Η εντολή while

Η εντολή `while` χρησιμοποιείται για επανάληψη. Με την εντολή `while <ΛΟΓΙΚΗ ΣΥΝΘΗΚΗ>:` δηλώνουμε ότι όλες οι εντολές "κάτω" από τη while θα τρέχουν μέχρι η `<ΛΟΓΙΚΗ ΣΥΝΘΗΚΗ>` να γίνει `False` .

In [1]:
```
# Τύπωσε όλους τους αριθμούς από το 0 μέχρι και το 9
a=0
while a<10:
    print (a)
    a += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

Τύπωσε όλους τους **μονούς** αριθμούς από το 1 μέχρι το 50

In [3]:
```
a=1

while a<50:
    if a%2 == 1:
        print (a)
    a+=1
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
```

Τύπωσε τη "προπαίδεια" του 8

In [4]:
```
a = 1
while a<=10:
    print (a*8)
    a+=1
```

```
8
16
24
32
40
48
56
64
72
80
```

Λίγο καλύτερα:

In [5]:
```
a = 1
while a<=10:
    print (a, 'fores to 8 mas kanei', a*8)
    a+=1
```

```
1 fores to 8 mas kanei 8
2 fores to 8 mas kanei 16
3 fores to 8 mas kanei 24
4 fores to 8 mas kanei 32
5 fores to 8 mas kanei 40
6 fores to 8 mas kanei 48
7 fores to 8 mas kanei 56
8 fores to 8 mas kanei 64
9 fores to 8 mas kanei 72
10 fores to 8 mas kanei 80
```

Τύπωσε τη προπαίδεια όλων των αριθμών από το 1 μέχρι το 10

In [6]:
```
a = 1
while a<=10:
    b=1
    while b<=10:
        print (a, 'fores to ', b, ' mas kanei', a*b)
        b+=1
    a+=1
```

```
1 fores to  1  mas kanei 1
1 fores to  2  mas kanei 2
1 fores to  3  mas kanei 3
1 fores to  4  mas kanei 4
1 fores to  5  mas kanei 5
1 fores to  6  mas kanei 6
1 fores to  7  mas kanei 7
1 fores to  8  mas kanei 8
1 fores to  9  mas kanei 9
1 fores to  10  mas kanei 10
2 fores to  1  mas kanei 2
2 fores to  2  mas kanei 4
2 fores to  3  mas kanei 6
2 fores to  4  mas kanei 8
2 fores to  5  mas kanei 10
2 fores to  6  mas kanei 12
2 fores to  7  mas kanei 14
2 fores to  8  mas kanei 16
2 fores to  9  mas kanei 18
2 fores to  10  mas kanei 20
3 fores to  1  mas kanei 3
3 fores to  2  mas kanei 6
3 fores to  3  mas kanei 9
```

```
3 fores to  4  mas kanei 12
3 fores to  5  mas kanei 15
3 fores to  6  mas kanei 18
3 fores to  7  mas kanei 21
3 fores to  8  mas kanei 24
3 fores to  9  mas kanei 27
3 fores to  10  mas kanei 30
4 fores to  1  mas kanei 4
4 fores to  2  mas kanei 8
4 fores to  3  mas kanei 12
4 fores to  4  mas kanei 16
4 fores to  5  mas kanei 20
4 fores to  6  mas kanei 24
4 fores to  7  mas kanei 28
4 fores to  8  mas kanei 32
4 fores to  9  mas kanei 36
4 fores to  10  mas kanei 40
5 fores to  1  mas kanei 5
5 fores to  2  mas kanei 10
5 fores to  3  mas kanei 15
5 fores to  4  mas kanei 20
5 fores to  5  mas kanei 25
5 fores to  6  mas kanei 30
5 fores to  7  mas kanei 35
5 fores to  8  mas kanei 40
5 fores to  9  mas kanei 45
5 fores to  10  mas kanei 50
6 fores to  1  mas kanei 6
6 fores to  2  mas kanei 12
6 fores to  3  mas kanei 18
6 fores to  4  mas kanei 24
6 fores to  5  mas kanei 30
6 fores to  6  mas kanei 36
6 fores to  7  mas kanei 42
6 fores to  8  mas kanei 48
6 fores to  9  mas kanei 54
6 fores to  10  mas kanei 60
7 fores to  1  mas kanei 7
7 fores to  2  mas kanei 14
7 fores to  3  mas kanei 21
7 fores to  4  mas kanei 28
7 fores to  5  mas kanei 35
7 fores to  6  mas kanei 42
7 fores to  7  mas kanei 49
7 fores to  8  mas kanei 56
7 fores to  9  mas kanei 63
7 fores to  10  mas kanei 70
8 fores to  1  mas kanei 8
8 fores to  2  mas kanei 16
8 fores to  3  mas kanei 24
8 fores to  4  mas kanei 32
8 fores to  5  mas kanei 40
8 fores to  6  mas kanei 48
8 fores to  7  mas kanei 56
8 fores to  8  mas kanei 64
8 fores to  9  mas kanei 72
8 fores to  10  mas kanei 80
9 fores to  1  mas kanei 9
9 fores to  2  mas kanei 18
9 fores to  3  mas kanei 27
9 fores to  4  mas kanei 36
9 fores to  5  mas kanei 45
9 fores to  6  mas kanei 54
9 fores to  7  mas kanei 63
9 fores to  8  mas kanei 72
9 fores to  9  mas kanei 81
9 fores to  10  mas kanei 90
10 fores to  1  mas kanei 10
10 fores to  2  mas kanei 20
10 fores to  3  mas kanei 30
10 fores to  4  mas kanei 40
10 fores to  5  mas kanei 50
```

```
10 fores to  6  mas kanei 60
10 fores to  7  mas kanei 70
10 fores to  8  mas kanei 80
10 fores to  9  mas kanei 90
```

Βρες πόσα ψηφία έχει ένας αριθμός:

In [7]:
```
a=51234123
# Πρώτος τρόπος (fast and better)
len(str(a))
```

Out[7]: 8

In [8]:
```
# Δεύτερος τρόπος
# Παρατηρούμε ότι όταν δαιρούμε (ακέραια διαίρεση) έναν αριθμό με το 10, τότε
# 51234123 // 10 --> 5123412

a=51234123
c=0
upoloipo = a
while upoloipo != 0:
    upoloipo = upoloipo // 10
    c += 1
print (c)
```

```
8
```

Πόσες φορές υπάρχει το γράμμα a σε ένα string;

In [9]:
```
# Πρώτος τροπος (better / faster)
a = 'zabarakatranemia'
a.count('a')
```

Out[9]: 6

In [10]:
```
# Δεύτερος τρόπος
index = 0
c = 0
while index < len(a):
    if a[index] == 'a':
        c += 1
    index += 1
print (c)
```

```
6
```

Αντιστροφή ενός string.

In [94]:
```
# Πρώτος τρόπος (better / faster)
a = 'zabarakatranemia'
a[::-1]
```

Out[94]: 'aimenartakarabaz'

In [95]:
```
# Δεύτερος τρόπος
index = len(a)-1
anapodo = ''
while index >= 0:
    anapodo = anapodo + a[index]
    index -= 1
print (anapodo)
```

```
aimenartakarabaz
```

Το άθροισμα όλων των αριθμών από το 1 μέχρι 20

In [102…
```
s = 0
c = 0
while c < 20:
    c += 1
    s += c
print (s)
```

```
210
```

In [103…
```
s = 0
c = 1
while c <= 20:
    s += c
    c += 1
print (s)
```

```
210
```

In [98]:
```
sum(range(1,21))
```

Out[98]:
```
210
```

Όπως και με τη for μπορούμε να κάνουμε break και continue. Με τη break βγαίνουμε τελείως από τη while και με τη continue μεταβαίνουμε στην αρχή της while όπου γίνεται η εκτίμηση της λογικής συνθήκης.

In [105…
```
a=0
while a<10:
    a += 1
    if a== 5:
        continue

    print (a)
```

```
1
2
3
4
6
7
8
9
10
```

Προσέξτε ότι το 5 δεν υπάρχει.

In [107…
```
a=0
while a<10:
    a += 1
    if a== 5:
        break

    print (a)
```

```
1
2
3
4
```

Και εδώ όταν το a γίνει 5 τότε βγαίνει τελείως από τη while.

Κάτι που χρησιμοποιείται σπάνια αλλά είναι ιδιαίτερα χρήσιμο είναι η else μετά τη while. Σε αυτή την else μπαίνει μόνο αν αν έχει συμβεί break μέσα στη while.

In [108…
```
a=0
while a<10:
    a += 1
    if a== 5:
        break

    print (a)
else:
    print ('No break happened')
```

```
1
2
3
4
```

In [110…
```
a=0
while a<10:
    a += 1
    #if a== 5:
    #    break

    print (a)
else:
    print ('No break happened')
```

```
1
2
3
4
5
6
7
8
9
10
No break happened
```

Τη while τη χρησιμοποιούμε πολλές φορές όταν θέλουμε να κάνουμε μία επανάληψη αλλά δεν ξέρουμε πόσες φορές πρέπει να γίνει αυτή η επανάληψη. Για παράδειγμα: αφήνουμε να πέσει μία μπάλα από το 1 μέτρο. Κάθε φορά που αναπηδάει φτάνει στο 90% του ύψους της. Μετά από 5 αναπηδήσεις σε τι ύψος θα έχει φτάσει. Εδώ ξέρουμε πόσες επαναλήψεις θα κάνουμε οπότε θα χρησιμοποιήσουμε for:

In [112…
```
height=1
for i in range(5):
    height -= 0.1*height
print (height)
```

```
0.5904900000000001
```

Ας δούμε τώρα ένα άλλο πρόβλημα: Πόσες αναπηδήσεις πρέπει να γίνουν ώστε το ύψος της μπάλας να γίνει μικρότερο από 0.5 μέτρα; Τώρα δεν ξέρουμε το πλήθος από επαναλήψεις (για την ακρίβεια αυτό είναι και το ζητούμενο), άρα "βολεύει" h while:

In [114…
```
height = 1
bounces = 0
while height > 0.5:
    bounces += 1
    height -= 0.1*height
print (bounces)
```

```
7
```

Ένα άλλο παράδειγμα: Η παρακάτω συνάρτηση ελέγχει αν ένας αριθμός είναι πρώτος ή όχι:

In [129…
```
def is_prime(n):
    for x in range(2, int(n**0.5)+1):
        if n%x==0:
            return False

    return True
```

Αν αρχίζουμε και αθροίζουμε όλους τους πρώτους ξεκινώντας από το 1 σε ποιον πρώτο αριθμό αυτό το άθροισμα θα ξεπεράσει το 1.000.000 ;

In [131…
```
s = 0
c = 1
while s < 1_000_000:
    if is_prime(c):
        s += c
    c += 1
print (c-1)
```

```
3943
```

In [ ]:
```
```

Tuples

Τα tuples είναι δομές δεδομένων που μοιάζουν με τη λίστα. Η διαφορά τους είναι ότι στα tuples δεν μπορούμε να αλλάξουμε μία τιμή. Αντί για αγκύλες (`[1,2,3]`) στα tuples χρησημοποιούμε παρενθέσεις (`(1,2,3)`).

In [84]:
```
a = (1,2,3)
type(a)
```

Out[84]:
```
tuple
```

In [85]:
```
a[2] = 7 #Πετάει μήνυμα λάθους. Δεν μπορούμε να το αλλάξουμε
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-85-42aa7089a9ac> in <module>
----> 1 a[2] = 7 #Πετάει μήνυμα λάθους. Δεν μπορούμε να το αλλάξουμε

TypeError: 'tuple' object does not support item assignment
```

In [86]:
```
b = [1,2,3]
b[2] = 7 # Αυτό είναι οκ, δεν πετάει μήνυμα λάθους
```

Παρόλο που στα tuples δεν μπορώ να προσθέσω ή να αφαιρέσω ένα στοιχείο μπορώ να βάλω στοιχεία στις λίστες ή στα dictionaries που περιέχουν:

In [87]:
```
a = (1,[4,5],10)
a[1].append(6)
print (a)
```

```
(1, [4, 5, 6], 10)
```

Επίσης μπορούμε να χρησιμοποιήσουμε τα tuples όπως ακριβώς τις λίστες για να κάνουμε επανάληψη, min, max, sort, ...

In [133…
```
for x in (1,2,3):
    print (x)
```

```
1
2
3
```

In [135…
```
sum((5,6,7))
```

Out[135… 18

Συναρτήσεις που επιστρέφουν παραπάνω από 1 τιμή

Στη python μία συνάρτηση μπορεί να επιστρέψει παραπάνω από μία τιμή:

In [88]:
```
def f():
    return 1,2
```

In [89]:
```
a,b = f()
print (a)
print (b)
```

```
1
2
```

Αν αποθηκεύσουμε σε μία μόνο μεταβλητή το αποτέλεσμα μίας συνάρτησης η οποία επιστρέφει παραπάνω από μία τιμές τότε αυτό που επιστρέφει είναι ένα tuple.

In [90]:
```
a = f()
```

In [91]:
```
print (a)
```

```
(1, 2)
```

In [92]:
```
type(a)
```

Out[92]:
```
tuple
```

In [ ]:

In [ ]:

Dictionaries

Μέχρι στιγμής έχουμε μάθει τους παρακάτω τύπους μεταβλητών:

In [13]:
```
a=0 # ακέραιοι
a=True # λογικοί
a="324234" # αλφαρηθμιτικά
a=5.6 # δεκαδικοί
a=[2,4,4] # λίστες
a=None # None
```

Τα dictionaries είναι ένα νέος τύπος μεταβλητής. Τα dictionaries έχουν δεδομένα με τη μορφή κλειδί --> τιμή. Κάθε κλειδί (key) είναι μοναδικό. Για παράδειγμα:

In [136…
```
a = {"mitsos": 50, "anna": 40}
```

In [15]:
```
print(a)
```

```
{'mitsos': 50, 'anna': 40}
```

In [16]:
```
a['mitsos']
```

Out[16]:
```
50
```

In [17]:
```
a['anna']
```

Out[17]:
```
40
```

Η `keys` επιστρέφει μία λίστα με όλα τα κλειδιά του dictionary

In [18]:
```
a.keys()
```

Out[18]:
```
dict_keys(['mitsos', 'anna'])
```

Η `values` επιστρέφει μία λίστα με όλες τις τιμές του dictionary

In [19]:
```
a.values()
```

Out[19]:
```
dict_values([50, 40])
```

Μπορούμε να προσθέσουμε ένα νέο ζευγάρι κλειδί,τιμή με τον εξής τρόπο:

In [137…
```
a["kitsos"] = 100
```

In [139…
```
print (a)
```
```
{'mitsos': 50, 'anna': 40, 'kitsos': 100}
```

Επίσης μπορούμε να αφαιρέσουμε ένα ζευγάρι κλειδί,τιμή με την εντολή del:

In [140…
```
del a['kitsos']
print (a)
```
```
{'mitsos': 50, 'anna': 40}
```

Το κλειδί μπορεί να είναι αριθμός, string και boolean και tuple. Ενώ το value μπορεί να είναι οτιδίποτε.

In [22]:
```
a[123] = 0.1
a[3.14] = "hello"
a[False] = [1,2,3]
a[(4,7)] = 4
```

In [23]:
```
print (a)
```
```
{'mitsos': 50, 'anna': 40, 'kitsos': 100, 123: 0.1, 3.14: 'hello', False: [1,
2, 3], (4, 7): 4}
```

In [24]:
```
# Προσοχή! False == 0 !
a[0]
```

Out[24]:
```
[1, 2, 3]
```

Το κλειδί ΔΕΝ μπορεί να είναι λίστα:

In [141…
```
a[[1,2,3]] = 0
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-141-6cebb9942dfe> in <module>
----> 1 a[[1,2,3]] = 0

TypeError: unhashable type: 'list'
```

Το κλειδί ΔΕΝ μπορεί να είναι ούτε dictionary:

In [142…
```
a[{}] = 0
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-142-b372ccb1b9be> in <module>
----> 1 a[{}] = 0

TypeError: unhashable type: 'dict'
```

Στη python μπορούμε να έχουμε dictionaries μέσα σε lists και lists μέσα σε dictionaries χωρίς κανένα περιορισμό

In [26]:
```
d = {"a": {2:"a"}, 3: ["hello", False, []], 3.1: True}
print (d)
```
```
{'a': {2: 'a'}, 3: ['hello', False, []], 3.1: True}
```

Μπορούμε να συνθέσουμε listes και dictionaries από άλλες listes και dictionaries:

In [27]:
```
[d, d, d["a"]]
```

Out[27]:
```
[{'a': {2: 'a'}, 3: ['hello', False, []], 3.1: True},
 {'a': {2: 'a'}, 3: ['hello', False, []], 3.1: True},
 {2: 'a'}]
```

In [28]:
```
{"a": d, "b": d[3]}
```

Out[28]:
```
{'a': {'a': {2: 'a'}, 3: ['hello', False, []], 3.1: True},
 'b': ['hello', False, []]}
```

Υπάρχει και το άδειο dictionary

In [29]:
```
a = {}
```

Η `len` επιστρέφει το πλήθος των εγγραφών που έχει ένα dictionary:

In [30]:
```
person = {"name": "alex", "age": 50, "occupation": "master"}
```

In [31]:
```
len(person)
```

Out[31]:
```
3
```

In [32]:
```
len({})
```

Out[32]:
```
0
```

Μπορούμε να ελέγξουμε αν ένα κλειδί υπάρχει σε ένα dictionary

In [33]:
```
"name" in person
```

Out[33]:
```
True
```

In [34]:
```
"alex" in person
```

Out[34]:
```
False
```

Μπορούμε να ελέγξουμε αν μία τιμή υπάρχει σε dictionary:

In [35]:
```
"alex" in person.values()
```

Out[35]:
```
True
```

Μπορούμε να κάνουμε επανάληψη σε όλα τα στοιχεία ενός dictionary:

In [36]:
```
for i in person:
    print (i)
```

```
name
age
occupation
```

In [37]:
```
for i in person:
    print ("key: {}  Value: {}".format(i, person[i]))
```

```
key: name  Value: alex
key: age  Value: 50
key: occupation  Value: master
```

Μπορούμε να μετατρέψουμε μία λίστα (ή tuple) σε dictionary με τη συνάρτηση dict. Πρέπει όμως η λίστα να αποτελείται από υπολίστες, όπου η κάθε υπολίστα έχει 2 στοιχεία. Σε αυτές

τις υπολίστες το πρώτο στοιχείο θα γίνει το κλειδί και το δεύτερο η τιμή:

In [146…
```
a = [("mitsos", 1), ('maria', 2), ('elenh', 4) ]
dict(a)
```

Out[146…
```
{'mitsos': 1, 'maria': 2, 'elenh': 4}
```

Μπορεί να γίνει και το αντίθετο με τη items()

In [148…
```
a = {'mitsos': 1, 'maria': 2, 'elenh': 4}
print(list(a.items()))
```

```
[('mitsos', 1), ('maria', 2), ('elenh', 4)]
```

Προσπέλαση στοιχείων σε dictionary

Μπορούμε να χρησιμοποιήσουμε πάνω από μια φορά το `[][]` ώστε να προσπελάσουμε κάποιο στοιχείο:

In [38]:
```
person = {"name": "alex", "age": 50, "occupation": "master", "exper": ["pytho
```

In [39]:
```
print (person)
```

```
{'name': 'alex', 'age': 50, 'occupation': 'master', 'exper': ['python', 'karat
e']}
```

In [40]:
```
print (person['exper'][0])
```

```
python
```

In [41]:
```
print (person['exper'][1])
```

```
karate
```

In [42]:
```
print (person['exper'])
```

```
['python', 'karate']
```

In [43]:
```
a = ["a", "b", {"name": "mitsos", "surnmae": "sdfsdfsdf"}]
```

In [45]:
```
a[0]
```

Out[45]:
```
'a'
```

In [47]:
```
a[1]
```

Out[47]:
```
'b'
```

In [48]:
```
a[2]
```

Out[48]:
```
{'name': 'mitsos', 'surnmae': 'sdfsdfsdf'}
```

In [49]:
```
a[2]['name']
```

Out[49]:
```
'mitsos'
```

Η συνάρτηση `a.get(b,c)` ελέγχει αν το b υπάρχει στο dictionary a. Αν υπάρχει επιστρέφει τη τιμή: `a[b]` . Αν δεν υπάρχει επιστρέφει το c:

In [157…
```
a = {"a": 1, "b": 2, "c": 3}
```

In [158…
```
a.get("mitsos", 50)
```

Out[158…
```
50
```

In [159…
```
a.get("a", 50)
```

Out[159…
```
1
```

Iteration σε ένα dictionary

Έστω ένα list και ένα dictionary:

In [50]:
```
a = [1,2,3]
b = {"a":1, "b":2, "c":3}
```

Μπορούμε να κάνουμε iterate (επανάληψη) σε ένα list ως εξής:

In [51]:
```
for x in a:
    print (x)
```

```
1
2
3
```

Το ίδιο μπορούμε να κάνουμε και σε ένα dictionary:

In [52]:
```
for x in b:
    print (x, b[x])
```

```
a 1
b 2
c 3
```

Μπορούμε όμως να πάρουμε τα ζευγάρια κλειδιά-τιμές του dictionary ως μία λίστα χρησιμοποιώντας την `items()`

In [53]:
```
list(b.items())
```

Out[53]:
```
[('a', 1), ('b', 2), ('c', 3)]
```

Άρα όπως έχουμε δει και από πριν μπορούμε να κάνουμε iterate και να αναθέσουμε σε δύο μεταβλητές το κλειδί-τιμή κάθε μέλους του dictionary:

In [54]:
```
for x,y in b.items():
    print (x,y)
```

```
a 1
b 2
c 3
```

Παραδείγματα με dictionary

Μετράμε πόσες φορές υπάρχει το κάθε στοιχείο μίας λίστας:

In [145…
```
a = [3,2,3,2,4,5,4,3,6,5,7,9,1,2,8,9,9]
d = {}

for x in a:
    if not x in d:
        d[x] = 0
    d[x] += 1

print (d)
```

```
{3: 3, 2: 3, 4: 2, 5: 2, 6: 1, 7: 1, 9: 3, 1: 1, 8: 1}
```

Βρες το value που έχει το μεγαλύτερο key:

In [149…
```
a = {1:3, 5:2, 3:1} # Το μεγαλύτερο key είναι το 5 και το value του 5 είναι 2

max_key = max(a.keys())
print(a[max_key])
```

```
2
```

Βρες το key που έχει το μεγαλύτερο value

In [151…
```
a = {1:3, 5:2, 3:1} # Το μεγαλύτερο value είναι το 3 που έχει το key 1
max( (v,k)  for k,v in a.items())[1]
```

Out[151… 1

Πως βγήκε αυτό; Ας το "σπάσουμε" σε βήματα:

In [156…
```
b = list(a.items()) # Μετατρέπουμε σε λίστα
print (b)

c = [(v,k) for k,v in b] # Αντιμεταθέτουμε τα ζευγαράκια κλειδί/τιμή
print(c)

d = max(c) # Παίρνουμε το tuple το οποίο έχει το μεγαλύτερο value
print(d)

e = d[1] # Παίρνουμε το δευτερο στοιχείο το οποίο είναι το κλειδί.
print (e)
```

```
[(1, 3), (5, 2), (3, 1)]
[(3, 1), (2, 5), (1, 3)]
(3, 1)
1
```

Dictionary Comprehension

Σε προηγούμενη διάλεξη είχαμε πει τα list comprehensions

In [55]:
```
# List comprehension
[x for x in [1,2,3,4] if x>2]
```

Out[55]: [3, 4]

Είχαμε πει ότι το παραπάνω είναι ισοδύναμο με:

In [56]:
```
a = []
for x in [1,2,3,4]:
    if x>2:
        a.append(x)
print (a)
```

```
[3, 4]
```

Το ίδιο μπορούμε να κάνουμε και με τα dictionaries:

In [58]:
```
{ x:x*10 for x in range(1,10)}
```

Out[58]:
```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 7: 70, 8: 80, 9: 90}
```

Αυτό είναι ισοδύναμο με:

In [59]:
```
a={}
for x in range(1,10):
    a[x] = x*10
print (a)
```

```
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60, 7: 70, 8: 80, 9: 90}
```

Ένα άλλο παράδειγμα:

In [60]:
```
{ x:'hello {}'.format(x*10) for x in range(1,10)}
```

Out[60]:
```
{1: 'hello 10',
 2: 'hello 20',
 3: 'hello 30',
 4: 'hello 40',
 5: 'hello 50',
 6: 'hello 60',
 7: 'hello 70',
 8: 'hello 80',
 9: 'hello 90'}
```

Σύνολα

Η `set` είναι μία δομή δεδομένων που μοντελοποιεί ένα σύνολο. Κάθε στοιχείο σε ένα set μπορεί να υπάρχει **μόνο μία φορά**:

In [61]:
```
set([1,2,3])
```

Out[61]:
```
{1, 2, 3}
```

In [62]:
```
set([1,2,3,2])
```

Out[62]:
```
{1, 2, 3}
```

In [63]:
```
a = set(['a','b', 'a'])
a
```

Out[63]:
```
{'a', 'b'}
```

In [64]:
```
'b' in a
```

Out[64]:
```
True
```

In [65]:
```
set("Hello World!")
```

Out[65]:
```
{' ', '!', 'H', 'W', 'd', 'e', 'l', 'o', 'r'}
```

Η πράξη `&` μεταξύ δύο set μας επιστρέφει την τομή των συνόλων:

In [66]:
```
a = set([1,2,3,4])
b = set([3,4,5,6])
a & b
```

Out[66]:
```
{3, 4}
```

Το ίδιο μπορεί να γίνει και με τη συνάρτηση intersection:

In [67]:
```
a.intersection(b)
```

Out[67]:
```
{3, 4}
```

Η πράξη `|` μεταξύ δύο set μας επιστρέφει την ένωση των συνόλων:

In [68]:
```
a | b
```

Out[68]:
```
{1, 2, 3, 4, 5, 6}
```

Το ίδιο μπορεί να γίνει με τη συνάρτηση union:

In [69]:
```
a.union(b)
```

Out[69]:
```
{1, 2, 3, 4, 5, 6}
```

Η πράξη `-` μεταξύ δύο σετ `α` και `β` μας επιστρέφει τα στοιχεία της `α` που δεν υπάρχουν στην `β`:

In [70]:
```
a - b
```

Out[70]:
```
{1, 2}
```

In [71]:
```
b - a
```

Out[71]:
```
{5, 6}
```

In [75]:
```
(a - b) & (b-a)
```

Out[75]:
```
set()
```

Μπορούμε να προσθέσουμε ένα στοιχείο σε ένα set με τη συνάρτηση add:

In [81]:
```
a = {1,2,3}
a.add(10)
print (a)
```

```
{10, 1, 2, 3}
```

Ένας άλλος τρόπος να προσθέσουμε ένα στοιχείο είναι να χρησιμοποιήσουμε τον τελεστή `|` :

In [82]:
```
a = {1,2,3}
a = a | {10}
print (a)
```

```
{10, 1, 2, 3}
```

Δεν μπορούμε να προσθέσουμε μία λίστα σε ένα set. Αυτό γίνεται γιατί μπορούμε να προσθέσουμε μόνο στοιχεία που ΔΕΝ αλλάζουν:

In [79]:
```
a.add([7,8,9])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-79-1706898a2cfb> in <module>
----> 1 a.add([7,8,9])

TypeError: unhashable type: 'list'
```

Τα sets είναι ένας επιπλέον τύπος δεδομένων:

In [73]:
```
type(set([1,2,3]))
```

Out[73]:
```
set
```

In [74]:
```
a = set([1,2,3])
type(a) is set
```

Out[74]:
```
True
```

set comprehension

Όπως ακριβώς με τις λίστες και τα dictionaries, μπορούμε να έχουμε comprehensions και με τα sets:

In [83]:
```
{x%4 for x in range(10)}
```

Out[83]:
```
{0, 1, 2, 3}
```

In [ ]:
```

```