Ελληνικό Μεσογειακό Πανεπιστήμιο
Σχολή Μηχανικών / Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

HMY01K06
Επιστημονικός Προγραμματισμός με Python

Διάλεξη Όγδοη
Αρχεία

Φθινόπωρο 2025

Τι είναι αρχείο;

- Οι πάσης φύσεως πληροφορίες και δεδομένα που επεξεργάζονται στον Η/Υ αποθηκεύονται στη βοηθητική μνήμη *(πχ σκληρό δίσκο)* με μορφή ειδικών δομών που ονομάζονται *αρχεία* (*files*)
- Κάθε αρχείο αποτελείται από μια *ακολουθία από bytes* τα οποία κωδικοποιούν το περιεχόμενό του

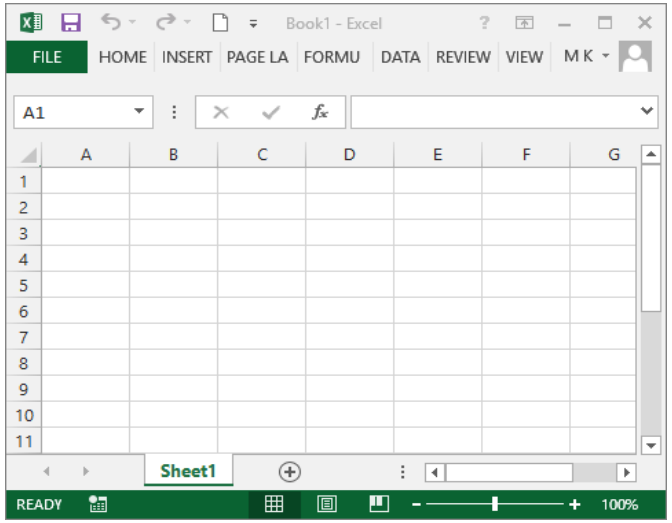
byte 0	*byte 1*	*byte 2*		*byte n*
10010110	00101101	11001101	...	10101101

- Ενα αρχείο χαρακτηρίζεται από το *όνομα* (*name*) και την *κατάληξή του* (*extension*) η οποία συνήθως υποδηλώνει και τον *τύπο* του, δηλαδή το *τι περιέχει* και το *πώς είναι κωδικοποιημένο* αυτό που περιέχει

`Image1.jpg` `Εργασία.pdf` `Μισθοδοσία.xlsx`

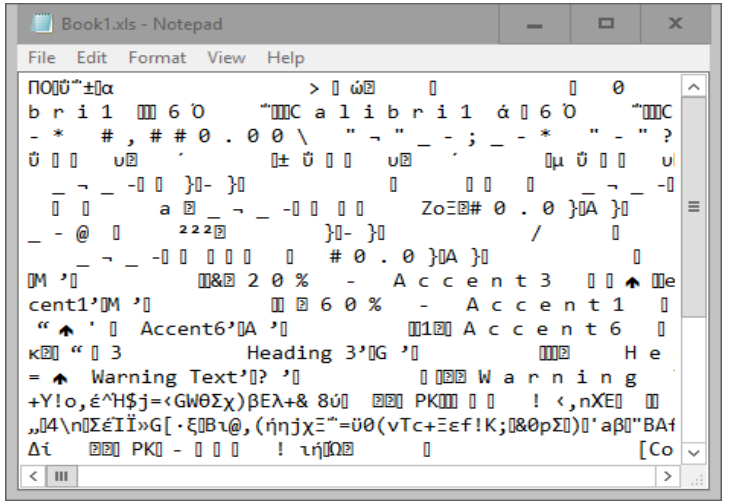
Χειρισμός αρχείων από προγράμματα

- Ο *τύπος* του αρχείου είναι εκείνος ο οποίος *καθορίζει* και το *πώς το χειριζόμαστε* δηλαδή από τι είδους προγράμματα αναγνωρίζεται ή/και δημιουργείται
- Αν κάποιο πρόγραμμα επιχειρήσει *ανοίξει* και να "*απεικονίσει*" ένα αρχείο με *διαφορετική κωδικοποίηση* από την αναμενόμενη, τότε το αποτέλεσμα είναι *απρόβλεπτο* (αυτό που συνήθως ονομάζουμε "*σκουπίδια*")



Άνοιγμα στο Excel

Book1.xls



Άνοιγμα στο Σημειωματάριο

Οι δυο βασικές κατηγορίες αρχείων

Ανάλογα με το *είδος του περιεχόμενου* και την *εσωτερική τους δομή* τα αρχεία διακρίνονται σε δυο βασικές κατηγορίες:

- Αρχεία κειμένου

  Περιέχουν κείμενο σε οποιαδήποτε μορφή και κωδικοποίηση

- Δυαδικά αρχεία

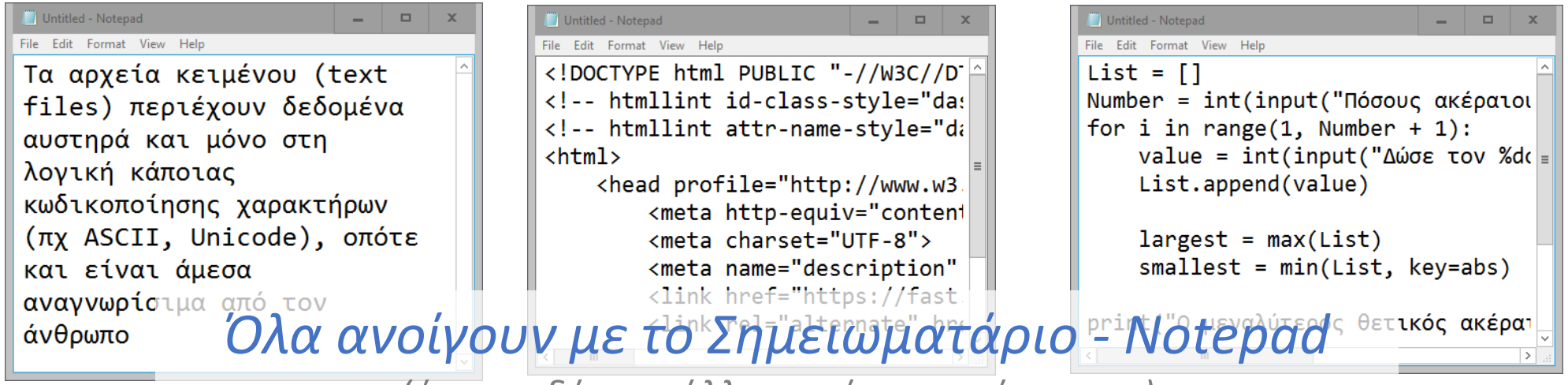
  Όλα τα υπόλοιπα είδη αρχείων

Η διάκριση αυτή γίνεται για λόγους *καλύτερου χειρισμού*. Όπως είπαμε νωρίτερα κάθε αρχείο αποτελείται από μια *ακολουθία από bytes* τα οποία κωδικοποιούν το περιεχόμενό του ανάλογα με το είδος του. Αν *δεν έχουμε πληροφορίες* για το περιεχόμενο ενός αρχείου *δεν υπάρχει δυνατότητα να καταλάβουμε* αν περιέχει κείμενο, εικόνες, βίντεο, εκτελέσιμο κώδικα κλπ.

Αρχεία κειμένου (text files)

Περιέχουν δεδομένα αυστηρά και μόνο στη λογική κάποιας κωδικοποίησης χαρακτήρων (πχ ASCII, Unicode), οπότε και είναι *άμεσα αναγνωρίσιμα από τον άνθρωπο* μέσω οποιουδήποτε *συντάκτη κειμένου (text editor)*

Παραδείγματα: Αρχεία *απλού*(*) κειμένου (.txt), ιστοσελίδες (.html, .css), πηγαία αρχεία γλωσσών προγραμματισμού (.cpp, .py, .ftn, .pas, .bas, .php…)



Όλα ανοίγουν με το Σημειωματάριο - Notepad

(ή οποιοδήποτε άλλο παρόμοιο πρόγραμμα)

(*)*απλό νοείται το κείμενο που δεν περιέχει άλλα γραφικά στοιχεία (εικόνες, πίνακες, γραφήματα κλπ.)*

Δυαδικά αρχεία (binary files):

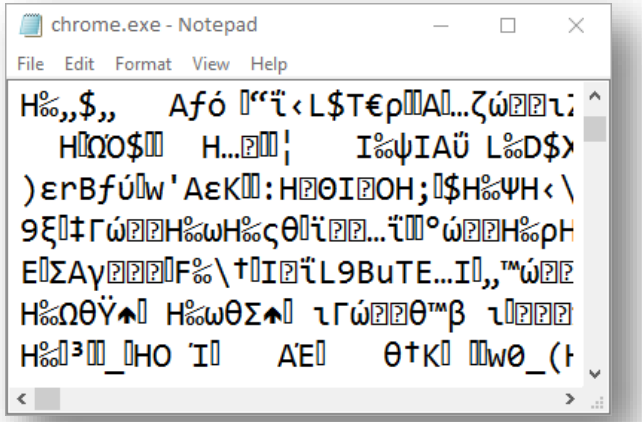
Τα δεδομένα που περιέχουν ακολουθούν *συγκεκριμένη κωδικοποίηση* και αναγνωρίζονται μόνο από τα *ειδικά προγράμματα* που τα χειρίζονται

Δεν είναι άμεσα αναγνώσιμα καθώς *δεν αποτελούνται από κωδικοποιημένους χαρακτήρες*

Παραδείγματα: Αρχεία πολυμέσων (.jpg, .bmp, .wav, .mp3, .ogg, .avi, .mp4, .mkv …), αρχεία εφαρμογών γραφείου (.doc, .xls, ppt) αρχεία βάσεων δεδομένων, ειδικών εφαρμογών…

… και φυσικά, *εκτελέσιμα αρχεία*, δηλαδή ***προγράμματα*** (.exe, .com, .cmd, .app …)

Αν επιχειρήσουμε να *ανοίξουμε ένα δυαδικό αρχείο* με ένα text editor θα δούμε αυτό που συνήθως ονομάζουμε "*σκουπίδια*", δηλαδή την προσπάθεια ερμηνείας των bytes του αρχείου σε χαρακτήρες κειμένου, που προφανώς δημιουργεί *τυχαία και ακατανόητα κείμενα*



Βασικές ενέργειες πάνω στα αρχεία

- Μπορούμε να *διαβάσουμε* δεδομένα από ένα αρχείο, ή να *γράψουμε* δεδομένα σε αρχείο που υπάρχει ή και να *δημιουργήσουμε* ένα αρχείο αν δεν υπάρχει
- Όταν θέλουμε να *διαβάσουμε* ή να *γράψουμε* σε ένα αρχείο, πρέπει πρώτα να το *ανοίξουμε*
- Όταν *τελειώσουμε*, πρέπει να *κλείσουμε*, ώστε να *αποδεσμευτούν* οι πόροι του συστήματος που είναι αφιερωμένοι στο αρχείο
- Επομένως η σειρά ενεργειών είναι η εξής:
    - *Δημιουργία ή Άνοιγμα* αρχείου
    - *Επεξεργασία* αρχείου *(διάβασμα/γράψιμο στοιχείων στο αρχείο)*
    - *Κλείσιμο* αρχείου

Αρχεία στην Python: Ας ξεκινήσουμε με ενα παράδειγμα

```
f = open('my_friends.txt', 'w')
f.write('Κώστας\n')
f.write('Γιώργος\n')
f.write('Μαρία\n')
f.close()
```

Δημιουργία αρχείου κειμένου

my_friends.txt

```
Κώστας\n
Γιώργος\n
Μαρία\n
```

```
f = open('my_friends.txt', 'r')
for line in f:
   print(line.rstrip())
f.close()
```

Διάβασμα αρχείου κειμένου
και εκτύπωση περιεχομένου

```
Κώστας
Γιώργος
Μαρία
```

Τι θα εμφανιζόταν αν παραλείπαμε το `.rstrip()`;

Θυμόμαστε

`rstrip()`: "Καθαρίζει" το δεξί άκρο μιας συμβολοσειράς από *κενά* και χαρακτήρες `\n`

`lstrip()`: Αντίστοιχα για το αριστερό άκρο

`strip()`: Και τα δύο άκρα ταυτόχρονα

Πως ανοίγουμε ένα αρχείο;

Στην Python χρησιμοποιούμε την συνάρτηση open() για το άνοιγμα αρχείων

αντικείμενο_αρχείου = open(*όνομα_αρχείου*, *mode*)

- *όνομα_αρχείου:* είναι ένα συντακτικά αποδεκτό μονοπάτι του Λ.Σ. προς ένα αρχείο
- *mode:* συμβολοσειρά που καθορίζει την *κατάσταση λειτουργίας του αρχείου*

Η εντολή open επιστρέφει ένα αντικείμενο τύπου file object το οποίο εκχωρούμε σε μια μεταβλητή *αντικείμενο_αρχείου* μέσω του οποίου αναφερόμαστε στη συνέχεια, στο αρχείο για κάθε είδους λειτουργία

```
f = open("friends.txt")   # ανοίγει ένα αρχείο κειμένου για διάβασμα στον φάκελο που βρισκόμαστε

f = open("friends.txt", "w")  # ανοίγει ένα αρχείο κειμένου για γράψιμο στον φάκελο που βρισκόμαστε

f = open("C:/Python312/friends.txt")   # ανοίγει ένα αρχείο κειμένου για διάβασμα
                                       χρησιμοποιώντας το πλήρες μονοπάτι
```

Καταστάσεις λειτουργίας (modes) αρχείων

- Το *mode* είναι μια συμβολοσειρά που αποτελείται *3 προαιρετικούς χαρακτήρες*
  δηλαδή η συμβολοσειρά mode μπορεί να έχει μήκος 1, 2 ή 3 ή να μην υπάρχει καθόλου

  - Ο ενας καθορίζει το αν το αρχείο θα ανοιχτεί για ανάγνωση ή εγγραφή *(τρία διαφορετικά είδη εγγραφής)* και μπορεί να είναι ενας από τους παρακάτω:

    - `r` (*read*) : Το αρχείο ανοίγει για *ανάγνωση (default – μπορεί να παραλειφθεί)*
      αν το αρχείο δεν υπάρχει, επιστρέφεται μήνυμα λάθους

    - `w` (*write*) : Το αρχείο ανοίγει κενό για *εγγραφή*
      αν το αρχείο *προϋπάρχει*, τότε το περιεχόμενό του *καθαρίζεται*

    - `a` (*append*) : Το αρχείο ανοίγει για *προσθήκη νέων εγγραφών στο τέλος*

    - `x` (*exclusive creation*) : Το αρχείο ανοίγει κενό για *εγγραφή*, *εφόσον δεν προϋπάρχει*
      αν το αρχείο προϋπάρχει, επιστρέφεται μήνυμα λάθους

Καταστάσεις λειτουργίας (modes) αρχείων

- Ένας δεύτερος χαρακτήρας καθορίζει το είδος των δεδομένων που περιέχει το αρχείο
  - `t` (*text*) : Το αρχείο θεωρείται *(ανοίγει ως) αρχείο κειμένου (default)*
  - `b` (*binary*) : Το αρχείο θεωρείται *(ανοίγει ως) δυαδικό αρχείο*

- Ενας τρίτος χαρακτήρας είναι το `"+"` που όταν υπάρχει, επιτρέπει να εκτελούνται ***και*** λειτουργίες ανάγνωσης σε αρχεία που έχουν ανοιχτεί για εγγραφή (και αντίστροφα)

- Η σειρά εμφάνισης των τριών χαρακτήρων δεν έχει σημασία (`"rb+"`, `"r+b"`, `"+br"`…)

Όλα ισοδύναμα

Παραδείγματα:

<Τίποτα> : Αρχείο *κειμένου(default)* που ανοίγει για *ανάγνωση (default)*

`"rb"` : *Δυαδικό* αρχείο που ανοίγει για *ανάγνωση (εναλλακτική μορφή: σκέτο* `"b"` – γιατί; *)*

`"a"` : Αρχείο *κειμένου* που ανοίγει για *προσάρτηση στο τέλος (εναλλακτικές μορφές:* `"ta"`, `"at"` *)*

`"x+b"` : *Δυαδικό* αρχείο που ανοίγει για *εγγραφή (και ανάγνωση).* Το αρχείο ***δεν πρέπει*** *να προϋπάρχει*

`"r+b"` : *Δυαδικό* αρχείο που ανοίγει για *ανάγνωση (και εγγραφή).* Το αρχείο ***πρέπει*** *να προϋπάρχει*

Χειρισμός αρχείων κειμένου

Η συνάρτηση `open()` δημιουργεί και επιστρέφει ένα *αντικείμενο_αρχείου* το οποίο εκχωρείται κατά κανόνα σε έναν *προσδιοριστή* που αποτελεί εφεξής το *όνομα του αρχείου μέσα στο πρόγραμμα*

```
>>> file_friends = open("my_friends.txt", "w")
```

Λόγω του `mode "w"`, *αν προϋπήρχε* το αρχείο `my_friends.txt`, τότε το περιεχόμενό του *χάνεται* και ανοίγει *καθαρό (κενό περιεχομένου)* για γράψιμο

```
>>> file_friends
<_io.TextIOWrapper name='C:/Test/my_friends.txt' mode='w' encoding ='cp1253'>

>>> type(file_friends)
<class '_io.TextIOWrapper'>
```

Χειρισμός αρχείων κειμένου

Κάθε *αντικείμενο_αρχείου* που δημιουργείται με το open(), έχει *μεθόδους* και *ιδιότητες* που επιστρέφουν συγκεκριμένες τιμές όπως:

```
>>> file_friends.name   # Ιδιότητα name
'C:/Test/my_friends.txt'

>>> file_friends.mode # Ιδιότητα mode
'w'

>>> file_friends.readable() # Μέθοδος readable()
False

>>> file_friends.writable() # Μέθοδος writable()
True
```

Εγγραφή σε αρχείο κειμένου

Για να *γράψουμε* κείμενο σε ένα αρχείο(*) χρησιμοποιούμε τη μέθοδο `.write()`

```
αντικείμενο_αρχείου_κειμένου.write(συμβολοσειρά)
```

```
>>> names = '''Γιώργος
Μαρία
Αντώνης
Πέτρος'''
>>> file_friends.write(names)
30
```

Η μέθοδος `.write()` επιστρέφει *το πλήθος των χαρακτήρων* που γράφτηκαν.
Σε αυτούς συμπεριλαμβάνονται και οι *χαρακτήρες αλλαγής γραμμής*

(*) *πρέπει να το έχουμε ανοίξει προηγουμένως ως αρχείο κειμένου και με δικαίωμα εγγραφής*

Κλείσιμο αρχείου

Όταν όλες μας οι ενέργειες ολοκληρωθούν, *πρέπει να κλείσουμε το αρχείο* με close()

```
αντικείμενο_αρχείου.close()
```

```
>>> file_friends.close() # Η μέθοδος close() δεν επιστρέφει κάτι.
>>> file_friends.closed
True
>>> file_friends.name
'C:/Test/my_friends.txt'
>>> file_friends.mode
'w'
>>> file_friends.write("Ελένη")
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    file_friends.write("Ελένη")
ValueError: I/O operation on closed file.
```

Ακόμα και μετά το κλείσιμο ενός αρχείου, μπορούμε να αναφερόμαστε στις ιδιότητές του….

…όμως δεν μπορούμε πια να έχουμε πρόσβαση στο περιεχόμενό του

Η έννοια του δείκτη ενός αρχείου

- Ο *δείκτης* ενός αρχείου *(file pointer)* αναφέρεται στη *θέση* μέσα στο αρχείο από όπου θα γίνει η *επόμενη ανάγνωση* ή *εγγραφή*
- Όταν *ανοίγουμε* ένα αρχείο, ο *δείκτης* τοποθετείται *στην αρχή του αρχείου*
- Καθώς *διαβάζουμε* ή *γράφουμε* δεδομένα στο αρχείο, ο δείκτης *μετακινείται αυξάνοντας την τιμή του* ανάλογα.

- Μπορούμε να *δούμε* την θέση του δείκτη μέσα στο αρχείο με τη μέθοδο `tell()`
- Μπορούμε να *αλλάξουμε* την θέση του δείκτη μέσα στο αρχείο *(μετακινούμενοι μπροστά ή πίσω)* με τη μέθοδο `seek()`
- Αυτό είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου χρειάζεται να *διαβάσουμε* ή να *γράψουμε* σε *συγκεκριμένα μέρη ενός αρχείου*, αντί να εργαζόμαστε από την αρχή μέχρι το τέλος σειριακά

Ανάγνωση αρχείου κειμένου

Για να *διαβάσουμε* το αρχείο κειμένου `my_friends.txt` που δημιουργήσαμε προηγουμένως, πρέπει αρχικά να το ανοίξουμε και πάλι, αυτή τη φορά όμως με δικαίωμα ανάγνωσης

```
>>> file_friends = open("my_friends.txt") ← απουσία mode: αρχείο κειμένου για ανάγνωση
>>> file_friends
<_io.TextIOWrapper name='C:/Test/my_friends.txt' mode='r' encoding='cp1253'>
```

Το αρχείο που προσπαθούμε να ανοίξουμε θα πρέπει να υπάρχει

```
>>> g = open("my_enemies.txt")
>>> Traceback (most recent call last):
  File "<pyshell#49>", line 1, in <module>
    g = open("C:/Users/MK/my_enemies.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'C:/Users/Test/my_enemies.txt'
```

Ανάγνωση αρχείου κειμένου

Έστω λοιπόν ότι ανοίγουμε κανονικά και πάλι το αρχείο `my_friends.txt` αυτή τη φορά *για ανάγνωση*

```
>>> file_friends = open("my_friends.txt")
```

Εφόσον όλα πάνε καλά, *το* `file_friends` *αναφέρεται πάλι στο ίδιο αρχείο*, όμως ορισμένα από τα *χαρακτηριστικά* του έχουν *–όπως ήταν αναμενόμενο-* *αλλάξει*:

```
>>> file_friends.mode
'r'
>>> file_friends.readable()
True
>>> file_friends.writable()
False
```

Ανάγνωση αρχείου κειμένου

Μπορούμε να χρησιμοποιήσουμε τις ακόλουθες *τρεις μεθόδους*, οι οποίες επιστρέφουν συμβολοσειρές χαρακτήρων

1. `αντικείμενο_αρχείου.read(k)` `#διάβασε το πολύ k χαρακτήρες`

   Κάθε φορά που καλείται, διαβάζει και επιστρέφει σε μια συμβολοσειρά *το πολύ*[(*)] k *χαρακτήρες*, ή *όλο το υπόλοιπο περιεχόμενο* του αρχείου, αν παραλειφθεί το k

2. `αντικείμενο_αρχείου.readline()` `#διάβασε μέχρι να συναντήσεις \n`

   Κάθε φορά που καλείται, διαβάζει και επιστρέφει τους χαρακτήρες *μέχρι και το τέλος της γραμμής (χαρακτήρα αλλαγής γραμμής \n).* Αν *δεν συναντήσει χαρακτήρα αλλαγής γραμμής*, τότε *συνεχίζει να διαβάζει μέχρι το τέλος του αρχείου*

3. `αντικείμενο_αρχείου.readlines()` `#διάβασε σε λίστα τις υπόλοιπες γραμμές`

   Όταν καλείται, *διαβάζει το υπόλοιπο του αρχείου* και επιστρέφει μια ***λίστα*** *συμβολοσειρών*, καθεμία από τις οποίες αντιστοιχεί σε *μια γραμμή του αρχείου* δηλαδή τερματίζεται με τον *χαρακτήρα αλλαγής γραμμής (\n)*

(*) *επιστρέφει λιγότερους από k αν δεν υπάρχουν αρκετοί να διαβάσει μέχρι το τέλος του αρχείου*

Ανάγνωση αρχείου κειμένου - Παραδείγματα

Παράδειγμα 1:

```
>>> f = open("C:/Test/my_friends.txt")
>>> f.read(5)
'Γιώργ'
  1 2 3 4 5
>>> f.read(7)
'ος\nΜαρί'
  1 2  3  4 5 6 7
>>> f.read()
'α\nΑντώνης\nΠέτρος'
>>> f.read()
''
>>> f.close()
```

my_friends.txt

```
Γιώργος\n
Μαρία\n
Αντώνης\n
Πέτρος
```

Θυμόμαστε την έννοια του *δείκτη* ενός αρχείου *(file index)*

- Είναι το *σημείο* εκείνο του αρχείου στο οποίο βρισκόμαστε μετά από κάθε εκτέλεση μιας εντολής *ανάγνωσης* ή *εγγραφής*
- Όταν *ανοίγει* ένα αρχείο, ο δείκτης του είναι *στο μηδέν*. Καθώς *προχωράμε* στην ανάγνωση ή στην εγγραφή του αρχείου, ο δείκτης *μετακινείται*
- Μπορούμε να *δούμε* την παρούσα θέση του δείκτη, ή να τον *μετακινήσουμε* σε άλλη θέση (*μπροστά / πίσω*) χρησιμοποιώντας τις εντολές **`seek()`** και **`tell()`**

Ανάγνωση αρχείου κειμένου - Παραδείγματα

Παράδειγμα 2:

```
>>> f = open("C:/Test/my_friends.txt")
>>> f.readline()
'Γιώργος\n'
>>> f.readline()
'Μαρία\n'
>>> f.readline()
'Αντώνης\n'
>>> f.readline()
'Πέτρος\n'
>>> f.readline()
''
>>> f.close()
```

my_friends.txt

```
Γιώργος\n
Μαρία\n
Αντώνης\n
Πέτρος
```

```
>>> f = open("C:/Test/my_friends.txt")
>>> f.read(3)
'Γιώ'
>>> f.readline()
'ργος\n'
```

```
>>> f = open(" C:/Test/my_friends.txt ")
>>> f.readlines() # επιστρέφει μια λίστα
['Γιώργος\n', 'Μαρία\n', 'Αντώνης\n', 'Πέτρος']
```

Το αρχείο ως επαναλήψιμος τύπος

Το *αντικείμενο_αρχείου* που επιστρέφει η `open()` είναι επαναλήψιμος τύπος *(iterable),* που σημαίνει ότι μπορούμε να εφαρμόσουμε πάνω του μια δομή επανάληψης όπως `for` ή `with`

```
>>> f = open("C:/Users/Test/my_friends.txt")
>>> n=0
>>> for line in f:
        n+=1
        print(n, "-", line.rstrip())
```

Θα εμφανίσει:
1-Γιώργος
2-Μαρία
3-Αντώνης
4-Πέτρος

Σε κάθε επανάληψη, η τοπική μεταβλητή του `for` επιστρέφει ως συμβολοσειρά την *επόμενη γραμμή* του αρχείου κειμένου, και το *μπλοκ του κώδικα* που ακολουθεί την *επεξεργάζεται*

Ήδη ξέρουμε πως μπορούμε να διαβάσουμε *όλο το αρχείο μονομιάς* με `.read()` ή `.readlines()`

```
>>> f.seek(0) # Επιστροφή στην κορυφή του αρχείου  (ή εναλλακτικά κλείσιμο και ξανα-άνοιγμα)
>>> my_list = f.readlines()
>>> my_list
['Γιώργος\n', 'Μαρία\n', 'Αντώνης\n', 'Πέτρος ']
```

Το αρχείο κειμένου διαβάζεται *ολόκληρο* ως *λίστα* στην *κεντρική μνήμη* του υπολογιστή
Πρέπει *να αποφεύγεται για μεγάλα αρχεία*, λόγω της *μεγάλης ποσότητας κεντρικής μνήμης* που απαιτείται

Διαχείριση εννοιολογικού πλαισίου (Context management)

Είναι η δυνατότητα διαχείρισης *των πόρων ενός συστήματος* με έναν τρόπο που να εξασφαλίζει ότι ορισμένες ενέργειες *(όπως η απελευθέρωση πόρων ή το κλείσιμο αρχείων)* εκτελούνται *αυτόματα* όταν ολοκληρωθεί ένας κύκλος εργασίας

Στη Python αυτό επιτυγχάνεται με τη δήλωση `with`, η οποία *εγγυάται την ορθή διαχείριση των πόρων*, ακόμα και αν συμβεί κάποιο σφάλμα κατά την εκτέλεση του κώδικα

Παράδειγμα χρήσης της δήλωσης `with` *για το άνοιγμα και το κλείσιμο ενός αρχείου:*

```
with open("C:/Users/Test/my_friends.txt") as f:
    for line in f:
        print(line.rstrip(), end = " " )
# Με το που βγαίνουμε από το with, το αρχείο κλείνει αυτόματα (δεν χρειάζεται να δώσουμε close)
```

```
Γιώργος Μαρία Αντώνης Πέτρος
```

Οι μέθοδοι .seek() και .tell()

- `.seek(k[,w])` : μετακινεί τον δείκτη ενός αρχείου στη θέση `k`. Η επόμενη ενέργεια επάνω στο αρχείο *(εγγραφή ή ανάγνωση)* ξεκινάει από τη νέα αυτή θέση του δείκτη
- `.tell()` : επιστρέφει την τιμή *(θέση)* του δείκτη ενός αρχείου *(στο άνοιγμα, είναι 0)*

Παράδειγμα 1:

```
>>> f = open("C:/Users/Test/my_friends.txt")
>>> f.tell()
0
>>> f.read(3)
'Γιώ'
>>> f.tell()
3
>>> f.seek(2)
2
>>> f.read(3)
'ώργ'
```

```
>>> f.tell()
5
>>> f.readline()
'ος\n'
>>> f.seek(5)
5
>>> f.readlines()
['ος\n', 'Μαρία\n', 'Αντώνης\n', 'Πέτρος']
```

my_friends.txt

```
Γιώργος\n
Μαρία\n
Αντώνης\n
Πέτρος
```

Οι μέθοδοι .seek() και .tell()

Παράδειγμα 2:


```
>>> f = open("C:/Users/Test/my_friends.txt", "r+") #διάβασμα + γράψιμο
>>> f.seek(5)
5   ο δείκτης του αρχείου πηγαίνει στη θέση 5
>>> f.read(1)
'ο'
>>> f.seek(5)
5     ξανά πίσω την θέση  5 (είχε πάει στην 6 – γιατί;)
>>> f.write("η")
1     επιστρέφει τον αριθμό των χαρακτήρων που έγραψε
>>> f.seek(0)
0     ο δείκτης πηγαίνει στην κορυφή του αρχείου
>>> f.readline()
'Γιώργης\n'
```

my_friends.txt

```
Γιώργος\n
Μαρία\n
Αντώνης\n
Πέτρος
```

Πριν

```
Γιώργης\n
Μαρία\n
Αντώνης\n
Πέτρος
```

Μετά

Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- *Κωδικοποίηση χαρακτήρα*: αντιστοίχηση του χαρακτήρα με έναν *μοναδικό θετικό ακέραιο*, ώστε να είναι εφικτή η αποθήκευσή του στη μνήμη του Η/Υ. (πχ. `"!"` ☞ **`33`** `"f"` ☞ **`102`**)
- Η πιο διαδεδομένη μέχρι πρόσφατα και ιστορικά πρώτη κωδικοποίηση *(από το 1963 ως ASA X3.4)* ονομάζεται κώδικας *ASCII* (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)
- Ο κώδικας ASCII διαθέτει *256* θέσεις και επομένως χρειάζεται *1 byte (8 bits)* για την αποθήκευση κάθε χαρακτήρα *($2^8 = 256$)*
- Οι *πρώτες 128 θέσεις* του κώδικα ASCII *(0-127)* χρησιμοποιούνται για την αποθήκευσή *όλων των γραμμάτων της Αγγλικής γλώσσας* (κεφαλαίων - πεζών) , των *ψηφίων 0-9* και σχεδόν όλων των *ειδικών χαρακτήρων* που χρησιμοποιούνται στις γλώσσες προγραμματισμού *(+ - * ^ # ! …)*
- Για τις *επόμενες 128 θέσεις (128-255)* υπάρχει *δυνατότητα επιλογής* των χαρακτήρων που επιθυμούμε να κωδικοποιήσουμε. (πχ στην *Ελλάδα*, θα θέλαμε να κωδικοποιήσουμε τα *Ελληνικά,* στην *Βουλγαρία* τα *Βουλγαρικά κοκ.*)
- Αυτό δημιουργεί πολλές *εναλλακτικές μορφές του κώδικα ASCII* που λέγονται *κωδικοσελίδες*

  π.χ. *Codepage 1251 : Cyrillic, Codepage 1252 : Western European, Codepage 1253 : Greek*

Κωδικοσελίδες 1253 και 1251 του κώδικα ASCII

Codepage 1253 - Greece Windows

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-																
1-																
2-		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8-	€		‚	ƒ	„	…	†	‡		‰		‹				
9-		‘	’	“	”	•	–	—		™		›				
A-		΅	Ά	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	―
B-	°	±	²	³	΄	µ	¶	·	Έ	Ή	Ί	»	Ό	½	Ύ	Ώ
C-	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
D-	Π	Ρ		Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ	ά	έ	ή	ί
E-	ΰ	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο
F-	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	ϊ	ϋ	ό	ύ	ώ	

(the Unicode equivalent code is displayed under each character)

Codepage 1251 - Cyrillic Windows

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-																
1-																
2-		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8-	Ђ	Ѓ	‚	ѓ	„	…	†	‡	€	‰	Љ	‹	Њ	Ќ	Ћ	Џ
9-	ђ	‘	’	“	”	•	–	—		™	љ	›	њ	ќ	ћ	џ
A-		Ў	ў	Ј	¤	Ґ	¦	§	Ё	©	Є	«	¬	-	®	Ї
B-	°	±	І	і	ґ	µ	¶	·	ё	№	є	»	ј	Ѕ	ѕ	ї
C-	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
D-	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
E-	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
F-	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

(the Unicode equivalent code is displayed under each character)

Πρώτοι 128 χαρακτήρες (θέσεις 0-127) πάντα Αγγλικά

Επόμενοι 128 χαρακτήρες (θέσεις 128-255) διαφορετικές γλώσσες ανάλογα με την κωδικοσελίδα

Βασικός *περιορισμός* του κώδικα ASCII είναι ότι *δεν υποστηρίζει πλήρως πολύγλωσσο κείμενο*

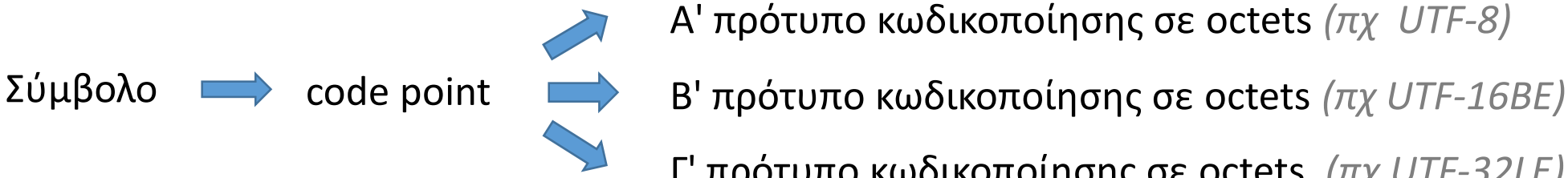
(πχ . δεν μπορούμε να έχουμε ταυτόχρονα Ελληνικά και Βουλγαρικά λόγω διαφορετικών κωδικοσελίδων)

Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- Για *πραγματικά πολύγλωσσο κείμενο*, χρειαζόμαστε μια κωδικοποίηση με *αρκετές θέσεις* ώστε να περιλαμβάνει *τα αλφάβητα όλων των γλωσσών του κόσμου* (και ίσως ακόμα περισσότερα σύμβολα)

- Η πιο γνωστή κωδικοποίηση που υποστηρίζει πολύγλωσσο κείμενο ονομάζεται ***Unicode*** και *έχει πρακτικά γίνει το νέο παγκόσμιο στάνταρ*, αντικαθιστώντας την κωδικοποίηση ASCII και γενικά όλες τις παλαιότερες κωδικοποιήσεις

- Με την κωδικοποίηση *Unicode* σε κάθε σύμβολο αντιστοιχεί ενας *μοναδικός ακέραιος θετικός αριθμός* που ονομάζεται *code point* ο οποίος είναι ανεξάρτητος από το πρότυπο που τελικά θα χρησιμοποιηθεί για τη μετατροπή του σε *bytes (που εδώ ονομάζονται octets)*



Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- Στην κωδικοποίηση Unicode κάθε *code point (μοναδικός ακέραιος αριθμός)* μπορεί να κωδικοποιηθεί σε *octets (δηλαδή bytes)* με διαφορετικά πρότυπα
- Το πιο *γνωστό και διαδεδομένο* πρότυπο του κώδικα Unicode ονομάζεται ***UTF-8** (8-bit Unicode Transformation Format)*
- *Σύμφωνα με το πρότυπο **UTF-8**, κάθε code point κωδικοποιείται σε 1,2,3 ή 4 octets, ανάλογα με την "σπουδαιότητα" του χαρακτήρα που αντιπροσωπεύει*

- Εκτός από το πρότυπο *UTF-8,* υπάρχουν ***6*** ακόμα πρότυπα τα οποία είναι λιγότερο διαδεδομένα
  - *UTF-16*
  - *UTF-16BE / UTF-16LE (παραλλαγές του UTF-16)*
  - *UTF-32*
  - *UTF-32BE / UTF-32LE (παραλλαγές του UTF-32)*

Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

Πρότυπο Unicode: κάθε χαρακτήρας αντιστοιχεί σε ένα *μοναδικό code-point*…

χαρακτήρας	. . .	g	. . .	ά	. . .	Ж	. . .	🤣	. . .
code –point (δεκαεξαδική μορφή)	. . .	0067	. . .	03AC	. . .	0416	. . .	1F923	. . .
code –point (δεκαδική μορφή) *ουσιαστικά η θέση του στον πίνακα*	. . .	103	. . .	940	. . .	1046	. . .	129315	. . .

UTF-8 Encoding:	0x67
UTF-16 Encoding:	0x0067
UTF-32 Encoding:	0x00000067

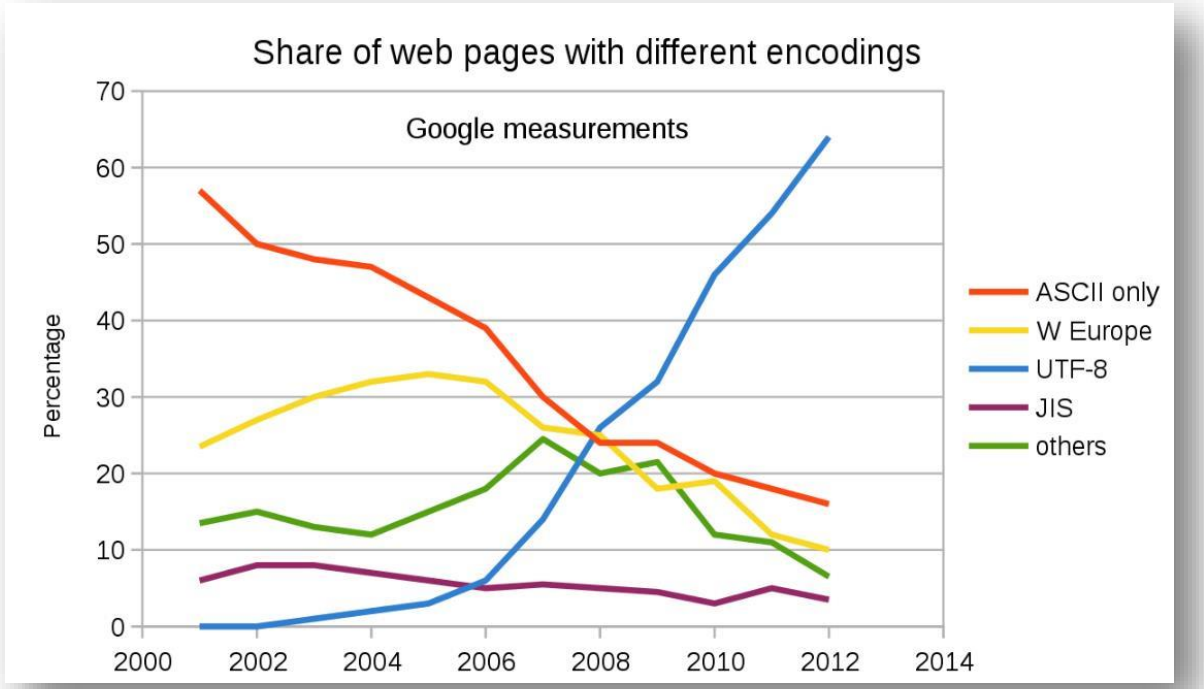
UTF-8 Encoding:	0xCE 0xAC
UTF-16 Encoding:	0x03AC
UTF-32 Encoding:	0x000003AC

UTF-8 Encoding:	0xD0 0x96
UTF-16 Encoding:	0x0416
UTF-32 Encoding:	0x00000416

UTF-8 Encoding:	0xF0 0x9F 0xA4 0xA3
UTF-16 Encoding:	0xD83E 0xDD23
UTF-32 Encoding:	0x0001F923

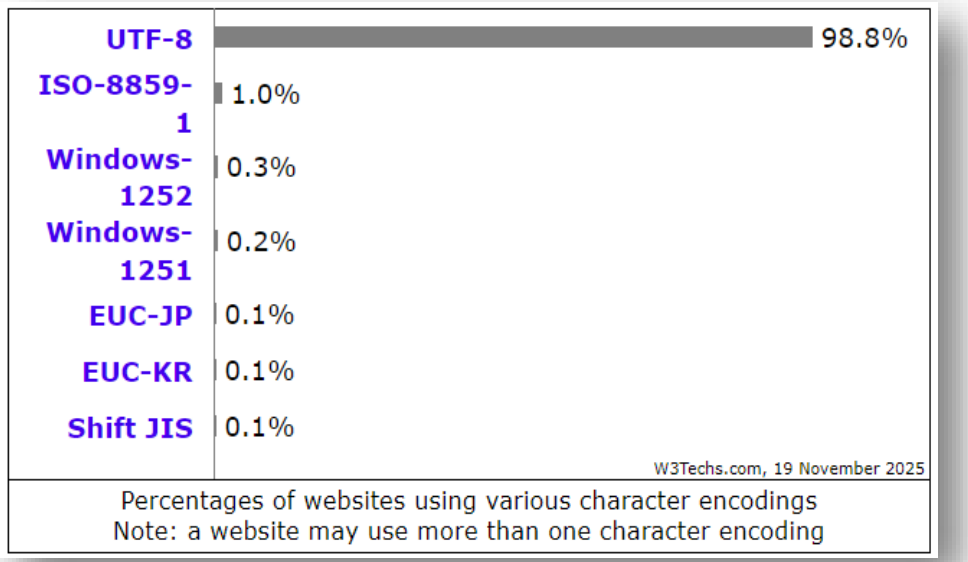
…το οποίο *κωδικοποιείται διαφορετικά* ανάλογα την *μέθοδο* που επιλέγουμε *(UTF-8…)*

Η κωδικοποίηση UTF-8 έχει επικρατήσει εδώ και αρκετά χρόνια στην δημιουργία ιστοσελίδων



https://en.wikipedia.org/wiki/Popularity_of_text_encodings

Νοέμβριος 2025



https://w3techs.com/technologies/overview/character_encoding

Αρχεία κειμένου και κωδικοποίηση χαρακτήρων

- Στην κωδικοποίηση *UTF-8* όλοι οι χαρακτήρες ***δεν*** *κωδικοποιούνται με τον ίδιο αριθμό bytes* (*κωδικοποίηση μεταβλητού μήκους* / *variable length encoding*)

- Οι χαρακτήρες με τη μεγαλύτερη συχνότητα εμφάνισης κωδικοποιούνται σε ένα byte (octet). *Οι χαρακτήρες αυτοί αντιστοιχούν εξ' ορισμού με τους πρώτους 128 χαρακτήρες του κώδικα ASCII. Έτσι, για κείμενα που περιέχουν μόνο Αγγλικά, οι κώδικες UTF-8 και ASCII ταυτίζονται*

- Οι επόμενοι *1920* χαρακτήρες κωδικοποιούνται σε *δυο bytes* και περιλαμβάνουν τα υπόλοιπα ειδικά σύμβολα όλων των Λατινογενών αλφαβήτων (*ë,î, ğ, ĩ* κλπ.), τα *Ελληνικά*, *Κυριλλικά*, *Αρμενικά*, *Εβραϊκά*, *Συριακά αλφάβητα* κ.α.

- Τα σύμβολα και οι χαρακτήρες των υπολοίπων γλωσσών του κόσμου (*Κινέζικα, Ιαπωνικά, Κορεάτικα* κλπ.) κωδικοποιούνται σε *3* ή *4* bytes

- *Πρακτικά για μας* αυτό σημαίνει ότι αν ένα *αρχείο* περιέχει κείμενο σε *κωδικοποίηση UTF-8* τότε
  - κάθε *Αγγλικός χαρακτήρας* ή/και ειδικό σύμβολο καταλαμβάνει *1 byte* στο αρχείο
  - κάθε *Ελληνικός χαρακτήρας* καταλαμβάνει *2 bytes* στο αρχείο

Πίσω στην Python

Πως ξέρω το είδος της κωδικοποίησης όταν γράφω/διαβάζω ένα αρχείο κειμένου;

```
>>> f = open("C:/Users/Test/my_friends.txt")
>>> f.encoding
'cp1253'
```

← κωδικοποίηση ASCII , Codepage 1253 (Ελληνικά)

Πως και από ποιόν αποφασίστηκε να χρησιμοποιηθεί ειδικά αυτή η κωδικοποίηση και όχι κάποια άλλη;

```
>>> import locale
>>> locale.getpreferredencoding()
'cp1253'
```

Μπορώ να καθορίσω ο ίδιος είδος της κωδικοποίησης όταν δημιουργώ ένα αρχείο κειμένου;

```
>>> f = open("C:/Users/Test/my_friends.txt", "w",  encoding = "utf-8")
```

Που θα βρω μια λίστα με όλους τους τύπους κωδικοποιήσεων που μπορώ να χρησιμοποιήσω;

https://docs.python.org/3.9/library/codecs.html#standard-encodings *(πάνω από 100 κωδ.)*

Παράδειγμα αρχείου με κωδικοποίηση cp1253 (Ελληνικά)

```
>>> f = open("Test1.txt", "w")
>>> f.write("Hello Πέτρο!")
12
>>> f.close()
```

Το αρχείο έχει μέγεθος 12 bytes
και περιέχει ισάριθμους χαρακτήρες

Byte	1	2	3	4	5	6	7	8	9	10	11	12
Character	1	2	3	4	5	6	7	8	9	10	11	12
	H	e	l	l	o		Π	έ	τ	ρ	o	!

```
>>> f = open("Test1.txt")

>>> f.seek(5)
5
>>> f.read(1)
' '
>>> f.tell()
6
>>> f.read(1)
'Π'
```

```
>>> f.tell()
7
>>> f.read(3)
'έτρ'
>>> f.tell()
10
```

Παραδείγματα αρχείων με κωδικοποίηση utf-8

```
>>> f = open("Test2.txt", "w", encoding = "utf-8")
>>> f.write("Hello Πέτρο!")
12
>>> f.close()
```

Το αρχείο έχει μέγεθος 17 bytes και περιέχει 12 χαρακτήρες

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Character	1	2	3	4	5	6	7		8		9		10		11		12
	H	e	l	l	o		Π		έ		τ		ρ		ο		!

```
>>> f = open("Test2.txt")
>>> f.readline()
Traceback (most recent call last):
UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position 13:
character maps to <undefined>

>>> f.close() # Δεν είναι απαραίτητο αν δοκιμάσουμε να το ξανα-ανοίξουμε
>>> f = open("Test2.txt", encoding = "utf-8")
>>> f.readline()
'Hello Πέτρο!'
```

```
>>> f.seek(5)
5
>>> f.read(1)
' '
>>> f.tell()
6
>>> f.read(1)
'Π'
>>> f.tell()
8 (γιατί;)
>>> f.read(1)
'έ'
>>> f.tell()
10 (γιατί;)
```

Παραδείγματα αρχείων με κωδικοποίηση utf-8

```
>>> f = open("Test3.txt", "w")
>>> f.write("Hello Πέτρο Питер!")
>>> Traceback (most recent call last):
UnicodeEncodeError: 'charmap' codec can't encode characters in position 12-16:
character maps to <undefined>

>>> f = open("Test3.txt", "w", encoding = "utf-8")
>>> f.write("Hello Πέτρο Питер!")
18    (τι μας δείχνει αυτός ο αριθμός;)
>>> f.close()
```

Το αρχείο έχει μέγεθος 28 bytes και περιέχει 18 χαρακτήρες

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Character	1	2	3	4	5	6	7		8		9		10		11		12	13		14		15		16		17		18
	H	e	l	l	o		Π		έ		τ		ρ		o			Π		и		т		e		p		!

Παραδείγματα αρχείων με κωδικοποίηση utf-8

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Character	1	2	3	4	5	6	7		8		9		10		11		12	13		14		15		16		17		18
	H	e	l	l	o		П		é		т		р		о			П		и		т		е		р		!

```
>>> f = open("Test3.txt", "r", encoding = "utf-8")
>>> f.read()
'Hello Πέτρο Питер!'
>>> f.seek(19)
19
>>> f.read(1)
'и'
>>> f.seek(22)
22
>>> f.read(1)
Traceback (most recent call last):UnicodeDecodeError:
'utf-8' codec can't decode byte 0x82 in position 0: invalid start byte
```

Η χρήση της μεθόδου *seek*, σε αρχεία κειμένου με κωδικοποίηση ***UTF*** *χρειάζεται προσοχή* καθώς μπορεί να οδηγήσει σε *απρόβλεπτα αποτελέσματα*

Αποθήκευση αριθμητικών δεδομένων σε αρχεία

Έστω ότι έχουμε να αποθηκεύσουμε 100 τετραψήφιους ακέραιους σε ένα αρχείο

Μέθοδος 1: Τους αποθηκεύουμε σε *αρχείο κειμένου*

Στην περίπτωση αυτή το αρχείο θα περιέχει *500 χαρακτήρες (δεν ξεχνάμε τα \n)* οπότε το μέγεθός του θα είναι *500 bytes* (γιατί;)

Μέθοδος 2: Τους αποθηκεύουμε σε *δυαδικο αρχείο*

Στην περίπτωση αυτή κάθε αριθμός κωδικοποιείται σε *δυαδική μορφή* και αποθηκεύεται σε *δυο bytes* οπότε το μέγεθος του αρχείου θα είναι μόνο *200 bytes* *(εξοικονόμηση μνήμης 60%)*

Πχ. ο δεκαδικός αριθμός `7348` σε δυαδική μορφή είναι `0011100010110100` οπότε

- σε δεκαδική μορφή αποθηκεύεται σε 4+1 bytes

'7'	'3'	'4'	'8'	'/n'

- σε δυαδική μορφή αποθηκεύεται σε 2 Bytes

0011100	10110100

Δυαδικά αρχεία

- Αρχεία που περιέχουν δεδομένα σε *δυαδική (binary) μορφή*, δηλαδή σε μορφή που δεν αποτελείται από κείμενο

- Αποτελούν την *συντριπτική πλειοψηφία* των αρχείων ενός υπολογιστή

- Πολύ πιο *γρήγορα* στην *δημιουργία* και *προσπέλασή* τους επειδή τα δεδομένα γράφονται και διαβάζονται στην "*φυσική*" δυαδική μορφή τους, *χωρίς να απαιτούνται μετατροπές*

- Αν περιέχουν *μόνο* αριθμητικά δεδομένα, είναι σημαντικά *μικρότερα* από τα αντίστοιχα αρχεία κειμένου *(βλ. προηγούμενο παράδειγμα)*

Αποθήκευση κειμένου σε δυαδικό αρχείο

Το κείμενο πρέπει αρχικά να κωδικοποιηθεί στην επιθυμητή κωδικοποίηση *(cp1253, utf-8, utf-16…)* με την μέθοδο .encode() η οποία μετατρέπει μια απλή συμβολοσειρά σε *byte string*(*) σύμφωνα με την επιλεγμένη μορφή κωδικοποίησης

```
>>> bdata_8 = 'Hello Πέτρο!'.encode('utf-8')
>>> bdata_8
b'Hello \xce\xa0\xce\xad\xcf\x84\xcf\x81\xce\xbf!'
```

utf-8: 17 bytes

```
>>> bdata_32 = 'Hello Πέτρο!'.encode('utf-32')
>>> bdata_32
b'\xff\xfe\x00\x00H\x00\x00\x00e\x00\x00\x00l\x00\x00\x00l\x00\x00\x00o\x00\x00\x00
\x00\x00\x00\xa0\x03\x00\x00\xad\x03\x00\x00\xc4\x03\x00\x00\xc1\x03\x00\x00\xbf\x03\x
00\x00!\x00\x00\x00'
```

utf-32: 52 bytes

(*)Στην Python 3, οι *byte strings* δημιουργούνται με την προσθήκη του προθέματος b πριν από την κυριολεκτική δήλωση string, π.χ. *b'some bytes'*. Διαφέρουν από τις κανονικές strings στο ότι κάθε στοιχείο της *byte string* είναι ένα byte (αντί για έναν χαρακτήρα Unicode), και έτσι δεν υποστηρίζουν πολλές από τις συνηθισμένες μεθόδους string, όπως πχ οι μετατροπές πεζών/κεφαλαίων

Αποθήκευση κειμένου σε δυαδικό αρχείο - συνέχεια

Στη συνέχεια το κωδικοποιημένο κείμενο αποθηκεύεται σε ένα δυαδικό αρχείο με την μέθοδο .write()

```
>>> f = open("TextBinary.bin", "wb")
>>> f.write(bdata_8)
17    # αριθμός bytes που γράφτηκαν στο αρχείο  (αν η κωδικοποίηση ήταν πχ. 'utf-32' θα επέστρεφε 52)
```

Όταν αργότερα ανοίξουμε το αρχείο και διαβάσουμε το περιεχόμενο, του θα πρέπει αντίστοιχα να το αποκωδικοποιήσουμε με την μέθοδο .decode()

```
>>> f = open("TextBinary.bin", "rb")
>>> s= f.read()
>>> s
b'Hello \xce\xa0\xce\xad\xcf\x84\xcf\x81\xce\xbf!'
>>> s.decode('utf-8') #αποκωδικοποίηση κειμένου για να μπορέσουμε να το διαβάσουμε πίσω
'Hello Πέτρο!'
```

Ολοκληρωμένο παράδειγμα αποθήκευσης κειμένου σε δυαδικό αρχείο

```
>>> bdata = 'Hello Πέτρο!'.encode('utf-8') #κωδικοποίηση κειμένου για αποθήκευση σε δυαδικό αρχείο
>>> bdata
b'Hello \xce\xa0\xce\xad\xcf\x84\xcf\x81\xce\xbf!'
>>> f = open("TextBinary.bin", "wb")
>>> f.write(bdata)
17    # αριθμός bytes που γράφτηκαν στο αρχείο  (αν η κωδικοποίηση ήταν πχ. 'utf-32' θα επέστρεφε 52)
>>> f.close()
>>> f = open("TextBinary.bin", "rb")
>>> f.read()
b'Hello \xce\xa0\xce\xad\xcf\x84\xcf\x81\xce\xbf!'
>>> f.seek(0) # επιστρέφουμε τον δείκτη αρχείου στην αρχή, για μπορέσουμε να το ξαναδιαβάσουμε
0
>>> s=f.read()
>>> s.decode('utf-8') #αποκωδικοποίηση κειμένου για να μπορέσουμε να το διαβάσουμε "σωστά"
'Hello Πέτρο!'
```

Η πλήρης μορφή της μεθόδου seek() για δυαδικά αρχεία

Για τα δυαδικά αρχεία, η μέθοδος seek() μπορεί προαιρετικά να δεχτεί και ένα δεύτερο όρισμα

`seek(n, whence)` *(whence : από πού;)*

- `n` : ακέραια τιμή που θα χρησιμοποιηθεί για να υπολογιστεί η *νέα θέση του δείκτη*
- *whence*: *(σημείο αναφοράς)* προαιρετική ακέραια τιμή που υποδηλώνει ποια θέση του αρχείου θα αποτελέσει *το σημείο αναφοράς* για τον υπολογισμό της νέας θέσης. Μπορεί να πάρει τις τιμές 0, 1 ή 2 ως εξής:

  - `0` Μετακίνηση n bytes από *αρχή* του αρχείου *(απόλυτη μετακίνηση)*
    Η τιμή του n πρέπει να είναι πάντα *θετική* ή *0*

  - `1` : Μετακίνηση n bytes από την *τρέχουσα θέση* του δείκτη *(σχετική μετακίνηση)*
    Η τιμή του n μπορεί να είναι *θετική* ή *αρνητική*

  - `2` : Μετακίνηση n bytes από το *τέλος* του αρχείου *(σχετική από το τέλος)*
    Η τιμή του n μπορεί να είναι *θετική* ή *αρνητική*. Αν είναι θετική, τότε το αρχείο *επιμηκύνεται* κατά n θέσεις, οι οποίες γεμίζουν με τον χαρακτήρα που έχει κωδικό ASCII 0 *(συμβολίζεται `\0`)*
    Προσοχή: ο χαρακτήρας \0 [κωδικός ASCII 0] είναι διαφορετικός από το ψηφίο "0" [κωδικός ASCII 48]

Όταν η παράμετρος *whence* απουσιάζει, τότε θεωρείται πως έχει την τιμή 0 *(default)*

Παράδειγμα επιμήκυνσης δυαδικού αρχείου με το seek()

```
>>> z= open("Test.bin", "w+b") # Δυαδικό αρχείο, που ανοίγει για γράψιμο, αλλά μπορούμε και να διαβάσουμε
>>> z.write(b"Some data") # Το b μπροστά από την συμβολοσειρά, σημαίνει την μετατροπή της σε binary
9
>>> z.seek(10,2) # Μετακίνηση του δείκτη 10 θέσεις μετά το τέλος του αρχείου
19
>>> z.write(b"Last Data")
9
>>> z.seek(0)
0
>>> z.read()
b'Some data\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00Last Data'
```

Παρατηρήστε τους 10 ενδιάμεσους χαρακτήρες με ASCII κωδικό 0
(εμφανίζονται σε δεκαεξαδική μορφή - δυο bytes)

Μια εξαίρεση: `seek(0,2)` για αρχεία κειμένου

Η μέθοδος seek() με δυο ορίσματα μπορεί *κατ' εξαίρεση* να χρησιμοποιηθεί για *αρχεία κειμένου*, *αποκλειστικά στη μορφή* `seek(0,2)`

Η μορφή αυτή σημαίνει μετάβαση `0` θέσεις από το τέλος του αρχείου, δηλαδή ουσιαστικά ο *δείκτης μεταβαίνει στο τέλος του αρχείου κειμένου*

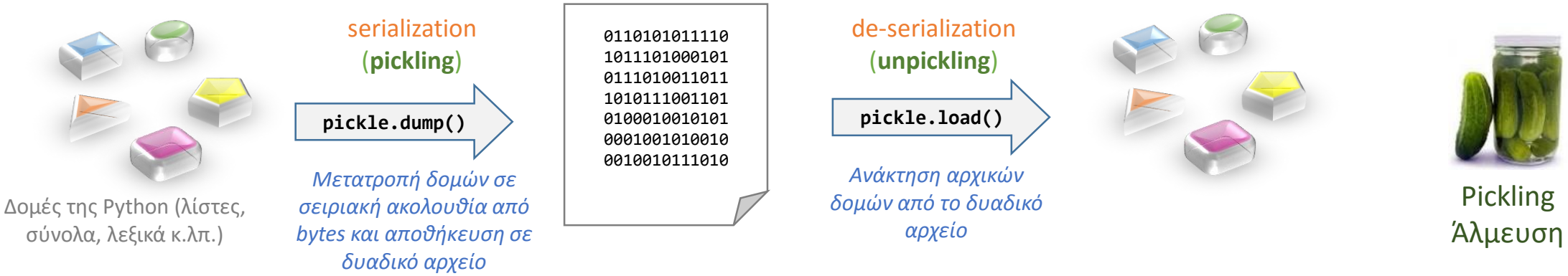
Αν αυτό συνδυαστεί με την μέθοδο `tell()` αποτελεί έναν *έμμεσο τρόπο* να πάρουμε *το μέγεθος ενός αρχείου κειμένου,* δίδοντας διαδοχικά τις εντολές

```
>>> f.seek(0,2) #Μετάβαση στο τελευταίο byte του αρχείου κειμένου
>>> f.tell()    #Θέση του τελευταίου byte του αρχείου κειμένου
                 (υποδηλώνει ουσιαστικά το μέγεθος του αρχείου σε bytes)
```

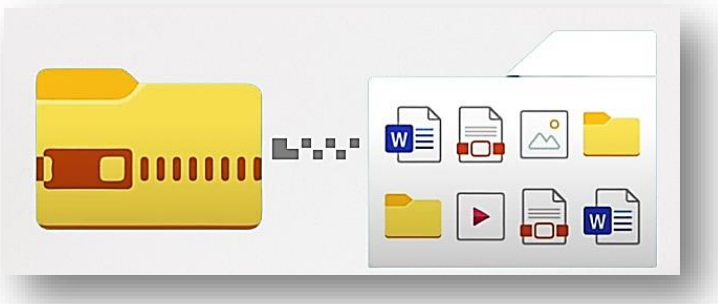
Σειριοποίηση / Από-σειριοποίηση (Pickling / Unpickling)

Γιατί γίνεται / πού χρειάζεται; *Πακετάρισμα δομών* με σκοπό

- είτε την αποθήκευσή τους σε δυαδικά αρχεία *χωρίς να χαθεί η διάρθρωσή τους*,
- είτε την *σειριακή μετάδοσή τους* σε άλλη συσκευή και την *ανασύστασή* τους εκεί



Σκεφτείτε το περίπου σαν το *zip/unzip* των windows, χωρίς την συμπίεση



Σειριοποίηση / Από-σειριοποίηση (Pickling / Unpickling)

Σειριοποίηση ή άλμευση (serialization ή pickling)

Μετατροπή μιας ή περισσότερων *δομών* / αντικειμένων της Python *(λεξικά, σύνολα, λίστες κλπ.. και συνδυασμοί τους)* σε *σειριακή ακολουθία από bytes*

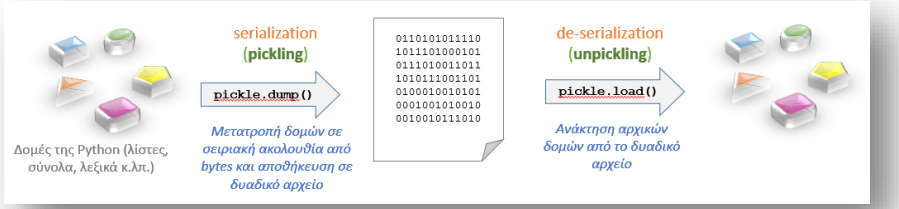
Αποσειριοποίηση ή ανασύσταση (de-serialization ή unpickling)

Ανασύσταση των δεδομένων στην *αρχική τους μορφή* εφαρμόζοντας την *αντίστροφη διαδικασία* πάνω σε μια σειριακή ακολουθία από bytes

Οι μέθοδοι που χρειαζόμαστε για αυτή την διαδικασία βρίσκονται στο δομοστοιχείο `pickle`, το οποίο και πρέπει να εισάγουμε στο πρόγραμμά μας `import pickle`

Το δομοστοιχείο `pickle` περιέχει 4 μεθόδους *(από δυο για κάθε ενέργεια)*

- `dump` και `dumps` για σειριοποίηση
- `load` και `loads` για αποσειριοποίηση



Σειριοποίηση / Από-σειριοποίηση (Pickling / Unpickling)

Άλμευση / ανασύσταση *με ενδιάμεση αποθήκευση σε αρχείο*

- `pickle.dump(<obj>, <file_obj>)`: μετατρέπει ένα *σύνθετο αντικείμενο* *(δομή)* σε *σειριακή μορφή* την οποία στη συνέχεια *αποθηκεύει* σε ένα *δυαδικό αρχείο*
- `<obj> = pickle.load(<file_obj>)`: *ανακτά* και *ανασυνθέτει στην αρχική του μορφή* ένα *αντικείμενο* που *έχει νωρίτερα αποθηκευτεί* *(σε σειριακή μορφή)* *σε* ένα *δυαδικό αρχείο* με την μέθοδο `.dump()`

Σειριοποίηση / αποσειριοποίηση *χωρίς απαραίτητα ενδιάμεση αποθήκευση σε αρχείο*

- `<byte_str> = pickle.dumps(<obj>)` : μετατρέπει ένα *σύνθετο αντικείμενο* *(δομή)* σε *σειριακή μορφή* και την αποθηκεύει ως *byte string* *(συμβολοσειρά από bytes)*
- `<obj> = pickle.loads(<byte_str>)` : *ανασυνθέτει* στην *αρχική του μορφή* ένα αντικείμενο που έχει νωρίτερα μετατραπεί σε *byte string* με την μέθοδο `.dumps`

Παράδειγμα pickling / unpickling

Πρώτος Τρόπος – χρήση των μεθόδων dump και load

```
>>> import pickle
>>> kids = {'Alice':19, 'Ben':22, "Carol":20}

>>> f = open("pickle_file.bin", "wb")
>>> pickle.dump(kids, f)
>>> f.close()

>>> f = open("pickle_file.bin", "rb")
>>> new_kids= pickle.load(f)
>>> new_kids
{'Alice': 19, 'Ben': 22, 'Carol': 20}
```

Παρόλο που γενικά δεν συνηθίζεται, είναι εφικτό να αποθηκεύσουμε και να ανακτήσουμε *περισσότερα από ένα pickled objects* στο ίδιο αρχείο

Παράδειγμα pickling / unpickling

Δεύτερος Τρόπος – χρήση των μεθόδων dumps και loads

```
>>> import pickle
>>> kids = {'Alice':19, 'Ben':22, "Carol":20}
>>> pickled_kids = pickle.dumps(kids) #μετατρέπει το λεξικό σε ένα <byte string>
>>> pickled_kids
b'\x80\x03}q\x00(X\x05\x00\x00\x00Aliceq\x01K\x13X\x03\x00\x00\x00Benq\x02K\x16X\x
05\x00\x00\x00Carolq\x03K\x14u.'
>>> f = open("pickle_file.bin", "wb")
>>> f.write(pickled_kids)
>>> f.close()
>>> f = open("pickle_file.bin", "rb")
>>> tmp = f.read() #το tmp είναι ένα <byte string>
>>> new_kids= pickle.loads(tmp)  #μετατρέπει το <byte string> στην αρχική του μορφή (λεξικό)
>>> new_kids {'Alice': 19, 'Ben': 22, 'Carol': 20}
```

Το b σημαίνει ότι η συμβολοσειρά αυτή ερμηνεύεται όχι σαν κείμενο, αλλά σαν αληλλουχία από bytes (byte string)

Τέλος Διάλεξης

Ερωτήσεις;

Τμήματα αυτής της διάλεξης περιέχουν πληροφορίες από πηγές που είναι ελεύθερες στο διαδίκτυο όπως η Βικιπαίδεια και ανοιχτές σημειώσεις παρεμφερών διαλέξεων