Ακολουθιακοί Τύποι Δεδομένων

Πλειάδες, Λίστες και Αλφαριθμητικά

Ακολουθιακοί Τύποι Δεδομένων

- Πλειάδα (tuple)
  1. Μια απλή αμετάβλητη διατεταγμένη ακολουθία δεδομένων.
  2. Τα δεδομένα μπορεί να είναι μεικτού τύπου, συμπεριλαμβανομένων των συλλογικών τύπων.
- Αλφαριθμητικά (Strings)
  1. Αμετάβλητα
  2. Εννοιολογικά μοιάζουν πολύ με τις πλειάδες
- Λίστα
  1. Μια μεταβλητή διατεταγμένη ακολουθία δεδομένων μεικτού τύπου.

Παρόμοιο συντακτικό

- Και οι τρεις αυτοί ακολουθιακοί τύποι έχουν παρόμοιο συντακτικό και λειτουργικότητα.
- Ειδοποιός διαφορά:
  - Οι πλειάδες και τα αλφαριθμητικά είναι αμετάβλητα.
  - Οι λίστες είναι μεταβλητές.
- Οι λειτουργίες που αναφέρονται στην ενότητα αυτή αφορούν όλους τους ακολουθιακούς τύπους.

Ακολουθιακοί τύποι 1

- **Οι πλειάδες ορίζονται χρησιμοποιώντας παρενθέσεις και κόμματα.**

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- **Οι λίστες ορίζονται χρησιμοποιώντας αγκύλες και κόμματα.**

```
>>> li = ["abc", 34, 4.34, 23]
```

- **Τα αλφαριθμητικά ορίζονται χρησιμοποιώντας εισαγωγικά και αποστρόφους (“, ‘, or “““).**

```
>>> st = "Hello World"
>>> st = 'Hello World'
>>> st = """This is a multi-line
string that uses triple quotes."""
```

Ακολουθιακοί τύποι 2

- **Μπορούμε να προσπελάσουμε μεμονωμένα μέλη μιας πλειάδας, λίστας ή αλφαριθμητικού χρησιμοποιώντας το συνηθισμένο συμβολισμό στοιχείων πινάκων.**
- ***Οι θέσεις πάντα ξεκινούν από το 0.***

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]
'abc'
>>> li = ["abc", 34, 4.34, 23]
>>> li[1]
34
>>> st = "Hello World"
>>> st[1]
'e'
```

Θετικοί και αρνητικοί δείκτες

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

- **Θετικός δείκτης: η μέτρηση ξεκινά από αριστερά, αρχίζοντας με το 0.**

```
>>> t[1]
'abc'
```

- **Αρνητικός δείκτης: η μέτρηση ξεκινά από δεξιά, αρχίζοντας με –1.**

```
>>> t[-3]
4.56
```

Τεμαχισμός: επιστροφή αντιγράφου ενός υποσυνόλου 1

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

- **Επιστροφή αντιγράφου του περιεχομένου με ένα υποσύνολο των αρχικών μελών. Η αντιγραφή ξεκινά από τον πρώτο δείκτη και σταματά ακριβώς πριν το δεύτερο δείκτη.**

```
>>> t[1:4]

('abc', 4.56, (2,3))
```

- **Επίσης, μπορούν να χρησιμοποιηθούν αρνητικοί δείκτες κατά τον τεμαχισμό.**

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Τεμαχισμός: επιστροφή αντιγράφου ενός υποσυνόλου 2

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

- **Όταν παραλείπεται ο πρώτος δείκτης τότε ο τεμαχισμός ξεκινά από την αρχή της δομής.**

```
>>> t[:2]

(23,'abc')
```

- **Όταν παραλείπεται ο δεύτερος δείκτης τότε ο τεμαχισμός σταματά στο τελευταίο στοιχείο της δομής.**

```
>>> t[2:]
(4.56, (2,3),'def')
```

Αντιγραφή ολόκληρης της ακολουθίας

Για αντιγραφή ολόκληρης της ακολουθίας χρησιμοποιούμε το [:].

```
>>> t[:]

(23, 'abc', 4.56, (2,3), 'def')
```

- **Σημειώστε τη διαφορά των δύο παρακάτω γραμμών που αφορά τις μεταβλητές ακολουθίες:**

```
>>> list2 = list1      # 2 ονόματα για μία αναφορά.
                       # Αν υπάρξει αλλαγή επηρεάζονται και οι δύο.

>>> list2 = list1[:]   # Δύο ανεξάρτητα αντίγραφα, δύο αναφορές.
```

Ο τελεστής in

- Λογικός έλεγχος αν περιέχεται μια τιμή:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- Για αλφαριθμητικά, έλεγχος μέρους αλφαριθμητικού:

```
>>> a = 'abcde’
>>> 'c’ in   a
True
>>> 'cd' in  a
True
>>> 'ac' in  a
False
```

Ο τελεστής +

- Ο τελεστής + παράγει μια νέα πλειάδα, λίστα ή αλφαριθμητικό του οποίου η τιμή είναι η συνένωση των ακολουθιών.

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> "Hello" + " " + "World"
'Hello World'
```

Ο τελεστής *

- Ο τελεστής * παράγει μια νέα πλειάδα, λίστα ή αλφαριθμητικό που επαναλαμβάνει το αρχικό περιεχόμενο.

```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> “Hello” * 3
‘HelloHelloHello’
```

Μεταβλητότητα: Πλειάδες εναντίον λιστών

Οι πλειάδες (tuples) είναι αμετάβλητες

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14

Traceback (most recent call last):
   File "<pyshell#75>", line 1, in -
   toplevel-  tu[2] = 3.14
TypeError: object doesn't support item assignment
```

- **Δεν μπορεί να γίνει αλλαγή σε μια πλειάδα.**
- **Μπορεί να δημιουργηθεί μια νέα πλειάδα και να ανατεθεί η αναφορά της σε ένα όνομα που έχει χρησιμοποιηθεί προηγουμένως.**

```
>>> t = (23, 'abc', 3.14, (2,3), 'def')
```

Οι λίστες (lists) είναι μεταβλητές

```
>>> li = ['abc', 23, 4.34, 23]
>>> li[1] = 45
>>> li
['abc', 45, 4.34, 23]
```

- **Οι λίστες μπορούν να αλλάξουν επιτόπου.**
- **Το όνομα *li* δείχνει ακόμη δείχνει στην ίδια αναφορά μνήμης όταν γίνει η αλλαγή.**
- **Η μεταβλητότητα των λιστών σημαίνει ότι δεν είναι τόσο γρήγορες όσο οι πλειάδες.**

Λειτουργίες μόνο για λίστες 1

```
>>> li = [1, 11, 3, 4, 5]

>>> li.append('a')
>>> li
[1, 11, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>>li
[1, 11, 'i', 3, 4, 5, 'a']
```

Λειτουργίες μόνο για λίστες 2

+ δημιουργεί μια νέα λίστα (με νέα αναφορά μνήμης)
***extend* λειτουργεί στη λίστα `li` επιτόπου.**

```
>>> li.extend([9, 8, 7])
>>>li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7]
```

***Περίπτωση σύγχυσης*:**

Η extend παίρνει μια λίστα σαν όρισμα. Η append παίρνει μια μεμονωμένη λίστα σαν όρισμα.

```
>>> li.append([10, 11, 12])
>>> li
 [1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7, [10, 11, 12]]
```

Λειτουργίες μόνο για λίστες 3

```
>>> li = ['a', 'b', 'c', 'b']

>>> li.index('b')      # δείκτης πρώτης εμφάνισης
1
>>> li.count('b')      # πλήθος εμφανίσεων
2
>>> li.remove('b')     # διαγραφή πρώτης εμφάνισης
>>> li
  ['a', 'c', 'b']
>>> li.clear()         # διαγραφή όλων των στοιχείων
>>> li                 της λίστας.
   []
```

Λειτουργίες μόνο για λίστες 4

```
>>> del li                  # διαγραφή της λίστας
>>> li = [5, 2, 6, 8]
>>> li.reverse()            # αντιστροφή της λίστας
>>> li
  [8, 6, 2, 5]

>>> li.sort()               # ταξινόμηση λίστας
>>> li
  [2, 5, 6, 8]

>>> li.sort(some_function)
    # ταξινόμηση λίστας με χρήση συνάρτηση ορισμένης από το
    χρήστη.
```

Λειτουργίες μόνο για λίστες 5

```
>>> li=[1,2,3,2,5,1,4,2,7,4,2,5]
>>> li.count(2)          # το πλήθος των φορών που συναντάται ένα στοιχείο
4

>>> id(li)
2388724468864
>>> li2=li.copy()
>>> li2
[1,2,3,2,5,1,4,2,7,4,2,5]
>>> id(li2)
2388757362688
```

Προσοχή! Δεν είναι εκχώρηση !! Οι δύο λίστες έχουν την ίδια τιμή, αλλά διαφορετικές ταυτότητες.

Λίστες ως στοίβες

```
>>> stack1=[]
>>> stack1.append(10)
>>> stack1.append(20)
>>> stack1.append(30)
>>> stack1.append(40)
>>> stack1
[10,20,30,40,50]
>>> stack1.pop()
50
>>> stack1.pop()
40
```

```
# η μέθοδος pop() χωρίς
παράμετρο εξάγει το τελευταίο
στοιχείο της λίστας
```

Λίστες ως ουρές

```
>>> queue1=[]
>>> queue1.append(10)
>>> queue1.append(20)
>>> queue1.append(30)
>>> queue1.append(40)
>>> queue1
[10,20,30,40,50]
>>> queue1.pop(0)
10
>>> queue1.pop()
50
```

η μέθοδος pop() με παράμετρο τον αριθμό 0 εξάγει το πρώτο στοιχείο της λίστας.

Λίστες εναντίον Πλειάδων

- Οι λίστες είναι πιο αργές, αλλά πιο ισχυρές από τις πλειάδες.
  - Οι λίστες μπορούν να μεταβληθούν και διαθέτουν πολλές λειτουργίες που μπορούν να εφαρμοστούν σε αυτές.
  - Οι πλειάδες δεν μεταβάλλονται και έχουν λιγότερα χαρακτηριστικά.
- Για τη μετατροπή μεταξύ πλειάδων και λιστών χρησιμοποιούνται οι συναρτήσεις list() και tuple().

```
li=list(tu)

tu=tuple(li)
```

Λειτουργίες με αλφαριθμητικά 1

```
>>> string1='This is a test string'
>>> string1.find('is')
2
>>> string1.rfind('is')
5

>>> string1.find('s',10,13)
12
>>> string1.index('is')
2
>>> string1.find('v')
-1
```

εκτελεί αναζήτηση και απαντά με την πρώτη θέση από αριστερά που βρίσκει το στοιχείο.

#εκτελεί αναζήτηση και απαντά με την δεξιότερη θέση που βρίσκει το στοιχείο.

εκτελεί αναζήτηση και απαντά με την πρώτη θέση από αριστερά που βρίσκει το στοιχείο ανάμεσα στα δοσμένα όρια.

#Επιστρέφει -1 αν δεν βρει το στοιχείο.

Λειτουργίες με αλφαριθμητικά 2

```
>>> string1='This is a test string'
>>> string1.replace('s', 'x')
'Thix ix a text xtring'
>>> string1.replace('is', 'was',1)
'Thwas is a test string'
```

εκτελεί αντικατάσταση της πρώτης παραμέτρου με τη δεύτερη δημιουργώντας αντίγραφο του αλφαριθμητικού.

#ο αριθμός της τρίτης παραμέτρου δείχνει το πλήθος των αντικαταστάσεων που θα γίνουν.

```
>>> 'Hello'.isalnum()
True
>>> 'Hello !!!'.isalnum()
False
>>> 'Hello123'.isalpha()
False
>>> '2022'.isdigit()
True
```

έλεγχος τύπου.

Λειτουργίες με αλφαριθμητικά 3

```
>>> 'This is a test string'.startswith("This")        # έλεγχος άκρων
True
>>> 'This is a test string'.startswith("str")
False
>>> 'This is a test string'.endswith("str")
False
>>> 'This is a test string'.endswith("ing")
True
```

Λειτουργίες με αλφαριθμητικά 4

```
>>> 'This is a test string'.isupper()
False
>>> 'This is a test string'.islower()
False
>>> 'This is a test string'.title()
'This Is A Test String'
>>> 'This is a test string'.upper()
'THIS IS A TEST STRING'
>>> 'This is a test string'.lower()
'this is a test string'
>>> 'This is a Test String'.swapcase()
'tHIS IS A tEST sTRING'
```

```
>>> "string".ljust(20)
'string              '
>>> "string".rjust(20)
'              string'
>>> "string".center(20)
'       string       '
```

Λειτουργίες με αλφαριθμητικά 5

```
>>> string1='    This is a test string    '          # περικοπή μέρους αλφαριθμητικού
>>> string1.lstrip()
'This is a test string    '
>>> string1.rstrip()
'    This is a test string'
>>> string2='This is a test string'
>>> string2.strip("This")
'is a test string'
>>> string2.split()                                  # διαχωρισμός αλφαριθμητικού.
['This','is','a','test','string']                    #Αν το αλφαριθμητικό απλώνεται σε
                                                     περισσότερες από μία γραμμές, μπορεί
                                                     να χρησιμοποιηθεί η splitlines()
```

Λειτουργίες με αλφαριθμητικά 6

```
>>> ' '.join(['This', 'is', 'a', 'test', 'string'])      # συνένωση
'This is a test string'
>>> ','.join(['This', 'is', 'a', 'test', 'string'])
'This,is,a,test,string'
```

Υπενθύμιση: συνένωση μπορεί να γίνει και με τους τελεστές '+' και '+='. Επειδή όμως τα αλφαριθμητικά δεν είναι μεταβαλλόμενα, δημιουργείται ένα νέο αντικείμενο με νέα ταυτότητα.