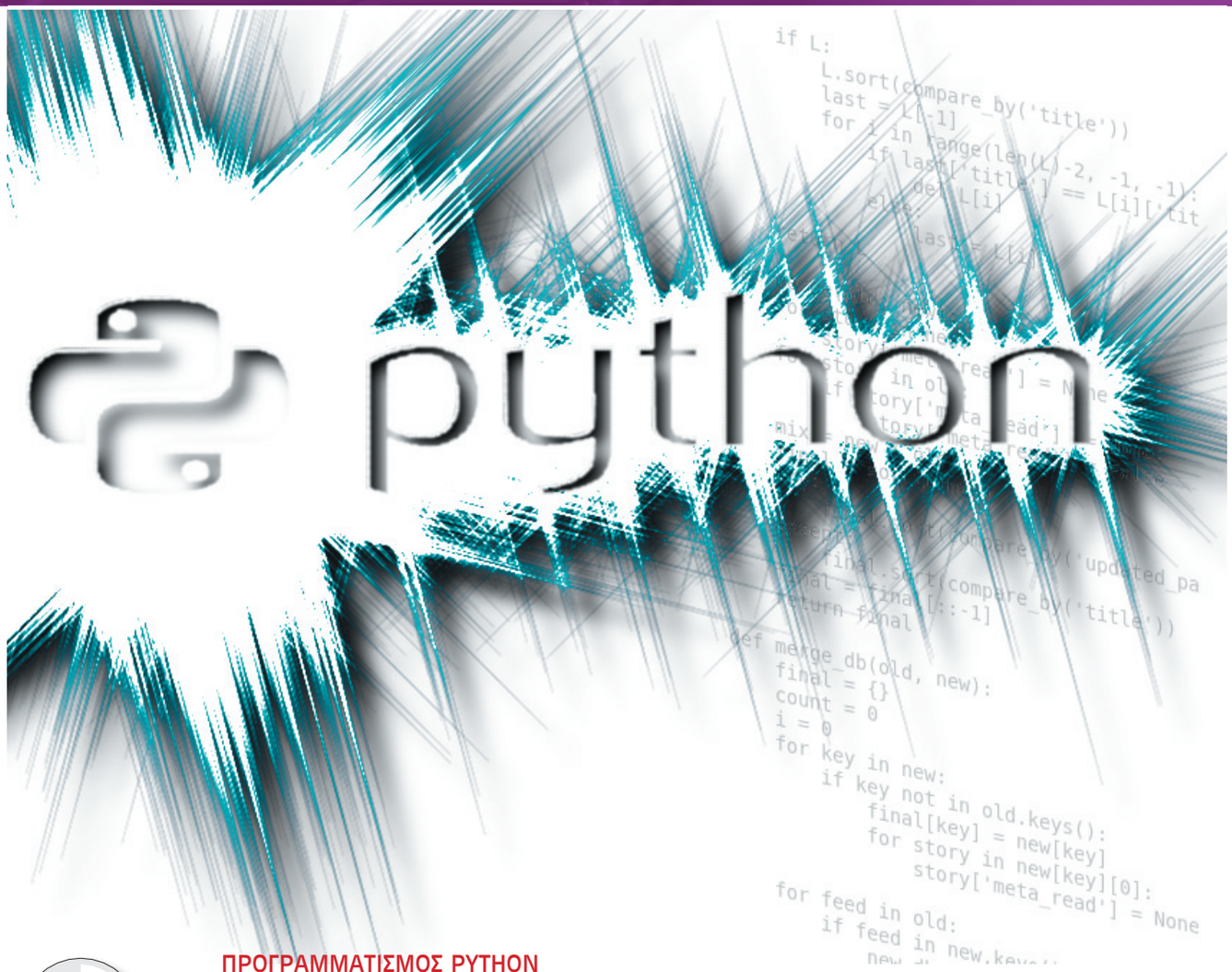
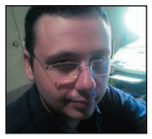


ΣΤΟ WEB
■ Κώδικας
http://ub0.cc/l/0l

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ PYTHON

Διαχείριση Βιβλιογραφίας με Python

ΜΕΡΟΣ 3ο 149 γραμμές κώδικα σε Python είναι αρκετές για να δαμάσουν την BibTeX...



ΤΟΥ ΒΑΣΙΛΗ ΚΑΡΑΚΟΪΔΑ
Ο Βασίλης Καρακόιδας είναι διδακτορικός φοιτητής του τμήματος Διοικητικής Επιστήμης και Τεχνολογίας του Οικ. Πανεπιστημίου Αθηνών. Ανήκει στην Ερευνητική Ομάδα SENSE του εργαστηρίου ISTLab. Εντός και εκτός των ακαδημαϊκών δραστηριοτήτων του τού αρέσει να ταλαιπωρεί τους υπολογιστές του, δοκιμάζοντας διάφορες διανομές του Linux και καταλήγοντας πάντα στο ότι η Debian είναι η καλύτερη, να γράφει scripts και να μαθαίνει καινούργιες τεχνολογίες.

Το τρίτο μέρος αυτής της σειράς οδηγών για Python θα αναλωθεί στο γενικότερο σχεδιασμό ενός ολοκληρωμένου προγράμματος, όχι μόνο σε τεχνικές που αποδίδουν κυρίως όταν γράφουμε μικρά scripts για να αυτοματοποιήσουμε μικρές εργασίες. Το πρόβλημα, λοιπόν, που καλούμαστε να λύσουμε αυτή τη φορά, είναι η διαχείριση βιβλιογραφίας με τη χρήση Python και BibTeX. Ο πηγαίος κώδικας που χρησιμοποιούμε στο tutorial, είναι διαθέσιμος στην ηλεκτρονική διεύθυνση http://www.linuxformat.gr/files/biblio-py.tar.gz.

Τι θα κάνουμε

Εκτός από τις τυπικές εφαρμογές επεξεργασίες κειμένου, σε όλα σχεδόν τα λειτουργικά συστήματα είναι διαθέσιμη η γλώσσα LaTeX, η οποία αποτελεί μία markup γλώσσα, με την οποία μπορεί κάποιος να παράγει πολύ όμορφα κείμενα. Γύρω από τη LaTeX έχουν αναπτυχθεί αρκετά εργαλεία που αυτοματοποιούν διάφορες διαδικασίες. Ενα από αυτά είναι η BibTeX, η οποία αυτοματοποιεί τη συντήρηση βιβλιογραφικών αναφορών, αλλά και την εύκολη χρήση τους μέσα σε κείμενα γραμμένα σε LaTeX. Παρόλα αυτά, η χρήση της BibTeX έχει επικρατήσει ως γενικό format για την συντήρηση της βιβλιογραφίας και χρησιμοποιείται ως "είσοδος" σε πάρα πολλές εφαρμογές, όπως το KBibTex στο KDE, το BibDesk στο Mac OS X (ναι, μου αρέσουν οι Mac) κ.λπ.

Οπως θα έχετε ήδη καταλάβει, η διαχείριση της βιβλιογραφίας αποτελούσε μέχρι πρόσφατα ένα αρκετά μεγάλο πρόβλημα για εμένα που λόγω των ερευνητικών δραστηριοτήτων μου, αποτελεί μία από τις κύριες ενασχολήσεις μου. Πραγματικά δοκίμασα σχεδόν τα πάντα, το KBibTex, το BibDesk, το Αιγαίον (http://aigaion.sourceforge.net/), αλλά δυστυχώς τίποτε δεν με βόλευε 100%, με αποτέλεσμα να καταλήγω μετά από λίγο να συντηρώ τη βιβλιογραφία χειροκίνητα. Το κύριο πρόβλημά μου ήταν το micro-management που θέλουν αυτές οι εφαρμογές, το user interface τους που απαιτούσε να κάνεις συνεχώς κλικ-κλικ με το ποντίκι (του οποίου δεν είμαι οπαδός, οφείλω να ομολογήσω).

Ηθελα εδώ και αρκετό καιρό λοιπόν να κάνω μία υλοποίηση σε Python (λόγω κυρίως των ευκολιών που παρέχει), που να μου δίνει

τη δυνατότητα να κάνω έρευνα αποδοτικά μέσα στη βιβλιογραφία μου, αλλά και να μη χρειάζεται ολόκληρη διαδικασία για να τη συντηρώ. Να είναι, δηλαδή, ανθεκτική σε λάθη, αλλά και να επιτρέπει να επεξεργάζομαι τα αρχεία μου με απλούς κειμενογράφους (vi, kate κ.λπ.), ώστε να κάνω εύκολα διορθώσεις και αλλαγές.

Σχεδιάζοντας τη λύση

Από τις απαιτήσεις που περιέγραψα στην προηγούμενη παράγραφο, είναι σαφές ότι αυτό που θα ήθελα, είναι σίγουρα ένα εργαλείο που να εκτελείται από τη γραμμή εντολών, χωρίς καμία γραφική διεπαφή.

Επίσης, οι λειτουργίες που με ενδιαφέρουν, αφορούν κυρίως στην έρευνα στα πεδία των εγγραφών της BibTeX (κυρίως author και title), αλλά και στη δυνατότητα εξαγωγής συγκεκριμένων εγγραφών με βάση το κλειδί τους. Τέλος, θα ήθελα να βλέπω κάποια απλά στατιστικά, όπως ποια αρχεία χρησιμοποιήθηκαν ως είσοδος και πόσες εγγραφές έχω στη βιβλιοθήκη μου.

Το συντακτικό ενός αρχείου BibTeX

Τα αρχεία που ακολουθούν το format της BibTeX, είναι αρχεία κειμένου, των οποίων οι εγγραφές μοιάζουν με το κείμενο που ακολουθεί:

```
@book{NNIE07,
title = {Semantics with {A}pplications: {A}n
{A}ppetizer},
publisher = {Springer},
year = {2007},
author = {Hane Riis Nielson and Flemming Nielson},
isbn = {2006939147}
}
```

Το @book δηλώνει ότι η εγγραφή αυτή αντιπροσωπεύει ένα βιβλίο. Το NNIE07 είναι το κλειδί της εγγραφής, το οποίο είναι και μοναδικό ανά εγγραφή (δεν είναι αναγκαίο, αλλά αποτελεί καλή πρακτική). Τα δεδομένα της εγγραφής είναι αποθηκευμένα σε ζεύγη κλειδιού-τιμής (key-value pair), με προφανή τρόπο, π.χ., το title αντιπροσωπεύει τον τίτλο του βιβλίου στη συγκεκριμένη περίπτωση.

Οι τύποι δεδομένων που μπορούμε να αποθηκεύουμε, είναι άρθρα (article), βιβλία (book), τεχνικές αναφορές (techreport) κ.ά. Ολα τα διαθέσιμα πρότυπα εγγραφών υπάρχουν στο αρχείο templates/bibtex.bib.

Η δομή του προγράμματος

Το πρόγραμμα Python που καλούμαστε να γράψουμε έχει δύο προφανή μέρη:

- τον αναγνώστη (parser) των αρχείων BibTeX και
- τον κώδικα διαχείρισης

Ο parser θα πρέπει να δέχεται ως είσοδο αρχεία BibTeX και να οργανώνει σε κάποια δομή (μία κλάση, όπως θα δούμε παρακάτω) και μετά να παρέχει τρόπο να προσπελαστούν τα δεδομένα. Ο κώδικας διαχείρισης θα πρέπει να καταλαβαίνει τις παραμέτρους εκτέλεσης και στη συνέχεια να εκτελεί αντίστοιχο κώδικα.

Διαβάζοντας ένα αρχείο BibTeX

Για να κατασκευάσουμε τον parser, χρειάζεται αρχικά να σχεδιάσουμε τη δομή στην οποία θα αποθηκεύουμε κάθε εγγραφή των BibTeX αρχείων. Αυτή είναι μία απλή κλάση, την οποία ονομάσαμε BibtexEntry και ορίζεται από τον ακόλουθο κώδικα:

```
class BibtexEntry:
def __init__(self):
        self.key = ''
        self.data = {}
        self.btype = ''
```

Η κλάση BibtexEntry περιέχει τρία πεδία, το κλειδί (key) κάθε εγγραφής, τον τύπο (btype) της, π.χ., αν είναι book ή article και ένα dictionary με τα υπόλοιπα πεδία της. Αυτά όλα ορίζονται όπως φαίνεται στον παραπάνω κώδικα στον constructor της (__init__(self) για τους μη μελετηρούς).

Ο parser ενός αρχείου BibTeX είναι η πολύ απλή συνάρτηση που ακολουθεί:

```
def parse_bib(bibfile):
bibitems = []
bib_file = open(bibfile, "r")
re_head = re.compile('@([a-zA-Z]+)[ ]*\{[ ]*(.*),')
current = None
for l in bib_file:
        mr = re_head.match(l.strip())
        if mr != None:
                if current == None:
                        current = BibtexEntry()
                else:
                        bibitems.append(current)
                        current = BibtexEntry()
                current.key = mr.group(2).strip()
                current.btype = mr.group(1).strip()
                continue
        try:
                l.index('=')
                kv_data = l.split('=')
                current.data[kv_data[0].strip().lower()]
= kv_data[1].strip()[1:-1]
        except ValueError:
                continue
bib_file.close()
return bibitems
```

Η συνάρτηση parse_bib δέχεται ως όρισμα το όνομα ενός αρχείου BibTeX. Στη συνέχεια, το διαβάζει και δημιουργεί instances της κλάσης BibtexEntry, τα οποία αποθηκεύει στη λίστα bibitems, την οποία επιστρέφει ως αποτέλεσμα. Η ροή που ακολουθεί είναι απλή και προβλεπόμενη. Αφού ανοίξει το αρχείο που δέχεται ως όρισμα και δημιουργήσει το bib_file (μεταβλητή που παρέχει πρόσβαση στα δεδομένα του αρχείου), τότε το διαβάζει γραμμή-γραμμή και προσπαθεί να αναγνωρίσει τις εγγραφές BibTeX. Η λογική του είναι πολύ απλή και θα τη δούμε βήμα-βήμα. Για κάθε γραμμή του αρχείου, λοιπόν:

1 Εξετάζει αρχικά αν πρόκειται για νέα εγγραφή. Αυτό το καταλαβαίνει μέσω της επικεφαλίδας. Αν, δηλαδή, συναντήσει ένα String της μορφής @book{FOO, τότε έχει συναντήσει καινούργια εγγραφή στο αρχείο. Αυτό το ανιχνεύει μέσω της κανονικής έκφρασης '@([a-zA-Z]+)[ ]*\{[ ]*(.*),'.

2 Αν αυτό ισχύει, τότε δημιουργεί ένα καινούργιο στιγμιότυπο της κλάσης BibtexEntry και το αποθηκεύει στη μεταβλητή current. Αν αυτή περιέχει ήδη κάποια προηγούμενη εγγραφή, τότε, πριν την αντικαταστήσει, την προσθέτει στη λίστα με τα αποτελέσματα.

3 Στην περίπτωση που δεν είναι επικεφαλίδα, τότε πρέπει να είναι ζεύγος κλειδιού-τιμής. Για να εξετάσουμε αν είναι, απλώς ψάχνουμε να δούμε την ύπαρξη του χαρακτήρα =. Αν υπάρχει, τότε ξεχωρίζουμε την τιμή και το κλειδί και τα αποθηκεύουμε στη μεταβλητή current, η οποία περιέχει την εγγραφή που επεξεργαζόμαστε εκείνη τη στιγμή. Αν ο χαρακτήρας δεν υπάρχει, η συνάρτηση index() πετάει εξαίρεση τύπου ValueError, οπότε τη γραμμή δεν την επεξεργαζόμαστε περαιτέρω.

Με την εντολή bib_file.close() κλείνουμε το αρχείο. Η μέθοδος που χρησιμοποιούμε για την ανάγνωση του BibTeX αρχείου, είναι

ΔΕΙΤΕ

BibTeX
http://en.wikipedia.org/wiki/BibTeX
LaTeX
http://en.wikipedia.org/wiki/LaTeX
os.walk()
http://docs.python.org/lib/os-file-dir.html
biblio-py
http://gaijin.dmst.aueb.gr/~bkarak/programs/biblio-py
KbibTeX
http://www.unix-ag.uni-kl.de/~fischer/kbibtex/
Aigaion
http://www.aigaion.nl/

ΤΙ ΚΑΛΥΠΤΕΙ ΤΟ TUTORIAL;

Αυτό το τρίτο μέρος της σειράς περιέχει παραδείγματα χρήσης της συνάρτησης getattr(). Πρόκειται για μία από τις συναρτήσεις που καταδεικνύουν το δυναμικό χαρακτήρα της Python. Επίσης, κατασκευάζεται ένας απλός parser για αρχεία BibTeX. Οι βιβλιοθήκες που χρησιμοποιούνται, είναι οι:

- os - λειτουργίες στο σύστημα (αρχεία κ.λπ.)
- re - Κανονικές εκφράσεις

αρκετά ανεκτική σε λάθη και δεν ελέγχει τη δομή των εγγραφών - απλώς προσπαθεί με έναν απλό τρόπο να φορτώσει τις εγγραφές. Αυτό είναι βολικό γιατί, αν υπάρχει λάθος σε κάποια εγγραφή, αυτό δεν θα εμποδίσει την εκτέλεση της λειτουργίας.

Το μόνο μειονέκτημα της μεθόδου που χρησιμοποιούμε για ανάγνωση, είναι η επεξεργασία ανά γραμμή που κάνουμε. Ο παρακάτω κώδικας,

```
for l in bib_file:
[...]
```

τυπώνει το αρχείο ανα γραμμή, οπότε αν μία εγγραφή είναι σε μία γραμμή, τότε θα έχουμε πρόβλημα με αυτή την εγγραφή.

Εκτύπωση της BibtexEntry

Ενα σημαντικό θέμα είναι η εκτύπωση σε ορθή μορφή μίας εγγραφής, η οποία είναι αποθηκευμένη σε ένα instance της κλάσης. Ο πιο προφανής τρόπος θα ήταν με τη δημιουργία μίας συνάρτησης που να το κάνει αυτό. Δηλαδή, ως μέθοδο της κλάσης BibtexEntry, μπορούμε να ορίσουμε μία μέθοδο tostring(), η οποία να επιστρέφει ένα string με τη λεκτική αναπαράσταση της εγγραφής BibTeX. Πρέπει, λοιπόν, στην κλάση να προσθέσουμε τον παρακάτω κώδικα:

```
class BibtexEntry:
def __init__(self):
        self.key = ''
        self.data = {}
        self.btype = ''
        def tostring(self):
        result = StringIO()
        result.write("@%s{%s,\n" % ( self.btype, self.key
))
        for k, v in self.data.iteritems():
                result.write("\t%s = {%s},\n" % ( k, v ))
        result.write('}\n')
        return result.getvalue()
```

Σε αυτήν την περίπτωση, όμως, κάθε φορά που θα θέλουμε να εκτυπώσουμε την κλάση στην οθόνη μας, θα πρέπει να γράφουμε:

```
bibentry = BibtexEntry()
print bibentry.tostring()
```

Δεν είναι κάπως κουραστικό αυτό; Φανταστείτε να έπρεπε να το κάνετε αυτό κάθε φορά που θέλατε να τυπώσετε έναν αριθμό. Κάθε αριθμός είναι object όπως η κλάση που έχουμε ορίσει. Για του λόγου το αληθές:

```
>>> i = 10
>>> dir(i)
['__abs__', '__add__', '__and__', ... ,'__truediv__', '__xor__']
```

Γιατί, όμως, αν κάνουμε print έναν αριθμό, δεν χρειάζεται να καλέσουμε την αντίστοιχη tostring();

Ολα τα objects στην Python έχουν μία συνάρτηση που καλείται και επιστρέφει τη λεκτική αναπαράσταση ενός αντικειμένου. Αυτή η συνάρτηση είναι η __str__() και υπάρχει σε κάθε αντικείμενο. Για παράδειγμα:

```
>>> class Foo:
... def __init__(self):
... self.__i = 1
...
>>> f = Foo()
>>> print f
<__main__.Foo instance at 0x6b3f0>
>>> class Foo:
... def __str__(self):
... return "it's me, Foo!"
...
>>> f = Foo()
>>> print f
it's me, Foo!
```

Οπως βλέπουμε στον παραπάνω κώδικα, ορίζοντας την κλάση Foo, όταν προσπαθούμε να την τυπώσουμε μέσω της print, τότε εμφανίζεται ο τύπος της f, καθώς και η διεύθυνσή της στη μνήμη. Αν ορίσουμε ξανά στη Foo τη μέθοδο __str__(), τότε τυπώνει το μήνυμα που εμείς θέσαμε.

Με αυτό τον τρόπο, θα αντιμετωπίσουμε και το πρόβλημα στην περίπτωσή μας. Η κλάση BibtexEntry αποκτά την εξής μορφή:

```
class BibtexEntry:
def __init__(self):
        self.key = ''
        self.data = {}
        self.btype = ''
def __str__(self):
        result = StringIO()
        result.write("@%s{%s,\n" % ( self.btype, self.key
 ))
        for k, v in self.data.iteritems():
                result.write("\t%s = {%s},\n" % ( k, v ))
        result.write('}\n')
        return result.getvalue()
```

Παράμετρος εκτέλεσης

Εφόσον έχουμε καταφέρει να διαβάσουμε τα αρχεία με τις εγγραφές, αυτό που να απομένει, είναι να υλοποιήσουμε τη διεπαφή της γραμμής εντολών και να υλοποιήσουμε τη λειτουργικότητα των παραμέτρων εκτέλεσης.

Οι βασικές παράμετροι εκτέλεσης είναι οι παρακάτω (απλώς εκτελούμε ./biblio.py):

```
$ python biblio.py
bibliography utility ver. 0.1.0
usage: biblio.py <directive> <arguments>
search - search title, author tags for specific entries
count - Count all bibtex entries
export - Extracts the selected keys
help - Prints the online help
```

Η δομή των παραμέτρων ακολουθεί τη δομή αρκετών γνωστών εφαρμογών, όπως το subversion. Ως πρώτη παράμετρος πάντα δηλώνεται η εργασία που θέλουμε να εκτελεστεί και ακολουθούν οι παράμετροι για τη συγκεκριμένη εργασία, π.χ.:

```
$ python biblio.py search Semantics
$ python biblio.py export BBSM01 NNIE07
```

Πώς, όμως, αναγνωρίζονται οι παράμετροι; Υπάρχουν πραγματικά αρκετές βιβλιοθήκες για να γίνει αυτό, με κύρια την getOpt της GNU. Να πω την αλήθεια, εγώ την αντιπαθώ και σκαρφίστηκα έναν "δικό μου" τρόπο να κάνω τη δουλειά μου σε αυτή την αρκετά περιορισμένη - οφείλω να ομολογήσω - περίπτωση. Χρησιμοποίησα τη συνάρτηση getattr() της Python. Ο σκοπός της getatrr() είναι απλός: προσπαθεί να εκτελέσει μία συνάρτηση από ένα object, του οποίου το όνομα δίνεται ως string. Για παράδειγμα:

```
>>> s = 'foo'
>>> getattr(s, "upper")()
'FOO'
```

Οπως βλέπουμε, για το string "s" καλούμε τη συνάρτηση "upper". Αυτό μας επιστρέφει τη μέθοδο s.upper(), την οποία καλούμε. Απίστευτο; Για να δούμε τώρα πώς χρησιμοποιήσαμε αυτήν τη συνάρτηση για την αναγνώριση των παραμέτρων εκτέλεσης. Ορίσαμε αρχικά την κλάση CmdDispatcher ως αυτή που θα περιέχει συναρτήσεις για όλες τις παραμέτρους:

```
class CmdDispatcher:
def __init__(self, args):
```

```
        [ ... ]
        def doSearch(self):
        [ ... ]
def doCount(self):
        [ ... ]
def doExport(self):
        [ ... ]
def doHelp(self):
        [ ... ]
def main():
# initialise the dispatcher and execute
cdisp = CmdDispatcher(sys.argv[2:])
try:
        getattr(cdisp,"do" + sys.argv[1].title())()
except (AttributeError, IndexError):
        cdisp.doHelp()
```

Η συνάρτηση main() εκτελείται πρώτη κατά την εκτέλεση του προγράμματος και κάνει τα εξής:

1 Δημιουργεί ένα instance της κλάσης CmdDispatcher και του περνάει ως όρισμα όλες τις παραμέτρους εκτέλεσης εκτός από τις δύο πρώτες, το όνομα του εκτελεσίμου και την εργασία που θα εκτελεσθεί, π.χ., search, count κ.λπ.

2 Αφού αρχικοποιηθεί η κλάση, κατασκευάζεται το όνομα της μεθόδου και καλείται.

3 Αν δεν υπάρχει, τότε δημιουργείται εξαίρεση και καλείται η συνάρτηση που δείχνει την επεξήγηση της λειτουργικότητας της γραμμής εντολών.

Ο κώδικας που απαρτίζει κάθε συνάρτηση, είναι απλός κώδικας Python και δεν χρειάζεται ιδιαίτερη μνεία, καθώς παρόμοιο κώδικα έχουμε αναλύσει στα προηγούμενα τεύχη.

Το package bibtex

Αν έχετε κοιτάξει ήδη τον κώδικα που συνοδεύει το παρόν άρθρο, ίσως να παρατηρήσατε ότι το αρχείο Python που περιέχει τον parser της BibTeX, βρίσκεται μέσα στον κατάλογο bibtex. Επίσης, εκεί υπάρχει ένα μυστηριώδες αρχείο Python που ονομάζεται __init__.py, το οποίο είναι κενό περιεχομένου.

Ακολουθώντας αυτή τη δομή, δημιουργήσαμε ένα package με τον parser και έτσι δίνουμε τη δυνατότητα να χρησιμοποιηθεί ως αυτοτελής βιβλιοθήκη και από άλλα προγράμματα Python που μπορεί να αναπτύξουμε στο μέλλον. Για να εισάγουμε τη βιβλιοθήκη στο πρόγραμμά μας, απλώς χρησιμοποιούμε τη γνωστή πλέον εντολή import με τον ακόλουθο τρόπο:

```
from bibtex import bibparse
bibparse.parse_bib(f)
```

Η δεύτερη εντολή είναι παράδειγμα κλήσης της συνάρτησης parse_bib() που είχαμε αναλύσει νωρίτερα.

Το __init__.py είναι αρχείο που περιέχει κάθε κατάλογος ενός package και περιέχει, αν απαιτείται, κώδικα Python που εκτελείται κατά την αρχικοποίηση του package. Αν αυτό το αρχείο παραλειφθεί, τότε η Python δεν αναγνωρίζει ως package το συγκεκριμένο κατάλογο.

Πού αποθηκεύω τα αρχεία BibTeX;

Τα αρχεία BibTeX τα αποθηκεύουμε στον κατάλογο ./repository/. Αν σε αυτό περιέχονται και άλλοι κατάλογοι, θα εισέλθει σε αυτούς και θα αναγνώσει όλα τα αρχεία BibTeX που θα βρει (.bib). Ενας εύκολος τρόπος για να προσπελάσουμε έναν κατάλογο και τους υποκαταλόγους του, είναι η συνάρτηση walk() του package 'os'. Η συνάρτηση που υλοποιεί αυτή τη λειτουργικότητα, είναι η scan_dirs():

```
def scan_dirs(path):
res = []
for root, dirs, files in os.walk(path):
        if '.svn' in dirs:
                dirs.remove('.svn')
        for d in dirs:
                fd = os.path.join(root, d)
                if os.path.islink(fd):
                        res_two = scan_dirs(fd)
                        res.extend(res_two)
        for f in files:
                if f.endswith('.bib'):
res.append(os.path.join(root, f))
return res
```

Κατά την υλοποίησή της, ανακάλυψα ότι η os.walk() δεν ακολουθεί symbolic links. Αυτό λύνεται αν απομονώσουμε τα symbolic links μέσω της συνάρτησης os.path.islink() πριν καλέσουμε την os.walk(). Αλλά, ας δούμε αυτή τη συνάρτηση αναλυτικά.

1 Η συνάρτηση δέχεται ως όρισμα τον κατάλογο που καλείται να ανιχνεύσει. Αυτό γίνεται αναδρομικά, δηλαδή, αν ο αρχικός κατάλογος περιέχει και άλλους καταλόγους, τότε ανιχνεύονται και αυτοί.

2 Καλώντας την os.walk() ξεκινάμε τη διαδικασία. Η μεταβλητή res είναι η λίστα με τα αποτελέσματα, στην προκειμένη περίπτωση τα αρχεία της BibTeX.

3 Επειδή χρησιμοποιώ Subversion, θέλω τα αρχεία που βρίσκονται στους καταλόγους του Subversion, να μην ανιχνεύονται (.svn). Οπότε αν τους συναντήσω σε κάποια επανάληψη του for, τους αφαιρώ από τη λίστα με τους καταλόγους (dirs). Εσείς αν θέλετε, τροποποιείτε αυτό το κομμάτι κώδικα.

4 Για όλους τους καταλόγους, ελέγχουμε αν είναι symbolic links. Αν βρούμε κάποιον που είναι link, τότε καλούμε αναδρομικά τη scan_dirs() με όρισμα αυτόν και, αφού εκτελεστεί, προσθέτουμε τα αποτελέσματα στην αρχική λίστα αποτελεσμάτων μέσω της συνάρτησης extend, η οποία προσθέτει στην περίπτωσή μας, τα αποτελέσματα της λίστας res_two στην res.

5 Τέλος, για όλα τα αρχεία ελέγχουμε αν αυτά έχουν την κατάληξη .bib και τα προσθέτουμε στη λίστα αποτελεσμάτων.

Ο λόγος που η os.walk() δεν ελέγχει symbolic links είναι η αποφυγή αναδρομών μεταξύ συνδέσμων που κάνουν κύκλο. Το αστείο είναι ότι το πρόγραμμα που έχουμε γράψει, είναι ευπαθές σε τέτοια φαινόμενα, αλλά είναι σωστό από τη σκοπιά των προγραμματιστών των βιβλιοθηκών να μην επιτρέπουν τέτοια φαινόμενα και να αφήνουν τους χρήστες των βιβλιοθηκών να τα αντιμετωπίζουν όπως νομίζουν.

Εν κατακλείδι

Μετά από δύο ευχάριστες ώρες προγραμματισμού σε Python, ήμουν σε θέση να πω ότι είχα αντιμετωπίσει το πρόβλημα σε ένα μεγάλο βαθμό. Μπορούσα με απλό τρόπο να βάζω αρχεία BibTeX που είχα στον κατάλογο repository/ και να χρησιμοποιώ το περιεχόμενό τους. Και αυτό μόνο με 149 γραμμές Python. Για του λόγου το αληθές:

```
$ find . ¦ grep ".py$" ¦ xargs wc -l
91 ./biblio.py
0 ./bibtex/__init__.py
58 ./bibtex/bibparse.py
149 total
```

Το πρόγραμμα, όμως, έχει κάποια προβλήματα, τα οποία καλό θα ήταν να αντιμετωπίσετε. Τα αφήνω ως... άσκηση:

- Πώς θα μπορούσαμε να περνάμε ως παράμετρο τον κατάλογο που περιέχει τα BibTeX αρχεία;
- Τι θα έπρεπε να κάνουμε, ώστε το package bibtex να μπορούμε να το κάνουμε install ως μία βιβλιοθήκη της Python, ώστε να μπορούν να επωφεληθούν όλα τα προγράμματα από την ύπαρξή της;
- Πώς θα μπορούσαμε να τροποποιήσουμε τον parser των αρχείων BibTeX, ώστε να μην έχει τα προβλήματα που αναφέραμε;

■

ΤΗΝ ΕΠΟΜΕΝΗ ΦΟΡΑ

Στο τελευταίο μέρος αυτής της σειράς θα δούμε μία ολοκληρωμένη εφαρμογή με γραφικό περιβάλλον.