Πρόγραμμα Μεταπτυχιακών Σπουδών
Πληροφορική και Υπολογιστική Βιοϊατρική

Θέματα Προγραμματισμού Η/Υ

Ενότητα 10:

Θεματική Ενότητα: Λίστες (Lists) και Πλειάδες (Tuples)

ΘΕΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Η/Υ

Θεματική Ενότητα 10

Λίστες (Lists) και Πλειάδες (Tuples)

Πληροφορική και Υπολογιστική Βιοϊατρική
Α. Κακαρούντας, Γ. Σπαθούλας, Π. Κοντού

Δομές δεδομένων

- Οι **δομές δεδομένων** αποτελούν τους διαφορετικούς δυνατούς τρόπους οργάνωσης και αποθήκευσης των δεδομένων, ώστε να χρησιμοποιηθούν/επεξεργαστούν από το πρόγραμμα όσο το δυνατόν πιο αποδοτικά (ανάλογα με το τι ακριβώς κάνει ο αλγόριθμος).

Λίστες (Lists)

- **Λίστα** (*list*) είναι η δομή δεδομένων (ή αντικείμενο - object) που περιέχει πολλαπλά δεδομένα. Κάθε τέτοιο δεδομένο ονομάζεται στοιχείο (*element*) της λίστας.
- Τα δεδομένα τους μπορούν να αλλάζουν.
- *Δυναμικές* δομές δεδομένων: τα περιεχόμενά τους μπορούν να αυξάνουν ή να μειώνονται κατά την εκτέλεση του προγράμματος.
- Χρήση δείκτη (*index*) για προσπέλαση συγκεκριμένου element της λίστας.

Λίστες - Παραδείγματα δημιουργίας

```
empty = [ ]
numbers = [3, 5, 7, 9]
names = ['Kostas', 'Giorgos', 'Maria', 'Anna']
info = ['Kostas', 'Xatzis', 'Mykonou 23', '26500', 'Patra']
```

- Χρήση του τελεστή επανάληψης (*):

```
numbers = [1, 2] * 3
```
→ [1, 2, 1, 2, 1, 2]

- Η *range()* δημιουργεί τις τιμές της λίστας αλλά <u>όχι</u> την λίστα σαν δομή δεδομένων. Αυτό το κάνει η συνάρτηση *list()* με είσοδο τις αριθμητικές τιμές που συμπεριλαμβάνει στη λίστα που δημιουργεί.

```
numbers = list(range(99,103))
```
→ [99, 100, 101, 102]

```
for n in numbers :
    print(n)
```
→
```
99
100
101
102
```

Indexing – προσπέλαση συγκεκριμένου στοιχείου.

```
my_list = [10, 20, 30, 40]
print(my_list[0], my_list[1], my_list[2], my_list[3])
```

- ή

```
for index in range(4) :
    print(my_list[index])
```

- ή

```
index = 0
while index < 4:
    print(my_list[index])
    index += 1
```

- Αλλαγή τιμής στοιχείου

```
my_list[1] = 200
```

• Πιο γενικά χρησιμοποιείται η συνάρτηση len(), που με είσοδο μια λίστα επιστρέφει το μήκος της. Εδώ χρήση του len(my_list) αντί για το 4.

• Χρήση index>= len(list) προκαλεί IndexError: list assignment index out of range.

• Συνένωση λιστών με χρήση του τελεστή +

```
more = [50, 60]
full_list = my_list + more
```

[10, 200, 30, 40, 50, 60]

Απομόνωση τμημάτων λίστας και εύρεση στοιχείου

Παρόμοια με τις αντίστοιχες λειτουργίες στα αλφαριθμητικά:

- *list [start : end]* *# end index not included*
- *[: end] ξεκινά από το 0*
- *[start :] πάει ως το len(list)*
- *Για αλλαγή βήματος: list [start : end : step]*

- *item* **in** *list* → *True/False*
- *Π.χ.*

```
codes = ['U135', 'X979', 'A123', 'S888']
verify  = input('Enter a product code: ')
if verify in codes :
        print(verify , 'was found in the list.')
else:
        print(verify , 'was not found in the list.')
```

Μέθοδοι για λίστες

- **append(*item*)** Προσθέτει το *στοιχείο στο τέλος της λίστας.*
- **index(*item*)** Επιστρέφει τον δείκτη του πρώτου στοιχείου του οποίου η τιμή ισούται με το *στοιχείο.* ValueError αν το *στοιχείο δεν*υπάρχει στη λίστα.
- **insert(*index , item*)** Εισάγει το *στοιχείο στη λίστα στη θέση που κα*θορίζει ο *δείκτης. Όταν ένα στοιχείο εισάγεται* στη λίστα, η λίστα επεκτείνεται σε μέγεθος για να συμπεριλάβει το νέο στοιχείο. Το στοιχείο που βρισκόταν προηγουμένως στη συγκεκριμένη θέση, και όλα τα στοιχεία μετά από αυτό, μετατοπίζονται κατά μία θέση προς το τέλος της λίστας. Αν καθορίσετε έναν δείκτη πέρα από το τέλος της λίστας, το *στοιχείο θα προστεθεί στο τέλος* της λίστας. Αν ο *δείκτης είναι αρνητικός και ανα*φέρεται σε μία μη-έγκυρη θέση, το *στοιχείο θα* προστεθεί στην αρχή της λίστας.
- **sort()** Ταξινομεί τα στοιχεία της λίστας σε αύξουσα σειρά (από τη μικρότερη προς τη μεγαλύτερη τιμή).
- **remove(*item*)** Αφαιρεί από τη λίστα το *στοιχείο στην πρώτη του* εμφάνιση. ValueError αν το *στοιχείο δεν υπάρχει στη λίστα.*
- **reverse()** Αντιστρέφει τη σειρά των στοιχείων στη λίστα.

Συναρτήσεις για λίστες

- Εναλλακτικά της μεθόδου remove που διαγράφει στοιχείο της λίστας με συγκεκριμένη τιμή, η συνάρτηση ***del*** διαγράφει στοιχείο συγκεκριμένου index. Π.χ. del my_list[2]
- Η συνάρτηση **min** δέχεται ως όρισμα μια αλληλουχία, όπως είναι μια λίστα, και επιστρέφει το στοιχείο εκείνο της αλληλουχίας που έχει την ελάχιστη τιμή. Π.χ. my_list = [5, 4, 3, 2, 50, 40, 30]

  print('Η ελάχιστη τιμή είναι', min(my_list))
- Ανάλογα λειτουργεί η συνάρτηση **max**.

Αντίγραφα λιστών

```
list1 = [1, 2, 3, 4]
list2 = list1
print(list1)  ⟶ [1, 2, 3, 4]
print(list2)  ⟶ [1, 2, 3, 4]
list1[2] = 33
print(list1)  ⟶ [1, 2, 33, 4]
print(list2)  ⟶ [1, 2, 33, 4]
```

- Άρα, οι μεταβλητές list1 και list2 αναφέρονται στην ίδια λίστα στη μνήμη
- Για δημιουργία αντιγράφου λίστας που να μην αναφέρεται σε άλλο σημείο της μνήμης αλλά με τα ίδια στοιχεία πρέπει να γίνει αντιγραφή των στοιχείων ένα-προς-ένα:

```
list2 = []          # κενή λίστα
for item in list1:                  # εναλλακτικά  list2 += list1  ή  list2 = list1[:]
    list2.append(item)
```

Παράδειγμα1: Άθροιση και μέσος όρος τιμών λίστας

- Το πρόγραμμα αυτό υπολογίζει το άθροισμα και τον μέσο όρο των τιμών μιας λίστας.

```
numbers = [2, 4, 6, 8, 10]          # Δημιουργία μιας λίστας.
total = 0                           # Μεταβλητή που θα χρησιμεύσει σαν αθροιστής.

# Υπολογισμός του αθροίσματος των στοιχείων της λίστας.
for value in numbers:
    total += value
print('Το άθροισμα των στοιχείων είναι', total)

# Υπολογισμός του μέσου όρου των στοιχείων.
average = total / len(numbers)
print('Ο μέσος όρος των στοιχείων είναι', average)
```

Οι Λίστες ως Ορίσματα Συναρτήσεων

- Η λίστα χρησιμοποιείται στις συναρτήσεις όπως και οποιαδήποτε άλλη μεταβλητή.
- Το πρόγραμμα αυτό χρησιμοποιεί μια συνάρτηση για να υπολογίσει το άθροισμα των τιμών σε μια λίστα:

```
# Η συνάρτηση get_total δέχεται μια λίστα ως όρισμα και επιστρέφει
    #   το άθροισμα των στοιχείων της λίστας.
def get_total(value_list):
    total = 0       # Μεταβλητή που θα χρησιμεύσει ως αθροιστής.
    # Υπολογισμός του αθροίσματος των στοιχείων της λίστας.
    for num in value_list:
            total += num
    return total    # Επιστροφή του αθροίσματος.

numbers = [2, 4, 6, 8, 10]          # Δημιουργία μιας λίστας.
print('Το άθροισμα των στοιχείων είναι', get_total(numbers) )
```

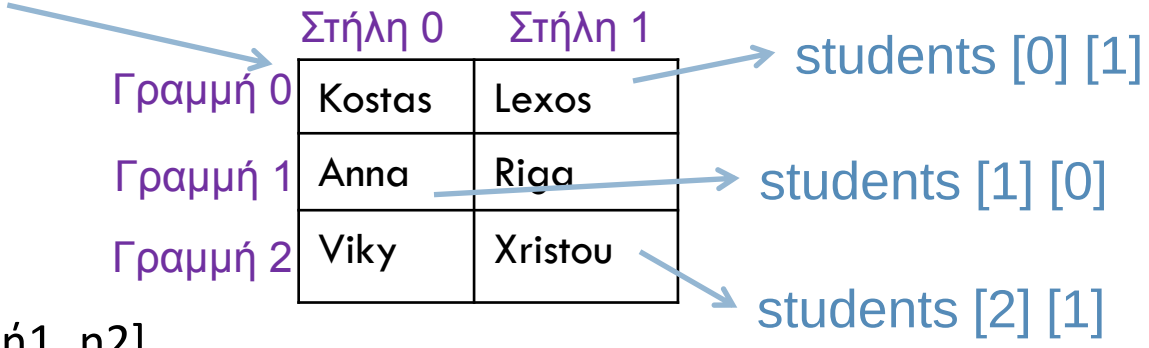
Συναρτήσεις που Επιστρέφουν Λίστες

- Το πρόγραμμα αυτό χρησιμοποιεί μια συνάρτηση για να δημιουργήσει μια λίστα. Η συνάρτηση **επιστρέφει** μια αναφορά στη λίστα.

```
# Η συνάρτηση get_values δέχεται 10 τιμές από το χρήστη και τις αποθηκεύει σε λίστα.
# Η συνάρτηση επιστρέφει μια αναφορά στη λίστα που δημιούργησε.
def get_values():
    values = [ ]              # Δημιουργία μιας κενής λίστας.
    count=0
    while count < 10:     # Είσοδος τιμών από τον χρήστη και προσθήκη τους στη λίστα.
        num = int(input('Δώσε έναν αριθμό: '))
        values.append(num)
        count += 1
    return values          # Επιστροφή της λίστας.
numbers = get_values()       # Είσοδος μιας λίστας που περιέχει τιμές.
print('Οι αριθμοί στη λίστα είναι:')
print(numbers)                    # Εμφάνιση των τιμών της λίστας.
```

Δισδιάστατες Λίστες

- Μια δισδιάστατη (ή ένθετη) λίστα είναι μια λίστα που τα στοιχεία της είναι άλλες λίστες.
- students = [['Kostas', 'Lexos'], ['Anna', 'Riga'], ['Viky', 'Xristou']]



- Για μέγεθος n1xn2:

όνομα=[[τιμή1_1, τιμή1_2, ..., τιμή1_n2],
[τιμή2_1, τιμή2_2, ..., τιμή2_n2], ...,
[τιμήn1 _1, τιμήn1 _2, ..., τιμήn1_n2]]

- Οι δισδιάστατες λίστες είναι πολύ χρήσιμες όταν δουλεύουμε με πολλαπλά σύνολα δεδομένων.

Π.χ. πίνακας βαθμολογίας φοιτητών σε πολλά μαθήματα

Παράδειγμα1: Άθροιση και μέσος όρος όλων των στοιχείων μιας 2D λίστας

- Το πρόγραμμα αυτό υπολογίζει το άθροισμα και τον μέσο όρο όλων των στοιχείων μιας δισδιάστατης λίστας.

```
numbers = [ [2, 4, 6, 8, 10], [1, 3, 5, 7, 9], [11, 12, 13, 14, 15] ]  # Δημιουργία λίστας.
total = 0          # Μεταβλητή που θα χρησιμεύσει σαν αθροιστής.

# Υπολογισμός του αθροίσματος των στοιχείων της λίστας.
for row in range(3):
    for col in range(5):
            total += numbers[row][col]
print('Το άθροισμα των στοιχείων είναι', total)

# Υπολογισμός του μέσου όρου των στοιχείων.
average = total / ( len(numbers)* len(numbers[0]) )
print('Ο μέσος όρος των στοιχείων είναι', average)
```

Παράδειγμα1α: Άθροιση και μέσος όρος των στοιχείων κάθε γραμμής μιας 2D λίστας

- Το πρόγραμμα αυτό υπολογίζει το άθροισμα και τον μέσο όρο των στοιχείων κάθε μιας γραμμής μιας δισδιάστατης λίστας.

```
numbers = [ [2, 4, 6, 8, 10], [1, 3, 5, 7, 9], [11, 12, 13, 14, 15] ]  # Δημιουργία λίστας.

# Υπολογισμός του αθροίσματος των στοιχείων της γραμμής της λίστας.
for row in range(3):
    total = 0                    # Μεταβλητή που θα χρησιμεύσει σαν αθροιστής.
    for col in range(5):
        total += numbers[row][col]
    print('Το άθροισμα των στοιχείων της', row, 'γραμμής είναι', total)

    # Υπολογισμός του μέσου όρου των στοιχείων της κάθε γραμμής.
    average = total / ( len(numbers[0]) )
    print('Ο μέσος όρος των στοιχείων της', row, 'γραμμής είναι', average)
```

Παράδειγμα 2: Υπολογισμός του μέγιστου/ελάχιστου στοιχείου κάθε γραμμής/στήλης μιας 2D λίστας

- Το πρόγραμμα αυτό βρίσκει το μέγιστο/ελάχιστο μεταξύ των στοιχείων κάθε μιας γραμμής/στήλης μιας δισδιάστατης λίστας.

```
numbers = [ [2, 4, 6, 8, 10], [1, 3, 5, 7, 9], [11, 12, 13, 14, 15] ]  # Δημιουργία λίστας.


# Υπολογισμός του μέγιστου των στοιχείων κάθε γραμμής της λίστας.
for row in range(3):
    print('Το μέγιστο στοιχείο της', row, 'γραμμής είναι', max(numbers[row]) )


# Υπολογισμός του ελάχιστου των στοιχείων κάθε στήλης της λίστας.
for col in range(5):
     print('Το ελάχιστο στοιχείο της', col, 'στήλης είναι', \
                        min(numbers[0][col], numbers[1][col]), numbers[2][col] ) )
```

Πλειάδες (Tuples)

- Μια ***πλειάδα*** είναι μια **αμετάβλητη** λίστα, που σημαίνει ότι το περιεχόμενό της δε μπορεί να αλλάξει.
- Χρήση ( ) για να ορίσουμε την αλληλουχία τιμών αντί για [ ].
- my_tuple = (1, 2, 3, 4, 5)
- Οι πλειάδες υποστηρίζουν τις ίδιες λειτουργίες με τις λίστες, εκτός από εκείνες που αλλάζουν το περιεχόμενο της λίστας:
  - Διαχείριση με δείκτες (μόνο για ανάκτηση τιμών των στοιχείων)
  - Μέθοδοι όπως η index
  - Ενσωματωμένες συναρτήσεις όπως οι len, min και max
  - Εκφράσεις τεμαχισμού
  - Τον τελεστή in
  - Τους τελεστές + και *
- Για πλειάδα με μόνο ένα στοιχείο: my_tuple = (3,)

Γιατί το
my_tuple = (3)
είναι ισοδύναμο
του my_tuple = 3
που εκχωρεί τιμή
σε απλή

Λόγοι ύπαρξης Πλειάδων στην Python και Μετατροπή από Λίστες σε Πλειάδες

- Υπολογιστική απόδοση: η επεξεργασία με πλειάδες είναι ταχύτερη από την επεξεργασία με λίστες
- Ασφάλεια: αποθήκευση δεδομένων χωρίς κίνδυνο να τροποποιηθεί κατά λάθος (ή με άλλο τρόπο) από τον κώδικα του προγράμματος
- Υπάρχουν ορισμένες πράξεις στην Python οι οποίες απαιτούν τη χρήση πλειάδων

```
number_tuple = (1, 2, 3)
number_list = list(number_tuple)
print(number_list) ⟶ [1, 2, 3]
back_to_tuple= tuple(number_list)
print(back_to_tuple) ⟶ (1, 2, 3)
```

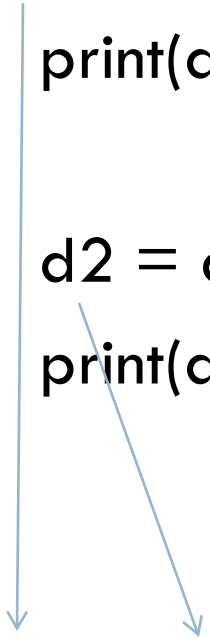
Αντικείμενα datetime

```
from datetime import datetime,timedelta
d1 = datetime.now()
print(d1)
```

2017-06-08 19:38:16.796501

```
d2 = datetime(1984, 1, 10, 23, 30)
print(d2)
```

1984-01-10 23:30:00



τα d1 και d2 είναι αντικείμενα **datetime**

Αντικείμενα datetime

```
print(d1.strftime("%B %d, %Y"))
```

June 08, 2017

```
td = d1-d2
print(td)
```

τα **td** είναι διαφορές χρόνου

12202 days, 20:12:31.179046

```
td = timedelta(hours=1)
print(d1)
```

2017-06-08 19:38:16.796501

```
print(d1+td)
```

2017-06-08 20:38:16.796501

ΘΕΜΑΤΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Η/Υ

Θεματική Ενότητα 10

Λίστες (Lists) και Πλειάδες (Tuples)

Πληροφορική και Υπολογιστική Βιοϊατρική
Α. Κακαρούντας, Γ. Σπαθούλας, Π. Κοντού