Σημειωματάριο Τετάρτης 15 Νοε. 2017

Συνάρτηση για υπολογισμό όλων των υποσυνόλων ενός συνόλου (δυναμοσύνολο)

Για να βρούμε τα υποσύνολα του συνόλου $S = \{s_1, s_2, \ldots, s_n\}$ τα χωρίζουμε σε δύο κατηγορίες. Στην κατηγορία 1 μπαίνουν όλα τα υποσύνολα που δεν περιέχουν το στοιχείο s_1 ενώ στην κατηγορία 2 αυτά που το περιέχουν.

Παρατηρούμε ότι τα υποσύνολα κατηγορίας 1 είναι ακριβώς όλα τα υποσύνολα του $S' = \{s_2, s_3, \ldots, s_n\}$. Επίσης τα υποσύνολα κατηγορίας 1 και τα υποσύνολα κατηγορίας 2 είναι σε 1 προς 1 αντιστοιχεία μέσω της απεικόνισης

$$A \to A \cup \{s_1\}$$

που απεικονίζει το τυχόν σύνολο A κατηγορίας 1 σε ένα μοναδικό υποσύνολο κατηγορίας 2.

Η συνάρτησή μας λοιπόν, `subsets(S)` λειτουργεί αναδρομικά. Η βασική περίπτωση (εκεί όπου σταματάνε οι αναδρομικές κλήσεις προς τον εαυτό της) είναι όταν το σύνολό μας είναι το κενό, όταν δηλ. S=`[]`, οπότε το μόνο υποσύνολο είναι το κενό και άρα η λίστα με όλα τα υποσύνολα (που επιστρέφει η `subsets`) είναι η `[ [] ]`.

Διαβάστε τα σχόλια μέσα στη συνάρτηση για να δείτε πώς δουλεύει.

In [7]:

```
import pprint as pp

def subsets(S):
    """
    Επιστρέφει μια λίστα με όλα τα υποσύνολα του S (που είναι επίσης μια λίστα)
    """
    if len(S) == 0: # Αν S είναι το κενό σύνολο τότε το μόνο υποσύνολο είναι το
 κενό.
        return [[]]

    L = subsets(S[1:]) # Εδώ υπολογίζουμε όλα τα υποσύνολα που δεν περιέχουν το
 πρώτο στοιχείο (αναδρομική κλήση)

    powerset = []
    for s in L: # Εδώ βάζουμε μαζί όλα τα υποσύνολα κατηγορίας 1 (που βρήκαμε με
 την αναδρομική κλήση) και για κάθε
                # υποσύνολο κατηγορίας 1 βάζουμε και το υποσύνολο κατηγοριας 2 π
ου προκύπτει αν προσθέσουμε στο
                # υποσύνολό μας το στοιχείο s_1
        powerset.append(s)
        powerset.append([S[0]] + s)
    return powerset

pp.pprint(subsets([1, 2, 3]), width=40) # Τυπώνουμε τη λίστα με τον pretty print
er σε πλάτος 40 χαρακτήρων
```

```
[[],
 [1],
 [2],
 [1, 2],
 [3],
 [1, 3],
 [2, 3],
 [1, 2, 3]]
```

Λεξικά (dictionaries)

Ένα dictionary στην python είναι σα μια λίστα, χωρίς εσωτερική διάταξη, που τα στοιχεία της γίνονται indexed όχι κατ' ανάγκη από φυσικούς αριθμούς αλλά και από λέξεις. Τα περιεχόμενα ενός λεξικού είναι ζεύγη της μορφής

```
key: object
```

όπου το κλειδί `key` μπορεί να είναι αριθμός ή string και το object μπορεί να είναι οτιδήποτε. Για κάθε κλειδί πρέπει να υπάρχει ένα μοναδικό ζεύγος (αν γράψουμε κι άλλο επικρατεί το τελευταίο στη σειρά).

Στο παρακάτω παράδειγμα βλέπουμε ένα λεξικό με όνομα `age`, τα ζεύγη του οποίου είναι της μορφής "όνομα: ηλικία". Παρατηρείστε ότι στον ορισμό του λεξικού χρησιμοποιούμε άγκιστρα `{}` και όχι αγκύλες `[]` όπως κάνουμε για τις λίστες. Όταν όμως αναφερόμαστε σε κάποιο στοιχείο όπως κάνουμε στη δεύτερη και στην τρίτη γραμμή του προγράμματος τότε χρησιμοποιούμε αγκύλες.

In [8]:

```
age = {"Mihalis": 48, "Manolis": 50, "Nikos": 12}
age["Nikos"] += 2
print(age["Nikos"])
print(age)
```

```
14
{'Nikos': 14, 'Mihalis': 48, 'Manolis': 50}
```

Εδώ βλέπουμε στη δεύτερη γραμμή το πώς ελέγχουμε αν ένα κλειδί `"Yannis"` υπάρχει σε ένα λεξικό. Αν δεν υπάρχει το προσθέτουμε με ηλικία 55 ενώ αν υπάρχει τυπώνουμε την ηλικία του (συμβαίνει το πρώτο). Στην προτελευταία γραμμή διαγράφουμε το ζεύγος με key `"Manolis"` από το λεξικό με την εντολή `del`.

In [10]:

```
age = {"Mihalis": 48, "Manolis": 50, "Nikos": 12}
if "Yannis" in age:
    print(age["Yannis"])
else:
    age["Yannis"] = 55
del age["Manolis"]
print(age)
```

```
{'Nikos': 12, 'Yannis': 55, 'Mihalis': 48}
```

Στο επόμενο πρόγραμμα βλέπουμε πώς διανύουμε όλα τα κλειδιά ενός λεξικού με το `for`.

In [12]:

```
D = { "Mihalis": 49, "Maria": 25, "Yannis": 150, "Eleni": 22, "Markos": 30}

for k in D:
    print("The age of {} is {}".format(k, D[k]))
```

```
The age of Eleni is 22
The age of Markos is 30
The age of Yannis is 150
The age of Mihalis is 49
The age of Maria is 25
```

Κι εδώ αυξάνουμε την ηλικία κάθε ατόμου (κλειδιού) στο λεξικό.

In [13]:

```
D = { "Mihalis": 49, "Maria": 25, "Yannis": 150, "Eleni": 22, "Markos": 30}

for k in D:
    D[k] += 1

for k in D:
    print("The age of {} is {}".format(k, D[k]))
```

```
The age of Eleni is 23
The age of Markos is 31
The age of Yannis is 151
The age of Mihalis is 50
The age of Maria is 26
```

Τα κλειδιά ενός λεξικού μπορούν να είναι αριθμοί ή strings (ή και tuples, αλλά δεν έχουμε μιλήσει ιδιαίτερα για αυτά). Δε μπορεί π.χ. να είναι λίστα, όπως φαίνεται παρακάτω όπου παράγεται σφάλμα όταν πάμε να ορίσουμε ένα λεξικό με κλειδί που είναι λίστα.

In [14]:

```
D = { [1, 2]: 3.5 }
```

```
---------------------------------------------------------------------
--------
TypeError                                 Traceback (most recent ca
ll last)
<ipython-input-14-38acac7a165f> in <module>()
----> 1 D = { [1, 2]: 3.5 }

TypeError: unhashable type: 'list'
```

Στο επόμενο πρόγραμμα φτιάχνουμε στο λεξικό H το ιστόγραμμα των τιμών που είναι στη λίστα L. Για κάθε τιμή (φυσικό αριθμό) k που εμφανίζεται στην L δηλ. βάζουμε το ζεύγος k:n όπου n είναι το πόσες φορές υπάρχει η τιμή k στη λίστα L.

In [15]:

```
L = [1, 1, 2, 3, 2, 1, -1, 5, 5, 10, 1, 2, -1] # Εδώ είναι οι αριθμοί των οποίων
 θέλουμε το ιστόγραμμα

H = {} # Αρχικά το λεξικό H είναι κενό.

for k in L: # Το k διανύει την L
    if k in H.keys(): # Αν υπάρχει ήδη ως κλειδί στο H τότε το έχουμε ξαναδεί κα
ι αυξάνουμε τον αριθμό του
        H[k] += 1 # μια παραπάνω εμφάνιση για το k
    else:
        H[k] = 1 # Δεν το έχουμε ξαναδεί το k οπότε το εισάγουμε με αριθμό εμφαν
ίσεων (προς το παρόν) ένα.
print(H) # Τυπώνουμε το λεξικό/ιστόγραμμα

D = H.copy() # Εδώ φτιάχνουμε ένα αντίγραφο του λεξικού

del H[10] # Σβήνουμε από το H το ζεύγος με κλειδί 10. Δε σβήνεται όμως από το D.
 Αν αντί για D=H.copy() είχαμε
          # κάνει H = D τα D και H θα ήταν απλά δύο διαφορετικά ονόματα για το
 ίδιο αντικείμενο και τότε
          # οποιαδήποτε αλλαγή στο ένα φαίνεται και στο άλλο.

print(D) # Τυπώνουμε το D

for k in sorted(H.keys()): # Εδώ διανύουμε τα κλειδιά του H, ταξινομημένα.
    print("Η τιμή {} εμφανίζεται {} {}.".format(k, H[k],"φορές" if H[k]>1 else
"φορά"))
```

```
{1: 4, 2: 3, 3: 1, 5: 2, 10: 1, -1: 2}
{1: 4, 2: 3, 3: 1, 5: 2, 10: 1, -1: 2}
Η τιμή -1 εμφανίζεται 2 φορές.
Η τιμή 1 εμφανίζεται 4 φορές.
Η τιμή 2 εμφανίζεται 3 φορές.
Η τιμή 3 εμφανίζεται 1 φορά.
Η τιμή 5 εμφανίζεται 2 φορές.
```