ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Γλώσσες Προγραμματισμού

Λεξικά – Πλειάδες – Σύνολα

Χαρά Πρασσά, chprassa@gmail.com

- Μία μεταβλητή τύπου λεξικού (dictionary) είναι μια συλλογή δεδομένων. Κάθε στοιχείο της συλλογής αυτής αποτελείται από δύο μέρη. Το πρώτο μέρος αναφέρεται ως **κλειδί (key)** και το δεύτερο μέρος ως **τιμή(value).**
- Το κλειδί χρησιμοποιείται για τον εντοπισμό μιας τιμής. Σε κάθε κλειδί αντιστοιχεί μία τιμή. Τα ζεύγη **key:value καταχωρούνται μέσα σε άγκιστρα { } και διαχωρίζονται με (,). Η τιμή μπορεί να είναι οποιουδήποτε τύπου(κλάσης).**

Σύνταξη:

```
όνομα_dictionary={key 1: value 1, key 2:value 2,…..key n:value n}
```

Παράδειγμα:

```
employee={
   'first_name': 'George',
   'last_name': 'Athanasopoulos',
   'Department': 'Marketing'
   }
print(employee)
```

Output

```
{'first_name': 'George', 'last_name': 'Athanasopoulos', 'Department': 'Marketing'}
```

- Δημιουργία λεξικού μέσω της dict()

```
employee2=dict(first_name='George', last_name='Athanasopoulos', Department='Marketing')
print(employee2)
```

Output

```
{'first_name': 'George', 'last_name': 'Athanasopoulos', 'Department': 'Marketing'}
```

- Δημιουργία κενού λεξικού

```
employee3=dict()
print(employee3)
```

Output

```
{}
```

- Δημιουργία κενού λεξικού

```
employee4={}
print(employee4)
```

Output

```
{}
```

❑ Χαρακτηριστικά λεξικών

- Αδιάτακτη δομή (δεν υπάρχει κάποια συγκεκριμένη σειρά των ζευγών κλειδί:τιμή)

Χρησιμοποιούμε λεξικά όταν δεν μας ενδιαφέρει η θέση των στοιχείων σε αντίθεση με τις λίστες όπου μας ενδιαφέρει η θέση κάθε στοιχείου

- Κλειδιά και τιμές οποιουδήποτε τύπου δεδομένων

Μπορούμε να έχουμε ως κλειδί έναν αριθμό, ένα αλφαριθμητικό ή και άλλες δομές όπως μια πλειάδα που θα δούμε στη συνέχεια. Το μόνο που πρέπει να προσέχουμε είναι η δομή που θα χρησιμοποιηθεί ως κλειδί να είναι μια αμετάλλακτη δομή της Python

- Κάθε κλειδί είναι μοναδικό

Αυτό δεν ισχύει με τις τιμές όπου μπορούμε να έχουμε όσες ίδιες τιμές θέλουμε μέσα σε ένα λεξικό

- Ανομοιογενής δομή

Σε ένα λεξικό το κλειδί σε ένα ζεύγος μπορεί να είναι ακέραιος ενώ σε κάποιο επόμενο ζεύγος να είναι αλφαριθμητικό

- Για να αποκτήσουμε πρόσβαση στην τιμή (value) ενός κλειδιού (key) του dictionary, πρέπει να γράψουμε το όνομα του λεξικού και αμέσως μετά μέσα σε square brackets([ ]) το κλειδί.

Παράδειγμα:

```
employee={
   'first_name': 'George',
   'last_name': 'Athanasopoulos',
   'Department': 'Marketing'
   }
print(employee['first_name'])
print(f"The employee first name is {employee['first_name']}")
```

Output

```
George
The employee first name is George
```

❑ Αν αναζητούσαμε ένα κλειδί που δεν υπάρχει στο λεξικό π.χ. print(employee[Specialization']) το πρόγραμμα θα επέστρεφε σφάλμα.

- Ένας δεύτερος τρόπος για να λάβουμε την τιμή κάποιου value είναι με την χρήση της get( ) μεθόδου. Η μέθοδος αυτή επιστρέφει την τιμή (value) ενός κλειδιού με όνομα kleidi στο λεξικό dictionary.

Σύνταξη: **`timi=dictionary.get(kleidi)`**

Παράδειγμα:

```
employee={
   'first_name': 'George',
   'last_name': 'Athanasopoulos',
   'Department': 'Marketing'
   }
print(employee.get('first_name', 'There is no first name key'))
print(employee.get('id', 'This employee has no company ID yet'))
```

Output

```
George
This employee has no company ID yet
```

- *Η διαφορά με τον προηγούμενο τρόπο είναι ότι αν μέσα στο get( ) αναζητήσουμε κάποιο key το οποίο δεν υπάρχει, το πρόγραμμα μας δεν θα σταματήσει. Δηλαδή δεν θα πάρουμε ένα exception. Απλά η Python θα μας δώσει None σαν απάντηση στο συγκεκριμένο αίτημα. Επίσης μπορούμε να προσθέσουμε και ένα δικό μας μήνυμα σε περίπτωση που δεν βρεθεί το κλειδί που αναζητάμε.*

- Η Python μας δίνει τη δυνατότητα να προσθέσουμε λίστες σε ένα λεξικό. Στο ακόλουθο παράδειγμα ορίζουμε μια λίστα σαν τιμή σε ένα λεξικό.

Παράδειγμα:

```
employee={
   'first_name': 'George',
    'last_name': 'Athanasopoulos',
    'Department': 'Marketing',
    'Specialization': ['Python','Matlab','R']
    }
print(employee['Specialization'][0])
```

Output

Python

- Υπάρχει ακόμα η δυνατότητα όχι μόνο να έχουμε πρόσβαση στα δεδομένα ενός λεξικού αλλά και να προσθέσουμε καινούργια στοιχεία σε αυτό.

Παράδειγμα:

```
employee={
   'first_name': 'George',
    'last_name': 'Athanasopoulos'
    }
employee['Department'] = 'Marketing'
print(employee)
```

Output

{'first_name': 'George', 'last_name': 'Athanasopoulos', 'Department': 'Marketing'}

- Για να αλλάξουμε μια υπάρχουσα τιμή, ακολουθούμε ακριβώς τα ίδια βήματα όπως και στο προηγούμενο παράδειγμα – δηλαδή προσθέτουμε ένα καινούργιο ζευγάρι key-value. Εάν το key υπάρχει ήδη τότε θα αντικατασταθεί το value με αυτό που προτείνουμε, ενώ αν δεν υπάρχει το key τότε θα δημιουργηθεί ένα καινούργιο key-value ζευγάρι στο λεξικό.

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
}

employee['specialization'] = 'Python'
print(employee)
employee['specialization'] = 'R'
print(employee)
```

Output

```
{'first_name': 'George', 'last_name': 'Athanasopoulos', 'specialization': 'Python'}
{'first_name': 'George', 'last_name': 'Athanasopoulos', 'specialization': 'R'}
```

- Για να διαγράψουμε ένα συγκεκριμένο key-value από ένα λεξικό χρησιμοποιούμε την μέθοδο del

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos',
   'Department': 'Marketing'
}
del employee['Department']
print(employee)
```

Output

```
{'first_name': 'George', 'last_name': 'Athanasopoulos'}
```

- Η μέθοδος .clear() διαγράφει το περιεχόμενο του λεξικού.

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
  }
employee.clear()
print(employee)
```

Output

```
{}
```

- Επειδή τα λεξικά συνήθως περιέχουν πάρα πολλά στοιχεία, για να μπορέσουμε να αποκτήσουμε μια γρήγορη εικόνα τους χρησιμοποιούμε τους βρόχους επανάληψης (loops). Μπορούμε να χρησιμοποιήσουμε ένα for loop για να δούμε μόνο τα keys, ή μόνο τα values ή ολόκληρο το ζευγάρι key-value.

1. Η μέθοδος keys() επιστρέφει όλα τα κλειδιά ενός λεξικού σε ένα αντικείμενο που κάθε στοιχείο του αντικειμένου αυτού είναι ιδίου τύπου με το κλειδί όπως ορίσθηκε στο λεξικό.

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
}
for key in employee.keys():
   print(key)
   print(employee[key])
```

Το αποτέλεσμα θα ήταν το ίδιο αν χρησιμοποιούσαμε την for χωρίς τη μέθοδο keys()

Output

```
first_name
George
last_name
Athanasopoulos
```

2. Η μέθοδος values() επιστρέφει τις τιμές(**values**) ενός λεξικού σ' ένα αντικείμενο τύπου **dict_values** που κάθε στοιχείο του αντικειμένου αυτού είναι ιδίου τύπου με την τιμή(**value**) όπως ορίσθηκε στο λεξικό.

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
}
for value in employee.values():
   print(value)
```

Output

George
Athanasopoulos

3. Η μέθοδος items() επιστρέφει όλα τα κλειδιά & τιμές ενός λεξικού.

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
}
for key, value in employee.items():
   print(f"The key is {key} and the value is {value}")
```

Output

The key is first_name and the value is George
The key is last_name and the value is Athanasopoulos

- Μπορούμε επίσης να χρησιμοποιήσουμε ένα dictionary σε ένα if statement για να ελέγξουμε αν υπάρχει κάποια τιμή.

Παράδειγμα:

```
employee = {
   'first_name': 'George',
   'last_name': 'Athanasopoulos'
}
if 'first_name' in employee:
   print(f"The employee has a name {employee['first_name']}")
```

Output

```
The employee has a name George
```

Παράδειγμα εξάσκησης: Έστω ότι μία εμπορική εταιρεία καταγράφει για κάθε πωλητή την αξία κάθε πώλησης που πραγματοποίησε σε μία ημέρα. Να γραφεί πρόγραμμα το οποίο για κάθε πωλητή να διαβάζει από την οθόνη:

- το όνομα του πωλητή
- και την αξία πώλησης

Να αποθηκεύει τα παραπάνω στοιχεία σε ένα λεξικό.
Η διαδικασία να επαναλαμβάνει μέχρις ότου να δοθεί ως όνομα του πωλητή η λέξη ΤΕΛΟΣ.
Μετά το τέλος εισαγωγής των στοιχείων να εμφανίζει για κάθε πωλητή τη συνολική αξία πωλήσεων του.

Επίλυση:

```
poliseis={} # Δημιουργία κενού λεξικού {key:value,....}
# key:επώνυμο
# value: συνολική αξία πωλήσεων
epan=True
while epan:
    eponymo=input("επώνυμο πωλητή ή ΤΕΛΟΣ ")
    if eponymo!="ΤΕΛΟΣ":
        axia=float(input("Αξία πωλήσεων "))
        if eponymo in poliseis: # αν υπάρχει εγγραφή στο λεξικό με αυτό το κλειδί
            s_axia=poliseis.get(eponymo) # διαβάζεται η τιμή (value) της εγγραφής
            s_axia= s_axia + axia # αυξάνεται το περιεχόμενο της μεταβλητής axia
            poliseis[eponymo]=s_axia # η εγγραφή με το αντίστοιχο κλειδί
                                     # ενημερώνεται με τη νέα τιμή
        else:
             poliseis[eponymo]=axia # εισάγεται μία νέα εγγραφή
    else: epan=False
for politi in poliseis:
    print(politi, poliseis.get(politi)) # για κάθε κλειδί του λεξικού εμφανίζει το
                                        # κλειδί και την τιμή της εγγραφής
                                        # (επώνυμο και αξία πωλήσεων)
```

- Μία σειρά στοιχείων (αριθμών, συμβολοσειρών κλπ.) μπορούν να ομαδοποιηθούν κάτω από ένα όνομα (μίας μεταβλητής) σε μία δομή η οποία ονομάζεται **πλειάδα**. Τα στοιχεία της πλειάδας γράφονται μέσα σε παρένθεση και χωρίζονται με κόμμα.
- Ουσιαστικά, οι πλειάδες είναι το ίδιο με τις λίστες με τη μόνη διαφορά ότι δεν μπορείς να εισάγεις νέα τιμή μέσα σε αυτές και επίσης δεν μπορείς να αλλάξεις τις τιμές που περιέχει με ανάθεση τιμής σε κάποιο στοιχείο της. Είναι δηλαδή *μια αμετάλλακτη δομή. Θυμηθείτε ότι και οι συμβολοσειρές είναι μη-μεταλλάξιμος τύπος.*

Παράδειγμα:

```
names_grades1=['Μαρία',9,'Αντώνης',8]
for name_or_grade in names_grades1:
    print(name_or_grade)
print('*' * 80)
names_grades2=('Μαρία',9,'Αντώνης',8)
for name_or_grade in names_grades2:
    print(name_or_grade)
```

Δημιουργία λίστας

Δημιουργία πλειάδας

Output

```
Μαρία
9
Αντώνης
8
********************************************************************************
Μαρία
9
Αντώνης
8
```

- *Εφόσον οι πλειάδες είναι η ίδια δομή με τις λίστες γιατί να χρησιμοποιήσουμε τις πλειάδες από τη στιγμή μάλιστα που δεν μπορούμε να εισάγουμε νέα δεδομένα σε αυτές;*

```
print(dir(names_grades1))  ή  print(dir(list()))  (εκτύπωση λίστας μεθόδων διαθέσιμων για λίστες)
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy',
'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy',
'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

```
print(dir(names_grades2))  ή  print(dir(tuple()))  (εκτύπωση λίστας μεθόδων διαθέσιμων για πλειάδες)
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'count', 'index']
```

- Εκτός από τις ειδικές μεθόδους που ξεκινούν με τις _ , οι μόνες μέθοδοι διαθέσιμες σε πλειάδες είναι οι count και index ενώ σε λίστες είναι διαθέσιμες πολύ περισσότερες append, …., sort. Αυτό σημαίνει πως όταν δημιουργούμε ένα στιγμιότυπο τύπου πλειάδας αυτό θα καταλαμβάνει λιγότερο χώρο στη μνήμη του υπολογιστή μας, άρα κάνει καλύτερη διαχείριση.

- Ένα ακόμα πλεονέκτημα της πλειάδας σε σχέση με τις λίστες είναι ότι πρόκειται για μια αποδοτικότερη δομή δεδομένων. Επειδή η πλειάδα είναι αμετάλλακτη δομή εκτελείται ταχύτερα από ότι η λίστα. Όταν έχουμε προγράμματα τα οποία διαχειρίζονται πάρα πολλά δεδομένα θα ήταν καλύτερο να χρησιμοποιήσετε μια πλειάδα αντί για λίστα ώστε να έχετε ένα αποδοτικότερο πρόγραμμα.

```
demodata1=(1,2,3,4,5)
print(type(demodata1))
```

```
<class 'tuple'>
```

```
demodata1=(1,2,3,4,5)
demodata2=3,5,7,9,11
demodata3=10
print(type(demodata1))
print(type(demodata2))
print(type(demodata3))
```

```
<class 'tuple'>
<class 'tuple'>
<class 'int'>
```

```
demodata1=(1,2,3,4,5)
demodata2=3,5,7,9,11
demodata3=(10)
print(type(demodata1))
print(type(demodata2))
print(type(demodata3))
```

```
<class 'tuple'>
<class 'tuple'>
<class 'int'>
```

```
demodata1=(1,2,3,4,5)
demodata2=3,5,7,9,11
demodata3=(10,)
print(type(demodata1))
print(type(demodata2))
print(type(demodata3))
```

Δήλωση πλειάδας που περιέχει ένα μόνο στοιχείο

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```

- Εστιάζοντας στην μεταβλητή demodata2 παρατηρούμε ότι ένας ακόμα τρόπος να ορίσουμε μία πλειάδα είναι να δώσουμε πολλές τιμές σε μία μεταβλητή, οπότε αυτή θεωρείται από την python μία μεταβλητή πλειάδας. Οι τιμές δεν είναι απαραίτητο να ανήκουν σε έναν τύπο. Στο ακόλουθο παράδειγμα η πλειάδα x αποτελείται από 2 στοιχεία ακεραίων (int) και μια συμβολοσειρά (string).

```
x=1,2,'three'
print(type(x))
```

<class 'tuple’>

- Επιπλέον, μια πλειάδα μπορεί να έχει σαν στοιχεία άλλες πλειάδες ή ακόμα και λίστες.

```
x=1,2,'three'
y=[4,5,'six']
z=(x,y)
print(type(x))
print(type(y))
print(type(z))
```

<class 'tuple'>
<class 'list'>
<class 'tuple’>

- Ορισμός κενής πλειάδας.

```
t = ()
print(type(t))
```

<class 'tuple’>

- Υπάρχει η δυνατότητα να συνενώσετε πλειάδες χρησιμοποιώντας τον τελεστή της συνένωσης +

```
demodata1=(1,2,3,4,5)
demodata1+=(6,7,8,9,10)
print(demodata1)
```

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

- Ο τελεστής += εκτελεί τη συνένωση και την ταυτόχρονη απόδοση. Είναι ισοδύναμος με το να γράφαμε demodata1=demodata1+(6,7,8,9,10)

- Επίσης μπορούμε να χρησιμοποιήσουμε το σύμβολο του πολλαπλασιασμού (*) για να πάρουμε μία σειρά αντιγράφων μίας πλειάδας.

```
demodata2=3,5,7,9,11
print(3*demodata2)
```

(3, 5, 7, 9, 11, 3, 5, 7, 9, 11, 3, 5, 7, 9, 11)

Η νέα πλειάδα έχει προκύψει από την επανάληψη 3 φορές της demodata2

- Για να προσπελάσουμε κάποιο στοιχείο της πλειάδας το κάνουμε με τον ίδιο τρόπο με τον οποίο προσπελαύνουμε στοιχεία σε μια λίστα.

```
demodata2=3,5,7,9,11
print(demodata2[2])
```

```
7
```

Το πρώτο στοιχείο της πλειάδας έχει το δείκτη 0

```
demodata2=3,5,7,9,11
print(demodata2[-1])
```

```
11
```

Το τελευταίο στοιχείο της πλειάδας έχει το δείκτη -1 όταν η μέτρηση των στοιχείων γίνεται αντίστροφα

- Τεμαχισμός: Μπορούμε να ανακτήσουμε ένα τμήμα της πλειάδας χρησιμοποιώντας ένα εύρος δεικτών. Το αποτέλεσμα είναι μία νέα πλειάδα. (Η δεικτοδότηση είναι ακριβώς όπως στις συμβολοσειρές και στις λίστες.)

```
demodata2=3,5,7,9,11
print(demodata2[0:2])
```

```
(3, 5)
```

Εκτυπώνεται νέα πλειάδα η οποία αντιστοιχεί στα στοιχεία της αρχικής πλειάδας στις θέσεις 0 (πρώτη) και 1 (δεύτερη)

- Η μέθοδος count() επιστρέφει το πλήθος της εμφάνισης του στοιχείου που εισάγεται σαν όρισμα στη συνάρτηση.

```
demodata4=(1,2,1,4,5,1)
print(demodata4.count(1))
```

3

Επιστρέφει το πλήθος της εμφάνισης του αριθμού 1 μέσα στην πλειάδα

- Η μέθοδος index() επιστρέφει την πρώτη θέση στην οποία βρέθηκε το όρισμα της συνάρτησης.

```
demodata4=(1,2,1,4,5,1)
print(demodata4.index(1))
```

0

```
demodata4=(1,2,1,4,5,1)
print(1 in demodata4)
```

True

Η τιμή 1 υπάρχει στην πλειάδα demodata4

```
demodata4=(1,2,1,4,5,1)
print(10 in demodata4)
```

False

Η τιμή 10 δεν υπάρχει στην πλειάδα demodata4

- Μπορούμε να διαγράψουμε μία πλειάδα με την εντολή del.

```
demodata4=(1,2,1,4,5,1)
del demodata4
print(demodata4)
```

⇩

Exception has occurred: NameError
name 'demodata4' is not defined
File "C:\Users\chpra\OneDrive\Desktop\ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ\test.py", line 3, in <module>
print(demodata4)

- len(demodata4): δίνει τον αριθμό στοιχείων πλειάδας.
- max(demodata4), min(demodata4): δίνουν το μέγιστο και ελάχιστο στοιχείο πλειάδας (όταν αυτά είναι συγκρίσιμα μεταξύ τους).
- tuple: μετατρέπει σε πλειάδα (π.χ., L=[1,2,3]; t=tuple(L)).
- list: μετατρέπει σε λίστα (π.χ., t=(1,2,3); L=list(t)).
- Μπορείτε επίσης να χρησιμοποιήσετε τις πλειάδες σαν keys σε ένα λεξικό κάτι που δεν επιτρέπεται από την Python με λίστες.

```
locations = {
    (35.6823, 139.7531): "Τόκιο",
    (39.9017, 116.3914): "Πεκίνο",
    (52.5194, 13.4063): "Βερολίνο"
}# Γεωγραφικό πλάτος και μήκος πρωτευουσών

print(locations[(39.9017, 116.3914)])
```

⇩

Πεκίνο

- Οι πλειάδες χρησιμοποιούνται αρκετά συχνά σε εφαρμογές όταν θέλουμε να έχουμε πρόσβαση σε μη μεταλλάξιμες τιμές π.χ. σε εφαρμογές tracking μέσω GPS, όπου μπορούμε να ζητήσουμε την τρέχουσα τοποθεσία μας σε γεωμετρικό μήκος και πλάτος . Αυτό δεν αλλάζει οπότε μπορεί να αποθηκευτεί μέσα σε πλειάδα. Γενικά σε οποιαδήποτε εφαρμογή θέλετε να *έχετε πρόσβαση μόνο ανάγνωσης σε δεδομένα και όχι αλλαγής* τότε είναι καλύτερο να χρησιμοποιείτε πλειάδες.

Παράδειγμα: Κατασκευάστε μία πλειάδα η οποία θα περιέχει την ακολουθία αριθμών Fibonacci Fi, i=1,…,n για έναν θετικό ακέραιο n (τον οποίο θα εισάγετε όταν εκτελείται το πρόγραμμα).

Στα Μαθηματικά, οι αριθμοί Fibonacci είναι οι αριθμοί της παρακάτω ακέραιας ακολουθίας:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Εξ ορισμού, οι πρώτοι δύο αριθμοί Fibonacci είναι το 0 και το 1, και κάθε επόμενος αριθμός είναι το άθροισμα των δύο προηγούμενων αριθμών.

Σε μαθηματικούς όρους, η ακολουθία Fn των αριθμών Fibonacci ορίζεται από τον εξής αναδρομικό τύπο:

$F_n = F_{n-1} + F_{n-2}$, με $F_0 = 0$ και $F_1 = 1$.

Επίλυση:

```
fibonacci=(0,1)
n=int(input("Εισάγετε ένα θετικό ακέραιο αριθμό: "))
if n==0:
    fibonacci=(0,)
elif n==1:
        fibonacci=(0,1)
else:
    for i in range(2,n+1):
         fibonacci+=(fibonacci[i-2]+fibonacci[i-1],)
print(fibonacci)
```

- Τα σύνολα είναι ουσιαστικά σχεδόν τα ίδια με τις λίστες με τη διαφορά ότι η σειρά των στοιχείων δεν μας ενδιαφέρουν και επίσης ότι δεν περιέχουν διπλότυπες εγγραφές. Ένα σύνολο δηλώνεται μέσα σε αγκύλες {}.

Παράδειγμα:

```
my_set={'Python','Matlab'}
print(type(my_set))
```

Output

```
<class 'set'>
```

- Εναλλακτικά, για να ορίσουμε ένα σύνολο χρησιμοποιούμε τη **συνάρτηση set( )** και δηλώνουμε μέσα σε **αγκύλες { }** τα δεδομένα μας.

Παράδειγμα:

```
my_set=set({'Python','Matlab'})
print(type(my_set))
print(my_set)
```

Output

```
<class 'set'>
{'Python', 'Matlab'}
```

- Τα σύνολα δεν περιέχουν διπλότυπες εγγραφές οπότε αν το στοιχείο 'Matlab' υπήρχε και 2η φορά η Python δεν θα το υπολόγιζε.

Παράδειγμα:

```
my_set={'Python','Matlab','Matlab'}
print(len(my_set))
```

Output

```
2
```

- Ένας εύκολος τρόπος για να μπορείτε να βρείτε όλες τις διαθέσιμες μεθόδους που υποστηρίζει κάποιο αντικείμενο, στη προκειμένη περίπτωση είναι ένα σύνολο, είναι να χρησιμοποιήσουμε τη συνάρτηση dir( ).

Παράδειγμα:

```
my_set= {'Python','Matlab'}
print(dir(my_set))
```

Output

```
['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```

- Για να λάβετε βοήθεια για κάποια μέθοδο σε ένα αντικείμενο μπορείτε να κάνετε χρήση της help()

Παράδειγμα:

```
help(set.add)
```

Output

Help on method_descriptor:

add(...)
Add an element to a set.

This has no effect if the element is already present.

- Για να εισάγω ένα στοιχείο σε ένα σύνολο χρησιμοποιώ τη μέθοδο .add() .

Παράδειγμα:

```
my_set= {'Python','Matlab'}
my_set.add('R')
print(my_set)
```

Output

{'Python', 'R', 'Matlab'} ← *κάθε φορά που εκτελείτε την εντολή, η σειρά των στοιχείων στο σύνολο αλλάζει, είναι δηλαδή τυχαία και δεν εμφανίζεται πάντα όπως τα έχετε δηλώσει στη δομή. Αυτό συμβαίνει γιατί τα σύνολα είναι αδιάτακτες δομές.*

- Αυτό που μας ενδιαφέρει στα σύνολα είναι αν κάποιο στοιχείο βρίσκεται σε αυτά και όχι η θέση του

```
my_set= {'Python','Matlab','R'}
print('Matlab' in my_set)
print('Fortran' in my_set)
```

Output

```
True
False
```

- Για να διαγράψω ένα στοιχείο από ένα σύνολο χρησιμοποιώ τη μέθοδο .remove()

```
my_set= {'Python','Matlab','R'}
my_set.remove('R')
print(my_set)
```

Output

```
{'Python', 'Matlab'}
```

- Εναλλακτικά, για να διαγράψω ένα στοιχείο από ένα σύνολο χρησιμοποιώ τη μέθοδο .discard()

```
my_set= {'Python','Matlab','R'}
my_set.discard('R')
print(my_set)
```

Output

```
{'Python', 'Matlab'}
```

- Υπάρχει μια πολύ χρήσιμη διαφορά ανάμεσα στη remove() και στη discard() μέθοδο που είναι σημαντικό να γνωρίζετε. Αν προσπαθήσετε να διαγράψετε ένα στοιχείο μέσα από ένα set με τη χρήση του remove() και αυτό το στοιχείο δεν υπάρχει, η Python θα σας επιστρέψει λάθος. Αν όμως για την ίδια περίπτωση χρησιμοποιήσετε το discard() τότε η Python απλά θα αγνοήσει την εντολή χωρίς να σας επιστρέψει λάθος στο κώδικα σας.

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
my_set.remove('Fortran')
print(my_set)
```

Output

```
Traceback (most recent call last):
  File "c:\Users\chpra\OneDrive\Desktop\ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ\xara.py", line 2, in <module>
    my_set.remove('Fortran')
KeyError: 'Fortran'
```

```
my_set= {'Python','Matlab','R'}
my_set.discard('Fortran')
print(my_set)
```

Output

```
{'R', 'Matlab', 'Python'}
```

- Για να διαγράψω τα στοιχεία ενός συνόλου χρησιμοποιώ τη μέθοδο .clear()

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
my_set.clear()
print(my_set)
```

Output

```
set()
```

- Μπορούμε να διαγράψουμε ένα σύνολο με την εντολή del

Παράδειγμα:

```
my_set= {'Python','Matlab','R’}
del my_set
```

- Όπως και στους προηγούμενους τύπους δεδομένων που έχουμε μελετήσει, έτσι και στα σύνολα μπορούμε να εκτελέσουμε μια δομή επανάληψης for για να πάρουμε μια λίστα με όλα τα δεδομένα που περιέχει

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
for language in my_set:
    print(language)
```

Output

```
Matlab
Python
R
```

- Η εντολή "x" in mySet, που ελέγχει αν το x υπάρχει στο σύνολο mySet

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
if 'Matlab' in my_set:
    print('Το στοιχείο υπάρχει στο σύνολο')
else:
    print('Το στοιχείο δεν υπάρχει στο σύνολο')
```

Output

Το στοιχείο υπάρχει στο σύνολο

- Η εντολή "x" not in mySet ελέγχει αν το x δεν υπάρχει στο σύνολο mySet

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
if 'Matlab' not in my_set:
    print('Το στοιχείο δεν υπάρχει στο σύνολο')
else:
    print('Το στοιχείο υπάρχει στο σύνολο')
```

Output

Το στοιχείο υπάρχει στο σύνολο

- Έστω τα σύνολα Α και Β. Για να ελέγξω ποια στοιχεία του συνόλου Α δεν υπάρχουν στο σύνολο Β θα χρησιμοποιήσω τη μέθοδο .difference() .

Παράδειγμα:

```
entire_set={'JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C'}
my_set= {'Python','Matlab','R'}
print(entire_set.difference(my_set))
```

ελέγχω ποια στοιχεία του entire_set δεν υπάρχουν στο my_set

Output

```
{'Java', 'SQL', 'JavaScript', 'C'}
```

```
entire_set={'JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C'}
my_set= {'Python','Matlab','R'}
print(my_set.difference(entire_set))
```

ελέγχω ποια στοιχεία του my_set δεν υπάρχουν στο entire_set

Output

```
{'R'}
```

- Μπορούμε να βρούμε τα κοινά στοιχεία δύο συνόλων με τη χρήση της μεθόδου intersection() .

Παράδειγμα:

```
entire_set={'JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C'}
my_set= {'Python','Matlab','R'}
print(entire_set.intersection(my_set))
```

Output

```
{'Matlab', 'Python'}
```

- Μπορούμε να βρούμε την ένωση δυο συνόλων με χρήση της μεθόδου .union()

Παράδειγμα:

```
entire_set={'JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C'}
my_set= {'Python','Matlab','R'}
print(entire_set.union(my_set))
```

Output

```
{'Matlab', 'R', 'Python', 'C', 'SQL', 'JavaScript', 'Java'}
```

- Λοιπές μέθοδοι συγκρίσεων συνόλων:

Παράδειγμα:

```
entire_set={'JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C'}
my_set= {'Python','Matlab'}

print(entire_set.isdisjoint(my_set))
```

Output

False

Επιστρέφει True αν δεν υπάρχουν κοινά στοιχεία μεταξύ των συνόλων entire_set και my_set. Εδώ επιστρέφει False καθώς τα στοιχεία 'Python' και 'Matlab' είναι κοινά στα 2 σύνολα.

```
print(entire_set.issubset(my_set))
```

Output

False

Επιστρέφει True όταν όλα τα στοιχεία του συνόλου entire_set υπάρχουν στο σύνολο my_set.

```
print(entire_set.issuperset(my_set))
```

Output

True

Επιστρέφει True όταν το σύνολο entire_set περιέχει όλα τα στοιχεία του συνόλου my_set.

- Επειδή τα σύνολα κρατάνε μόνο μοναδικά στοιχεία, μπορούμε να χρησιμοποιήσουμε αυτό το χαρακτηριστικό τους για να βρούμε τις μοναδικές τιμές μιας λίστας μετατρέποντας τη λίστα σε σύνολο και πάλι πίσω σε λίστα. Σκεφτείτε την περίπτωση να έχετε λάβει εκατοντάδες χιλιάδες εγγραφές από μια βάση και θέλετε γρήγορα να επιλέξετε μόνο τις μοναδικές τιμές. Αυτή η τακτική θα σας σώσει πολύ χρόνο.

Παράδειγμα:

```
entire_list=['JavaScript', 'Java','Matlab', 'Python', 'SQL', 'C','R','C','Matlab','Python','C']
unique_list=list(set(entire_list))
print(unique_list)
```

Output

['Matlab', 'C', 'R', 'SQL', 'Java', 'Python', 'JavaScript']

- Για να δημιουργήσουμε ένα αντίγραφο του συνόλου κάνουμε χρήση της μεθόδου .copy()

Παράδειγμα:

```
my_set= {'Python','Matlab','R'}
my_set2=my_set.copy()
my_set.add('C')
print(my_set)
print(my_set2)
```

Τι θα γινόταν αν γράφαμε την εντολή
`my_set2=my_set;`

Output

{'Matlab', 'C', 'R', 'Python'}
{'Matlab', 'R', 'Python'}