
print(LetList)
```

Output:

['b','a','n','a','n','a']

Η Python προσφέρει έναν πιο γρήγορο τρόπο για να κάνουμε την ίδια διαδικασία.

Η μέθοδος ονομάζεται **list comprehension** και παρέχει τον ορισμό και δημιουργία μιας λίστας στηριζόμενη σε προϋπάρχουσα λίστα

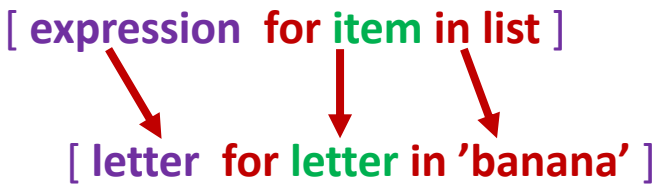
```
word = 'banana'
LetList = [ letter for letter in word]
print(LetList)
```

Στο παραπάνω παράδειγμα, δημιουργείται μια νέα λίστα με όνομα *LetList* και περιέχει τα στοιχεία της *word* που είναι η string μεταβλητή για τη λίστα/ακολουθία 'banana'

List Comprehension

Η σύνταξη της δομής list comprehension είναι η ακόλουθη:

[expression for item in list]



Στο προηγούμενο παράδειγμα είχαμε :

Παρόλο η σταθερά 'banana' είναι τύπου string και όχι list, η δομή ερμηνεύει την ακολουθία ως list. Το ίδιο συμβαίνει και με τα tuples.

Η δομή της list comprehension μπορεί να συνδυαστεί με μια άλλη δομή list comprehension, ή μια συνθήκη.

List Comprehension – με συνθήκη

Χρησιμοποιώντας μια συνθήκη στη θέση της expression στη list comprehension, μπορούμε να μεταβάλλουμε μια υπάρχουσα list ή tuple

```
AList = [ x for x in range(20) if x%4 == 0]
print(AList)
```

Output:

[4, 8,12,16]

Η λίστα AList γεμίζει με τα στοιχεία της λιστας [0,19] τα οποία διαιρούνται με το 4.

List Comprehension – με φωλιασμένη συνθήκη

Μπορεί στη θέση της συνθήκης να υπάρχει φωλιασμένη συνθήκη κάνοντας την συνθήκη αρκετά πιο πολύπλοκη για την επιλογή από την προϋπάρχουσα list

```
AList = [x for x in range(100) if x%2 == 0 if x%5 == 0]
print(AList)
```

Output:

```
[ 0,
 10,
 20,
 30,
 40,
 50,
 60,
 70,
 80,
 90]
```

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList αυτά που ικανοποιούν τις συνθήκες:

- x διαιρείται με το 2
- x διαιρείται με το 5

❑ Αν ικανοποιούνται και οι 2 συνθήκες, το στοιχείο x φυλάσσεται στην list AList

List Comprehension – με if … else… συνθήκη

Θα μπορούσε η συνθήκη υπόθεσης να είναι περισσότερο πολύπλοκη:

```
AList = [‘Even’ if x%2 == 0 else ‘Odd’ for x in range(10)]
print(AList)
```

Output:

[‘Even’,
‘Odd’,
’Even’,
‘Odd’,
‘Even’,
‘Odd’,
’Even’,
‘Odd’,
‘Even’,
‘Odd’]

Η list comprehension στην περίπτωση αυτή επιλέγει ως στοιχεία της list AList :

- ‘Even’ αν το x διαιρείται με το 2
- ‘Odd’ αν το x δεν διαιρείται με το 2

List Comprehension – με φωλιασμένο loop

Σε ασκήσεις του προηγούμενου εργαστηρίου υπάρχει στις προτεινόμενες λύσεις η χρήση ενός φωλιασμένου loop

```
AList = [[random.randint(0,100) for y in range(3)] for x in range(4)]
print(AList)
```

Στην περίπτωση αυτή δημιουργείται μια list με 3 στήλες και 4 γραμμές

Ο τρόπος που δουλεύει το φωλιασμένο loop στην περίπτωση αυτή, είναι να πάρει το x μια τιμή από το εύρος [0,4) (αυτή θα είναι η γραμμή της AList) και κατόπιν να επιλέξει τιμές του y στο διάστημα [0,3) (οι στήλες της γραμμής).

Η τιμή κάθε στοιχείου επιλέγεται τυχαία στο κλειστό διάστημα [0,100] με τη randint

Μέθοδος map()

Μια συνάρτηση μπορεί να εφαρμοστεί σε όλα τα στοιχεία ενός array ή μιας list χρησιμοποιώντας την μέθοδο **map**

Για παράδειγμα θα μπορούσαμε να έχουμε μια συνάρτηση που πολλαπλασιάζει τα στοιχεία μιας λίστας με 2 και αφαιρεί 1

```
def f(x):
    return 2*x - 1

newlist =list(map(f,oldlist))
```

Εφαρμόζει τη συνάρτηση f σε κάθε στοιχείο της λίστας oldlist και δημιουργεί μια νέα λίστα με τα αποτελέσματα με όνομα newlist