Αρχεία

Πριν ξεκινήσουμε για τα αρχεία μπορούμε να πούμε ότι το jupyter έχει έναν μηχανισμό για να σώζει απευθείας το περιεχόμενο ενός κελιού σε ένα αρχείο. Για να γίνει αυτό απλά γράφουμε `%%writefile NAME.TXT` στην αρχή του κελιού. Όπου NAME.TXT είναι το όνομα του αρχείου. Ας το δοκιμάσουμε:

In [2]:
```
%%writefile test.txt
aaa
bbb
ccc
```

```
Writing test.txt
```

Εκτελώντας το παραπάνω κελί δημιουργήθηκε το αρχείο `test.txt` . Για να δούμε τώρα πως μπορούμε να το ανοίξουμε και να το επεξεργαστούμε από τη python. Το αρχείο αυτό μπορώ να το "διαβάσω" αποθηκεύοντας όλο το περιεχόμενό του σε μία μεταβλητή με αυτόν τον τρόπο:

In [56]:
```
with open('test.txt') as f:
    d = f.read()
```

In [57]:
```
print (d)
```

```
aaa
bbb
ccc
```

Στο παραπάνω η f είναι μία νέα τύπου μεταβλητή η οποία αναπαριστάει ένα αρχείο. Οι εντολές που μπορούμε να κάνουμε μέσα σε ένα with open.. είναι:

- `f.read()` : Επιστρέφει ολόκληρο το αρχείο σε ένα string. Προσοχή! Ακατάλληλο για πολύ μεγάλα αρχεία!
- `f.readline()` : Επιστρέφει την επόμενη γραμμή του αρχείου.
- `f.readlines()` : Επιτρέφει μία λίστα με όλες τις γραμμές του αρχείου. (Για την ακρίβεια με όλες τις επόμενες γραμμές του αρχείου.

In [59]:
```
with open('test.txt') as f:
    d = f.readline()

print (d) # Η 1η γραμμή του αρχείου
```

```
aaa
```

In [60]:
```
with open('test.txt') as f:
    d = f.readlines()

print (d) # Μία λίστα με όλες τις γραμμές του αρχείου
```

```
['aaa\n', 'bbb\n', 'ccc\n']
```

Παρατηρούμε ότι κάθε γραμμή τελείωνει με new line (\n).

Η μεταβλητή που βάζουμε στη with (στο παράδειγμά μας η f) μπορεί να μπει μέσα σε for. Ή

αλλίως είναι "iterable". Σε αυτή τη περίπτωση, κάνουμε for σε όλες τις γραμμές του αρχείου:

In [61]:
```
with open('test.txt') as f:
    for line in f:
        print (line)
```

```
aaa

bbb

ccc
```

Η print τύπωσε και το new line στο τέλος της κάθε γραμμής. Για να το αφαιρέσουμε συνηθίζεται να κάνουμε:

In [62]:
```
with open('test.txt') as f:
    for line in f:
        print (line.strip('\n'))
```

```
aaa
bbb
ccc
```

Ο μηχανισμός `with` είναι και ο "επίσημος" τρόπος με τον οποίο η python προτείνει να ανοίγετε αρχεία. Υπάρχει όμως και άλλος ένας. Η άμεση χρήση της εντολής `open` :

In [3]:
```
f = open('test.txt')
```

Αφού το ανοίξω μπορώ να διαβάσω όλα τα περιεχόμενά του:

In [4]:
```
a= f.read()
print (a)
```

```
aaa
bbb
ccc
```

Αν επιχειρήσω να ξαναδιαβάσω το αρχείο τότε θα μου επιστρέψει ένα άδειο string:

In [5]:
```
f.read()
```

Out[5]:
```
''
```

Όταν διαβάζουμε ένα αρχείο με τη python και φτάνουμε στο τέλος, τότε η python δεν συνεχίζει να το διαβάζει από την αρχή. Αντίθετα επιτρέφει ένα άδειο string.

Για να ξαναδιαβάσουμε ένα αρχείο πρέπει να το κλείσουμε:

In [6]:
```
f.close()
```

Και να το ξανα-ανοίξουμε:

In [7]:
```
f = open('test.txt')
a= f.read()
print (a)
```

```
aaa
bbb
ccc
```

Μπορούμε να διαβάσουμε μία μόνο γραμμή από το αρχείο:

In [8]:
```
f = open('test.txt')
line = f.readline()
print (line)
```

```
aaa
```

Παρατηρείστε ότι η `line` περιέχει και το enter (`'\n'`) που υπάρχει στο τέλος της γραμμής:

In [9]:
```
line
```

Out[9]:
```
'aaa\n'
```

Αν καλέσουμε τη `readline` ξανά, τότε θα διαβάσει την επόμενη γραμμή:

In [10]:
```
f.readline()
```

Out[10]:
```
'bbb\n'
```

In [11]:
```
f.readline()
```

Out[11]:
```
'ccc\n'
```

Αν φτάσουμε στο τέλος τότε η `f.readline()` επιστρέφει το άδειο string:

In [12]:
```
f.readline()
```

Out[12]:
```
''
```

Επίσης μπορούμε να κάνουμε iterate (δηλαδή να εφαρμόσουμε τη `for`) σε ένα αρχείο που έχουμε ανοίξει:

In [13]:
```
f = open('test.txt')
for line in f:
    print (line)
```

```
aaa

bbb

ccc
```

In [15]:
```
f = open('test.txt')
for line in f:
    print (line.strip('\n'))
```

```
aaa
```

```
bbb
```

Ένας (ασυνήθιστος) τρόπος να κάνουμε iterate ένα αρχείο είναι να εκμεταλευτούμε το γεγονός ότι η f.readline() επιστρέφει το άδειο string (το οποίο κάνει evaluate ως False):

In [64]:
```
f = open('test.txt')
while True:
    line = f.readline()
    if not line:
        break
    print (line.strip('\n'))
f.close()
```

```
aaa
bbb
ccc
```

Η χρήση της `with` αντί για την open/close έχει όμως σημασία και στην αποτελεσματικότητα του προγράμματός μας. Όπως μας λέει και το επίσημο documentation αν ξεχάσουμε να καλέσουμε τη close() η python ΔΕΝ εγγυάται ότι θα εγγραφούν σωστά τα δεδομένα μας στον δίσκο!

Δημιουργία αρχείων

Μπορούμε να δημιουργήσουμε ένα καινούργιο αρχείο:

In [65]:
```
with open('results.txt', 'w') as f:
    f.write('Hello World\n')
```

In [66]:
```
!cat results.txt # For OSX/Linux
```

```
Hello World
```

In [ ]:
```
!type results.txt # For Windows
```

Προσέξτε το `'w'` .

Προσοχή!!! Αν το αρχείο `results.txt` υπάρχει ήδη τότε το διαγράφει!

Τι είναι όμως αυτό το `'w'` ; Όταν ανοίγουμε ένα αρχείο δηλώνουμε στη python για ποιο λόγο θέλουμε να το ανοίξουμε. Αυτό καλείται mode. Εδώ υπάρχει μία λίστα με όλα τα modes. Τα βασικά είναι:

- 'r' για διάβασμα. Αυτό είναι και το default mode. Όταν δεν βάζουμε τίποτα (π.χ. `f=open('test.txt')` τότε είναι το ίδιο σαν βάζουμε το 'r' (δλδ `f=open('test.txt', 'r')`)
- 'w' για γράψιμο
- 'a' για "append" δηλαδή για να προσθέσουμε σε ένα αρχείο
- 'x' για να γράψουμε σε ένα αρχείο αν και μόνο αν αυτό δεν υπάρχει. Για να το αποφύγετε λοιπόν τη διαγραφή ενός αρχείου με το 'w' μπορείτε να χρησιμοποιήτε αυτό το mode='x'.

Όπως και με το διάβασμα, έτσι και με το γράψιμο θα πρέπει εσείς να φροντίσετε να βάζετε enter (\n) στο τέλος του αρχείου:

In [67]:
```
# Δημιουργώ και γράφω στο αρχείο:
with open('results.txt', 'w') as f:
    f.write('hello')
    f.write('world')
```

In [69]:
```
# Διαβάζω από το αρχείο που μόλις έφτιαξα:
with open('results.txt') as f:
    data = f.read()
print (data)
```

```
helloworld
```

Παρατηρήστε ότι αν δεν βάλουμε enter τότε τα δεδομένα γράφονται στην ίδια γραμμή!

In [70]:
```
# Δημιουργώ και γράφω στο αρχείο:
with open('results.txt', 'w') as f:
    f.write('hello' + '\n')
    f.write('world' + '\n')
```

In [73]:
```
# Διαβάζω από το αρχείο που μόλις έφτιαξα:
with open('results.txt') as f:
    data = f.read()
print (data)
```

```
hello
world
```

Επίσης το όρισμα που βάζετε στη f.write πρέπει να είναι πάντα string:

In [74]:
```
with open('results.txt', 'w') as f:
    f.write(10)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-74-a433d4705467> in <module>
      1 with open('results.txt', 'w') as f:
----> 2     f.write(10)

TypeError: write() argument must be str, not int
```

Πολλές φορές θέλω να γράψω σε δύο ξεχωριστά αρχεία το περιεχόμενο κάποιας επεξεργασίας. Για παράδειγμα έστω η λίστα: `a=[1,2,3,4,5]` . Βάλε στο αρχείο results_1.txt το γινόμενο επί δύο κάθε στοιχείου και στο αρχείο results_2.txt το μισό κάθε στοιχείου. Ένας τρόπος είναι απλά να ανοίξουμε δύο αρχεία το ένα μετά το άλλο:

In [76]:
```
a=[1,2,3,4,5]
with open('results_1.txt', 'w') as f:
    for i in a:
        f.write(str(i*2)+'\n')

with open('results_2.txt', 'w') as f:
    for i in a:
        f.write(str(i/2) + '\n')
```

Ένας πιο όμορφος τρόπος όμως είναι:

In [77]:
```
a=[1,2,3,4,5]
with open('results_1.txt', 'w') as f1, open('results_2.txt', 'w') as f2:
    for i in a:
        f1.write(str(i*2)+'\n')
        f2.write(str(i/2) + '\n')
```

Οι μηχανισμοί που έχουμε μάθει μέχρι στιγμής για τον χειρισμό των strings μπορούν να συνδυαστούν άψογα με το διάβασμα και το γράψιμο αρχείων. Ας δούμε μερικά παραδείγματα:

Παράδειγμα 1

Σε αυτό το link υπάρχει ένας σύνδεσμος για το αρχείο mim2gene.txt το οποίο είναι "a tab-delimited file linking MIM numbers with NCBI Gene IDs, Ensembl Gene IDs, and HGNC Approved Gene Symbols. ". Τα περισσότερα αρχεία στη βιολογία έχουν αυτό περίπου το format το οποίο αξίζει να το εξερευνήσουμε! Αφού κατεβάσετε το αρχείο μπορούμε για αρχή να τυπώσουμε τις πρώτες 10 γραμμές του:

In [78]:
```
with open('mim2gene.txt') as f:
    for i, line in enumerate(f):
        print (line.strip('\n'))
        if i >= 10:
            break
```

```
# Copyright (c) 1966-2021 Johns Hopkins University. Use of this file adheres t
o the terms specified at https://omim.org/help/agreement.
# Generated: 2021-03-17
# This file provides links between the genes in OMIM and other gene identifier
s.
# THIS IS NOT A TABLE OF GENE-PHENOTYPE RELATIONSHIPS.
# MIM Number    MIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)
        Entrez Gene ID (NCBI)   Approved Gene Symbol (HGNC)     Ensembl Gene I
D (Ensembl)
100050  predominantly phenotypes
100070  phenotype       100329167
100100  phenotype
100200  predominantly phenotypes
100300  phenotype
100500  moved/removed
```

Παρατηρούμε ότι κάποιες γραμμές ξεκινάνε με "#" είναι συνιθισμένο με αυτόν τον τρόπο να δηλώνουμε ότι αυτή η γραμμή περιέχει κάποιο σχολιασμό και όχι κάποια δεδομένα. Παρόλα αυτά η 5η γραμμή περιέχει τα ονόματα κάθε στήλης. Ας τα εξάγουμε όλα αυτά!

In [80]:
```
with open('mim2gene.txt') as f:
    for i, line in enumerate(f):

        if i<4:
            continue

        # this is the header!
        header = line
        break

header
```

Out[80]:
```
'# MIM Number\tMIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)\tEntr
ez Gene ID (NCBI)\tApproved Gene Symbol (HGNC)\tEnsembl Gene ID (Ensembl)\n'
```

Τι είναι αυτά τα \t τα οποία υπάρχουν στο string; Αυτα είναι tabs. Όπως το \n είναι ένας ειδικός χαρακτήρας για να δηλώσει μία νέα γραμμή, έτσι το \t δηλώνει ένα tab. Είναι πολύ συνιθισμένο να χρησιμοποιούμε αυτόν τον χαρακτήρα για να διαχωρίζουμε τις στήλες μεταξύ τους σε αρχεία που περιέχουν πολλές στήλες.

Για αρχή μπορούμε να "πετάξουμε" το "#" στην αρχή της επικεφαλίδας:

In [81]:
```
header = header[1:]
header
```

Out[81]:
```
' MIM Number\tMIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)\tEntre
z Gene ID (NCBI)\tApproved Gene Symbol (HGNC)\tEnsembl Gene ID (Ensembl)\n'
```

Τώρα μπορούμε να κάνουμε split με βάση το tab:

In [82]:
```
header = header.split('\t')
header
```

Out[82]:
```
[' MIM Number',
 'MIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)',
 'Entrez Gene ID (NCBI)',
 'Approved Gene Symbol (HGNC)',
 'Ensembl Gene ID (Ensembl)\n']
```

Χμμ φαίνεται ότι πρώτο και το τελευταίο στοιχείο έχουν έξτρα χαρακτήρες στην αρχή και στο τέλος. Ας τους αφαιρέσσουμε.

In [83]:
```
header = [x.strip() for x in header]
header
```

Out[83]:
```
['MIM Number',
 'MIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)',
 'Entrez Gene ID (NCBI)',
 'Approved Gene Symbol (HGNC)',
 'Ensembl Gene ID (Ensembl)']
```

Τέλεια, τώρα έχουμε μία λίστα με την επικεφαλίδα του αρχείου. Ας δούμε και τις υπόλοιπες τιμές:

In [87]:
```
header = None
data = []

with open('mim2gene.txt') as f:
    for i, line in enumerate(f):

        if i<4: # περιττά σχόλια
            continue

        if not header:
            # this is the header!
            header = line

            # Εξηγήσαμε τι κάνουν αυτά
            header = header[1:]
            header = header.split('\t')
            header = [x.strip() for x in header]
            continue

        # Here are the rest of the data
        data.append(line.strip('\n').split('\t'))
```

Ωραία ας δω πως είναι τα 10 πρώτα στοιχεία του αρχείου:

In [88]:
```
data[:10]
```

Out[88]:
```
[['100050', 'predominantly phenotypes', '', '', ''],
 ['100070', 'phenotype', '100329167', '', ''],
 ['100100', 'phenotype', '', '', ''],
 ['100200', 'predominantly phenotypes', '', '', ''],
 ['100300', 'phenotype', '', '', ''],
 ['100500', 'moved/removed', '', '', ''],
 ['100600', 'phenotype', '', '', ''],
 ['100640', 'gene', '216', 'ALDH1A1', 'ENSG00000165092'],
 ['100650', 'gene/phenotype', '217', 'ALDH2', 'ENSG00000111275'],
 ['100660', 'gene', '218', 'ALDH3A1', 'ENSG00000108602']]
```

Ας πάρω μόνο αυτά που το 5ο στοιχείο δεν είναι άδειο:

In [89]:
```
data = [x for x in data if x[4]]
```

Μπορούμε κάθε στοιχείο της λίστα να το κάνουμε dictionary χρησιμοποιώντας το header σαν κλειδιά!

In [91]:
```
data = [dict(zip(header,x)) for x in data]
```

In [92]:
```
data[100]
```

Out[92]:
```
{'MIM Number': '104615',
 'MIM Entry Type (see FAQ 1.3 at https://omim.org/help/faq)': 'gene',
 'Entrez Gene ID (NCBI)': '6541',
 'Approved Gene Symbol (HGNC)': 'SLC7A1',
 'Ensembl Gene ID (Ensembl)': 'ENSG00000139514'}
```

Τέλος ας αποθηκεύσουμε σε ένα αρχείο τις στήλες Ensembl Gene ID (Ensembl) και MIM Number:

In [93]:
```
with open('results.txt', 'w') as f:
    # Save header:
    f.write('MIM Number\tEnsembl Gene ID (Ensembl)' + '\n')

    # Save data
    for x in data:
        s = '\t'.join([x['MIM Number'], x['Ensembl Gene ID (Ensembl)']])
        f.write(s + '\n')
```

In [95]:
```
!head results.txt # τυπώνει τις πρώτες 10 γραμμές σε περβάλλον OSX/Linux
```

```
MIM Number      Ensembl Gene ID (Ensembl)
100640  ENSG00000165092
100650  ENSG00000111275
100660  ENSG00000108602
100670  ENSG00000137124
100678  ENSG00000120437
100690  ENSG00000138435
100710  ENSG00000170175
100720  ENSG00000135902
100725  ENSG00000108556
```

In [55]:
```
f=open('results.txt', 'x')
```

```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-55-0b070d8efefb> in <module>
----> 1 f=open('results.txt', 'x')

FileExistsError: [Errno 17] File exists: 'results.txt'
```

Παράδειγμα 2

Σε αυτό το link υπάρχει ένα αρχείο με το πολύ γνωστό IRIS dataset. Κατεβάστε το και σώστε το στο ίδιο directory με τη python. **ΠΡΟΣΟΧΗ!** Αυτό το αρχείο δεν πρέπει να είναι το ίδιο με το αυθεντικό γιατί περιέχει μόνο 149 δεδομένα (αντί για 150) που έχει το αυθεντικό. Για τους σκοπούς του παραδείγματος αυτού όμως αυτό δεν μας επηρεάζει.

Το όνομα του αρχείο είναι: `Wq2PpbZy.txt` .

Ας παίξουμε μαζί του!

Αρχικά το ανοίγουμε:

In [21]:
```
with open('Wq2PpbZy.txt') as f:
    data = f.read()
print(data[:300]) #  τυπώνω μόνο τους 300 πρώτους χαρακτήρες για λόγους μεγέθ
```

```
Sepal length    Sepal width     Petal length    Petal width     Species
5.2     3.5     1.4     0.2     I. setosa
4.9     3.0     1.4     0.2     I. setosa
4.7     3.2     1.3     0.2     I. setosa
4.6     3.1     1.5     0.2     I. setosa
5.0     3.6     1.4     0.3     I. setosa
5.4     3.9     1.7     0.4     I. setosa
4.6     3.4     1.4     0.3     I. setosa
5.0     3.4     1.5     0.2     I. setos
```

Μπορούμε να πάρουμε μία λίστα με όλες τις γραμμές:

In [22]:
```
data = data.split('\n')
data[:5]
```

Out[22]:
```
['Sepal length \tSepal width \tPetal length \tPetal width \tSpecies',
 '5.2 \t3.5 \t1.4 \t0.2 \tI. setosa',
 '4.9 \t3.0 \t1.4 \t0.2 \tI. setosa',
 '4.7 \t3.2 \t1.3 \t0.2 \tI. setosa',
 '4.6 \t3.1 \t1.5 \t0.2 \tI. setosa']
```

Φαίνεται ότι το αρχείο διαχωρίζει τις κολόνες με tabs. Ας το κάνουμε split με βάση τα tabs:

In [23]:
```
data = [x.split('\t') for x in data]
data[:5]
```

Out[23]:
```
[['Sepal length ', 'Sepal width ', 'Petal length ', 'Petal width ', 'Species
'],
 ['5.2 ', '3.5 ', '1.4 ', '0.2 ', 'I. setosa'],
 ['4.9 ', '3.0 ', '1.4 ', '0.2 ', 'I. setosa'],
 ['4.7 ', '3.2 ', '1.3 ', '0.2 ', 'I. setosa'],
 ['4.6 ', '3.1 ', '1.5 ', '0.2 ', 'I. setosa']]
```

Παρατηρούμε ότι υπάρχουν spaces (κενά) στο τέλος κάποιων strings. Ας τα "περάσουμε" όλα από ένα `strip` :

In [24]:
```
data = [[y.strip() for y in x] for x in data]
data[:5]
```

Out[24]:
```
[['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'Species'],
 ['5.2', '3.5', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.0', '1.4', '0.2', 'I. setosa'],
 ['4.7', '3.2', '1.3', '0.2', 'I. setosa'],
 ['4.6', '3.1', '1.5', '0.2', 'I. setosa']]
```

Φαίνεται ότι το πρώτο στοιχείο της λίστας είναι το header. Ας το πάρουμε:

In [25]:
```
header = data[0]
header
```

Out[25]:
```
['Sepal length', 'Sepal width', 'Petal length', 'Petal width', 'Species']
```

Επίσης φαίνεται ότι όλα τα υπόλοιπα στοιχεία (εκτός το πρώτο) είναι τα δεδομένα. Ας πάρουμε μόνο αυτά:

In [26]:
```
data = data[1:]
data[:5]
```

Out[26]:
```
[['5.2', '3.5', '1.4', '0.2', 'I. setosa'],
 ['4.9', '3.0', '1.4', '0.2', 'I. setosa'],
 ['4.7', '3.2', '1.3', '0.2', 'I. setosa'],
 ['4.6', '3.1', '1.5', '0.2', 'I. setosa'],
 ['5.0', '3.6', '1.4', '0.3', 'I. setosa']]
```

Παρατηρώ ότι οι τιμές στις 4 πρώτες στήλες είναι σε string. Τις μετατρέπουμε σε float:

In [27]:
```
data = [list(map(float, x[:4])) + [x[-1]]  for x in data]
data[:5]
```

Out[27]:
```
[[5.2, 3.5, 1.4, 0.2, 'I. setosa'],
 [4.9, 3.0, 1.4, 0.2, 'I. setosa'],
 [4.7, 3.2, 1.3, 0.2, 'I. setosa'],
 [4.6, 3.1, 1.5, 0.2, 'I. setosa'],
 [5.0, 3.6, 1.4, 0.3, 'I. setosa']]
```

Τώρα μπορώ να κάνω μια μικρή "περιήγηση" σε αυτό. Για παράδειγμα, Ποια συνολικά species υπάρχουν;

In [28]:
```
species = set([x[-1] for x in data])
species
```

Out[28]:
```
{'I. setosa', 'I. versicolor', 'I. virginica'}
```

Πόσα data έχουμε για κάθε species;

In [29]:
```
b = [(x, sum([y[-1]==x for y in data])) for x in species]
b
```

Out[29]:
```
[('I. versicolor', 50), ('I. virginica', 49), ('I. setosa', 50)]
```

Αυτό μπορούμε να το μετατρέψουμε και σε dictionary

In [31]:
```
dict(b)
```

Out[31]:
```
{'I. versicolor': 50, 'I. virginica': 49, 'I. setosa': 50}
```

Ποιος είναι ο μέσος όρος του "Sepal Length" για όλα τα data;

In [33]:
```
def average(x):
    return sum(x)/len(x)

average([x[header.index('Sepal length')] for x in data])
```

Out[33]:
```
5.831543624161076
```

Ποιος είναι ο μέσος όρος του "Sepal length" για κάθε species ξεχωριστά;

In [34]:
```
[(x, average([y[0] for y in data if y[-1] == x])) for x in species]
```

Out[34]:
```
[('I. versicolor', 5.936),
 ('I. virginica', 6.565306122448979),
 ('I. setosa', 5.007999999999999)]
```

Πόσα "I. setosa" έχουν Petal length > 1.5 ;

In [35]:
```
sum([x[2] > 1.5 and x[-1] == 'I. setosa' for x in data])
```

Out[35]:
```
13
```

Ποια είναι αυτά;

In [36]:
```
[x for x in data if x[2] > 1.5 and x[-1] == 'I. setosa']
```

Out[36]:
```
[[5.4, 3.9, 1.7, 0.4, 'I. setosa'],
 [4.8, 3.4, 1.6, 0.2, 'I. setosa'],
 [5.7, 3.8, 1.7, 0.3, 'I. setosa'],
 [5.4, 3.4, 1.7, 0.2, 'I. setosa'],
 [5.1, 3.3, 1.7, 0.5, 'I. setosa'],
 [4.8, 3.4, 1.9, 0.2, 'I. setosa'],
 [5.0, 3.0, 1.6, 0.2, 'I. setosa'],
 [5.0, 3.4, 1.6, 0.4, 'I. setosa'],
 [4.7, 3.2, 1.6, 0.2, 'I. setosa'],
 [4.8, 3.1, 1.6, 0.2, 'I. setosa'],
 [5.0, 3.5, 1.6, 0.6, 'I. setosa'],
 [5.1, 3.8, 1.9, 0.4, 'I. setosa'],
```

```
 [5.1, 3.8, 1.6, 0.2, 'I. setosa']]
```

Ποια είναι τα indexes των παραπάνω (I. setosa που έχουν Petal length > 1.5)

In [37]:
```
[i for i,x in enumerate(data) if x[2]>1.5 and x[-1] == 'I. setosa']
```

Out[37]:
```
[5, 11, 18, 20, 23, 24, 25, 26, 29, 30, 43, 44, 46]
```

Ποιο είναι το minimum Sepal length για όλα;

In [38]:
```
min(data, key=lambda x:x[header.index('Sepal length')]) # θα δούμε τη lambda
```

Out[38]:
```
[4.3, 3.0, 1.1, 0.1, 'I. setosa']
```

Που είναι το mimimum Sepal Length για όλα τα είδη;

In [39]:
```
[min([(y[0], y) for y in data if y[-1] == x]) for x in species]
```

Out[39]:
```
[(4.9, [4.9, 2.4, 3.3, 1.0, 'I. versicolor']),
 (4.9, [4.9, 2.5, 4.5, 1.7, 'I. virginica']),
 (4.3, [4.3, 3.0, 1.1, 0.1, 'I. setosa'])]
```

Ποιο είναι το index των παραπάνω;

In [40]:
```
[min([(y[0], y, i) for i,y in enumerate(data) if y[-1] == x]) for x in species
```

Out[40]:
```
[(4.9, [4.9, 2.4, 3.3, 1.0, 'I. versicolor'], 57),
 (4.9, [4.9, 2.5, 4.5, 1.7, 'I. virginica'], 106),
 (4.3, [4.3, 3.0, 1.1, 0.1, 'I. setosa'], 13)]
```

Ποιο είναι το εύρος (δηλαδή το μικρότερο και το μεγαλύτερο) του "Sepal Length" για κάθε είδος ξεχωριστά;

In [41]:
```
# Επιστρέφει το εύρος μίας λίστας
def my_range(l):
    return min(l), max(l)

my_range([4,5,6,3,4,7,8,9])
```

Out[41]:
```
(3, 9)
```

In [42]:
```
[ (my_range([y[0] for y in data if y[-1] ==x]), x) for x in species]
```

Out[42]:
```
[((4.9, 7.0), 'I. versicolor'),
 ((4.9, 7.9), 'I. virginica'),
 ((4.3, 5.8), 'I. setosa')]
```

Ας βάλουμε και τα indexes των μικρότερων και μεγαλύτερων:

In [43]:
```
# Επιστρέφει το index του μεγαλύτερου στοιχείου
def max_index(l):
    return max(range(len(l)), key=lambda x : l[x])

max_index([3,6,5,8,7])
```

Out[43]:
```
3
```

In [44]:
```
# Επιστρέφει το index του μικρότερου στοιχείου
def min_index(l):
    return min(range(len(l)), key=lambda x : l[x])

min_index([3,6,5,8,7])
```

Out[44]: 0

In [45]:
```
def my_range_2(l):
    return 'min:{} min_index:{} max:{} max_index: {}'.format(
        min(l), min_index(l), max(l), max_index(l)
    )
```

In [46]:
```
[ (my_range_2([y[0] for y in data if y[-1] ==x]), x) for x in species]
```

Out[46]:
```
[('min:4.9 min_index:7 max:7.0 max_index: 0', 'I. versicolor'),
 ('min:4.9 min_index:6 max:7.9 max_index: 31', 'I. virginica'),
 ('min:4.3 min_index:13 max:5.8 max_index: 14', 'I. setosa')]
```

Ποιο dataset έχει το μεγαλύτερο εμβαδό sepal. Ως εμβαδό ορίζουμε το γινόμε Sepal width*Sepal length

In [47]:
```
max([(x[0]*x[1], x, i) for i,x in enumerate(data)])
```

Out[47]:
```
(30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131)
```

Ποιο dataset έχει το μεγαλύτερο εμβαδό για κάθε είδος;

Φτιάχνουμε μία συνάρτηση η οποία παίρνει μία λίστα από data και υπολογίζει ποιο από αυτα έχει το μεγαλύτερο εμβαδό:

In [48]:
```
f = lambda d : max([(x[0]*x[1], x, data.index(x)) for x in d])
```

π.χ: από όλα τα data, το μεγαλύτερο εμβαδό το έχει:

In [49]:
```
f(data)
```

Out[49]:
```
(30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131)
```

Το εφαρμόζουμε για κάθε είδος ξεχωριστά:

In [50]:
```
[f([y for y in data if y[-1] == x]) for x in species]
```

Out[50]:
```
[(22.400000000000002, [7.0, 3.2, 4.7, 1.4, 'I. versicolor'], 50),
 (30.02, [7.9, 3.8, 6.4, 2.0, 'I. virginica'], 131),
 (25.080000000000002, [5.7, 4.4, 1.5, 0.4, 'I. setosa'], 15)]
```

String formatting

Το να "μπλέκουμε" μεταβλητές μέσα σε strings γρήγορα και εύκολα είναι ένα από τα πιο κοινά πράγματα που κάνουμε και ένας από τους βασικούς λόγους δημιουργίας της python. Ας υποθέσουμε λοιπόν ότι έχουμε τις μεταβλητές:

In [96]:
```
name = "Mitsos"
age = 40
```

Και θέλουμε να τις βάλουμε στο string: `My name is .... and I am ... years old`.
Έχουμε τις εξής επιλογές:

1. string concatenation

Από τις πιο βασικές επιλογές. Δεν μας δίνει πολλές δυνατότητες αλλά είναι πολύ απλή:

In [97]:
```
print ('My name is ' + name + ' and I am ' + str(age) + ' years old')
```

```
My name is Mitsos and I am 40 years old
```

2. Use the format command:

In [98]:
```
print ('My name is {} and I am {} years old'.format(name, age))
```

```
My name is Mitsos and I am 40 years old
```

3. Use the format command with placeholders:

In [102…
```
print ('My name is {NAME} and I am {AGE} years old'.format(NAME=name, =age))
```

```
My name is Mitsos and I am 40 years old
```

Αυτό μας επιτρέπει να χρησιμοποιήσουμε το ίδιο placeholder πολλές φορές

In [105…
```
a = 'James'
b = 'Bond'

print ('My name is {SURNAME}. {NAME} {SURNAME}.'.format(NAME=a, SURNAME=b))
```

```
My name is Bond. James Bond.
```

4. Use the format command for "clever" formatting

Η format έχει ένα υπο-σετ εντολών για να τυπώνετε αριθμούς σε με συγκεκριμένη ακρίβεια π.χ. 3 δεκαδικά ψηφία:

In [107…
```
a=234/1345
print (a)
print ('the result is {0:.3f} '.format(a))
```

```
0.17397769516728626
the result is 0.174
```

Η format υποστηρίζει στην ουσία μία [mini-language]https://docs.python.org/3/library/string.html#formatspec γλώσσα μέσα στη γλώσσα για να φορμάρετε strings πιο έξυπνα. Για παράδειγμα: δεξιά στοίχιση με μέγιστο 30 χαρακτήρες:

In [114…
```
a = 'Hello'
b = 'Heraklion'

print ('{0:>30}'.format(a))
print ('{0:>30}'.format(b))
```

```
                         Hello
                     Heraklion
```

5. Ο τελεστής %

Αντί για τη format μπορούμε να χρησιμοποιήσουμε τον τελεστή `%` . Αυτό δεν συνηθίζεται

In [118…
```
print ('My name is %s and I am %s years old' % (name, age))
```
```
My name is Mitsos and I am 40 years old
```

6. f strings

Τα f strings είναι η πιο σύγχρονη μέθοδος για να φορμάρετε strings μαζί με μεταβλητές. Τα f strings έχουν τη διαφορά με τις υπόλοιπες μεθόδους ότι η αντικατάσταση γίνεται κατά τη διάρκεια δημιουργίας του string. Δηλαδή όλες οι άλλες μέθοδοι (εκτός από τη 1η) πρώτα φτιάχνουν το string χωρίς τις μεταβλητές στη μνήμη και μετά αντικαθιστούν τις μεταβλητές όπου χρειάζονται. Σαν συνέπεια είναι εξαιρετικά γρήγορα. Ένα f-string δηλώνεται βάζοντας το f πριν τα αυτάκια. Μέσα στο string μπορείτε να βάλετε απευθείας τις μεταβλητές σας:

In [120…
```
print (f'My name is {name} and I am {age} years old')
```
```
My name is Mitsos and I am 40 years old
```

pass

Σε οποιοδίποτε καινούργιο indentation (for, while, def, if, ...) πρέπει υποχρεωτικά να έχουμε τουλάχιστον μία εντολή:

In [121…
```
if True:
```
```
  File "<ipython-input-121-2c8a33b52dfb>", line 1
    if True:
            ^
SyntaxError: unexpected EOF while parsing
```

Αν για κάποιο λόγο δεν θέλουμε να βάλουμε καμία εντολή, τότε μπορούμε να χρησιμοποιήσουμε τη `pass` :

In [122…
```
if True:
    pass
```

In [123…
```
def f():
    pass
```

Το `pass` μπορούμε να το βάλουμε οπουδίποτε και δεν κάνει.. τίποτα:

In [124…
```
print ('Hello')
pass
print ('World')
```
```
Hello
World
```

Ο τελεστής is

Έχουμε δει πολλές φορές τον τελεστή `is` . Το έχουμε χρησιμοποιήσει για να δούμε τι τύπος είναι μία μεταβλητή:

In [125…
```
a = [1,2,3]
type(a) is list
```

Out[125…
```
True
```

In [126…
```
type(a) is str
```

Out[126…
```
False
```

Η is κοιτάει αν δύο μεταβλητές είναι ίδιες. Ή αλλιώς κοιτάει αν αναφέρονται στην ίδια θέση μνήμης. Αυτό είναι διαφορετικό από το ίσες: Δύο μεταβλητές είναι *ίσες* (==) αν έχουν την ίδια τιμή. Δύο μεταβλητές είναι *ίδιες* αν αναφέρονται στην ίδια θέση μνήμης. Παραδείγματα:

In [127…
```
a = [1,2,3]
b = [1,2,3]
print (a==b)
print (a is b)
```

```
True
False
```

In [128…
```
a = [1,2,3]
b = a
print (a==b)
print (a is b)
```

```
True
True
```

Όταν δύο μηταβλητές είναι ίδιες (και όχι απλά ίσες) τότε αν αλλάξεις την μία, αλλάζει και η άλλη:

In [129…
```
a = [1,2,3]
b=a
b[0] = 100
print (a)
```

```
[100, 2, 3]
```

Ένα άλλο παράδειγμα:

In [130…
```
a = [1,2,3,4]
b = [a,a,a]
print (b)
a[0]=100
print (b)
```

```
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
[[100, 2, 3, 4], [100, 2, 3, 4], [100, 2, 3, 4]]
```

Παρατηρούμε ότι αλλάξαμε το πρώτο στοιχείο μόνο της πρώτης υπολίστας (`a[0]=100`) και παρόλα αυτά αλλάξαν όλες! Αυτό έγινε γιατί όλες οι τιμές της λίστας είναι η ίδια μεταβλητή `b = [a,a,a]` !

Αυτό δημιουργεί το εξής περιέργο φαινόμενο:

In [143…
```
a = [[]] * 3 # Η εσωτερική λίστα ([]) είναι ίδια και για τα 3!
print (a)
```

```
[[], [], []]
```

In [144…
```
a[0].append('mitsos')
print (a)
```

```
[['mitsos'], ['mitsos'], ['mitsos']]
```

Το παραπάνω δεν ισχύει για τα primitive types δηλαδή για τα int, str, float, bool, complex:

In [134…
```
a = 1000
b = 1000
a is b
```

Out[134…
```
False
```

In [135…
```
a = 1000
b = 1000
a=b
a is b
```

Out[135…
```
True
```

Παρόλο που είναι ίδιες, αν αλλάξω τη b ΔΕΝ αλλάζει η a:

In [136…
```
b = b + 1
a is b
```

Out[136…
```
False
```

Δηλαδή κάθε φορά που αλλάζω τη τιμή μιας μεταβλητής αλλάζει και η θέση μνήμης της. Για να δούμε ποιος είναι ο κωδικός της θέσης μνήμης μίας μεταβλητής χρησιμοποιούμε την `id` :

In [140…
```
a = 1
id(a)
```

Out[140…
```
4398221664
```

In [141…
```
a = a + 1
id(a) # Η θέση άλλαξε!
```

Out[141…
```
4398221696
```

In [142…
```
a = [1,2,3]
b = a
print (id(a))
print (id(b)) # Ίδια θέση μνήμης!
```

```
140652702590144
140652702590144
```

Ternary operator

με το ternary operator μπορούμε να γράψουμε μία `if ... else ...` σε μία εντολή. Η δομή είναι:

```
a = EXPRESSION_IF_THE_CONDITION_IS_TRUE if CONDITION else
```

```
EXPRESSION_IF_THE_CONDITION_IS_FALSE
```

π.χ.:

In [147…
```
age = 40
if age>=18:
    status = 'adult'
else:
    status = 'not adult'
print (status)
```

```
adult
```

Το ίδιο με ternary operator:

In [148…
```
status = "adult" if age>=18 else "not adult"
print (status)
```

```
adult
```

In [145…
```
a = 1 if 5<3 else 8
print (a)
```

```
8
```

In [146…
```
a = 1 if 3<5 else 8
print (a)
```

```
1
```

Μπορούμε να συνθέσουμε πολλούς ternary operators μαζί:

In [149…
```
a = (1 if 3<5 else 6) if 5>6 else (6 if 4>1 else 7)
print (a)
```

```
6
```

Αν βγάλω τις παρενθέσεις θα έχει διαφορετική τιμή η `a` . (Γιατί;)

In [150…
```
a = 1 if 3<5 else 6 if 5>6 else 6 if 4>1 else 7
print (a)
```

```
1
```

lambda functions

Οι lambda functions είναι ειδικές συναρτήσεις που έχουν τις παρακάτω ιδιότητες:

- Δεν έχουν όνομα.
- Περιέχουν μόνο αυτό που κάνουν return. Δηλαδή δεν μπορούν να έχουν πάνω από μία γραμμές.

In [152…
```
f = lambda x : x/2
f(10)
```

Out[152…
```
5.0
```

Το παραπάνω είναι ισοδύναμο με:

In [154…
```
def f(x):
    return x/2
f(10)
```

Out[154…
```
5.0
```

Προσέχτε ότι μπορούμε να παραλείψουμε τελείως το γράμμα `f` που είναι το όνομα της συνάρτησης:

In [155…
```
(lambda x : x/2)(10)
```

Out[155…
```
5.0
```

Οι lambda συναρτήσεις έχουν ακριβώς τον ίδιο τύπο με τις "κανονικές" συναρτήσεις:

In [156…
```
type(lambda x:x)
```

Out[156…
```
function
```

Γιατί είναι χρήσιμες οι συναρτήσεις lambda; Πολλές φορές χρειάζεται να χρησιμοποιήσουμε μία συνάρτηση που κάνει κάτι απλό ή πρόκειται να τη χρησιμοποιήσουμε μία φορά, οπότε δεν υπάρχει λόγος να της δώσουμε όνομα ως μεταβλητή. Αυτό συμβαίνει συχνά όταν θέλουμε να δώσουμε μία συνάρτηση σαν όρισμα σε μία άλλη συνάρτηση, ή σε συναρτήσεις που επιστρέφουν συναρτήσεις ή σε συναρτήσεις που είναι μέσα σε λίστες και dictionaries.

Π.χ:

In [157…
```
a = {
    'accurate': lambda x:x/3, # Δεκαδική διαίρεση με 3
    'not_accurate': lambda x:x//3 # Ακέραια διαίρεση με 3
}
print(a['accurate'](55))
print (a['not_accurate'](55))
```

```
18.333333333333332
18
```

Εκεί όμως που κλασσικά χρησιμοποιούνται οι lambda συναρτήσεις είναι στις συναρτήσεις map, filter, min, max, sorted. Για παράδειγμα έστω η λίστα:

In [158…
```
a = [1,2,3,4,5,6,7,8,9,10]
```

Πολλαπλασίασε όλα τα στοιχεία με το 2:

In [159…
```
def f(x):
    return x*2
print (list(map(f, a)))
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

In [160…
```
print (list(map(lambda x:x*2, a)))
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Πάρε μόνο τους μονούς αριθμούς:

In [161…
```
def f(x):
    return x%2==1
print (list(filter(f, a)))
```
```
[1, 3, 5, 7, 9]
```

In [162…
```
print (list(filter(lambda x:x%2==1, a)))
```
```
[1, 3, 5, 7, 9]
```

Έστω η λίστα:

In [166…
```
cities = ['edessa', 'bethleem', 'Heraklion', ]
```

Ταξινόμησε τες με βάση το πλήθος από "e":

In [167…
```
def f(x):
    return x.count('e')
print (sorted(cities, key=f))
```
```
['Heraklion', 'edessa', 'bethleem']
```

In [168…
```
print (sorted(cities, key=lambda x: x.count('e')))
```
```
['Heraklion', 'edessa', 'bethleem']
```

Έστω η λίστα:

In [170…
```
cities = [
    ('Heraklion', 200_000),
    ('Athens', 1_000_000),
    ('Thessaloniki', 500_000),
]
```

Ποια είναι η πόλη με τους περισσότερους κατοίκους;

In [171…
```
def f(x):
    return x[1]

max(cities, key=f)
```

Out[171…
```
('Athens', 1000000)
```

In [172…
```
max(cities, key=lambda x:x[1])
```

Out[172…
```
('Athens', 1000000)
```

Variable scoping ή Variable visibility

Με αυτόν τον περίεργο όρο αναφερόμαστε στην "εμβέλεια" που έχουν οι μεταβλητές σε μία γλώσσα προγραμματισμού. 'Η αλλιώς από ποιες συναρτήσεις και δομές μπορούμε να προσπελάσουμε μία μεταβλητή. Περισσότερα: https://en.wikipedia.org/wiki/Scope_%28computer_science%29

Ξεκινάμε λέγοντας ότι μία μεταβλητή που έχει οριστεί έξω από τη συνάρτηση μπορεί να προσπελάστεί μέσα στη συνάρτηση (αυτό είναι κάτι που γενικότερα το αποφεύγουμε).

In [173…
```
a = 3

def f():
    print (a)

f()
```

```
3
```

Παρόλα αυτά αν μία μεταβλητή **οριστεί** μέσα σε μία συνάρτηση τότε "ανεξαρτητοποιείται". Δηλαδή ορίζεται για τη συνάρτηση και τη συνάρτηση μόνο:

In [174…
```
a = 3
def f():
    a=5

f()
print (a)
```

```
3
```

Παρατηρούμε ότι παρόλο που θέσαμε με 5 την a μέσα στη συνάρτηση, η a έξω από τη συνάρτηση δεν άλλαξε. Αυτό έγινε επειδή οι δύο μεταβλητές παρόλο που έχουν το ίδιο όνομα, είναι διαφορετικές.

Μία απορία είναι: πως γίνεται όταν κάνω print(a) στο 1ο παράδειγμα να αναφέρομαι στην "έξω" a και όταν κάνω a=5 να αναφέρομαι στη "μέσα" a; Αυτό έχει προκαλέσει αρκετή σύγχηση αλλά η python έχει την εξής αρχή: αν θέσεις μία νέα τιμή σε μία μεταβλητή οπουδήποτε μέσα σε μία συνάρτηση τότε την ξαναορίζεις. Για παράδειγμα:

In [176…
```
a=5
def f():
    print (a)
    a=5

f()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-176-835a5d405592> in <module>
      4     a=5
      5
----> 6 f()

<ipython-input-176-835a5d405592> in f()
      1 a=5
      2 def f():
----> 3     print (a)
      4     a=5
      5

UnboundLocalError: local variable 'a' referenced before assignment
```

Τι έγινε εδώ; Γιατί πέταξε αυτό το λάθος; Όπως είπαμε και πριν: αν θέσεις μία νέα τιμή σε μία μεταβλητή οπουδήποτε μέσα σε μία συνάρτηση τότε την ξαναορίζεις. Άρα η a ξαναορίστηκε μέσα στη συνάρτηση. Όταν λοιπόν πάμε να κάνουμε print(a), τότε του λέμε να τυπώσει τη μεταβλητή a της συνάρτησης η οποία.. δεν έχει ακόμα οριστεί!

Αν για κάποιο λόγο θέλουμε να πούμε ότι η a στη f αναφέρεται στη "έξω" a και όχι στη

"μέσα" πρέπει να το δηλώσουμε:

In [178…
```
a=5
def f():
    global a

    print (a)
    a=6

f()
print (a)
```

```
5
6
```

Εδώ παρατηρούμε το εξής: αλλάξαμε τη a μέσα στη συνάρτηση και άλλαξε και έξω!

Επίσης είναι προφανές ότι μία μεταβλητή που έχει οριστεί μέσα σε μία συνάρτηση δεν "υπάρχει" έξω από αυτή:

In [182…
```
def f(x):
    t = "hello"
print (t)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-182-4a4591587f68> in <module>
      1 def f(x):
      2     t = "hello"
----> 3 print (t)

NameError: name 't' is not defined
```

Τι γίνεται με τις παραμέτρους των συναρτήσεων; Αν αλλάξουμε μία παράμετρο μέσα στη συνάρτηση, θα αλλάξει και έξω; Η απάντηση είναι.. εξαρτάται! Αν η παράμετρο είναι int, float, string, bool, complex, None (ή αλλιώς primitive data types). Τότε δεν αλλάζει:

In [180…
```
def f(x):
    x += " mitsos"

x = "My name is:"
f(x)
print (x)
```

```
My name is:
```

Αν όμως είναι list, dictionary, set, class (θα τα μάθουμε αργότερα). Τότε αλλάζει!

In [181…
```
def f(l):
    l.append(5)

a = [1,2,3,4]
f(a)
print (a)
```

```
[1, 2, 3, 4, 5]
```