

Arrays στην Python

Πίνακες της βιβλιοθήκης NumPy: **numpy arrays**

Μέχρι τώρα είδαμε αντικείμενα τύπου list ή tuple τα οποία είναι γηγενή αντικείμενα της γλώσσας PYTHON

Υπάρχουν όμως και αντικείμενα τα οποία μοιάζουν περισσότερο με τους πίνακες μαθηματικών και αυτά τα αντικείμενα ορίζονται μέσα στη βιβλιοθήκη NumPy.

Από τη στιγμή που ορίζονται στη NumPy, θα πρέπει η βιβλιοθήκη αυτή να γίνεται πάντοτε `import` για να μπορέσουμε να έχουμε πρόσβαση στα αντικείμενα και μεθόδους της.

Στα επόμενα, υποθέτω ότι έχω κάνει `import` το `numpy`:

`import numpy as np`

Στο μάθημα θα προσπαθήσουμε να χρησιμοποιήσουμε `numpy arrays`

Μπορούμε να ορίσουμε έναν 1-D (μονοδιάστατο) πίνακα με την εντολή:

`Name = np.zeros(N)`

Ορίζεται ένας πίνακας `Name` (ή ότι άλλο όνομα θέλουμε) ενώ το αντικείμενο **`N`** ορίζει ότι το πλήθος των στοιχείων του πίνακα τα οποία έχουν μηδέν ως αρχικές τιμές

Το όρισμα της μεθόδου **`zeros`**, ορίζει τη μορφή του πίνακα

Θα μπορούσαμε να ορίσουμε ένα 2-D (δυσδιάστατο) πίνακα με την εντολή:

`Name = np.zeros((NR,NC))`

Προσοχή: το όρισμα της μεθόδου `zeros` είναι και πάλι ένα αντικείμενο!

Είναι ένα tuple και τα στοιχεία του, **`NR`** και **`NC`**, ορίζουν τον αριθμό των γραμμών και στηλών του πίνακα αντίστοιχα.

Θα μπορούσαμε να δώσουμε μια list: `Name=np.zeros([NR,NC])`

numpy arrays

```
>>>
>>> import numpy as np
>>>
>>> a = np.zeros(4)
>>> a
array([0., 0., 0., 0.])
>>>
>>> b = np.zeros((3,2))
>>> b
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
>>>
>>> c = np.zeros([3,2])
>>> c
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
>>>
>>> d = np.ones([3,2])
>>> d
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>>
>>>
>>> e = np.eye(3)
>>> e
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> □
```

1-D πίνακας με 4 στοιχεία

όλα τα στοιχεία είναι μηδέν

ορισμός 2-D πίνακας με tuple ως όρισμα, 3 γραμμές & 2 στήλες

ορισμός 2-D πίνακας με list ως όρισμα, 3 γραμμές & 2 στήλες

ορισμός 2-D πίνακας με list ως όρισμα, 3 γραμμές & 2 στήλες
 Η χρήση της μεθόδου **ones** γεμίζει τα στοιχεία του πίνακα με 1

Η χρήση της μεθόδου **eye(N)**

Η πλήρης σύνταξη είναι **eye(NR, NC, K, data-type, stored)**

Δημιουργία 2-D πίνακα $NR \times NC$ και όλα τα στοιχεία της K -διαγωνίου με 1 και όλα τα υπόλοιπα στοιχεία 0.

NR: υποχρεωτικό, αριθμός γραμμών

NC: μη υποχρεωτικό, αριθμός στηλών

K: μη υποχρεωτικό, αριθμός διαγωνίου, 0 κύρια, >0 πάνω, <0 κάτω

Data-type: int, float, complex, είδος των στοιχείων του πίνακα

numpy arrays

Μπορούμε να δημιουργήσουμε 1-D arrays, ως διανύσματα στήλης ή γραμμής

Διάνυσμα γραμμής: `V= np.zeros([1,3])` Διάνυσμα V με 1 γραμμή και 3 στήλες

Διάνυσμα στήλης: `C= np.zeros([3,1])` Διάνυσμα C με 3 γραμμές και 1 στήλη

Αποτελούν ιδιαίτερες περιπτώσεις 2-D πινάκων

Άλλοι τρόποι ορισμού και συμπλήρωσης στοιχείων πίνακα είναι:

Name = np.random.random([2,3]) Δημιουργία ενός πίνακα 2x3 και συμπλήρωση με τυχαίους αριθμούς

Name = np.random.rand(2,3) Ισοδύναμος τρόπος με την προηγούμενη εντολή

Name = np.array ([2,3,4]) Δημιουργία ενός 1-D πίνακα με 3 στοιχεία οι τιμές των οποίων είναι 2,3,4

Name = np.array ([[2], [3], [4]]) Μετατροπή κάθε στοιχείου σε list. Επιστροφή ενός 2-D πίνακα διανύσματος στήλης με 3 στοιχεία

Name = np.array ([[2, 3, 4], [5, 6, 7]]) Δημιουργία ενός πίνακα 2x3 και συμπλήρωση των στοιχείων της γραμμής με συγκεκριμένες τιμές

Μέθοδοι των numpy array

Οι arrays ως αντικείμενα της numpy έχουν μια σειρά από μεθόδους που βοηθούν στην ευκολότερη χρήση τους:

- | | | |
|----------------|---|--|
| sum() | Σύνταξη: <code>name.sum()</code> | Επιστρέφει το άθροισμα των στοιχείων του πίνακα |
| mean() | Σύνταξη: <code>name.mean()</code> | Επιστρέφει τη μέση τιμή των στοιχείων του πίνακα |
| std() | Σύνταξη: <code>name.std()</code> | Επιστρέφει την τυπική απόκλιση των στοιχείων του πίνακα |
| size() | Σύνταξη: <code>np.size(name)</code>
Σύνταξη: <code>name.size</code> | Επιστρέφει το πλήθος των στοιχείων του πίνακα
Διαφορετικός τρόπος ως μέθοδος του πίνακα, Δεν θέλει () |
| shape() | Σύνταξη: <code>np.shape (name)</code>
Σύνταξη: <code>name.shape</code> | Επιστρέφει τη μορφή του πίνακα, αριθμό γραμμών & στηλών
Διαφορετικός τρόπος ως μέθοδος του πίνακα, Δεν θέλει () |

numpy arrays - μέθοδος `np.arange`

Σε αρκετές εφαρμογές θέλουμε πίνακες αλλά δεν θέλουμε να καθορίσουμε ακριβώς τις τιμές κάθε στοιχείου του. Σε περιπτώσεις, χρειαζόμαστε πίνακες ή ακολουθίες που τα στοιχεία τους έχουν τιμές που είναι χωρισμένες σε συγκεκριμένες αποστάσεις. Δύο μέθοδοι για την numpy:

`np.arange(start, end, increment)`

start: προαιρετικό όρισμα. Η αρχική τιμή της ακολουθίας. Αν δεν οριστεί θεωρείται 0

end: υποχρεωτικό όρισμα. Η τελική τιμή της ακολουθίας. **Δεν** συμπεριλαμβάνεται στην ακολουθία

increment: προαιρετικό όρισμα. Η αύξηση των τιμών της ακολουθίας.
Αν δεν οριστεί θεωρείται 1

Δημιουργείται ο πίνακας με στοιχεία των οποίων οι τιμές είναι στο διάστημα [start, end) και διαφέρουν μεταξύ τους κατά increment

```
a = np.arange(2.1, 5.1, 0.1)
```

Η μέθοδος `np.arange()` είναι παρόμοια με την μέθοδο `range()` της python. Ωστόσο:

Η `np.arange` δημιουργεί πίνακα ενώ η `range()` μία list

Οι τιμές που δίνει η `np.arange` είναι τύπου `int` ή `float` ενώ με `range()` μόνο `int`

Προσοχή: για περιπτώσεις μη-`int` τα αποτελέσματα δεν είναι σταθερά και πρέπει να χρησιμοποιείται η μέθοδος `np.linspace`

numpy arrays - μέθοδος np.linspace

Η μέθοδος np.linspace() είναι παρόμοια με την μέθοδο np.arange() ωστόσο Προσφέρει σωστή αντιμετώπιση των ακραίων τιμών

Η σύνταξη της εντολής είναι: **np.linspace(start, end, number-of-steps)**

Δημιουργεί έναν 1-D πίνακα με τιμές που έχουν ίσες αποστάσεις, ξεκινούν από start, **συμπεριλαμβάνουν την τελική τιμή end** και είναι number-of-steps σε πλήθος

Δεν χρειάζεται να ορίσετε την απόσταση μεταξύ διαδοχικών αριθμών, αλλά η python υπολογίζει τη σωστή απόσταση

Ποιες οι τιμές των πινάκων a και b με τις δυο παρακάτω εντολές: ?

```
a = np.arange(0,10,2)
```

```
b = np.linspace(0,10,6)
```

```
>>>
>>> a=np.arange(0,10,2)
>>> b=np.linspace(0,10,6)
>>>
>>> a
array([0, 2, 4, 6, 8])
>>>
>>> b
array([ 0.,  2.,  4.,  6.,  8., 10.])
>>> 
```

Ποια από τις δύο μεθόδους θα χρησιμοποιούσατε για να βρείτε τις τιμές μιας συνάρτησης σε ένα συγκεκριμένο διάστημα?

np.linspace: γιατί επιτρέπει τον ακριβή προσδιορισμό της αρχικής και τελικής τιμής των στοιχείων του πίνακα

numpy arrays – σύνδεση πινάκων (concatenation)

Υπάρχουν δύο μέθοδοι για να κατασκευαστεί ένας πίνακας συνδέοντας μικρότερους μεταξύ τους

Οι μέθοδοι αυτοί είναι:

np.hstack(list or tuple): Ο πίνακας έχει τον ίδιο αριθμό γραμμών όπως και οι αρχικοί επιμέρους πίνακες.

Οι πίνακες που χρησιμοποιούνται πρέπει να έχουν τον ίδιο αριθμό γραμμών

np.vstack(list or tuple): Ο πίνακας έχει τον ίδιο αριθμό στηλών όπως και οι αρχικοί επιμέρους πίνακες.

Οι πίνακες που χρησιμοποιούνται πρέπει να έχουν τον ίδιο αριθμό στηλών

```
>>> a=np.zeros( (2,3) )
>>> b=np.ones( (2,3) )
>>> h=np.hstack([a,b])
>>> v=np.vstack([a,b])
>>>
>>> h
array([[0., 0., 0., 1., 1., 1.],
       [0., 0., 0., 1., 1., 1.]])
>>>
>>>
>>> v
array([[0., 0., 0.],
       [0., 0., 0.],
       [1., 1., 1.],
       [1., 1., 1.]])
```


numpy arrays – Πρόσβαση στα στοιχεία του πίνακα

Για να αποκτήσουμε πρόσβαση στα στοιχεία ενός numpy πίνακα:

```
A = np.array( [2, 4, 5] )
A[0]
A[1] = 100
print(A)
```

Η θέση του στοιχείου του πίνακα που επιθυμούμε περικλείεται σε `[]` και όχι σε παρενθέσεις

Η αρίθμηση των στοιχείων του πίνακα ξεκινά από το μηδέν και όχι το 1. Για N-πλήθος στοιχείων πηγαίνει από 0 έως N-1

Αρνητικοί δείκτες δηλώνουν πρόσβαση από δεξιά προς τα αριστερά στα στοιχεία ενός πίνακα με `A[-1]` το τελευταίο στοιχείο του πίνακα (N-th)

```
A = np.array( [ [2, 3, 5], [7, 11, 13] ] )
A[0]           Πρόσβαση στα στοιχεία της 1ης γραμμής
A[0][1]        Πρόσβαση στο στοιχείο της 1ης γραμμής και 2ης στήλης
A[1][2] = 999  Ανάθεση στο στοιχείο της 2ης γραμμής και 3ης στήλης
```

Θα μπορούσαμε να έχουμε το ίδιο αποτέλεσμα χρησιμοποιώντας τη συντομότερη γραφή που φαίνεται παρακάτω:

```
A = np.array( [ [2, 3, 5], [7, 11, 13] ] )
A[0, 1]
A[1, 2] = 999
```

Αναθέσεις τιμών σε μεταβλητές και πίνακες

Εν γένει υπάρχει μικρή διαφορά μεταξύ ανάθεσης τιμής σε μεταβλητή και ανάθεση τιμής σε στοιχείο πίνακα.

Ωστόσο υπάρχει διαφορά στον τρόπο με τον οποίο οι αναθέσεις αυτές χειρίζονται από την Python.

Η ανάθεση και εξέταση ενός στοιχείου του πίνακα γίνεται με μεθόδους των πινάκων της βιβλιοθήκης `numpy`

Η ανάθεση και εξέταση τιμής μιας μεταβλητής γίνεται με Python, η οποία συνδέει την μεταβλητή με ένα νέο αντικείμενο.

Δεν υπάρχει διαφορά έως ότου υπάρξουν δύο μεταβλητές που συνδέονται με το ίδιο αντικείμενο.

```
f = 2.5  
g = f  
A = np.zeros(3)  
B = A
```

Όταν δώσουμε τις παρακάτω εντολές.

```
g = 3.5  
A[0] = 1  
B[1] = 3
```

Η μεταβλητή *f* παραμένει αμετάβλητη αλλά οι αλλαγές στον ένα πίνακα διαδίδονται και στην μεταβλητή που έχει πρόσβαση στις ίδιες θέσεις μνήμης.

Εξαγωγή πληροφορίας από τμήμα πίνακα - **Slicing**

Συχνά χρειάζεται να πάρουμε πληροφορία για ένα τμήμα ενός πίνακα (ομάδα στοιχείων) και όχι για ένα στοιχείο του μόνο

Για παράδειγμα μπορεί να θέλουμε να εξετάσουμε τα 10 πρώτα στοιχεία ενός πίνακα ή τα στοιχεία κάποιας στήλης του.

Αυτό ονομάζεται slicing ενός πίνακα και επιτυγχάνεται με την χρήση του τελεστή :

Για έναν 1-D πίνακα η πιο γενική σύνταξη είναι: `A[start:end:stride]`

Για παράδειγμα: `A[start]`, `A[start+stride]`, `A[start+2*stride]`, ... `A[start+M*stride]`

όπου $start + M*stride < end$

Παράδειγμα: Έστω θέλουμε τα 10 πρώτα στοιχεία ενός πίνακα `A[0:10:1]` ή `A[:10]`

Η ίδια σύνταξη ισχύει και για πολυδιάστατους πίνακες και δημιουργία slice σε κάποια διάσταση.

Παράδειγμα: Έστω θέλουμε τμήμα της 3^{ης} στήλης ενός πίνακα: `A[start:end:stride,2]`

Αν δοθεί ένας πίνακας και δεν γνωρίζουμε τον αριθμό των γραμμών τότε μπορούμε να τις βρούμε ως

```
N = np.size(A, 1)
```

```
x = A[0:N:1, 0]
```

```
y = A[0:N:1, 1]
```

Η χρήση απλά του τελεστή : αντιπροσωπεύει όλες τις δυνατές τιμές

```
x = A[:, 0]
```

```
y = A[:, 1]
```

Ο τελεστής **is** και **is not**

Οι τελεστές == και !=

Όπως έχουμε δει, όταν θέλουμε να εξετάσουμε αν δύο αντικείμενα έχουν το ίδιο περιεχόμενο, χρησιμοποιούμε τους τελεστές == και !=

Ο παραπάνω έλεγχος είναι διαφορετικός από το να προσπαθήσουμε να ελέγξουμε αν **τα δύο αντικείμενα είναι το ίδιο και το αυτό**. Δηλαδή αν αναφέρονται στην ίδια διεύθυνση μνήμης


Ο έλεγχος αυτός γίνεται με τους τελεστές:

is και **is not**

Οι δύο λίστες είναι δύο διαφορετικά αντικείμενα

```
>>> 2 == 2
True
>>> 2 != 2
False
>>>
```

```
>>> [2,3] == [3,2]
False
>>> [2,3] == [2,3]
True
>>> [2,3] is [2,3]
False
```



Τελεστές is και is not

True: Τα δύο αντικείμενα είναι το ίδιο και το αυτό και δείχνουν στην ίδια διεύθυνση της μνήμης.

True: Τα δύο αντικείμενα είναι δύο κενές λίστες και το περιεχόμενό τους είναι ίδιο

False: Τα δύο αντικείμενα είναι δύο διαφορετικές λίστες που αντιστοιχούν σε διαφορετικές θέσεις στη μνήμη

False: Η ένωση (concatenation) δύο λιστών δίνει σαν αποτέλεσμα μια νέα λίστα και επομένως δεν αντιστοιχεί πλέον στην ίδια διεύθυνση μνήμης.

Ποιο το αποτέλεσμα του παρακάτω;

```
>>> a=5
>>> b=5
>>> if a is b:
>>>     print("True")
>>> else:
>>>     print("False")
```

True: οι μεταβλητές a και b αναφέρονται στο ίδιο αντικείμενο (5) που είναι αποθηκευμένο στην ίδια διεύθυνση μνήμης

```
>>> list1[]
>>> list2[]
>>> list3 = list1
>>> if list3 is list1:
>>>     print("True")
>>> else:
>>>     print("False")
>>> if list1 == list2:
>>>     print("True")
>>> else:
>>>     print("False")
>>> if list1 is list2:
>>>     print("True")
>>> else:
>>>     print("False")
>>> list3 = list3 + list2
>>> if list1 is list3:
>>>     print("True")
>>> else:
>>>     print("False")
```

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(), floa()].

```
>>> 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**)

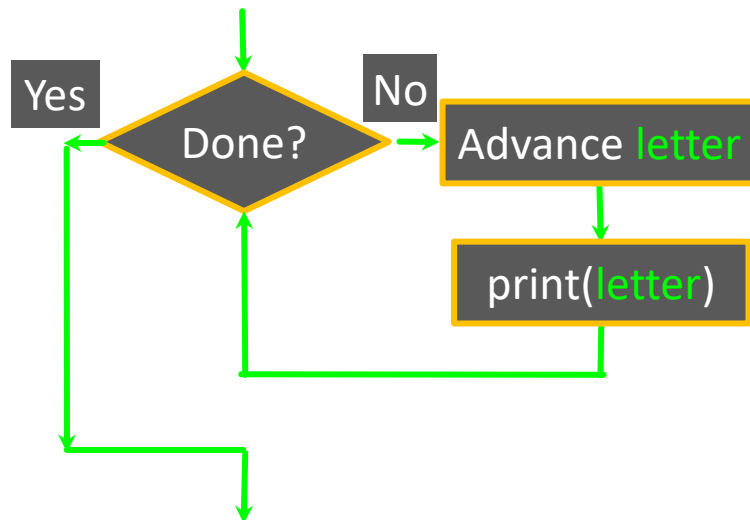
Το κύριο σώμα του loop (οι εντολές του loop) εκτελούνται μια φορά για κάθε τιμή της ακολουθίας

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 αντικειμένου 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

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

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
```

Αρχεία

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\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\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()**

Μέτρηση γραμμών σε ένα file

Ανοίγουμε ένα file για ανάγνωση

Χρήση ενός for loop για ανάγνωση
κάθε γραμμής του file

Μέτρηση των γραμμών και
εκτύπωση του αριθμού

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
```

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

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

Εισαγωγή 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. Ανάλογοι μέθοδοι υπάρχουν και στο module NumPy
- ❑ Το module SciPy περιέχει μία μέθοδο, **scipy.io**, που επιτρέπει την ανάγνωση και γραφή αρχείων σε διάφορες μορφές

Εισαγωγή δεδομένων στην Python

- ❑ Θα πρέπει πρώτα να δείτε που βρίσκεται το αρχείο που θέλετε να εισαγάγετε στο πρόγραμμά σας.
- ❑ Για να εισαγάγετε τα δεδομένα ενός αρχείου σε έναν numpy array μπορείτε να δώσετε την εντολή **np.loadtxt**
 - Η εντολή προσπαθεί να εισαγάγει ένα txt file και να μετατρέψει τα δεδομένα σε έναν array. Υποθέτει ότι το αρχείο είναι .tsv αρχείο.
- ❑ Για εισαγωγή από ένα .csv αρχείο πρέπει να δώσετε την οδηγία για το είδος του αρχείου:
data_set = np.loadtxt ("Test.csv", delimiter=',')
- ❑ Προσέξτε πως χρησιμοποιείται η εντολή
 - Η συνάρτηση επιστρέφει έναν array, data_set
 - Θα πρέπει να δώσουμε το όνομα του αρχείου στη συνάρτηση **np.loadtxt**
 - Θα πρέπει να προσδιορίσουμε τον τρόπο που περιέχονται τα στοιχεία στο file ως string. Στην προκειμένη περίπτωση ","
- ❑ Ένας τρόπος για να περάσουμε το όνομα του αρχείου είναι δίνοντας το απόλυτο path (directory) του αρχείου στον υπολογιστή.

```
data_file = "/Users/username/Downloads/PMLdata/01HIVseries/HIVseries.csv"  
data_set = np.loadtxt(data_file, delimiter=',')
```

Εισαγωγή δεδομένων στην Python

```
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()
data_set = np.array(temp_data)
```

Η 1^η γραμμή δημιουργεί ένα αντικείμενο που μπορεί να διαβάσει το file

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

Παρατηρήστε ότι χρησιμοποιούμε λίστα και όχι array, γιατί δεν ξέρουμε πόσα δεδομένα έχει το αρχείο (πόσες γραμμές δηλαδή)

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

Μέσα στο loop υπάρχει η εντολή ***line.split(',')*** που αποτελεί μία μέθοδο των strings, για να αναλύσει την γραμμή σε νούμερα, χρησιμοποιώντας το "," για να διαχωρίσει τα νούμερα.

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

Εισαγωγή δεδομένων από το web

Μπορούμε να εισαγάγουμε δεδομένα διαβάζοντας κάποιο αρχείο απευθείας από το web

Αυτό μπορούμε να το κάνουμε με τη εισαγωγή της βιβλιοθήκης **urllib** και τις ακόλουθες εντολές:

```
import numpy as np
import urllib.request
web_file = urllib.request.urlopen("http://www2.ucy.ac.cy/~fotis/phy140/LAB06/mydata.txt")
data_set = np.loadtxt(web_file, skiprows=3, usecols=(0,1,2), unpack=False)
```

Μετά την κλήση της συνάρτησης **urlopen**, η Python μπορεί να διαβάσει τα data στο **web_file** χρησιμοποιώντας την μέθοδο **np.loadtxt** σαν να ήταν αρχείο που βρίσκονταν στο δίσκο του υπολογιστή μας.

skiprows=n Αν υπάρχουν σχόλια ή γραμμές που θέλουμε να αγνοηθούν (n αριθμός γραμμών)

usecols=(n1,n2,...,n3) Αν υπάρχουν πολλές στήλες, ποιες θα θέλαμε να χρησιμοποιήσουμε ξεκινώντας η αρίθμηση από το 0. **By default** διαβάζονται όλες

unpack=False/True Αν είναι True επιστρέφει τον πίνακα σε ανάστροφη μορφή.
Default False

```
>>> from io import StringIO # StringIO behaves like a file object
>>> c = StringIO("0 1\n2 3")
>>> np.loadtxt(c)
array([[0., 1.],
       [2., 3.]])
```

delimiter=" " String που δηλώνει πως χωρίζονται τα δεδομένα στο file. **Default** κενά " "

Αποθήκευση δεδομένων

```
x = np.linspace(0, 1, 1001)
y = 3*np.sin(x)**3 - np.sin(x)

np.save('x_values', x)
np.save('y_values', y)
np.savetxt('x_values.dat', x)
np.savetxt('y_values.dat', y)
np.savez('xy_values', x_vals=x, y_vals=y)
```

Οι εντολές **np.save** και **np.savez** δημιουργούν αρχεία με το ακρωνύμιο **.npy** και **.npz** αντίστοιχα και αποθηκεύουν τα δεδομένα των arrays *x*, και *y*.

Τα δεδομένα αποθηκεύονται σε binary μορφή - όχι αναγνώσιμα από άνθρωπο

Η εντολή **np.savetxt** αποθηκεύει τα δεδομένα ως text σε αρχείο με το ακρωνύμιο **.dat** που δίνεται στην εντολή

Ο πίνακας που αποθηκεύεται μπορεί να είναι 1-D ή 2-D

Τα δεδομένα αποθηκεύονται διατηρώντας κενά μεταξύ διαφορετικών τιμών (.tsv αρχείο). Αν θέλατε να αποθηκεύσετε τις τιμές διατηρώντας “ , ” ως διαχωρισμό θα πρέπει να χρησιμοποιηθεί το πεδίο του delimiter (delimiter=“,”)

```
np.savetxt("x_values.csv", x, delimiter=“,”)
```

Αποθήκευση δεδομένων

Για να χρησιμοποιηθούν files στα οποία έχουμε αποθηκεύσει δεδομένα μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις **np.load()** και **np.loadtxt()**.

```
x2 = np.load('x_values.npy')  
y2 = np.loadtxt('y_values.dat')
```

Οι μεταβλητές x2 και y2 είναι δύο arrays

Συναρτήσεις / Functions

Συναρτήσεις χρήστη

Σε αρκετές περιπτώσεις χρειάζεται να επαναλάβουμε τις ίδιες εντολές κώδικα σε διάφορα σημεία του προγράμματός μας. Στις περιπτώσεις αυτές μπορούμε να ορίσουμε μια δομή η οποία εμπεριέχει κάποιο όνομα που μπορεί να κληθεί σε κάποιο σημείο του προγράμματος που απαιτείται για το αποτέλεσμα ενός υπολογισμού.

Έστω για παράδειγμα, θέλουμε να υπολογίσουμε είτε κάποιο άθροισμα ή το παραγοντικό ενός αριθμού.

Θυμηθείτε ότι το παραγοντικό ενός αριθμού ορίζεται ως:

$$n! = \prod_{k=1}^n k$$

Μπορούμε να γράψουμε στην Python το πρόγραμμα ως

```
#!/usr/bin/python3

# Υπολογισμος του παραγοντικου

f=1.0
n = int(input("Give a number to calculate the factorial "))
for k in range(1,n+1):
    f *= k
print("The factorial of %d is %d" % (n,f))
```

Ωστόσο δεν θέλουμε να επαναλαμβάνουμε τον ίδιο κώδικα αν θέλουμε να υπολογίσουμε το παραγοντικό διάφορων αριθμών

Συναρτήσεις χρήστη

Ένας περισσότερο αποδοτικός τρόπος είναι να γράψουμε μια δική μας συνάρτηση για τον υπολογισμό του παραγοντικού ενός οποιοδήποτε αριθμού που περνάμε ως όρισμα

```
#!/usr/bin/python3

# Υπολογισμός του παραγοντικού με συνάρτηση

def factorial(n): ← Ορισμός μιας function
    f=1.0
    for k in range(1,n+1):
        f *= k
    return f

# Κύριο πρόγραμμα

n = int(input("Give a number to calculate the factorial "))
f = factorial(n) ← Call of the function
print("The factorial of %d is %d" % (n,f))
```

Με την κλήση της συνάρτησης το πρόγραμμα πηγαίνει στο τμήμα που ορίζεται από την εντολή `def factorial(n):`

Ακολουθούν οι υπολογισμοί του παραγοντικού και όταν φθάσει στην τελική γραμμή `return f` επιστρέφει στο σημείο του προγράμματος που κάλεσε την συνάρτηση και η τιμή της συνάρτησης είναι αυτή μετά το `return` που είναι η τελική τιμή της μεταβλητής `f`

Προσοχή: οι μεταβλητές ή οι τιμές τους που ορίζονται μέσα στη συνάρτηση, υπάρχουν όσο είμαστε μέσα στο τμήμα της συνάρτησης και καταστρέφονται μετά **local variables**

Συναρτήσεις χρήστη

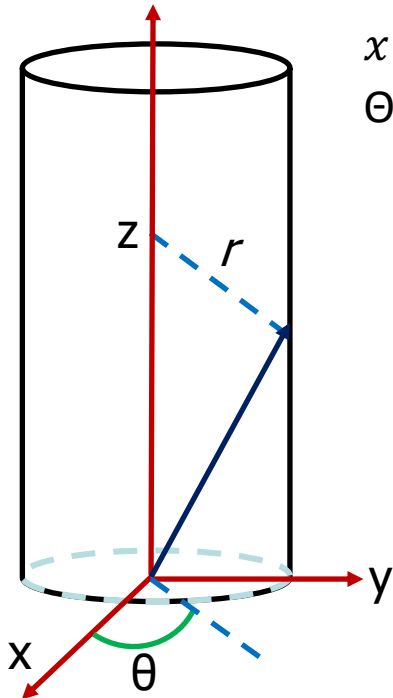
Οι συναρτήσεις μπορούν να έχουν περισσότερα από 1 ορίσματα

Έστω για παράδειγμα θέλουμε να υπολογίσουμε τη απόσταση ενός σημείου από την αρχή του συστήματος συντεταγμένων και το σημείο δίνεται σε κυλινδρικές συντεταγμένες, r , θ , z

Ο απλούστερος τρόπος για να κάνουμε τους υπολογισμούς είναι να μετρατρέψουμε τις κυλινδρικές συντεταγμένες σε καρτεσιανές όπως φαίνεται στο διπλανό σχήμα:

$$x = r \cos \theta \quad y = r \sin \theta \quad \text{και} \quad d = \sqrt{x^2 + y^2 + z^2}$$

Θα γράφαμε επομένως μια συνάρτηση ως:



```
#!/usr/bin/python3
```

```
from math import cos, sin, sqrt
```

```
def distance(r,theta,z):
```

```
    x = r*cos(theta)
```

```
    y = r*sin(theta)
```

```
    d = sqrt(x**2 + y**2 + z**2)
```

```
    return d
```

```
# Kyrio programma
```

```
r = float(input("Give the radius of the circle "))
```

```
ang = float(input("Give the azimuthal angle "))
```

```
thez = float(input("Give the z coordinate "))
```

```
dist = distance(r,ang,thez)
```

```
print("The particle is %.5f units for the origin" % dist)
```

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορούν να επιστρέψουν ως αποτέλεσμα οποιοδήποτε τύπο αντικειμένου integer, float, string, complex, list, array.

Θα μπορούσαμε να γράψουμε μια συνάρτηση που μετατρέπει μεταξύ πολικών και καρτεσιανών συντεταγμένων ως εξής:

```
#!/usr/bin/python3

#Metatropi apo polikes se kartesianes suntetagmenes
from math import cos, sin, sqrt,pi

def cartesian(r,theta):
    x = r*cos(theta)
    y = r*sin(theta)
    position = [x,y]
    return position

# Kyrio programma

r = float(input("Give the radius of the circle "))
ang = float(input("Give the azimuthal angle "))
ang = ang*pi/180.
cart = cartesian(r,ang)
print("The particle is @ x = %.5f and y = %.5f" % (cart[0],cart[1]))
□
```

Θα μπορούσαμε αντί να επιστρέψουμε τον array απευθείας ως `return array([x,y],float)`

Θα μπορούσαμε να επιστρέψουμε επίσης **return x,y** και να την καλέσουμε ως

α,b = cartesian(r,theta)

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορεί να μην επιστρέφουν κάποια τιμή.

Επιτρέπεται οι συναρτήσεις να μην περιέχουν `return` εντολή

Γιατί να θέλει να γράψει κάποιος μια τέτοια συνάρτηση;

Έστω θέλετε να τυπώσετε τις συνιστώσες ενός διανύσματος με μια εντολή της μορφής:

```
print("(" + r[0] + ", " + r[1] + ", " + r[2] + ")")
```

Επειδή είναι αρκετά μπερδεμένη έκφραση, θα μπορούσατε να γράψετε μια συνάρτηση που να κάνει αυτό όταν της περνάτε ένα διάνυσμα:

```
#!/usr/bin/python3

#Metatropi apo polikes se kartesianes suntetagmenes
from math import cos, sin, sqrt, pi

def print_vector(r):
    print("(" + r[0] + ", " + r[1] + ", " + r[2] + ")")

# Kyrio programma

rp = [3,2,1]
print_vector(rp)
```

Συναρτήσεις χρήστη

Οι συναρτήσεις μπορεί να οριστούν οπουδήποτε στο πρόγραμμα αλλά πάντοτε η εντολή **def function_name()** : πρέπει να προηγείται της πρώτης κλήσης της συνάρτησης

Καλή πρακτική να βρίσκονται στην αρχή του προγράμματος που γράφεται

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 λιστα με τα αποτελέσματα με όνομα newlist

Μπορείτε να βάλετε τους ορισμούς διαφόρων συναρτήσεων σε ένα file, p.x. myfuncs.py
Όταν θέλετε να χρησιμοποιήσετε κάποια συνάρτηση μπορείτε να δώσετε την εντολή:

from myfuncs import afunction όπου **afunction** είναι το όνομα μιας συνάρτησης

Οπτικοποίηση δεδομένων

Οπτικοποίηση δεδομένων

Το γεγονός ότι οι μελέτες διαφόρων φυσικών διεργασιών γίνεται με πειραματικές διατάξεις και επομένως μετρήσεις διαφόρων φυσικών μεγεθών, σημαίνει ότι θα πρέπει να είμαστε σε θέση να μελετήσουμε συσχετίσεις μεταξύ των μεγεθών αλλά και τα χαρακτηριστικά του κάθε μεγέθους που μετρήθηκε.

Θα πρέπει επομένως να μπορούμε να οπτικοποιήσουμε με ένα περισσότερο γραφικό τρόπο το τι έχουμε μετρήσει

Θα πρέπει επομένως να μπορούμε να χρησιμοποιήσουμε τα γραφικά πακέτα τα οποία μας προσφέρονται από την PYTHON

Θα χρησιμοποιήσουμε την βιβλιοθήκη **matplotlib**

Η εντολή που θα πρέπει να δώσουμε είναι: **import matplotlib.pyplot as plt**

Η εντολή `plot` και σχετιζόμενες εντολές

Η μέθοδος `pyplot` θα κάνει ένα 2-Δ γράφημα αν δοθεί ένα σετ ζευγών (x,y).

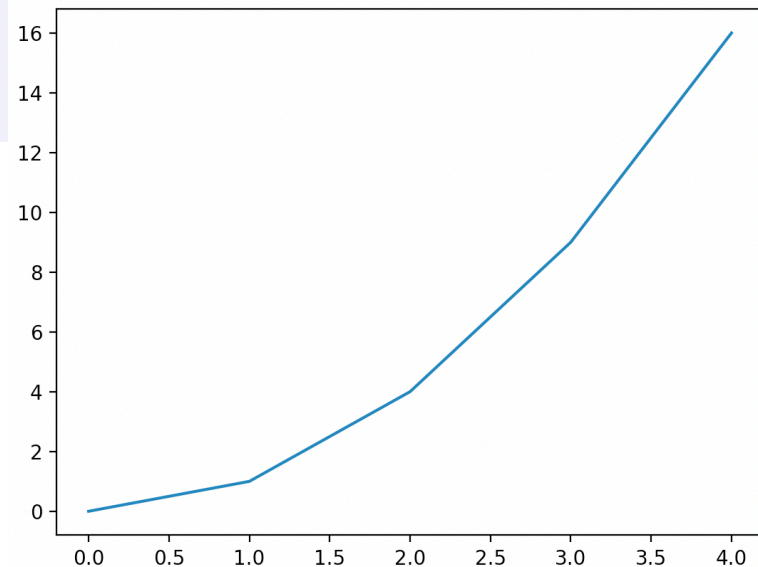
Για παράδειγμα θα έχουμε:

```
import numpy as np, matplotlib.pyplot as plt
num_points = 5
x_min, x_max = 0, 4
x_values = np.linspace(x_min, x_max, num_points)
y_values = x_values**2
plt.plot(x_values, y_values)
plt.show()
```

Η εντολή **`plt.show()`** χρειάζεται για να δούμε το plots τα οποία έχουν δημιουργηθεί.

Μπορούμε να δημιουργήσουμε διάφορα γραφήματα χρησιμοποιώντας παράθυρα με την εντολή **`plt.figure()`**.

Για να δούμε όλα τα αποτελέσματα ταυτόχρονα μπορούμε να δώσουμε την εντολή **`plt.show()`**



Η εντολή plot και σχετιζόμενες εντολές

Έστω για παράδειγμα θέλουμε το γράφημα του συνημιτόνου από -2 έως 1.

Θα πρέπει πρώτα να διακριτοποιήσουμε τις τιμές κατά μήκος του x-άξονα και να εκτιμήσουμε την τιμή της συνάρτησης για κάθε τιμή του x.

```
import numpy as np
import matplotlib.pyplot as plt

xvals = np.arange(-1, 2, 0.01)
yvals = np.cos(xvals)
plt.plot(xvals, yvals)
plt.show()
```

Έστω ότι θέλουμε να συγκρίνουμε το αποτέλεσμα αυτό με το αποτέλεσμα που θα είχαμε αν χρησιμοποιούσαμε το ανάπτυγμα Taylor του συνημιτόνου και κρατούσαμε μόνο τους δύο πρώτους όρους. Επομένως θα είχαμε $\cos(x) = 1 - 0.5 * x * x$

```
newyvals = 1 - 0.5*xvals*xvals
plt.plot(xvals, newyvals, 'r--')
plt.title('Example plots')
plt.xlabel('Input')
plt.ylabel('Function values')
plt.show()
```

Το 3^ο όρισμα προσδιορίζει να γίνει το γράφημα με με κοκκινή (r) διακεκομμένη γραμμή (--)

Σε όλα τα γραφήματα θα πρέπει να υπάρχουν κατάλληλα ονοματισμένοι άξονες. Αυτό γίνεται με τις εντολές **title**, **xlabel** και **ylabel** που δίνουν τίτλο και ονόματα στους x, y- άξονες

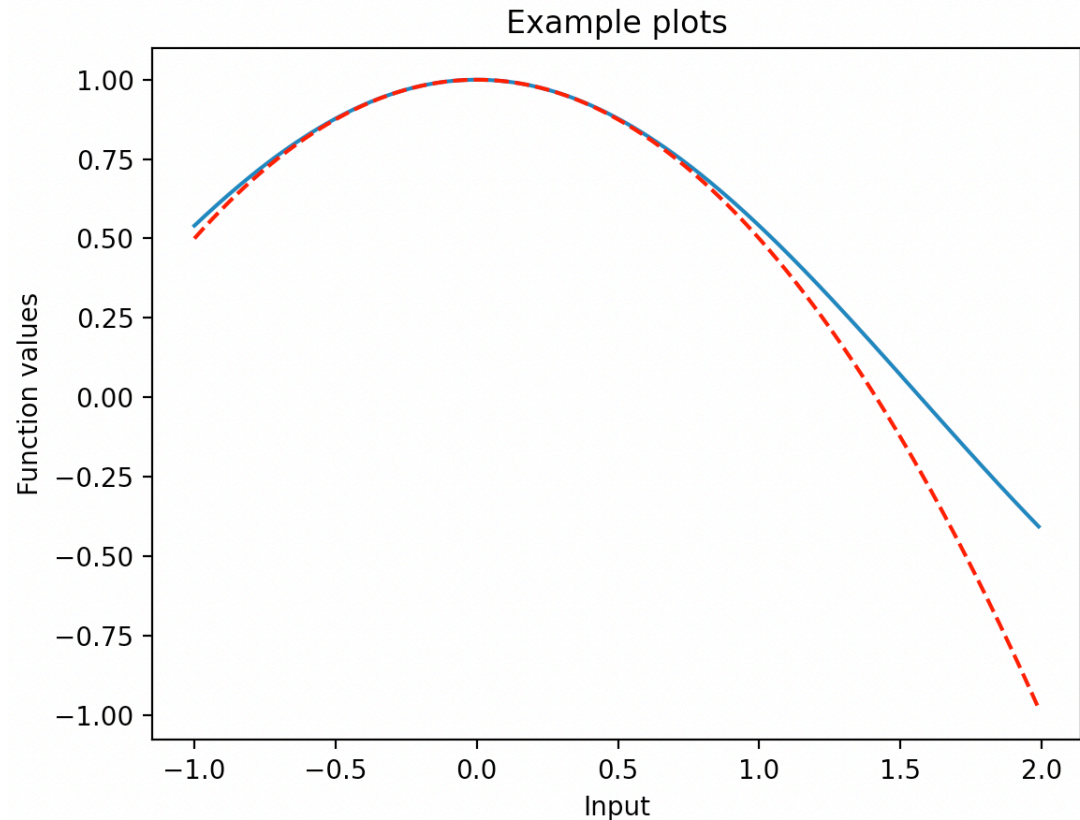
Η εντολή plot και σχετιζόμενες εντολές

Αν θέλουμε και τα δύο γραφήματα μαζί, θα πρέπει να δώσουμε την εντολή `show()` στο τέλος αφού έχουμε κάνει όλα τα γραφήματα που θέλουμε:

```
import numpy as np
import matplotlib.pyplot as plt

xvals = np.arange(-1, 2, 0.01)
yvals = np.cos(xvals)
plt.plot(xvals, yvals)
#plt.show()

newyvals = 1 - 0.5*xvals*xvals
plt.plot(xvals, newyvals, 'r--')
plt.title('Example plots')
plt.xlabel('Input')
plt.ylabel('Function values')
plt.show()
```



Οι επιλογές στην εντολή plot

Μπορούμε να αλλάξουμε το χρώμα της καμπύλης του γραφήματος δίνοντας το όρισμα:

plt.plot(x_value,y_value,'r')

r: κόκκινο **k**: μαύρο

b: μπλέ **g**: πράσινο

Μπορούμε να αλλάξουμε το style της γραμμής της καμπύλης του γραφήματος:

'.' διακεκομμένη γραμμή **'-'** συμπαγής γραμμή

'--' dashed γραμμή

Μπορούμε να αλλάξουμε το σύμβολο των σημείων του γραφήματος:

'.' μικρή κουκκίδα σε κάθε σημείο **'o'** ανοικτός κύκλος

Μπορούμε να αλλάξουμε τα όρια ενός άξονα σε κάποιο γράφημα δίνοντας την εντολής:

plt.xlim([x_min,x_max]) ή **plt.xlim((x_min,x_max))** ή **plt.xlim(xmax=x_max,xmin=x_min)**

plt.ylim([ymin,ymax])

Μπορούμε να δούμε διάφορες επιλογές της εντολής plot δίνοντας την εντολή: **help(plt.plot)**

Άξονες λογαριθμικός ή ημιλογαριθμικός

Μπορούμε να αλλάξουμε τον γ-άξονα από γραμμικό σε ημιλογαριθμικό άξονα:

plt.semilogy

Αν επιθυμούμε ημιλογαριθμικό τον x-άξονα τότε μπορούμε να γράψουμε:

plt.semilogx

Αν επιθυμούμε λογαριθμικούς και τους δύο άξονες τότε δίνουμε την εντολή: :

plt.loglog

Κάνοντας τα plots ευπαρουσίαστα

Τίτλος:

Μπορούμε να δώσουμε τίτλο στο γράφημα με την εντολή:

```
plt.title("Title", size=24, weight='bold')
```

Ονομασία αξόνων:

```
plt.xlabel("speed")
```

```
plt.ylabel("kinetic energy")
```

Αλλαγή του στυλ της γραμμής:

Μπορούμε να αλλάξουμε το στυλ της γραμμής αφού σχεδιάσουμε τη γραμμή:

```
plt.setp (lines[0], linestyle='--', linewidth=3, color='r')
```

Αλλαγή της πρώτης γραμμής (lines[0]) σε --, με πάχος 3 και χρώματος κόκκινο

Εισαγωγή legend: (περιγραφή μιας γραμμής για παράδειγμα)

```
plt.plot(xvalue, yvalue, label="Energy 1")
```

```
plt.plot(xvalue, yvalue*yvalue, label="Energy 2")
```

```
plt.legend()    εμφανίζει το legend
```

Εισαγωγή σφαλμάτων:

Μπορούμε να κάνουμε ένα γράφημα με σφάλματα (error bars)

```
plt.errorbar(xvalue, yvalue, yerr=y_errors, xerr=x_errors)
```

Πολλαπλές καμπύλες στο ίδιο γράφημα

Μπορούμε να κάνουμε πολλές καμπύλες στο ίδιο γράφημα:

```
x = np.linspace(0, 1, 51)
y1 = np.exp(x)
y2 = x**2
plt.plot(x, y1, x, y2)    Εισαγωγή πολλαπλών ζευγών (x,y1) και (x,y2)
```

Η python θα διαλέξει διαφορετικά χρώματα μόνο του

Αποθήκευση γραφημάτων

Μπορείτε να αποθηκεύσετε τα γραφήματα σε pdf format με την εντολή:

```
plt.savefig("myfigures.pdf")
```