ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επισκόπηση των γλωσσών προγραμματισμού για εξόρυξη δεδομένων και ανάπτυξη τέτοιου προγράμματος για τον ιστοχώρο του χρυσού οδηγού.

Ποταμίτης Ιωάννης

Αρ. Μητρώου: **04/2520**

Επιβλέπων καθηγητής:

Δρ. Μηνάς Δασυγένης

ΠΡΟΛΟΓΟΣ

Ο Tim Berners-Lee είχε φανταστεί ένα δίκτυο δεδομένων που συνδέονται μεταξύ τους και είναι ανοιχτά προς όλους για να τα μοιραστούν και να τα χρησιμοποιήσουν. Αυτή τη λογική ακολουθεί και η τεχνική της εξόρυξης δεδομένων από το διαδίκτυο. Είναι απλά μια πράξη κατά την οποία παίρνουμε δεδομένα από μια πηγή, τα αναλύουμε και τα χρησιμοποιούμε σε δικές μας εφαρμογές.

Σήμερα όμως πολλοί ιστότοποι παρέχουν RSS feeds ή δημόσια APIs, με τα οποία μπορούμε να έχουμε πρόσβαση στα δεδομένα τους. Για ποιό λόγο λοιπόν πρέπει να χρησιμοποιούμε τεχνικές εξόρυξης δεδομένων; Η χρήση αυτών των τεχνικών είναι σημαντική λόγω του ότι ελάχιστοι ιστότοποι προσφέρουν APIs μέχρι στιγμής αλλά και λόγω της εμφάνισης των microformats, των τεχνολογιών του Σημασιολογικού Ιστού, και του έργου της κοινότητας W3C Linking Open Data και του κινήματος Ανοιχτών Δεδομένων (Open Data Movement).

Οι τεχνικές που περιγράφονται σε αυτή τη πτυχιακή εργασία είναι χρήσιμες για τη πρόσβαση σε κάθε είδους πληροφορία που βρίσκεται στο διαδίκτυο.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία διαπραγματεύεται τα ζητήματα εξαγωγής δεδομένων από το διαδίκτυο, των τεχνικών που χρησιμοποιούνται, καθώς και των τρόπων υλοποίησης των τεχνικών αυτών. Γίνεται μια περιγραφή των τρόπων με τους οποίους μπορούμε να εξάγουμε δεδομένα από το διαδίκτυο, χρησιμοποιώντας τις πιο γνωστές γλώσσες προγραμματισμού. Αυτή η περιγραφή έχει ως σκοπό την αναζήτηση του βέλτιστου τρόπου εξαγωγής δεδομένων. Τέλος, αναπτύσσεται μια εφαρμογή εξόρυξης δεδομένων από το διαδικτυακό τόπο του χρυσού οδηγού του ΟΤΕ, για την εξειδίκευση του αναγνώστη/σπουδαστή πάνω στο θέμα των τεχνικών εξόρυξης δεδομένων.

ABSTRACT

This thesis deals with the issues of web scraping, the techniques that are used and the visualization of these techniques. Additionally the ways in which we can export data with the most popular programming language are described. This description is intended to find the best way to export data from the web. Finally, we created a data mining application that parses and scrapes data from OTE yellow pages for the specificity of the reader/students on the topic of web scraping techniques.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΙΣΑΓΩΓΗ

Η παρούσα πτυχιακή εργασία θα μπορούσαμε να πούμε ότι αποτελείται από τρία μέρη. Το πρώτο και το δεύτερο μέρος αποτελούν μια βιβλιογραφική έρευνα, ενώ στο τρίτο τμήμα αναπτύσσεται μια ρεαλιστική εφαρμογή για την εξόρυξη δεδομένων από τον ιστοχώρο του χρυσού οδηγού.

Συγκεκριμένα το πρώτο τμήμα αποτελείται από τρία κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή στις έννοιες της εξόρυξης δεδομένων από το διαδίκτυο. Στο δεύτερο κεφάλαιο περιγράφονται τα κύρια χαρακτηριστικά που διέπουν τα προγράμματα εξόρυξης δεδομένων. Τέλος, στο τρίτο κεφάλαιο γίνεται μια περιγραφή των κύριων τεχνολογιών που χρησιμοποιούνται κατά τη διαδικασία δημιουργίας τέτοιων προγραμμάτων.

Το δεύτερο μέρος της πτυχιακής αποτελείται από πέντε κεφάλαια, στα οποία γίνεται περιγραφή τον τρόπων με τους οποίους μπορούμε να εξάγουμε δεδομένα από το διαδίκτυο χρησιμοποιώντας τις ποιο γνωστές γλώσσες προγραμματισμού. Οι γλώσσες που περιγράφονται είναι η Ruby, η Perl, η Python, η PHP, η Java και μια βιβλιοθήκη του περιβάλλοντος .NET.

Στο τρίτο μέρος γίνεται η αναλυτική περιγραφή της πορείας δημιουργίας μιας εφαρμογής, η οποία έχει σαν σκοπό την εξαγωγή δεδομένων από τη διαδικτυακή τοποθεσία του χρυσού οδηγού του ΟΤΕ. Το πρόγραμμα αυτό στέλνει ειδικές διαμορφωμένες αιτήσεις στον ιστοχώρο και λαμβάνει απαντήσεις από αυτόν. Οι απαντήσεις αναλύονται και τίθενται υπό επεξεργασία, προκειμένου να βρεθούν τα χρήσιμα για τον χρήστη δεδομένα και να τοποθετηθούν σε ένα αρχείο. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την ανάπτυξη αυτής της εφαρμογής είναι η Python.

ΚΕΦΑΛΑΙΟ 1
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΟ ΔΙΑΔΙΚΤΥΟ: ΘΕΜΕΛΙΩΔΕΙΣ ΕΝΝΟΙΕΣ

Εισαγωγή
Οι περισσότερες ιστοσελίδες στο διαδίκτυο περιέχουν πληθώρα πληροφοριών σε μορφή κειμένου. Χρησιμοποιώντας τεχνικές εξόρυξης δεδομένων μπορούμε να συγκεντρώσουμε πληροφορίες και δεδομένα από πολλές διαφορετικές ιστοσελίδες και διαδικτυακές τοποθεσίες. Σε αυτό το κεφάλαιο θα δούμε τις βασικές έννοιες και τεχνικές για τη συλλογή όλων αυτών των δεδομένων από τις ιστοσελίδες, ενώ στο επόμενο κεφάλαιο θα μιλήσουμε για τα κύρια χαρακτηριστικά των προγραμμάτων που συλλέγουν δεδομένα από το διαδίκτυο.

1.1 Τι ονομάζουμε εξόρυξη δεδομένων από το διαδίκτυο;
Η εξόρυξη δεδομένων (στα αγγλικά το συναντάμε ως *web scraping*, *screen scraping, web harvesting* ή *web data extraction*) είναι μια διαδικασία άντλησης πληροφοριών από ιστοσελίδες του παγκόσμιου ιστού. Η διαδικασία αυτή περιλαμβάνει την ανάκτηση ενός ημι-δομημένου εγγράφου από το διαδίκτυο, συνήθως μιας ιστοσελίδας σε μια γλώσσα σήμανσης όπως η HTML, ή η XHTML, και την ανάλυση του εγγράφου, προκειμένου να εξαχθούν συγκεκριμένα στοιχεία από αυτό.

Με τα προγράμματα που χρησιμοποιούν αυτή τη τεχνική μπορούμε να συνδεθούμε και να ζητήσουμε μια ιστοσελίδα, όπως ακριβώς θα έκανε και ένας περιηγητής ιστού. Αφού ο διακομιστής μας στείλει την ιστοσελίδα εμείς στη συνέχεια μπορούμε να την επεξεργαστούμε και να εξάγουμε δεδομένα από αυτήν. Η εξόρυξη των δεδομένων μπορεί να γίνει από έναν μόνο ιστότοπο ή από χιλιάδες διαφορετικές ιστοσελίδες.

Η διαδικασία της εξόρυξης δεδομένων δίνει μεγάλο βάρος στη μετατροπή του ημι-δομημένου περιεχομένου της ιστοσελίδας σε μια δομημένη μορφή. Στη συνέχεια, αυτά τα δεδομένα μπορούν να αποθηκευτούν και να αναλυθούν σε μια τοπική βάση ή ένα υπολογιστικό φύλλο.

1.2 Τεχνικές για την εξόρυξη δεδομένων από το διαδίκτυο
Η εξόρυξη δεδομένων είναι μια διαδικασία αυτόματης συλλογής πληροφοριών από το διαδίκτυο. Σίγουρα υπάρχουν διάφορα επίπεδα αυτοματισμού που οι υφιστάμενες τεχνολογίες μπορούν να προσφέρουν:

- **Copy-Paste**: Μερικές φορές ακόμη και το καλύτερο πρόγραμμα εξόρυξης δεδομένων δεν μπορεί να αντικαταστήσει τη χειροκίνητη αντιγραφή και επικόλληση. Σε μερικές περιπτώσεις αυτό μπορεί να είναι και η μοναδική λύση, όταν ο ιστοχώρος από τον οποίο θέλουμε να εξάγουμε τα δεδομένα εμποδίζει τέτοιου είδους προγράμματα.

- **Κανονικές Εκφράσεις** (Regular Expressions): Μια απλή, όμως ισχυρή προσέγγιση για την εξαγωγή πληροφοριών από ιστοσελίδες, μπορεί να βασίζεται στην εντολή grep του UNIX, ή στις κανονικές εκφράσεις που μπορούμε να χρησιμοποιήσουμε μέσω γλωσσών προγραμματισμού (π.χ. Perl ή Python).
- **HTTP Προγραμματισμός**: Στατικές και δυναμικές ιστοσελίδες μπορούν να ανακτηθούν μέσω HTTP αιτήσεων στον απομακρυσμένο διακομιστή χρησιμοποιώντας socket προγραμματισμό.
- **DOM Parsing**: Με την ενσωμάτωση της λειτουργικότητας ενός προγράμματος περιήγησης, όπως ο Internet Explorer ή ο Mozilla Firefox, τα προγράμματα μπορούν να ανακτήσουν το δυναμικό περιεχόμενο που παράγεται από τα scripts στη πλευρά του πελάτη (client) και να έχουν πρόσβαση στα δεδομένα της σελίδας μέσα από το DOM (Document Object Model) δέντρο που παράγεται από τον HTML κώδικα.
- **HTML parsers**: Μερικές γλώσσες ερωτημάτων, όπως η γλώσσα ερωτημάτων της XML (XQL) και η γλώσσα ερωτημάτων υπερ-κειμένου (HTQL), μπορούν να χρησιμοποιηθούν για να αναλύσουν και να ανακτήσουν το περιεχόμενο σελίδων HTML.
- **Χρήση λογισμικού**: Υπάρχουν πολλά εργαλεία λογισμικού διαθέσιμα, τα οποία μπορούν να χρησιμοποιηθούν για να προσομοιώσουν μια εξόρυξη δεδομένων από το διαδίκτυο. Αυτά τα προγράμματα λογισμικού μπορεί να παρέχουν ένα γραφικό περιβάλλον και μια βάση δεδομένων, στην οποία μπορούν να αποθηκεύονται τα δεδομένα που ανακτούμε από τον ιστό.

1.3 Λόγοι χρήσης των τεχνικών εξόρυξης δεδομένων

Αν και όλο περισσότεροι ιστότοποι προσφέρουν τα δεδομένα τους μέσω της χρήσης υπηρεσιών διαδικτύου (web services), η απουσία μιας τέτοιας υπηρεσίας που να είναι προσαρμοσμένη στις μηχανές και να προσφέρει όλα τα δεδομένα του ιστότοπου σε αυτές είναι μια πολύ συνηθισμένη κατάσταση. Σε αυτές τις περιπτώσεις είναι απαραίτητη η χρήση τεχνικών εξόρυξη, για την αυτοματοποίηση της εξαγωγής των δεδομένων που διαθέτει ο εκάστοτε ιστότοπος.

Μια άλλη χρήση των τεχνικών εξόρυξης δεδομένων είναι για τη δημιουργία αυτόματων πρακτόρων (automated agents), γνωστών και ως ανιχνευτές ιστού (web crawlers, web spiders ή web robots). Ένας ανιχνευτής ιστού είναι ένα πρόγραμμα υπολογιστή που περιηγείται στο Παγκόσμιο Ιστό, αναζητώντας και αποθηκεύοντας πληροφορίες για την μετέπειτα επεξεργασία τους.

Τέλος, η χρήση των τεχνικών εξόρυξης δεδομένων μπορεί να χρησιμοποιηθεί και για τη δοκιμή διαδικτυακών εφαρμογών. Η δυνατότητα της προσομοίωσης της λειτουργίας ενός φυλλομετρητή μας επιτρέπει να ενεργήσουμε ως πελάτης και να διαπιστώσουμε εάν η εφαρμογή θα επιστρέψει τα αναμενόμενα αποτελέσματα.

1.4 Κατάλληλη χρήση των τεχνικών εξόρυξης δεδομένων.

Μερικοί πάροχοι δεδομένων προσφέρουν απλά έναν δικτυακό τόπο, ενώ άλλοι προσφέρουν κάποιο API (Application Programming Interface) που δεν προσφέρει τα δεδομένα σε μια ευνοϊκή μορφή, ή δεν προσφέρει καθόλου δεδομένα. Σε αυτές τις περιπτώσεις φαίνεται να μην υπάρχει εναλλακτική λύση, εκτός από τη χρήση τεχνικών εξόρυξης δεδομένων.

Βέβαια σε αντίθετη περίπτωση, δηλαδή εάν έχουμε τη δυνατότητα πρόσβασης στα δεδομένα ενός ιστότοπου μέσω ενός API που αυτός προσφέρει, τότε η καλύτερη επιλογή είναι να κάνουμε χρήση αυτού του API. Όχι μόνο γιατί μπορεί να είναι πιο εύκολο στη χρήση, αλλά και γιατί η εξόρυξη δεδομένων με τη χρήση γλωσσών προγραμματισμού υποκρύπτει έναν μεγάλο κίνδυνο. Ο κίνδυνος αυτός είναι ότι, σε περίπτωση έστω και μικρής αλλαγής του κώδικα των σελίδων του ιστότοπου, η εφαρμογή μας υπάρχει περίπτωση να σταματήσει να λειτουργεί.

Με λίγα λόγια, οι τεχνικές εξόρυξης δεδομένων πρέπει να χρησιμοποιούνται ως τελευταία λύση, όταν δεν υπάρχουν άλλες εφικτές εναλλακτικές λύσεις για την απόκτηση των απαραίτητων δεδομένων.

1.5 Είναι η εξόρυξη δεδομένων από το διαδίκτυο νόμιμη;

Η εξόρυξη δεδομένων είναι μια τεχνική με την οποία τις περισσότερες φορές κατεβάζουμε μεγάλου όγκου δεδομένα από το διαδίκτυο με αυτοματοποιημένο τρόπο. Το λογισμικό που κατασκευάζουμε και χρησιμοποιούμε μιμείται κυριολεκτικά τη πρόσβαση ενός χρήστη στον ιστότοπο, συλλέγοντας δεδομένα από σελίδα σε σελίδα. Γενικά, θα λέγαμε ότι είναι μια τεχνική που επιτρέπει την πρόσβαση σε τεράστιες ποσότητες δεδομένων σε σχετικά μικρό χρονικό διάστημα.

Βέβαια, αυτή η τακτική ενέχει κάποια αρνητικά σημεία για τον ιδιοκτήτη του δικτυακού ιστότοπου. Αυτά σε γενικές γραμμές είναι τα εξής:

1. Είναι ένας τρόπος πρόσβασης στην ιστοσελίδα ο οποίος δεν είχε προβλεφθεί – οι περισσότερες ιστοσελίδες αναπτύσσονται για περιήγηση μέσω κάποιου φυλλομετρητή.
2. Μερικές φορές οι άνθρωποι που συγκεντρώνουν αυτά τα δεδομένα τα χρησιμοποιούν για σκοπούς που δεν είναι σύμφωνοι με τους όρους χρήσης του ιστότοπου – είτε τα αναδημοσιεύουν μέσω κάποιου δικού τους ιστότοπου, είτε τα χρησιμοποιούν για όφελος κάποιας εταιρείας.
3. Τέτοιου είδους προγράμματα μπορεί να προκαλέσουν μεγάλο φόρτο σε ένα διακομιστή (web server), γεγονός το οποίο μπορεί να προκαλέσει την επιβράδυνση της ανταπόκρισης του δικτυακού τόπου.

Πολλοί ιστότοποι που έχουν προβλήματα με τέτοιου είδους προγράμματα χρησιμοποιούν μια τεχνική η οποία ονομάζεται *CAPTCHA*. Με αυτή εμφανίζεται ένα κείμενο, το οποίο είναι αναγνώσιμο μόνο από άνθρωπο και το οποίο θα σας ζητηθεί να επαναλάβατε μέσα σε ένα πεδίο. Σε μερικούς ιστότοπους ο μηχανισμός *CAPTCHA* δεν εμφανίζεται μόνο σε φόρμες, για να εξακριβωθεί ότι συμπληρώνονται από κάποιο

υπαρκτό πρόσωπο, αλλά εμφανίζεται και μετά από αλλεπάλληλες αιτήσεις για σελίδες του ιστότοπου σε πολύ μικρό χρονικό διάστημα. Εκτός από αυτόν, υπάρχουν κι άλλοι τρόποι με τους οποίους μπορεί να αποτραπεί η πρόσβαση τέτοιον προγραμμάτων στους ιστότοπους, όμως το ζήτημα εδώ δεν είναι το πώς θα αποτραπεί, αλλά το αν είναι νόμιμη αυτή η πρόσβαση και εξαγωγή των δεδομένων.

Δεδομένου ότι το διαδίκτυο είναι ακόμα αρκετά νέο και τέτοιες δραστηριότητες διασχίζουν διεθνή σύνορα, είναι πολύ δύσκολο να βρεθεί ένας γενικός, παγκόσμιος νόμος που θα εξειδικεύεται σε αυτή τη πολύ συγκεκριμένη δραστηριότητα. Γενικά η εξόρυξη δεδομένων από μια σελίδα μπορεί να είναι αντίθετη από τους όρους χρήσης της, όμως η επιβολή αυτών τον όρων είναι πολύ ασαφής.

Πριν κατασκευάσουμε μια εφαρμογή η οποία συλλέγει δεδομένα από έναν συγκεκριμένο διαδικτυακό ιστότοπο, θα πρέπει να σιγουρευτούμε για το κατά πόσο μια τέτοια ενέργεια είναι επιτρεπτή μέσω των όρων χρήσης του ιστότοπου, ή κάποιον άλλων όρων ή ανακοινώσεων που είναι δημοσιευμένα ή διατίθενται μέσω του ιστότοπου.

Βέβαια η εξόρυξη δεδομένων δεν φαίνεται ότι θα σταματήσει, όπως και τα νομικά προβλήματα με την Napster δεν εμπόδισαν τους ανθρώπους από το γράψιμο peer-to-peer λογισμικού, ή την πιο πρόσφατη μήνυση στο YouTube η οποία δεν πτόησε τον κόσμο από το να ανεβάζει βίντεο με προστατευόμενο περιεχόμενο (copyrighted).

Συνοψίζοντας, εάν έχουμε σκοπό να εξάγουμε δεδομένα από έναν διαδικτυακό τόπο, θα πρέπει να σεβαστούμε τους όρους χρήσης. Διαφορετικές ιστοσελίδες έχουν διαφορετικούς όρους και προϋποθέσεις για τη χρήση και την επαναχρησιμοποίηση των δεδομένων που φιλοξενούνται στους κεντρικούς υπολογιστές τους. Αν σκοπός μας είναι η αναδημοσίευση στοιχείων και πληροφοριών άλλων ανθρώπων, πρέπει να ερχόμαστε σε επικοινωνία με τους ιδιοκτήτες του ιστοχώρου και να ζητάμε την άδειά τους, εκτός και αν επιτρέπεται η επαναχρησιμοποίηση τους. Τέλος, πρέπει να είμαστε προσεκτικοί ώστε να μην υποβάλουμε πάρα πολλά αιτήματα για ιστοσελίδες σε μικρό χρονικό διάστημα. Σε αντίθετη περίπτωση αυτό μπορεί να θεωρηθεί ως επίθεση άρνησης παροχής υπηρεσιών (denial-of-service attack) και η IP διεύθυνσή μας να μπλοκαριστεί από μελλοντικά αιτήματα, ή ακόμα να αντιμετωπίσουμε και νομικές συνέπειες.

1.6 Ποιο το μέλλον της χρήσης τεχνικών εξόρυξης δεδομένων;

Σήμερα ο παγκόσμιος ιστός διαθέτει κολοσσιαίες ποσότητες πληροφορίας για τους ανθρώπους, οι οποίες όμως είναι κρυμμένες από τους υπολογιστές. Είναι παράδοξο το γεγονός ότι οι πληροφορίες είναι στοιβαγμένες μέσα σε σελίδες HTML, διαμορφωμένες με τέτοιο τρόπο που είναι δύσκολο για τους ηλεκτρονικούς υπολογιστές να τις επεξεργαστούν. Με ποιο τρόπο λοιπόν μπορεί να αλλάξει αυτό;

Το Web 3.0, το οποίο είναι ένας πρόδρομος του πραγματικού σημασιολογικού ιστού, είναι αυτό που θα αλλάξει τον τρόπο που παρέχονται οι πληροφορίες στο διαδίκτυο. Αυτό που εννοούμε με τον όρο web 3.0 είναι ότι οι μεγάλες ιστοσελίδες θα

μετατραπούν σε διαδικτυακές υπηρεσίες και θα παρέχουν με αποτελεσματικότητα τις πληροφορίες τους στον κόσμο.

Όπως είπαμε και πριν, πολλές ιστοσελίδες προσφέρουν είδη τις πληροφορίες τους μέσω ενός API. Ορισμένες από αυτές είναι η Amazon, η del.icio.us και η Flickr. Στο μέλλον θα ακολουθήσουν πολλές περισσότερες.

1.6.1 Η χρήση των API

Όμως τι όφελος μπορεί να έχει μια εταιρεία διανέμοντας τις πληροφορίες τις μέσω ενός API; Για να απαντήσουμε σε αυτό το ερώτημα, ας πάρουμε ως παράδειγμα το Amazon E-Commerce API. Μέσω αυτού του API η Amazon προσφέρει ανοιχτή πρόσβαση στο κατάλογο προϊόντων της. Ο λόγος που το κάνει αυτό είναι γιατί οι εφαρμογές που δημιουργούνται μέσω αυτού του API δημιουργούν πρόσθετη πρόσβαση χρηστών στα προϊόντα της Amazon. Με άλλα λόγια, η Amazon μέσω ενός API έδωσε τη δυνατότητα σε άλλους να δημιουργήσουν πρόσθετους τρόπους πρόσβασης στο κατάλογό της.

Η del.icio.us είναι επίσης γνωστή ως μια από τις πρώτες εταιρείες που άνοιξαν ένα υποσύνολο των λειτουργιών του δικτυακού τους τόπου μέσω ενός API. Ωστόσο, το API της del.icio.us είναι τελείως διαφορετικό από αυτό της Amazon γιατί δεν επιτρέπει στο κοινό τη πρόσβαση στα δεδομένα του διαδικτυακού τόπου. Για παράδειγμα, μια εφαρμογή που χρησιμοποιεί το API της del.icio.us μπορεί να προσθέσει προγραμματιστικά μια νέα ιστοσελίδα, αλλά όχι να βρει ποιες ιστοσελίδες έχουν ήδη προστεθεί στο del.icio.us.

1.6.2 Η χρήση των τεχνικών εξόρυξης δεδομένων

Παρά το γεγονός ότι το API του del.icio.us δεν προσφέρει πρόσβαση στα δεδομένα του ιστοχώρου, πολλές εταιρείες καταφέρνουν να συλλέξουν πληροφορίες που είναι αποθηκευμένες σε αυτό. Πως τα καταφέρνουν; Η απάντηση είναι απλή. Κάνοντας χρήση τεχνικών εξόρυξης δεδομένων. Στο del.icio.us οι διευθύνσεις url είναι τυποποιημένες. Για παράδειγμα, αν ψάχναμε ιστοσελίδες με ετικέτα «programming» το url θα ήταν «http://www.delicious.com/tag/programming», ενώ αν κάναμε αναζήτηση σχετικά με ταινίες αυτό θα ήταν «http://www.delicious.com/tag/ταινίες». Όπως παρατηρούμε η δομή των url είναι πάντα ίδια. Οπότε, δίνοντας μια ετικέτα σε κάποιο πρόγραμμα, θα μπορούσε να μας επιστρέψει όλες της σελίδες που έχουν τη συγκεκριμένη ετικέτα.

Επίλογος

Σε αυτό το κεφάλαιο πήραμε μια πρώτη γεύση για την έννοια της εξόρυξης δεδομένων από το διαδίκτυο. Πως μπορούμε όμως να δημιουργήσουμε ένα πρόγραμμα που θα συλλέγει τις πληροφορίες που εμείς θέλουμε από το διαδίκτυο; Πριν απαντήσουμε σε αυτό το ερώτημα είναι καλό να δούμε τα χαρακτηριστικά που παρουσιάζουν τέτοιου είδους προγράμματα. Γνωρίζοντας αυτά τα χαρακτηριστικά, θα προχωρήσουμε στη περιγραφή των εργαλείων που υπάρχουν σε κάθε γλώσσα προγραμματισμού, έτσι ώστε να είμαστε σε θέση να δημιουργήσουμε ένα πρόγραμμα εξόρυξης δεδομένων.

ΚΕΦΑΛΑΙΟ 2
ΔΗΜΙΟΥΡΓΙΑ ΠΡΟΓΡΑΜΜΑΤΩΝ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

Εισαγωγή
Σε αυτό το κεφάλαιο θα δούμε τις θεμελιώδεις τεχνικές για τη δημιουργία προγραμμάτων αξιοποίησης των πόρων του διαδικτύου. Για να δημιουργήσουμε τέτοια προγράμματα θα πρέπει να σταματήσουμε να σκεφτόμαστε σαν κοινοί χρήστες του διαδικτύου. Δηλαδή θα πρέπει να βγούμε από την έννοια της κοινής περιήγησης στον παγκόσμιο ιστό μέσω ενός φυλλομετρητή.

Τα προγράμματα περιήγησης έχουν σχεδιαστεί για να κάνουν κάποια πράγματα καλά, αλλά τους λείπει η ικανότητα να κάνουν συγκεκριμένα πράγματα εξαιρετικά καλά. Από την άλλη πλευρά, προγράμματα όπως οι *spiders* ή οι *scrapers* έχουν προγραμματιστεί για να εκτελούν συγκεκριμένα καθήκοντα στη τελειότητα.

2.1 Spiders και Scrapers
Υπάρχει πληθώρα προγραμμάτων τα οποία συλλέγουν και επεξεργάζονται δεδομένα από το διαδίκτυο. Αλλά παρόλο το πλήθος αυτών των προγραμμάτων, όλα έχουν κάποια κοινά χαρακτηριστικά. Διακρίνουμε δυο κατηγορίες τέτοιων προγραμμάτων, τους *spiders* και τους *scrapers*.

Θα μπορούσαμε να πούμε ότι οι *spiders* είναι προγράμματα τα οποία ακολουθούν συνδέσμους συλλέγοντας περιεχόμενα των ιστοσελίδων, αρχεία ή σύνθεση και των δύο, ενώ οι *scrapers* συλλέγουν συγκεκριμένα κομμάτια πληροφορίας μέσα από αυτά τα αρχεία. *Spiders* και *scrapers* δουλεύουν πολλές φορές σε συνεργασία. Για παράδειγμα, θα μπορούσαμε να έχουμε ένα πρόγραμμα στο οποίο θα υπήρχε ένας spider για να ακολουθεί συνδέσμους και ένας scraper για να συλλέγει συγκεκριμένα κομμάτια πληροφορίας (Κάτι παρόμοιο θα κάνουμε στην εφαρμογή εξόρυξης δεδομένων από τη σελίδα του χρυσού οδηγού).

2.2 Γιατί να δημιουργήσουμε έναν spider ή έναν scraper;
Φυσικά, μπορούμε να επισκεφτούμε οποιαδήποτε στιγμή την ιστοσελίδα που επιθυμούμε, αλλά δεν θα ήταν καλύτερο να είχαμε ένα πρόγραμμα που θα το έκανε αυτό για εμάς και θα μας εμφάνιζε μόνο τη πληροφορία που χρειαζόμασταν; Επιπλέον, με ένα τέτοιο πρόγραμμα θα μπορούσαμε να συλλέξουμε δεδομένα από χιλιάδες διαφορετικές ιστοσελίδες και να τα αποθηκεύουμε σε ένα διαφορετικό τύπο δεδομένων, να συμπληρώσουμε φόρμες, να συγκρίνουμε τιμές προϊόντων από διάφορες σελίδες, να ειδοποιηθούμε για ένα σχόλιο που έγινε στο ιστολόγιο, στο forum ή στην ιστοσελίδα μας.

2.3 Τα 4 βασικά βήματα δημιουργίας ενός scraper.

Για τη δημιουργία ενός προγράμματος εξόρυξης δεδομένων από έναν ιστοχώρο ακολουθούνται τέσσερα βασικά βήματα:

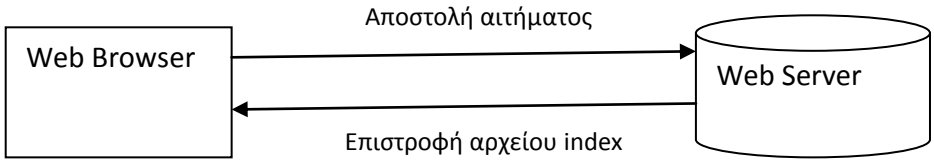
1. **Εξερεύνηση**. Η εξερεύνηση είναι μια επαναληπτική διαδικασία, κατά την οποία βρίσκουμε τις πληροφορίες τις οποίες θέλουμε να εξάγουμε. Κατ' αρχήν είναι απαραίτητη μια γενική ιδέα για το πως κατανέμονται οι πληροφορίες στις σελίδες του ιστότοπου. Αυτό γίνεται πολύ απλά με τη χρήση ενός φυλλομετρητή (web browser). Αργότερα, θα μας χρειαστεί η ακριβής θέση της πληροφορίας στο DOM δέντρο της σελίδας. Για να γίνει αυτό συνιστάται ένα εξειδικευμένο εργαλείο, όπως το Firebug (τί είναι το DOM δέντρο αλλά και το εργαλείο firebug θα το δούμε στο επόμενο κεφάλαιο).
2. **Λήψη**. Αφού γίνει γνωστή η δομή των σελίδων, το επόμενο βήμα είναι η λήψη τους. Αυτό συνήθως δε γίνεται εύκολα, αφού χρειάζεται να μεταφερόμαστε από τη μια σελίδα του ιστοχώρου σε άλλη. Ένα εργαλείο που κάνει καλά αυτή τη δουλειά είναι ένας προσομοιωτής browser, όπως το mechanize της python και της perl, επειδή επιτρέπει στον προγραμματιστή να γράφει κώδικα διαισθητικά, με τα ίδια βήματα που ακολουθήθηκαν με τη χρήση του φυλλομετρητή στο στάδιο της εξερεύνησης.
3. **Εξόρυξη των δεδομένων**. Μόλις το περιεχόμενο της σελίδας είναι διαθέσιμο, η εξαγωγή των δεδομένων από τον ιστοχώρο είναι θέμα ενός καλού αναλυτή κώδικα (parser), που θα πάρει ακριβώς τα στοιχεία που βρήκαμε στο στάδιο της εξερεύνησης. Μια πολύ καλή βιβλιοθήκη γι' αυτό το σκοπό είναι η BeautifulSoup της Python η οποία μπορεί να πραγματοποιήσει αυτή τη διαδικασία εύκολα και γρήγορα. Εκτός από τη BeautifulSoup θα δούμε πολλές ακόμα βιβλιοθήκες για την ανάλυση του κώδικα στα επόμενα κεφάλαια.
4. **Επεξεργασία και αποθήκευση των δεδομένων**. Τα προγράμματα εξόρυξης δεδομένων είναι ικανά να συλλέξουν τεράστιες ποσότητες δεδομένων. Το πλήθος των δεδομένων που ένας απλός scraper μπορεί να συλλέξει, δουλεύοντας απλά για μερικές μέρες, είναι κολοσσιαίο. Δεδομένου ότι κανείς από μας δεν έχει απεριόριστες δυνατότητες αποθήκευσης, η διαχείριση του όγκου των δεδομένων και η οργάνωσή τους είναι σημαντική.

2.4 Λήψη των ιστοσελίδων

Το πιο σημαντικό πράγμα που κάνει ένας scraper είναι να μεταφέρει τις ιστοσελίδες από το διακομιστή στον υπολογιστή μας. Από τη στιγμή που η σελίδα βρίσκεται στον υπολογιστή μας, μπορούμε να την επεξεργαστούμε και να εξάγουμε δεδομένα από αυτήν.

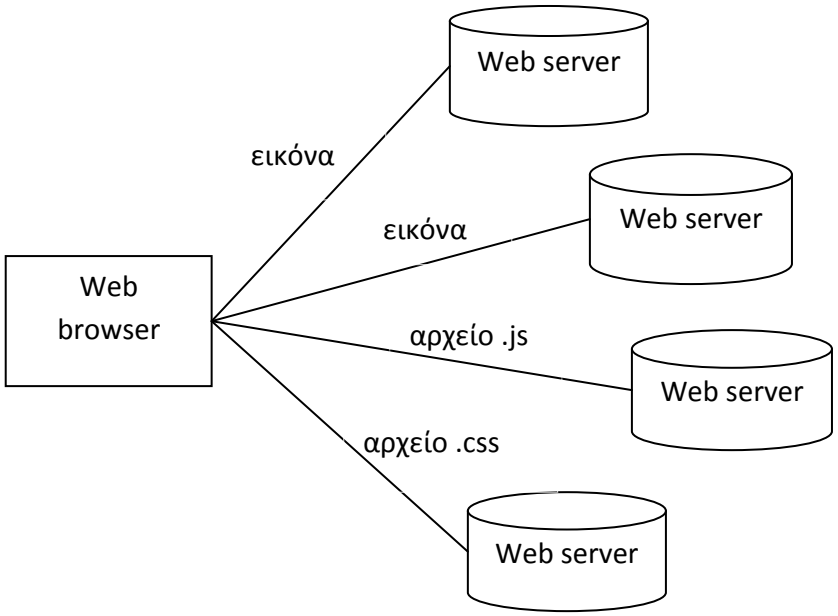
Για τους περισσότερους ανθρώπους, ο παγκόσμιος ιστός εμφανίζεται ως μια συλλογή από ιστοσελίδες. Αλλά στη πραγματικότητα ο παγκόσμιος ιστός είναι μια συλλογή από αρχεία που διαμορφώνουν αυτές τις ιστοσελίδες. Τα αρχεία αυτά μπορεί να υπάρχουν σε διακομιστές οπουδήποτε στο κόσμο, και δημιουργούν μια ιστοσελίδα όταν βρίσκονται όλα μαζί. Όλη αυτή τη διαδικασία ένας φυλλομετρητής την απλοποιεί αρκετά. Οπότε, εάν θέλουμε να κατασκευάσουμε τέτοιου είδους προγράμματα, θα πρέπει να γνωρίζουμε τις θεμελιώδεις αρχές με τις οποίες το διαδίκτυο λειτουργεί.

Όταν ο φυλλομετρητής ζητά μία ιστοσελίδα, ο διακομιστής αρχικά στέλνει ένα default αρχείο, ή ένα index αρχείο το οποίο δείχνει τη τοποθεσία όλων των άλλων αρχείων που χρειάζονται για να συνθέσουν την ιστοσελίδα.



Σχήμα - Επικοινωνία client/server

Κατά κανόνα, αυτό το αρχείο index περιέχει αναφορές στα άλλα αρχεία που συνθέτουν την ιστοσελίδα. Αυτά μπορεί να είναι εικόνες, αρχεία JavaScript, αρχεία φύλλων στυλ (CSS files), αρχεία πολυμέσων (π.χ. Flash, QuickTime) κ.α.



Σχήμα – Επικοινωνία client/server με την ύπαρξη πολλών διαφορετικών διακομιστών

Για παράδειγμα, εάν ζητήσετε μια ιστοσελίδα με αναφορές σε τέσσερα διαφορετικά σημεία, στη πραγματικότητα θα πραγματοποιηθούν πέντε διαφορετικές λήψεις αρχείων (μια για την ιστοσελίδα και μια για κάθε άλλο αντικείμενο στο οποίο αναφέρεται η ιστοσελίδα). Συνήθως όλα τα αρχεία βρίσκονται στον ίδιο διακομιστή, αλλά θα μπορούσαν κάλλιστα να βρίσκονται μερικά ή όλα σε διαφορετικά domains.

Όπως είδαμε, σε αυτή τη διαδικασία συμμετέχουν δυο μέρη: ο πελάτης (*web client*) και ο διακομιστής (*web server*). Αυτά τα δυο μέρη χρησιμοποιούν μια κοινή μέθοδο επικοινωνίας (συνήθως το πρωτόκολλο επικοινωνίας ***HTTP – HyperText Transfer***

Protocol) για την αποστολή αιτήσεων (requests) από τον πελάτη και την αποστολή απαντήσεων (*responses*) από τον διακομιστή. Παραδείγματα πελατών δεν είναι μόνο οι φυλλομετρητές, αλλά και οι αυτόματοι πράκτορες (*automated agents*) όπως οι *crawlers*, οι *spiders* και οι *scrapers*.

Σε επόμενα κεφάλαια θα δούμε βιβλιοθήκες γλωσσών προγραμματισμού που διευκολύνουν την μεταφορά αρχείων μέσω πρωτοκόλλων επικοινωνίας όπως τα *FTP*, *FTPS*, *HTTP*, *HTTPS*, *Gopher*, *TELNET* και *LDAP*.

2.5 Η ανατομία ενός URL

Για τη σωστή λήψη των ιστοσελίδων που μας ενδιαφέρουν είναι καλό να γνωρίζουμε όλα τα μέρη ενός URL (**Uniform Resource Locator**).

http://user:pass@www.domain.com:8080/path/to/file.ext?query=&var=value#anchor

- Το *http* είναι το πρωτόκολλο επικοινωνίας που χρησιμοποιείται για την αλληλεπίδραση πελάτη και διακομιστή. Ένα άλλο παράδειγμα είναι το https το οποίο είναι ισοδύναμο του http, μόνο που χρησιμοποιείται ένα πιστοποιητικό SSL για τη κρυπτογράφηση της επικοινωνίας ανάμεσα σε πελάτη και διακομιστή.
- Το *user@pass* είναι ένα προαιρετικό στοιχείο που χρησιμοποιείται σε περίπτωση ανάγκης ταυτοποίησης του χρήστη (μέσω username και password) για την πρόσβαση στους πόρους της ιστοσελίδας.
- Το *:8080* είναι ακόμα ένα προαιρετικό στοιχείο για την ενημέρωση του πελάτη, ότι ο διακομιστής δέχεται τις εισερχόμενες αιτήσεις στη θύρα (port) 8080. Σε περίπτωση απουσίας αυτού του στοιχείου οι περισσότεροι πελάτες χρησιμοποιούν την standard HTTP θύρα 80.
- */path/to/file.ext*: ακριβής προσδιορισμός ενός αρχείου μέσα στον server.
- *query=&var=value:* Αλφαριθμητικό για το πέρασμα παραμέτρων.
- *#anchor:* Προσδιορισμός μιας συγκεκριμένης τοποθεσίας μέσα στο έγγραφο.

2.6 Η ανατομία μια σελίδας HTML

Όταν γνωρίζουμε τη δομή ενός αρχείου *HTML,* είναι πολύ πιο εύκολο να εξάγουμε δεδομένα και πληροφορίες από μια ιστοσελίδα. Κατά τη δημιουργία ενός *scraper,* κυρίως τα *html* αρχεία είναι αυτά από τα οποία παίρνουμε τις πληροφορίες, και δεν είναι τίποτα παραπάνω από απλά αρχεία κειμένου με μια ιδιαίτερη μορφοποίηση.

Για να καταλάβουμε πώς το πρόγραμμά μας θα εξάγει δεδομένα από την ιστοσελίδα θα πρέπει να ξεκινήσουμε από τα βασικά. Καταρχήν, από το πώς μια σελίδα *HTML* φαίνεται και μετά στο πώς μπορεί να οργανωθεί η πληροφορία στο σώμα της ιστοσελίδας.

Ο κώδικας μιας απλής *HTML* σελίδας μοιάζει με τον παρακάτω:

```
1: <html>
2: <head>
3:     <title> Τίτλος της ιστοσελίδας </title>
4: </head>
5: <body>
6:  Σώμα της ιστοσελίδας
7: </body>
8: </html>
```

Αν το μόνο που χρειαζόμασταν ήταν ο τίτλος της ιστοσελίδας τότε όλα θα ήταν πολύ απλά. Όμως, αν χρειαζόμαστε κάτι από το σώμα της ιστοσελίδας, παραδείγματος χάριν μια ημερομηνία ή μια τιμή ενός προϊόντος, χρειάζεται λίγη δουλειά παραπάνω. Πολλές φορές στο σώμα μιας ιστοσελίδας υπάρχουν πίνακες, κώδικας JavaScript και άλλα κομμάτια κώδικα που δυσχεραίνουν τη δουλειά μας.

Συνήθως, σημαντικές πληροφορίες σε μια ιστοσελίδα βρίσκονται μέσα στα στοιχεία <hx>, όπου x ένας αριθμός από το 1 έως το 6. Μερικές φορές, μπορούμε να πάρουμε χρήσιμες πληροφορίες από μια ιστοσελίδα, εξάγοντας μόνο τις κεφαλίδες. Για παράδειγμα, μπορούμε να φτιάξουμε έναν spider ο οποίος θα εξάγει από μια ιστοσελίδα ειδήσεων μόνο τις επικεφαλίδες και τις υποκεφαλίδες των ειδήσεων, αγνοώντας το υπόλοιπο κείμενο.

2.7 Ανάλυσης του κώδικα

Η ανάλυση (*parsing*) του κώδικα είναι η διαδικασία διαχωρισμού των χρήσιμων ή επιθυμητών πληροφοριών από τις άχρηστες ή ανεπιθύμητες. Για παράδειγμα, αν δημιουργούσαμε κάποιον *scraper* που ακολουθεί συνδέσμους σε ιστοσελίδες, θα πρέπει να διαχωριστούν αυτοί οι σύνδεσμοι από τον υπόλοιπο *HTML* κώδικα. Ομοίως, εάν κατασκευάζαμε ένα πρόγραμμα που κατεβάζει όλες τις εικόνες από μια ιστοσελίδα, θα έπρεπε να δημιουργήσουμε μια μέθοδο για την αναγνώριση όλων των αναφορών στα αρχεία εικόνων της ιστοσελίδας.

Ένα από τα προβλήματα που ενδεχομένως μπορεί να αντιμετωπίσουμε κατά την ανάλυση του κώδικα είναι η κακή σύνταξή του. Η πλειοψηφία των *HTML* σελίδων στο διαδίκτυο είναι άσχημα μορφοποιημένες (π.χ. λείπουν ετικέτες κλεισίματος). Ευτυχώς υπάρχουν βιβλιοθήκες λογισμικού όπως η *HTML Tidy* που ‘καθαρίζουν’ τον κακογραμμένο κώδικα. Ακόμα υπάρχουν βιβλιοθήκες γλωσσών προγραμματισμού για την ανάλυση *HTML* κώδικα (*html parsers*), οι οποίες αγνοούν τέτοιου είδους κώδικα με αποτέλεσμα να μην έχουν κάποιο πρόβλημα. Μια από αυτές είναι η *BeautifulSoup* της *Python,* την οποία και θα χρησιμοποιήσουμε για την ανάλυση των σελίδων του Χρυσού Οδηγού.

2.8 Αυτοματοποιημένη υποβολή φορμών

Εκτός από το κατέβασμα και την ανάλυση των ιστοσελίδων, άλλη μια τεχνική που χρησιμοποιείται συχνά σε αυτές τις περιπτώσεις είναι η αυτοματοποιημένη συμπλήρωση και αποστολή φορμών. Με αυτό το τρόπο οι *scrapers* αναζητούν πληροφορίες και λειτουργούν για λογαριασμό μας. Στα επόμενα κεφάλαια θα δούμε διάφορες βιβλιοθήκες γλωσσών που μπορούν να υποβάλουν μια φόρμα. Μερικές από αυτές είναι η *Watir* και η *Selenium* της *Ruby* ή η *Mechanize* της *Python*.

2.9 Διαχείριση μεγάλου όγκου δεδομένων

Όπως είπαμε και πριν, τα προγράμματα εξόρυξης δεδομένων είναι ικανά να συλλέξουν τεράστιες ποσότητες δεδομένων. Οπότε είναι αναγκαία η χρήση μεθόδων αποθήκευσης και οργάνωσής τους.

Η οργάνωση των δεδομένων που το πρόγραμμά μας θα συλλέξει χρειάζεται προγραμματισμό. Η απόφαση που θα πάρουμε για τον τρόπο αποθήκευσης των δεδομένων πρέπει να ανταποκρίνεται στις ανάγκες του εκάστοτε προβλήματος που επιχειρούμε να λύσουμε. Για παράδειγμα, εάν τα στοιχεία είναι κυρίως σε μορφή κειμένου, το οποίο έχει την ανάγκη ταξινόμησης και αναζήτησης, τότε μια καλή επιλογή θα ήταν η δημιουργία μιας σχεσιακής βάσης δεδομένων.

Επίλογος

Είδαμε τα κύρια χαρακτηριστικά που έχουν σχεδόν όλα τα προγράμματα εξόρυξης δεδομένων. Τώρα πια είμαστε έτοιμοι να προγραμματίσουμε. Στα επόμενα κεφάλαια θα δώσουμε σάρκα και οστά στα προγράμματα, περιγράφοντας πληθώρα εργαλείων που μας βοηθούν στην εξόρυξη δεδομένων σε πολλές διαφορετικές γλώσσες προγραμματισμού.

ΚΕΦΑΛΑΙΟ 3
ΧΡΗΣΙΜΕΣ ΤΕΧΝΟΛΟΓΙΕΣ

Εισαγωγή
Πριν προχωρήσουμε στη περιγραφή της διαδικασίας εξόρυξης δεδομένων με κάθε γλώσσα προγραμματισμού, θα ήταν χρήσιμο να περιγράψουμε μερικές τεχνολογίες που χρησιμοποιούνται κατά τη διάρκεια ανάπτυξης τέτοιων προγραμμάτων. Αυτό θα μας βοηθήσει στη καλύτερη κατανόηση της λειτουργίας των βιβλιοθηκών, αλλά και των παραδειγμάτων που θα δούμε στα επόμενα κεφάλαια.

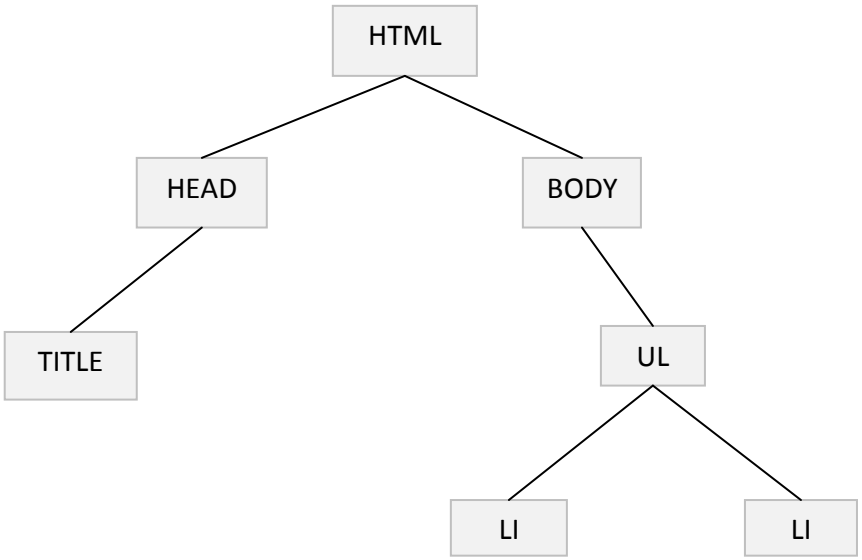
3.1 Document Object Model
Το πρώτο βήμα κατά τη διαδικασία εξόρυξης δεδομένων είναι η φόρτωση του αρχείου και μετά η εξαγωγή των επιθυμητών δεδομένων από αυτό. Ωστόσο, αυτό απαιτεί μια καλή γνώση του τρόπου δομής των HTML/XHTML ή XML εγγράφων. Το **DOM** (**Document Object Model**) είναι μια γενική τεχνολογία, ανεξάρτητη από πλατφόρμα και γλώσσα προγραμματισμού, για την αναπαράσταση και την αλληλεπίδραση με τα αντικείμενα αυτών των εγγράφων.

Η πιο σημαντική σύμβαση που χρησιμοποιείται από το DOM είναι η αναπαράσταση ενός εγγράφου όπως ένα **δέντρο (tree)**. Πιο συγκεκριμένα, το έγγραφο αναπαρίσταται ως ένα οικογενειακό δέντρο. Ένα δέντρο είναι μια ιεραρχική δομή που αποτελείται από **κόμβους (nodes)**. Οι κόμβοι είναι για τα δέντρα ότι είναι τα στοιχεία τους πίνακες: απλά αντικείμενα που βρίσκονται μέσα στη δομή δεδομένων.

Κάθε ξεχωριστός κόμβος μπορεί να έχει κανέναν έως πολλούς κόμβους **παιδιά (childs)**. Όλοι οι κόμβοι του δέντρου έχουν ένα κόμβο **γονέα (parent)**, με μια μόνο εξαίρεση: το κόμβο **ρίζα (root)**. Ας πάρουμε για παράδειγμα το παρακάτω κομμάτι HMTL κώδικα:

```
1: <html>
2: <title>DOM example</title>
3: <body>
4:     <ul>
5:         <li>list item 1</li>
6:         <li>list item 2</li>
7:     </ul>
8: </body>
9: </html>
```

Αυτό μπορεί να αναπαρασταθεί με το παρακάτω DOM δέντρο:



Σχήμα – Αναπαράσταση ενός DOM δέντρου

Στην εικόνα βλέπουμε για παράδειγμα, ότι το *html* είναι ο κόμβος γονέα (*root node*), το *head* και το *body* είναι το πρώτο και το δεύτερο αντίστοιχα παιδί του *html* ή ότι οι κόμβοι *li* είναι απόγονοι του στοιχείου *body* και έχουν πατέρα το στοιχείο *ul*.

3.2 Κανονικές Εκφράσεις (Regular Expressions)

Μια κανονική έκφραση είναι ένα συγκεκριμένο είδος πρότυπου κειμένου που μπορούμε να χρησιμοποιήσουμε σε πολλές γλώσσες προγραμματισμού. Στα προγράμματα εξόρυξης δεδομένων μπορούμε να χρησιμοποιήσουμε τις κανονικές εκφράσεις για να εξάγουμε συγκεκριμένα κομμάτια από τα HTML αρχεία που λαμβάνουμε. Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε απλές κανονικές εκφράσεις για την εύρεση όλων των συνδέσμων του εγγράφου ή την εύρεση μιας παραγράφου με συγκεκριμένη ιδιότητα id ή class.

Μια κανονική έκφραση συνήθως περιλαμβάνει μεταχαρακτήρες (metacharacters) οι οποίοι λειτουργούν ως ‘μπαλαντέρ’. Παρακάτω είναι μια λίστα με μερικούς μεταχαρακτήρες για μια γρήγορη ματιά:

Μεταχαρακτήρας	Ταίριασμα
.	Ταιριάζει οποιοδήποτε χαρακτήρα εκτός από την αλλαγή γραμμής
x	Ταιριάζει οποιοδήποτε στιγμιότυπο του ‘x’
[]	Ταιριάζει οποιονδήποτε χαρακτήρα από ένα συγκεκριμένο εύρος (π.χ. [0-9], [a-z], [A-Za-z0-9] …)
|	Λογικός τελεστής OR – [x|y] θα ταιριάξει ένα στιγμιότυπο του x ή του y
{x}	Ο χαρακτήρας που προηγείται θα ταυτιστεί ακριβώς x φορές (π.χ. το x{3} θα βρει ακριβώς τρεις εμφανίσεις του x
*****	Ο χαρακτήρας που προηγείται θα ταυτιστεί μηδέν ή περισσότερες φορές.

+	Ο χαρακτήρας που προηγείται θα ταυτιστεί μία ή περισσότερες φορές.
?	Ο χαρακτήρας που προηγείται είναι προαιρετικός και θα ταυτιστεί το πολύ μία φορά.
^	Ταιριάζει την αρχή μιας γραμμής.
$	Ταιριάζει το τέλος μιας γραμμής.

Πίνακας - Μεταχαρακτήρες των κανονικών εκφράσεων

Ας πούμε ότι θέλαμε όλους τους συνδέσμους ενός εγγράφου. Με ποια κανονική έκφραση θα το κάναμε αυτό;

```
<a href="[^"]+">[^<]+</a>
```

Ξεκινάμε με το *<a href=",* όπως ξεκινούν όλοι οι σύνδεσμοι σε ένα html αρχείο. Μετά βλέπουμε το *[^"]+* που σημαίνει ότι ακολουθεί οποιοσδήποτε χαρακτήρας, εκτός από διπλά εισαγωγικά (“). Έπειτα βρίσκουμε το πρώτο στιγμιότυπο ενός διπλού εισαγωγικού και ενός συμβόλου κλεισίματος (>). Οι ίδιες αρχές ισχύουν και στο εναπομείναν κομμάτι.

3.3 XPath

Κάπως παρόμοια με το τρόπο που οι κανονικές εκφράσεις μας επιτρέπουν να βρούμε συγκεκριμένα κομμάτια μέσα σε ένα κείμενο, έτσι και η XPath μας επιτρέπει να βρούμε συγκεκριμένους κόμβους μέσα σε XML συμβατά έγγραφα. Και οι δύο τεχνολογίες εκπληρώνουν τον σκοπό τους χρησιμοποιώντας μια σύνταξη που αποτελείτε από μεταχαρακτήρες (metacharacters).

Η XPath χρησιμοποιεί εκφράσεις με μορφή διαδρομών ή μονοπατιών (paths) για να επιλέξει κάποιο κόμβο ενός XML εγγράφου. Αυτές οι εκφράσεις, πολλές φορές είναι παρόμοιες με τα μονοπάτια που βλέπουμε στο παραδοσιακό σύστημα αρχείων του υπολογιστή μας. Βέβαια, υπάρχουν και εκφράσεις πιο πολύπλοκες, για το προσδιορισμό ομάδας στοιχείων από τα XML έγγραφα. Οι πιο σημαντικοί ειδικοί χαρακτήρες για τη δημιουργία αυτών των μονοπατιών είναι οι παρακάτω:

Ειδικοί χαρακτήρες	**Περιγραφή**
Όνομα στοιχείου (κόμβου)	Επιλογή όλων των στοιχείων παιδιών του κόμβου
/	Επιλογή ξεκινώντας από το στοιχείο ρίζα (απόλυτο μονοπάτι προς το στοιχείο)
//	Επιλογή των κόμβων του εγγράφου από το τρέχων κόμβο που ταιριάζουν με την επιλογή χωρίς να έχει σημασία το που βρίσκονται
.	Επιλογή του τρέχον κόμβου
..	Επιλογή του πατέρα του τρέχον κόμβου
@	Επιλογή ιδιοτήτων
*****	Επιλογή όλων των στοιχείων-κόμβων
@*	Επιλογή όλων των ιδιοτήτων-κόμβων
|	Λογικός τελεστής AND

Πίνακας - Ειδικοί χαρακτήρες της XPath

Ας πάρουμε για παράδειγμα το μικρό κομμάτι HTML κώδικα που είδαμε στην ενότητα 3.1 για το Document Object Model. Με την έκφραση «ul/li» θα επιλέγαμε όλα τα στοιχεία li που είναι παιδιά ενός στοιχείου ul, ενώ με την έκφραση «//li» θα επιλέγαμε όλα τα στοιχεία λίστας του εγγράφου χωρίς να έχει σημασία το που βρίσκονται. Για να επιλέξουμε τα στοιχεία head και body θα χρησιμοποιούσαμε την έκφραση «//head | //body». Μερικά ακόμη παραδείγματα θα δούμε στην επόμενη παράγραφο, όταν θα συγκρίνουμε τις εκφράσεις XPath με αυτές των CSS Selectors.

3.4 CSS Selectors

Τα CSS Selectors είναι ακόμα ένας τρόπος προσδιορισμού και εύρεσης στοιχείων μέσα στα HTML έγγραφα. Οι εκφράσεις που χρησιμοποιούνται είναι παρόμοιες με τις XPath εκφράσεις. Όμως αφού μοιάζουν, ποιός ο λόγος να τις μάθουμε;

Ένας λόγος που τα CSS Selectors μπορεί να μας φανούν χρήσιμα, είναι ότι μπορεί να έχουμε κάποια εμπειρία με τα CSS (Cascading Style Sheets), οπότε η χρήση βιβλιοθηκών που χρησιμοποιούν αυτό τον τρόπο προσδιορισμού των στοιχείων γίνεται πιο εύκολη. Επίσης οι εκφράσεις των CSS Selectors είναι σχεδόν πάντοτε πιο απλές και κατανοητές από τις εκφράσεις XPath. Παρακάτω, ακολουθούν μερικά παραδείγματα εκφράσεων CSS Selectors τα οποία θα αντιπαραθέσουμε με αυτές του XPath.

```
1: <html>
2: <body>
3: <div id="nav">
4:     <ul class="horizontal">
5:         <li><a href="/home">Home</a></li>
6:         <li><a href="/about-us">About Us</a></li>
7:         <li><a href="/contact-us">Contact Us</a></li>
8:     </ul>
9: </div>
10: </body>
11: </html>
```

Ας δούμε τους βασικούς τρόπους επιλογής και τα αποτελέσματά τους όταν θα εφαρμοστούν στο παραπάνω *HTML* έγγραφο:

- **#nav -** Θα επιλέξει το στοιχείο *div* επειδή έχει ιδιότητα *id* ίση με *nav*.
- **li -** Θα επιλέξει όλα τα *li* στοιχεία
- **.horizontal -** Θα επιλέξει το στοιχείο *ul* επειδή έχει κλάση ίση με horizontal
- * - Θα επιλέξει όλα τα στοιχεία του εγγράφου
- **li, a** - Θα επιλέξει όλα τα στοιχεία *li* και a του εγγράφου.

Στον επόμενο πίνακα βλέπουμε στη μία πλευρά τα *CSS Selector* και στην άλλη τις αντίστοιχες *XPath* εκφράσεις:

CSS	XPath
#nav	//*[@id=”nav”]
li	//li
.horizontal	//*[class=”horizontal”]
*****	//*
li, a	//li | //a

Πίνακας 3 - Σύγκριση εκφράσεων XPath με CSS Selectors

Βλέπουμε ότι οι εκφράσεις *XPath* είναι λίγο πολυπλοκότερες και αυτό δυσχεραίνει τη κατάσταση, εάν θέλουμε να κάνουμε μια πιο πολύπλοκη αναζήτηση από αυτές που είδαμε ως τώρα. Βέβαια δεν θα μπορούσαμε εδώ να δείξουμε όλα τα χαρακτηριστικά και των εκφράσεων *XPath* αλλά και των *CSS Selectors*. Περισσότερες πληροφορίες μπορούν να βρεθούν στην ιστοσελίδα http://www.w3.org/TR/css3-selectors για τα *CSS Selectors* και στην ιστοσελίδα http://www.w3.org/TR/xpath/ για την *XPath*.

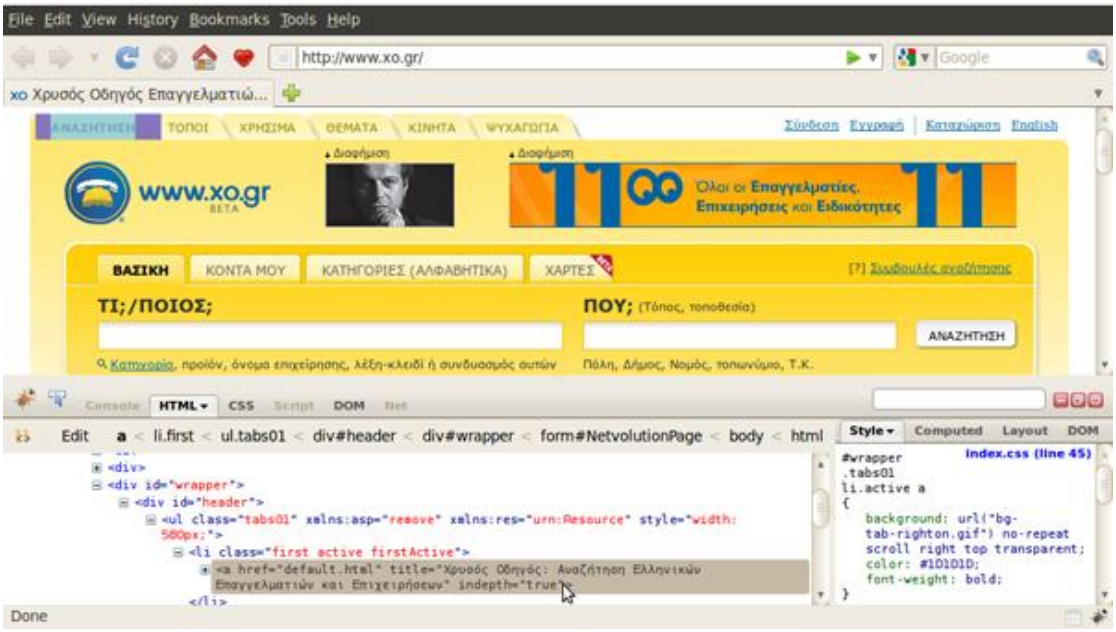
3.5 Χρήσιμα εργαλεία του Firefox

Κατά τη διαδικασία εξαγωγής δεδομένων από ιστοσελίδες χρειάζεται πολλές φορές να βλέπουμε το πηγαίο κώδικα των ιστοσελίδων. Αυτό μπορούμε να το κάνουμε με διάφορους τρόπους, όμως η χρήση του πρόσθετου εργαλείου Firebug του Firefox είναι η ιδανική λύση. Εννοείται ότι για να δουλέψουμε με το *Firebug* θα πρέπει να έχουμε εγκατεστημένο στον υπολογιστή μας τον *Firefox* (http://www.mozilla.com/en-US/firefox/) και το *add-on Firebug* (https://addons.mozilla.org/en-US/firefox/addon/1843).

Παρακάτω θα δημιουργήσουμε μια εφαρμογή, η οποία θα εξάγει δεδομένα από τη σελίδα του χρυσού οδηγού (http://www.xo.gr/). Αν ανοίξετε αυτή τη σελίδα στον *Firefox* και μετά επιλέξετε *Tools* → *Firebug* → *Open Firebug,* θα δείτε στην οθόνη σας κάτι σαν αυτό στην εικόνα 1. Το *Firebug* δείχνει τον κώδικα της σελίδας σε δενδρική μορφή. Κάθε φορά που περνάτε τον κέρσορα του ποντικιού πάνω από ένα στοιχείο του δέντρου αυτό γραμμοσκιάζεται. Στην *εικόνα 1* έχω τον κέρσορα πάνω από το HTML στοιχείο:

```
<a href="default.html" title="Χρυσός Οδηγός: Αναζήτηση Ελληνικών
Επαγγελματιών και Επιχειρήσεων" indepth="true"><span>ΑΝΑΖΗΤΗΣΗ</span></a>
```

Το αποτέλεσμα είναι το στοιχείο αυτό να έχει αποκτήσει ένα μπλε φόντο:

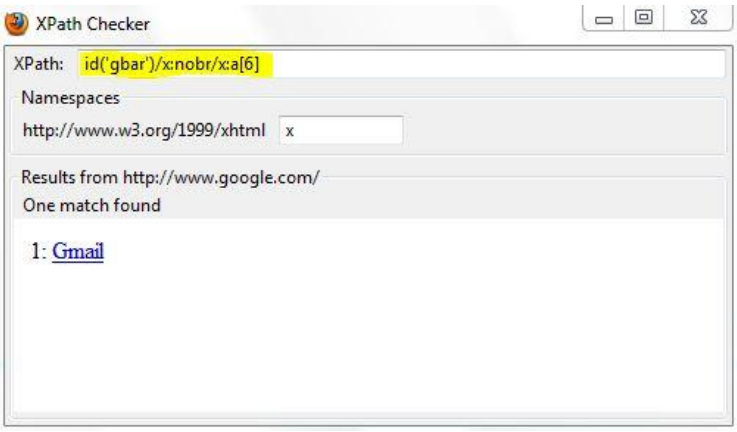


Εικόνα 1 - Εύρεση στοιχείο στη σελίδα του χρυσού οδηγού με το firebug

Μια άλλη δυνατότητα του Firebug που μπορεί να σας φανεί χρήσιμη είναι ότι μπορούμε να αντιγράψουμε το *XPath* κάθε στοιχείου. Κάνοντας δεξί κλικ σε κάθε *HTML* στοιχείο εμφανίζετε ένα αναδυόμενο μενού (*popup menu*) στο οποίο υπάρχει η επιλογή *'copy XPath'*. Το *XPath* του στοιχείου που είδαμε στην εικόνα 1 είναι:

/html/body/form/div[3]/div/ul/li/a

Ένα ακόμα χρήσιμο εργαλείο του Firefox είναι το *XPath Checker*. Έχοντας εγκαταστήσει αυτό το add-on στον Firefox, κάνοντας δεξί κλικ σε οποιοδήποτε στοιχείο της σελίδας και επιλέγοντας *'view XPath'* στο αναδυόμενο μενού που εμφανίζεται, μπορούμε να δούμε το *XPath* του στοιχείου. Εκτός αυτού, μπορούμε να δοκιμάσουμε μερικά *XPath* για να δούμε εάν αντιστοιχούν σε κάποια στοιχεία της σελίδας. Για παράδειγμα, κάνοντας κλικ στο σύνδεσμο "*Gmail*" που εμφανίζεται στη κεντρική σελίδα του Google, μας εμφανίζεται το παρακάτω παράθυρο με το *XPath* του στοιχείου:



Εικόνα 2 – Xpath Checker

Επίλογος

Σε αυτό το κεφάλαιο είδαμε κάποιες πολύ χρήσιμες τεχνολογίες, των οποίων η περιγραφή θα μας βοηθήσει στη καλύτερη κατανόηση του τρόπου λειτουργίας πολλών βιβλιοθηκών που θα δούμε στα επόμενα κεφάλαια. Στα επόμενα πέντε κεφάλαια περιγράφονται οι τρόποι με τους οποίους μπορούμε να εξάγουμε δεδομένα από το διαδίκτυο με τις γνωστότερες γλώσσες προγραμματισμού.

ΚΕΦΑΛΑΙΟ 4
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ RUBY

Εισαγωγή

Σε αυτό το κεφάλαιο θα αναφερθούμε στους τρόπους με τους οποίους μπορούμε να κάνουμε εξόρυξη δεδομένων από τον ιστό μέσω της γλώσσας προγραμματισμού *Ruby*. Καταρχήν θα διαπιστώσουμε ότι υπάρχουν δύο κύριοι τρόποι προσέγγισης αυτής της διαδικασίας: Πρώτον, βλέποντας τα περιεχόμενα μιας ιστοσελίδας σε μορφή κειμένου και δεύτερον, βλέποντας αυτά ως ένα 'δέντρο'.

Αν δούμε τα περιεχόμενα της ιστοσελίδας σε μορφή κειμένου, τότε ο καλύτερος τρόπος για να εξάγουμε τα δεδομένα είναι χρησιμοποιώντας κανονικές εκφράσεις. Αντίθετα, αν προσεγγίσουμε τα περιεχόμενα της σελίδας ως δέντρο, τότε οι επιλογές μας είναι περισσότερες. Όλες αυτές οι περιπτώσεις περιγράφονται στις επόμενες παραγράφους.

4.1 Η βιβλιοθήκη open-uri

Η *Ruby* περιέχει μια πολύ ευέλικτη βιβλιοθήκη που περικλείει όλες τις συνδέσεις *HTTP/HTTPS* και *FTP* μέσω της μεθόδου *open*. Τη βιβλιοθήκη *open-uri* θα τη χρησιμοποιήσουμε σχεδόν σε όλα τα παραδείγματα αυτού του κεφαλαίου. Δείτε για παράδειγμα πόσο εύκολα μπορεί να ανοίξει ένα http/https/ftp *URL* σαν ένα συνηθισμένο αρχείο:

```
1: open("http://www.google.com/") {|f|
2:   f.each_line {|line| p line}
3: }
```

4.2 Η ιστοσελίδα ως αρχείο κειμένου

Ο πιο εύκολος, αλλά σε πολλές περιπτώσεις ανεπαρκής τρόπος για την εξόρυξη πληροφοριών, είναι να δούμε την ιστοσελίδα ως αρχείο κειμένου. Με αυτό το τρόπο μπορούμε να χρησιμοποιήσουμε κανονικές εκφράσεις για να πάρουμε τα ζητούμενα πεδία κειμένου από την ιστοσελίδα. Για παράδειγμα, εάν θέλετε να εξάγετε το όνομα και τη τιμή ενός προϊόντος από ένα ηλεκτρονικό κατάστημα, και ξέρετε ότι και τα δύο βρίσκονται στο ίδιο *html* στοιχείο, όπως παρακάτω:

```
<td>Toshiba Satellite L655-11U        799.00€</td>
```

μπορείτε να το εξάγετε με την παρακάτω εντολή:

```
scan(page, /<td>(.*)\s+(\d+\.\d{2}€)<\/td>/)
```

Ας δούμε τώρα ένα πραγματικό αλλά και πολύ απλό παράδειγμα

```
1: require 'open-uri'
2:
3: url = "http://www.google.com/search?q=ATEI+of+Thessaloniki"
4: open(url) {
5:     |page| page_content = page.read()
6:     links = page_content.scan(/<a class=l.*?href=\"(.*?)\"/).flatten
7:     links.each {|link| puts link}
8: }
```

Αυτό το παράδειγμα παρουσιάζει τις βασικές έννοιες, δηλαδή πώς να 'φορτώσουμε' τα περιεχόμενα μιας ιστοσελίδας σε μια μεταβλητή τύπου αλφαριθμητικού (string) (γραμμή 4) και πώς να εξάγουμε τους συνδέσμους των αποτελεσμάτων από μια σελίδα του *Google* (γραμμή 5). Το πρόγραμμά μας θα συλλέξει τους πρώτους δέκα συνδέσμους από τη σελίδα αποτελεσμάτων του *Google* για το ερώτημα «*ATEI of Thessaloniki*» (γραμμή 6).

Η βασική γραμμή κώδικα είναι η έκτη. Σε αυτήν μέσω μιας κανονικής έκφρασης βρίσκουμε όλους τους συνδέσμους που έχουν το html πεδίο '*class*' ίσο με '*l*'. Στις σελίδες του Google, οι σύνδεσμοι με αυτό το όνομα στην ιδιότητα *class* είναι οι σύνδεσμοι των αποτελεσμάτων του ερωτήματος (στη συγκεκριμένη περίπτωση του ερωτήματος: «*ATEI of Thessaloniki*»).

Στις περισσότερες των περιπτώσεων όμως οι πληροφορίες που θα αναζητούμε δεν θα περιέχονται μέσα σε μία μόνο html ετικέτα ή θα χωρίζονται με έναν τρόπο στον οποίο οι κανονικές εκφράσεις δεν θα είναι ο καλύτερος τρόπος για την αναζήτησή τους. Σε αυτές τις περιπτώσεις είναι καλύτερα να δούμε την html σελίδα σε μορφή δέντρου.

4.3 HTree και REXML

Η 'δενδροειδής' προσέγγιση, μολονότι επιτρέπει πιο ισχυρές τεχνικές, έχει και αυτή τα μειονεκτήματά τις. Ένα αρχείο *html* μπορεί να δείχνει πολύ ωραίο σε έναν browser αλλά στην ουσία να είναι πολύ δύσμορφο και κακογραμμένο (για παράδειγμα μπορεί να υπάρχουν ετικέτες που δεν έχουν κλείσει σωστά ή ετικέτες που λείπουν τελείως).

Όμως υπάρχει λύση και ονομάζεται *HTree*. Αυτή η βιβλιοθήκη της Ruby, κατασκευασμένη από τον Tanaka Akira, είναι σε θέση να βάλει σε τάξη τον δύσμορφο *html* κώδικα και να τον μετατρέψει σε *XML*. Στην ουσία το αρχείο εισόδου μετατρέπεται σε αρχείο *REXML* (Η *REXML* είναι η *standart XML/XPath* βιβλιοθήκη της *Ruby*).

Αφού προ-επεξεργαστούμε το περιεχόμενο της σελίδας με το *HTree*, μπορούμε να χρησιμοποιήσουμε την *XPath* η οποία είναι μια ισχυρή γλώσσα ερωτημάτων για XML έγγραφα και ιδιαίτερα κατάλληλη για την εξαγωγή δεδομένων από το web.

Ας ξαναδούμε το πρόγραμμα που κάναμε προηγουμένως, αλλά αυτή τη φορά χρησιμοποιώντας τις βιβλιοθήκες *HTree* και *REXML* και όχι κανονικές εκφράσεις:

```
1: require 'open-uri'
2: require 'htree'
3: require 'rexml/document'
4:
5: url = "http://www.google.com/search?q=ruby"
6: open(url) {
7:   |page| page_content = page.read()
8:   doc = HTree(page_content).to_rexml
9:   doc.root.each_element('//a[@class="l"]') {
10:         |elem| puts elem.attribute('href').value
11:   }
12: }
```

Η βιβλιοθήκη *HTree* χρησιμοποιείται μόνο στην όγδοη γραμμή – μετατρέπει τον *html* κώδικα της σελίδα (ο οποίος έχει καταχωρηθεί στη μεταβλητή *page_content*) σε *REXML*. Η μαγεία του κώδικα όμως βρίσκεται στην ένατη γραμμή. Εκεί επιλέγουμε όλες τις ετικέτες *'a'* οι οποίες έχουν την ιδιότητα *'class'* ίση με 'l'. Στη συνέχεια, εκτυπώνουμε την τιμή της ιδιότητας *'href'* για κάθε ένα από αυτά τα στοιχεία.

Θεωρώ ότι αυτή η προσέγγιση είναι πολύ πιο φυσική από τη χρήση κανονικών εκφράσεων. Το μοναδικό μελανό σημείο είναι ότι πρέπει να είμαστε εξοικειωμένοι, όχι μόνο με τη γλώσσα προγραμματισμού Ruby, αλλά και με την *XPath*.

4.4 Hpricot

Το *Hpricot* είναι ένας γρήγορος και εύχρηστος *HTML* parser για τη *Ruby*. Βασίζεται στην *HTree* και τη *JQuery*, έτσι μπορεί να παρέχει την ίδια λειτουργικότητα με το συνδυασμό *HTree* και *REXML* που είδαμε στη προηγούμενη παράγραφο, αλλά με δυο μεγάλες διαφορές: καλύτερη απόδοση και ευκολότερη χρήση. Τώρα θα δούμε το γνωστό μας παράδειγμα για ακόμη μια φορά:

```
1: require 'rubygems'
2: require 'hpricot'
3: require 'open-uri'
4:
5: doc = Hpricot(open('http://www.google.com/search?q=ruby'))
6: links = doc/"//a[@class=l]"
7: links.map.each {|link| puts link.attributes['href']}
```

Πιστεύω ότι αυτός είναι ο ευκολότερος τρόπος που έχουμε δει ως τώρα. Στη πραγματικότητα αυτό το απλό παράδειγμα δεν μπορεί να δείξει παρά μόνο ελάχιστη από τη δύναμη του *Hpricot*. Αν φτιάχνουμε κάτι μικρό και δεν μας είναι αναγκαία η χρήση του *ScRUBYt!,* που θα δούμε παρακάτω, σίγουρα, η καλύτερη επιλογή είναι το *Hpricot*.

Η υποστήριξη της *XPath* από το *Hpricot* είναι σχεδόν πλήρης. Για να βρούμε το *XPath* κάποιου αντικειμένου στη σελίδα μπορούμε να χρησιμοποιήσουμε το *firebug*. Στο παρακάτω κομμάτι κώδικα θα δούμε ακόμα μερικά παραδείγματα χρήσης της *XPath*

από το *Hpricot*. Περισσότερα παραδείγματα, αλλά και πληροφορίες γι' αυτή τη βιβλιοθήκη, βρίσκονται στην ιστοσελίδα http://hpricot.com/.

```
1: require 'hpricot'
2: require 'open-uri'
3: doc = Hpricot(URI.parse( "http://google.com/" ).read )
4:
5: doc.search( "/html/body//p" )
6: doc.search( "//p" )
7: doc.search( "//p/a" )
8: doc.search( "//a[@src]" )
9: doc.search( "//a[@src='google.com']" )
```

4.5 Nokogiri

Ένας ακόμη *HTML/XML parser* για τη *Ruby* είναι ο *Nokogiri*. Τα πλεονεκτήματα του *Nokogiri* έναντι του *HPricot* είναι η γρηγορότερη φόρτωση και ανάλυση ενός εγγράφου αλλά και η υποστήριξη επιλογέων *CSS3*. Ακολουθεί ένα παράδειγμα που κάνει χρήση αυτής της βιβλιοθήκης:

```
1: require 'nokogiri'
2: require 'open-uri'
3:
4: doc = Nokogiri::HTML(open('http://www.google.com/search?q=tenderlove'))
5:
6: # Αναζήτηση στοιχείων μέσω επιλογέων css
7: doc.css('h3.r a.l').each do |link|
8:   puts link.content
9: end
10:
11: # Αναζήτηση κόμβων μέσω εκφράσεων xpath
12: doc.xpath('//h3/a[@class="l"]').each do |link|
13:   puts link.content
14: end
15:
16: # Συνδυασμός και των δύο.
17: doc.search('h3.r a.l', '//h3/a[@class="l"]').each do |link|
18:   puts link.content
19: end
```

4.6 RubyfulSoup

Η *RubyfulSoup* είναι μια πολύ ισχυρή βιβλιοθήκη της Ruby που προσφέρει παρόμοιες δυνατότητες με την *HTree*. Η *RubyfulSoup* είναι μια πολύ καλή επιλογή γι' αυτούς που δεν είναι εξοικειωμένοι με τις τεχνολογίες *XML* και *XPath*. Είναι ένα "όλα σε ένα", αποτελεσματικό εργαλείο για την προσπέλαση και την εξαγωγή δεδομένων από ιστοσελίδες και με σύνταξη παρόμοια με αυτή της Ruby. Παρακάτω δίνεται το γνωστό μας παράδειγμα με τη χρήση της RubyfulSoup:

```
1: require 'rubygems'
2: require 'rubyful_soup'
3: require 'open-uri'
4:
5: url = "http://www.google.com/search?q=ruby"
6: open(url) {
7:     |page| page_content = page.read()
8:     soup = BeautifulSoup.new(page_content)
9:     result = soup.find_all('a', :attrs => {'class' => 'l'})
10:    result.each { |tag| puts tag['href'] }
11: }
```

Όπως μπορούμε να δούμε, οι διαφορές των *HTree + XPath* με την *RubyfulSoup* είναι ελάχιστες. Στην ουσία οι διαφορές περιορίζονται στη μορφή του ερωτήματος στην ένατη γραμμή του κώδικα, η οποία βρίσκει τους συνδέσμους με τη ζητούμενη ιδιότητα *class*. Η άλλη διαφορά είναι στην ένατη γραμμή, όπου παίρνουμε μόνο την ιδιότητα '*href*' των συνδέσμων.

Θεωρώ την *RubyfulSoup* το ιδανικό εργαλείο για την εξαγωγή δεδομένων από μια μόνο ιστοσελίδα. Τι κάνουμε όμως εάν θέλουμε να εξάγουμε δεδομένα από πολλές ιστοσελίδες, να συμπληρώσουμε φόρμες και να ακολουθήσουμε συνδέσμους; Δεν υπάρχει κανένας λόγος ανησυχίας γιατί η επόμενη βιβλιοθήκη κάνει αυτό ακριβώς το πράγμα.

4.7 Mechanize

Σήμερα η πλειοψηφία των δεδομένων στον ιστό βρίσκεται βαθιά μέσα σε βάσεις δεδομένων. Αυτά είναι προσβάσιμα μέσω ερωτημάτων σε διάφορες φόρμες των ιστοσελίδων. Για παράδειγμα, εάν θέλετε να μάθετε πληροφορίες για μια πτήση από τη Θεσσαλονίκη στη Βαρκελώνη, δεν θα το ψάξετε στο *Google*, αλλά θα επισκεφθείτε έναν ιστότοπο όπως το *airtickets.gr*, θα συμπληρώσετε την φόρμα στη κεντρική σελίδα και θα κάνετε αποστολή του ερωτήματός σας πατώντας το κουμπί «Αναζήτηση». Τα αποτελέσματα που θα εμφανιστούν δεν είναι διαθέσιμα μέσω μιας στατικής ιστοσελίδας αφού δημιουργούνται δυναμικά ανάλογα με το ερώτημα που θέσατε στη φόρμα. Εδώ ακριβώς χρειαζόμαστε εργαλεία όπως το *Mechanize*.

Το *Mechanize* μαζί με το *Watir* που θα δούμε σε επόμενη παράγραφο ανήκουν στα εργαλεία εξόρυξης δεδομένων τα οποία προσομοιώνουν τη λειτουργία ενός browser. Για να δούμε πώς ακριβώς λειτουργεί το *Mechanize* θα αλλάξουμε λίγο το γνωστό μας παράδειγμα:

```
1: require 'rubygems'
2: require 'mechanize'
3:
4: agent = WWW::Mechanize.new
5: page = agent.get('http://www.google.com')
6:
7: search_form = page.forms.with.name("f").first
8: search_form.fields.name("q").first.value = "ruby"
9: search_results = agent.submit(search_form)
```

Αυτό το απλό παράδειγμα απεικονίζει τη διαφορά του *RubyfulSoup* με το *Mechanize*. Το *Mechanize* είναι σε θέση να οδηγεί τον φυλλομετρητή ιστού όπως ακριβώς θα έκανε και ένας άνθρωπος: Συμπληρώνει το πεδίο της φόρμας και κάνει κλικ στο κουμπί 'Αναζήτηση'. Δεν μπορεί όμως να εξάγει τους συνδέσμους όπως κάναμε με το *RubyfulSoup*. Επομένως ο συνδυασμός και τον δύο είναι σχεδόν απαραίτητος.

4.8 ScRUBYt!

Η βιβλιοθήκη *ScRUBYt*! είναι ένα ευέλικτο προγραμματιστικό περιβάλλον για τη δημιουργία προγραμμάτων εξόρυξης δεδομένων από το διαδίκτυο, εύκολο στην εκμάθηση αλλά και στη χρήση. Η βιβλιοθήκη είναι γραμμένη σε *Ruby* και βασίζεται στο *Hpricot* και το *Mechanize*. Η *ScRUBYt*! μπορεί να πλοηγηθεί στον ιστό (κάνοντας κλικ σε συνδέσμους και συμπληρώνοντας πεδία φορμών, χάρη στο *mechanize*), αλλά και να εξάγει και να σώζει δεδομένα από τις σελίδες που επισκέπτεται πάρα πολύ εύκολα (χάρη στο *Hpricot*).

Ας δούμε όμως δύο παραδείγματα χρήσης της βιβλιοθήκης *ScRUBYt*! για να καταλάβουμε πώς ακριβώς λειτουργεί. Για αρχή ας δούμε ένα απλό παράδειγμα. Αν θέλουμε να δοκιμάσουμε το παρακάτω κομμάτι κώδικα, δημιουργούμε ένα αρχείο με κατάληξη *.rb* και του δίνουμε ένα όνομα (π.χ. *scrubyt_test.rb*)

```
1: require 'rubygems'
2: require 'scrubyt'
3:
4:  data = Scrubyt::Extractor.define do
5:  fetch 'http://google.com'
6:  title '//head/title'
7:  end
8:
9: data.to_xml.write($stdout, 1)
```

Αυτό που κάνει το παραπάνω κομμάτι κώδικα είναι να εκτυπώνει το τίτλο της κεντρικής σελίδας του *Google*. Στο επόμενο παράδειγμα, εκτός απ' το να εξάγουμε κάποια στοιχεία από τη σελίδα του Google, θα δούμε και τη χρήση του *Mechanize* μέσω της βιβλιοθήκης *ScRUBYt*! συμπληρώνοντας το πεδίο αναζήτησης της σελίδας του *Google*:

```
1: equire 'rubygems'
2: require 'scrubyt'
3:
4: google_data = Scrubyt::Extractor.define do
5:     fetch 'http://www.google.com/ncr'
6:     fill_textfield 'q', 'ruby'
7:     submit
8:
9:     result 'Ruby Programming Language' do
10:      link 'href', :type => :attribute
11:    end
12: end
13:
```

```
14: google_data.to_xml.write($stdout, 1)
15: Scrubyt::ResultDumper.print_statistics(google_data)
```

Παρακάτω μπορείτε να δείτε το αποτέλεσμα εκτέλεσης του προγράμματος:

```
1: <root>
2:   <result>
3:     <link>http://www.ruby-lang.org/</link>
4:   </result>
5:   <result>
6:     <link>http://www.ruby-lang.org/en/20020101.html</link>
7:   </result>
8:   <result>
9:     <link>http://en.wikipedia.org/wiki/Ruby_programming_language</link>
10:  </result>
11:  <result>
12:    <link>http://en.wikipedia.org/wiki/Ruby</link>
13:  </result>
14:  <result>
15:    <link>http://www.rubyonrails.org/</link>
16:  </result>
17:  <result>
18:    <link>http://www.rubycentral.com/</link>
19:  </result>
20:  <result>
21:    <link>http://www.rubycentral.com/book/</link>
22:  </result>
23:  <result>
24:    <link>http://www.w3.org/TR/ruby/</link>
25:  </result>
26:  <result>
27:    <link>http://poignantguide.net/</link>
28:  </result>
29:  <result>
30:    <link>http://www.zenspider.com/Languages/Ruby/QuickRef.html</link>
31:  </result>
32: </root>
33:
34:  result extracted 10 instances.
35:  link extracted 10 instances.
```

Παρατηρούμε ότι το δεύτερο κομμάτι κώδικα είναι σχετικά μεγαλύτερο από τα παρόμοια κομμάτια κώδικα που είδαμε στις προηγούμενες βιβλιοθήκες της Ruby. Υπάρχουν όμως μερικά σημεία που παρουσιάζουν ενδιαφέρον. Καταρχήν, όπως προείπαμε, δεν 'φορτώνουμε' τη σελίδα με τα αποτελέσματα του Google κατευθείαν, αλλά έχουμε τη δυνατότητα να συμπληρώσουμε το πεδίο αναζήτησης και να κάνουμε submit τη φόρμα. Το άλλο ενδιαφέρον στοιχείο είναι ότι δεν χρειάζεται να κάνουμε απαραίτητα χρήση της XQuery για την εύρεση κάποιων στοιχείων στη σελίδα, αλλά αρκεί η αντιγραφή και επικόλληση αυτών στον κώδικά μας. Τέλος, ένα βοηθητικό στοιχείο είναι ότι μπορούμε να έχουμε κάποια απλά στατιστικά στοιχεία σχετικά με το πόσα αντικείμενα κατεβάσαμε από την ιστοσελίδα.

Τα δύο παραδείγματα που είδαμε για τη βιβλιοθήκη ScRUBYt! είναι μόνο η κορυφή του παγόβουνου. Υπάρχουν πολλά περισσότερα που μπορείτε να κάνετε με αυτή τη

βιβλιοθήκη. Για περισσότερες πληροφορίες, tutorials αλλά και νέες εκδόσεις μπορούμε να επισκεφτούμε τη σελίδα http://scrubyt.org/.

4.9 Watir και Selenium

Πολλές φορές χρειάζεται να γράψουμε scripts που καθοδηγούν κάποιον φυλλομετρητή ιστού. Τα δύο πιο γνωστά εργαλεία στη Ruby που κάνουν αυτό ακριβώς είναι το Watir (wtr.rubyforge.org) και το Selenium (www.openqa.org/selenium). Και τα δύο εργαλεία είναι ανοιχτού κώδικα. Παρακάτω βλέπετε ένα μικρό παράδειγμα για την καθοδήγηση του Internet Explorer με το Watir:

```
1: def Watir test
2:       br = IE.new
3:       br.goto('http://www.google.com' )
4:       #Αν δούμε τον HTML κώδικα της σελίδας, παρατηρούμε οτι
5:       #η Google ονομάζει το πεδίο αναζήτησης 'q'
6:       br.text_field(:name, "q" ).set("ΤΕΙ Θεσσαλονίκης" )
7:       # 'btnI' είναι το όνομα του κουμπιού "I'm Feeling Lucky".
8:       br.button(:name, "btnI" ).click
9: end
```

Το Watir ονομάζεται έτσι από τα αρχικά το λέξεων *'Web Application Testing in Ruby'* Όταν πρωτοεκδόθηκε μπορούσε να χρησιμοποιηθεί μόνο στα Windows οδηγώντας τον Internet Explorer όπως θα έκανε και ένας άνθρωπος (κάνοντας κλικ σε συνδέσμους και κουμπιά, συμπληρώνοντας πεδία σε φόρμες κ.α.). Τελικά, όμως, εκδόθηκε μια νέα έκδοση, το FireWatir, που υποστηρίζει Linux και Mac OS μέσω του Firefox. Ενδεχομένως, το Watir να είναι ποιο πλούσιο σε χαρακτηριστικά και δυνατότητες από το Mechanize.

Επίλογος

Σε αυτό το κεφάλαιο αναφερθήκαμε σε όλες τις βιβλιοθήκες τις Ruby οι οποίες μπορούν να χρησιμοποιηθούν για τη δημιουργία ενός προγράμματος εξόρυξης δεδομένων από το διαδίκτυο. Χρησιμοποιώντας μία μόνο απ' όλες αυτές τις βιβλιοθήκες σίγουρα δεν μπορούμε να φτιάξουμε ένα απαιτητικό πρόγραμμα. Οπότε ο συνδυασμός αυτών πολλές φορές επιβάλλεται.

Για ένα πολύπλοκο σενάριο, ο συνδυασμός των WWW::Mechanize ή Watir (για την πλοήγηση), της RubyfulSoup (για την εξόρυξη των δεδομένων) και των HTree+REXML (για τις περιπτώσεις που η RubyfulSoup δεν μπορεί να μας βοηθήσει) είναι ο ιδανικός.

ΚΕΦΑΛΑΙΟ 5
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ PERL

Εισαγωγή

Το μεγάλο πλεονέκτημα της *Perl* είναι το μεγάλο πλήθος από επεκτάσεις (*modules*) που διαθέτει, επεκτείνοντας με αυτές τις δυνατότητες της γλώσσας. Και στη περίπτωση του *web scraping* αυτό δεν αποτελεί εξαίρεση. Η *Perl* διαθέτει μια πληθώρα από modules, τα οποία μπορούν να μας βοηθήσουν και σε αυτό το τομέα.

Η λέξη *module* δεν χρειάζεται να μας φοβίζει, καθώς δεν είναι τίποτε άλλο παρά ένα κομμάτι κώδικα σε *Perl,* το οποίο έχει γραφτεί από κάποιον άλλον και μπορεί να δώσει πρόσθετη λειτουργικότητα στο δικό μας πρόγραμμα. Σε αυτό το κεφάλαιο θα αναλύσουμε τα σημαντικότερα από αυτά για το τομέα που μας ενδιαφέρει – την εξόρυξη δεδομένων από το διαδίκτυο.

5.1 Απλή ανάκτηση ιστοσελίδων με το LWP::Simple

Το *LWP* είναι μια ομάδα από modules τις *Perl* για τη πρόσβαση σε δεδομένα του Ιστού. Κάθε *module* έχει και ένα ξεχωριστό *documentation*. Βέβαια υπάρχουν τόσα πολλά modules σε αυτή τη βιβλιοθήκη που είναι δύσκολο ακόμα και να αποφασίσεις πού να ψάξεις για να βρεις πληροφορίες, ακόμα και για το πιο απλό πράγμα.

Αν θέλουμε απλά να έχουμε πρόσβαση σε ένα *url* μπορούμε απλά να χρησιμοποιήσουμε τις συναρτήσεις του *LWP*::*Simple*. Σε ένα πρόγραμμα *Perl*, μπορούμε απλά να καλέσουμε τη συνάρτηση *get($url*), όπου *url* είναι η διεύθυνση της σελίδας στην οποία υπάρχουν τα δεδομένα που μας ενδιαφέρουν. Δείτε, για παράδειγμα, το παρακάτω κομμάτι κώδικα:

```
1: #!/usr/bin/perl
2: use strict;
3: use warnings;
4: use LWP::Simple;
5:
6: my $url = 'http://www.it.teithe.gr/';
7: my $content = get($url) or die 'Unable to get page';
8:
9: exit 0;
```

Όπως μπορούμε να δούμε η χρήση του *LWP*::*Simple* είναι απλή. Αυτό που κάνουμε στο παραπάνω πρόγραμμα είναι να αποθηκεύουμε τα περιεχόμενα της σελίδας σε μια μεταβλητή, την $content. Εκτός από την get(), δυο ακόμα χρήσιμες μέθοδοι του *LWP*::*Simple* είναι οι *getprint*() και *getstore*() που εκτυπώνουν και αποθηκεύουν αντίστοιχα τα περιεχόμενα της ιστοσελίδας.

5.2 HTML::TreeBuilder και HTML::Element

Το *HTML*::*TreeBuilder* και το *HTML*::*Element* περιλαμβάνονται στο *HTML-Tree* το οποίο είναι μια σουίτα από modules για τη δημιουργία δέντρων από κώδικα HTML. Το *HTML*::*TreeBuilder* είναι το module που δημιουργεί το δέντρο (χρησιμοποιεί το *HTML*::*Parser* για να διαχωρίσει τον html κώδικα σε κομμάτια). Το δέντρο το οποίο θα προκύψει αποτελείται από κόμβους οι οποίοι είναι στιγμιότυπα της κλάσης *HTML*::*Element*.

Τα δέντρα είναι μια καλή μέθοδος για να αναπαραστήσουμε HTML κώδικα. Για παράδειγμα, το στοιχείο <*head*> είναι παιδί του στοιχείου <*html*> και πατέρας του στοιχείου <*title*>. Σε καθέναν από τους κόμβους του δέντρου μπορούμε να θέσουμε ερωτήματα για το γονέα (*parent*), τα αδέρφια (*siblings*) ή τα παιδιά (*children*) του. Στο παρακάτω κομμάτι κώδικα χρησιμοποιούμε τη βιβλιοθήκη *HTML*::*Simple* για να ανακτήσουμε τα περιεχόμενα της σελίδας του Google και μετά δημιουργούμε το δέντρο στην όγδοη γραμμή μέσω της μεθόδου *new_from_content().*

```
1: #!/usr/bin/perl -w
2: use strict;
3: use LWP::Simple;
4: use HTML::TreeBuilder;
5:
6: my $url = 'http://www.it.teithe.gr/';
7: my $page = get( $url ) or die 'Unable to get page';
8: my $p = HTML::TreeBuilder->new_from_content( $page );
```

Μια ακόμα χρήσιμη μέθοδο είναι η *look_down()* η οποία ξεκινάει από τη κορυφή του δέντρου και κατεβαίνοντας βλέπει ποιοι κόμβοι ταιριάζουν στις δοθείσες συνθήκες. Για παράδειγμα, στο παρακάτω κομμάτι κώδικα βρίσκουμε και αποθηκεύουμε όλους τους συνδέσμους που ταιριάζουν σε μια κανονική έκφραση:

```
my @links = $p->look_down(
    _tag => 'a',
    href => qr{^ \Q http://www.it.teithe.gr/\E \w+ $}x
);
```

5.3 WWW::Mechanize

Το *WWW*::*Mechanize* το είδαμε και στη *Ruby, και* θα το δούμε και σε επόμενο κεφάλαιο στη Python. Και στις τρεις αυτές γλώσσες προγραμματισμού έχει παρόμοια χαρακτηριστικά. Η βασική του λειτουργία είναι η αυτοματοποίηση της πλοήγησης και της αλληλεπίδρασης με τις ιστοσελίδες. Ας δούμε λοιπόν ένα *script* που κάνει χρήση του *WWW*::*Mechanize,* για να καταλάβουμε πώς μπορούμε να το χρησιμοποιήσουμε στη *Perl*:

```
1: #!/usr/bin/perl
2: use WWW::Mechanize;
3: $url = 'http://www.google.com';
4: $m->get($url);
5: $link = $m->find_link(text => 'Advanced Search');
6: print "The Google advanced search URL is: $link->url()\n";
```

Όπως με κάθε άλλο *module* στη *Perl,* συμπεριλαμβάνουμε το *WWW*::*Mechanize* στο κώδικά μας, χρησιμοποιώντας τη λέξη κλειδί '*use*' (γραμμή 2). Για να ανακτήσουμε μια ιστοσελίδα χρησιμοποιούμε τη μέθοδο *get()* (γραμμή 4). Τέλος για να βρούμε έναν σύνδεσμο στην ιστοσελίδα χρησιμοποιούμε τη μέθοδο *find_link()* (γραμμή 5). Υπάρχουν πολλοί τρόποι για να βρούμε έναν σύνδεσμο. όπως για παράδειγμα μπορούμε να χρησιμοποιήσουμε κείμενο, κανονική έκφραση, url ή ιδιότητες όπως το *id* και το *class*. Παρακάτω μπορούμε να δούμε μερικά παραδείγματα:

```
$m->find_link(text => 'string');
$m->find_link(text_regex => qr/regex/i);
$m->find_link(url => 'http://url/');
$m->find_link(url_regex => qr/domain/i);
$m->find_link(text => 'string', url => 'url');
```

Αφού βρούμε τον σύνδεσμο μπορούμε να τον ακολουθήσουμε χρησιμοποιώντας τη μέθοδο *follow_link().* Στο επόμενο παράδειγμα θα δούμε πως μπορούμε να συμπληρώσουμε και να υποβάλουμε μια φόρμα με το WWW::Mechanize.

```
1: #!/usr/bin/perl
2: use WWW::Mechanize;
3: $url = 'http://www.google.com/ncr';
4: $m->get($url);
5: $m->form_name('f');
6: $m->field('q', 'test search');
7: $response = $m->submit();
8: print $response->content();
```

Όπως μπορούμε να δούμε, όλα γίνονται μέσα σε λίγες γραμμές κώδικα. Αφού ανακτήσουμε τη κεντρική σελίδα του Google, βρίσκουμε τη φόρμα βάσει του ονόματός της (αυτό μπορεί να γίνει και βάσει του id ή προσπελάζοντας όλες τις φόρμες της σελίδας και καταλήγοντας σε αυτή που ζητάμε), συμπληρώνουμε το πεδίο με το όνομα 'q' και τέλος υποβάλλουμε τη φόρμα με τη βοήθεια της μεθόδου *submit().*

Στο παράδειγμά μας, αντί της μεθόδου *submit,* η οποία υποβάλει κατευθείαν τη φόρμα, θα μπορούσαμε να χρησιμοποιήσουμε τη μέθοδο *click_button*() για να κάνουμε κλικ το κουμπί '*Google Search*' ή το '*I'm Feeling Lucky*'. Αυτό θα το κάναμε με τις παρακάτω εντολές:

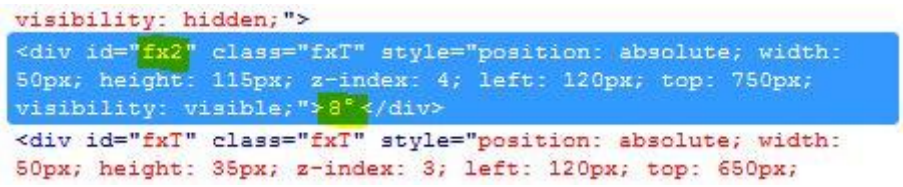
```
$m->click_button(name => 'btnG'); #για το κουμπί 'Google Search'
$m->click_button(name => 'btnI'); #για το κουμπί 'I'm Feeling Lucky'
```

5.4 Web::Scraper

Το *Web*::*Scraper* θα λέγαμε ότι είναι μια εργαλειοθήκη για την εξόρυξη δεδομένων από το διαδίκτυο, εμπνευσμένη από τη βιβλιοθήκη *scrapi.rb* της Ruby. Δεν υπάρχει όμως καλύτερος τρόπος για να καταλάβουμε πως λειτουργεί από ένα απλό παράδειγμα. Στο παρακάτω κομμάτι κώδικα θα πάρουμε και θα εκτυπώσουμε τη τιμή της θερμοκρασίας για τη Θεσσαλονίκη από την ιστοσελίδα *meteo.gr*:

```
1: #!/usr/bin/perl
2: use strict;
3: use warnings;
4: use Web::Scraper;
5: use URI;
6: my $s = scraper {
7:     process "div#fx2", temperature => 'TEXT';
8:     result 'temperature';
9: };
10: my $uri = URI->new('http://www.meteo.gr/');
11: print $s->scrape($uri);
```

Στη παρακάτω εικόνα βλέπουμε ότι η θερμοκρασία για τη πόλη της Θεσσαλονίκης βρίσκεται μέσα σε ένα *div* με *id* ίσο με *'fx2'*. Τώρα ο κώδικας γίνεται σίγουρα πιο κατανοητός. Στην έβδομη γραμμή του κώδικα, στην οποία υπάρχει και η μεγαλύτερη δυσκολία, με τη χρήση του άγκιστρου επιλέγουμε το div με *id* ίσο με *'fx2'* και έπειτα δίνουμε τη τιμή ***TEXT*** στη μεταβλητή *temperature*. Το ***TEXT*** είναι το κείμενο που υπάρχει μεταξύ των ετικετών ανοίγματος και κλεισίματος του *div*.



Εικόνα 3 – Εύρεση κώδικα από το meteo.gr με το firebug

Αν θέλαμε να επιλέξουμε ένα *html* στοιχείο βάση της ιδιότητας *class* και όχι τις *id,* τότε αντί για το άγκιστρο(#) θα έπρεπε να χρησιμοποιήσουμε τελεία(.). Για παράδειγμα:

```
process "div.ClsName" …
```

Όπως είπαμε και πριν το *TEXT* είναι το κείμενο μεταξύ δύο ετικετών. Αν όμως, αντί γι' αυτό θέλαμε τη τιμή μιας ιδιότητας τότε θα έπρεπε να χρησιμοποιήσουμε το σύμβολο at (@) όπως παρακάτω:

```
# <img src="www.test.com/myfile.jpg" title="Hello" />
'titles[]' => '@title';);
```

Επίλογος

Η ευελιξία της Perl, αλλά και το πλούσιο σύνολο των διαθέσιμων μονάδων (modules) που διαθέτει, την κάνει ένα πολύ καλό εργαλείο για την ανάπτυξη προγραμμάτων εξόρυξης δεδομένων. Βέβαια, πολλές φορές ο ακατανόητος κώδικας της Perl απωθεί τους προγραμματιστές που δεν είναι εξοικειωμένοι μαζί της από το να τη χρησιμοποιήσουν.

ΚΕΦΑΛΑΙΟ 6
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ PYTHON

Εισαγωγή

Η εξόρυξη δεδομένων από το διαδίκτυο με τη γλώσσα προγραμματισμού Python είναι αρκετά εύκολη. Η χρήση κανονικών εκφράσεων είναι μια πρώτη επιλογή. Βέβαια η χρήση κανονικών εκφράσεων, όπως και σε κάθε άλλη γλώσσα προγραμματισμού, παρουσιάζει κάποια μειονεκτήματα τα οποία θα δούμε σε αυτό το κεφάλαιο.

Για την αποφυγή των αδυναμιών και των προβλημάτων που εγκυμονεί η χρήση των κανονικών εκφράσεων και στη Python, υπάρχουν βιβλιοθήκες που κάνουν την εξόρυξη δεδομένων εύκολη διαδικασία. Οι περισσότερο ευρέως χρησιμοποιούμενες βιβλιοθήκες είναι η *BeautifulSoup* και η Mechanize. Αυτές τις δύο, αλλά και μερικές ακόμα θα τις δούμε παρακάτω.

6.1 Η βιβλιοθήκη urllib και οι κανονικές εκφράσεις

Η βιβλιοθήκη *urllib* μας βοηθάει να κατεβάσουμε το πηγαίο κώδικα *HTML* μιας σελίδας από το διαδίκτυο. Αφού πάρουμε τα περιεχόμενα της σελίδας, μπορούμε να χρησιμοποιήσουμε κανονικές εκφράσεις (*regular expression*) για να εξάγουμε τη πληροφορία που επιθυμούμε.

Ας πούμε για παράδειγμα ότι θέλουμε να εξάγουμε όλους τους συνδέσμους της βασικής σελίδας του χρυσού οδηγού. Παρακάτω βλέπουμε ένα πρόγραμμα το οποίο χρησιμοποιεί τις βιβλιοθήκες '*urllib*' και '*re*' γι' αυτόν ακριβώς το σκοπό:

```
1: from urllib import urlopen
2: import re
3:
4: p = re.compile('<a .*? href="(.*?)">(.*?)</a>')
5: text = urlopen('http://www.xo.gr/').read()
6: for url, name in p.findall(text):
7:     print '%s (%s)' % (name, url)
```

Ένα ενδεικτικό απόσπασμα από το αποτέλεσμα της εκτέλεσης του προγράμματος είναι το ακόλουθο:

Επικοινωνία (default_076.html)
Βοήθεια (default_077.html)
Συνήθεις Ερωτήσεις (default_078.html)
Σχόλια για "Δικός μου Χρυσός Οδηγός" (default_079.html)

Όμως υπάρχουν μερικές βασικές αδυναμίες αυτής της προσέγγισης:

1 Οι κανονικές εκφράσεις δεν είναι απολύτως κατανοητές. (Όσο πιο περίπλοκος είναι ο κώδικας HTML τόσο πιο ακατανόητες και περίπλοκες γίνονται και οι κανονικές εκφράσεις) Για ακόμα πιο περίπλοκο κώδικα *HTML*, οι κανονικές εκφράσεις γίνονται ακόμα πιο ακατανόητες και περίπλοκες.
2 Μια κανονική έκφραση βασίζετε εξ ολοκλήρου πάνω στον *HTML* κώδικα, με αποτέλεσμα μια μικρή αλλαγή σε αυτόν να προκαλέσει πρόβλημα στο πρόγραμμα.
3 Δεν διαχειρίζεται σωστά κάποιες ιδιαιτερότητες της *HTML*, όπως για παράδειγμα οι χαρακτήρες *&* και * *.

6.2 HTMLParser

Στη έκδοση 3 της *Python* αυτή η βιβλιοθήκη ονομάζεται html.parser. Η βιβλιοθήκη *HTMLParser* ορίζει τη μέθοδο *HTMLParser,* η οποία μπορεί να χρησιμοποιηθεί για να αναλύσει *HTML* και *XHTML* αρχεία. Δείτε παρακάτω ένα απλό παράδειγμα:

```
1: import HTMLParser
2: h = HTMLParser.HTMLParser()
3: h.feed('<html></html>')
4: f= h.get starttag text()
5: print f
6: h.close()
```

Βέβαια, οι ικανότητες του *HTMLParser* είναι αρκετά περιορισμένες. Στη πραγματικότητα, με πολύ περίπλοκο ή δύσμορφο κώδικα αυτή η βιβλιοθήκη μπορεί να αποτύχει. Γι' αυτό το λόγο παρακάτω θα δούμε τη βιβλιοθήκη *BeautifulSoup* που αντιμετωπίζει με τον καλύτερο δυνατό τρόπο αυτού του είδους τα προβλήματα.

6.3 BeautifulSoup

Η βασική βιβλιοθήκη της Python περιέχει το *HTMLParser* για την επεξεργασία και την ανάλυση HTML κώδικα. Ωστόσο, παρουσιάζει κάποια προβλήματα με HTML κώδικα ο οποίος δεν είναι καλά δομημένος. Υπάρχει όμως μια βιβλιοθήκη της Python, η οποία δουλεύει και με κώδικα που μπορεί να είναι περίπλοκος ή/και δύσμορφος, και αυτή η βιβλιοθήκη είναι η *BeautifulSoup*.

Με τη *BeautifulSoup* εγκατεστημένη, η εξαγωγή πληροφοριών από μια ιστοσελίδα είναι εξαιρετικά εύκολη. Μας επιτρέπει να έχουμε πρόσβαση σε κάθε στοιχείο της σελίδας με βάση το τύπο του, το *id* του, ή οποιαδήποτε από τις ιδιότητές του. Δείτε για παράδειγμα το παρακάτω απόσπασμα κώδικα:

```
1: from BeautifulSoup import BeautifulSoup
2: import re
3: import urllib2
4:
5: url = 'http://blogsearch.google.com/blogsearch?q=python'
6: response = urllib2.urlopen(url)
7: html = response.read()
8:
9: soup = BeautifulSoup(html)
```

```
10: links = soup.findAll('a', id=re.compile("^p-"))
11: for link in links:
12:     print link['href']
```

Το αποτέλεσμα της εκτέλεσης αυτού του script θα είναι παρόμοιο με το παρακάτω:

```
http://learnpython.wordpress.com/2010/12/12/python-in-the-news/
http://geert.vanderkelen.org/post/435/
http://www.h-online.com/news/item/Python-3-2-Beta-1-arrives-1148567.html
http://www.listshow.net/201012/debian-devel-changes/9840-accepted-python-apt-071001-source-all-amd64.html
http://www.daniweb.com/forums/thread331122.html
http://www.holovaty.com/writing/python-outlook-web-access/
http://www.blog.pythonlibrary.org/2010/12/09/mini-book-review-mysql-for-python/
http://www.findpdf.us/details-187317.pdf?core-python-programming-wesley-j-chun
http://www.dosyatube.com/python-in-a-nutshell-hotfileserve-download/
http://warezhere.com/python-in-a-nutshell-rapidshare-megaupload-rar.html
```

Ας δούμε όμως πώς ακριβώς λειτουργεί αυτό το κομμάτι κώδικα. Αφού κατεβάσαμε τον πηγαίο κώδικα της σελίδας του Google Blog Search με το ερώτημα 'python' μέσω της βιβλιοθήκης *urllib2,* περάσαμε όλον αυτόν τον κώδικα στο *BeautifulSoup* (γραμμή 9). Ο λόγος που το κάνουμε αυτό είναι γιατί δεν θέλουμε ολόκληρο τον κώδικα, αλλά μερικά αποσπάσματα που θα τα εξάγουμε με τη βοήθεια αυτής της βιβλιοθήκης. Αυτό γίνεται στην δέκατη γραμμή, όπου βρίσκουμε όλους τους συνδέσμους στους οποίους το id τους ξεκινάει με 'p-'. Για να ξεχωρίσουμε αυτούς τους συνδέσμους από τους υπόλοιπους, χρησιμοποιούμε μια κανονική έκφραση. Στο τέλος του προγράμματος εκτυπώνουμε την ιδιότητα *href* κάθε συνδέσμου που βρήκαμε.

Το μόνο μειονέκτημα αυτής της βιβλιοθήκης είναι ότι στην τελευταία έκδοση της Python, δηλαδή την έκδοση 3, παρουσιάζει αρκετά προβλήματα. Οπότε, εάν θέλουμε να χρησιμοποιήσουμε τη συγκεκριμένη βιβλιοθήκη, καλό θα ήταν να το κάνουμε με παλαιότερες εκδόσεις της Python. Τη βιβλιοθήκη μπορούμε να τη κατεβάσουμε από τη σελίδα http://crummy.com/software/BeautifulSoup ως απλό αρχείο και να το τοποθετήσουμε στον κατάλογο εργασίας μας ή στο κατάλογο της Python (π.χ. C:\Python\Lib\site_packages).

6.4 lxml

Μια άλλη καλή επιλογή parser html και xml αρχείων για τη Python, εκτός του BeautifulSoup, είναι το lxml. Η βιβλιοθήκη lxml είναι πολύ πιο γρήγορη από τη BeautifulSoup. Η επιλογή στοιχείων της σελίδας γίνεται σχετικά εύκολα με τη βοήθεια XPath εκφράσεων και CSS selectors. Αλλά έχει και αδυναμίες. Η σημαντικότερη αδυναμία της είναι ότι μπορεί να παρουσιάσει προβλήματα με html κώδικα ο οποίος δεν είναι σωστά μορφοποιημένος, κάτι στο οποίο η βιβλιοθήκη BeautifulSoup δεν παρουσιάζει ιδιαίτερο πρόβλημα. Ένα άλλο μικρό μειονέκτημα είναι η δυσκολία στην εγκατάσταση της βιβλιοθήκης. Ας δούμε ένα παράδειγμα για να καταλάβουμε πως λειτουργεί:

```
1: import lxml.etree
2: import lxml.html
3:
4: samplehtml = """<html><body>
5: <h1>Παράδειγμα lxml</h1>
6: <ul class="numbers">
7:   <li class="1">Ένα</li>
8:   <li class="2">Δύο</li>
9:   <li class="1" id="third">Τρία</li>
10: </ul>
11: </body></html>"""
12:
13: root = lxml.html.fromstring(samplehtml)
14:
15: #Για να φορτώσουμε τη σελίδα απευθείας από ένα url:
16: #root = lxml.html.parse(url).getroot()
17:
18: #χρησιμοποιούμε τη μέθοδο cssselect για να επιλέξουμε στοιχεία
19: print root.cssselect("li.1")              # επιστέφει 2 στοιχεία
20: print root.cssselect("ul #third")         # επιστέφει 1 στοιχείο
21: print root.cssselect(".numbers li")       # επιστέφει 3 στοιχεία
```

Μια δυνατότητα του lxml που θεωρώ αρκετά σημαντική είναι ότι μπορεί να μετατρέψει όλους τους συνδέσμους της σελίδας σε απόλυτους. Απόλυτο ονομάζεται ένα url το οποίο αρχίζει με το «http://». Σε αντίθετη περίπτωση ονομάζεται σχετικό url. Αυτό γίνεται με τη βοήθεια της μεθόδου *make_links_absolute().* Και δεν είναι μόνο οι σύνδεσμοι <a> που μετατρέπονται σε απόλυτους αλλά και τα μονοπάτια (paths) προς τα αρχεία μορφοποίησης, τα paths τον ιδιοτήτων background κ.α.

6.5 Mechanize

Τη βιβλιοθήκη mechanize την είδαμε και στη γλώσσα προγραμματισμού Ruby αλλά και στη Perl. Η λειτουργικότητά της είναι παρόμοια και στις τρεις αυτές γλώσσες,, οπότε δεν θα επεκταθούμε εδώ στη περιγραφή των δυνατοτήτων της, παρά μόνο θα δούμε πώς μπορούμε να τη χρησιμοποιήσουμε στη Python μέσω ενός απλού παραδείγματος.

```
1: import mechanize
2:
3: # Browser
4: br = mechanize.Browser()
5:
6: r = br.open('http://google.com')
7: html = r.read()
8:
9: # εμφάνιση του πηγαίου κώδικα
10: print html
11:
12: # εμφάνιση του τίτλου της σελίδας
13: print br.title()
14:
15: # εμφάνιση των φορμών της σελίδας
16: for f in br.forms():
17:     print f
18:
```

```
19: # επιλογή της πρώτης φόρμας
20: br.select_form(nr=0)
21:
22: # συμπλήρωση της φόρμας με τη λέξη python
23: br.form['q']='python'
24:
25: # υποβολή της φόρμας
26: br.submit()
```

Στο παράδειγμά μας είδαμε τις μεθόδους *read()* για την ανάγνωση του κώδικα μιας σελίδας, την *title()* για την εμφάνιση του τίτλου της σελίδας, την *forms()* η οποία επιστρέφει όλες τις φόρμες της σελίδας και τέλος την *submit()* η οποία υποβάλει τη φόρμα. Εκτός από αυτές, μερικές ακόμα χρήσιμες μέθοδοι του mechanize είναι οι: *geturl*() η οποία επιστρέφει το *url* της σελίδας, η *follow_link*() για να ακολουθήσουμε έναν σύνδεσμο και η *back*() για να επιστρέψουμε στην προηγούμενη σελίδα. Στην ιστοσελίδα *http*://*wwwsearch*.*sourceforge*.net/*mechanize*/ μπορούμε να κατεβάσουμε την βιβλιοθήκη, αλλά και να δούμε μερικά ακόμα χρήσιμα παραδείγματα για τη χρήση της.

6.6 scrape.py

Υπάρχουν κι άλλα εργαλεία για εξόρυξη δεδομένων στην *Python*. Ένα από αυτά είναι το *scrape*.*py* του *Ka-Ping Yee* (μπορείτε να το βρείτε στη σελίδα *http://zesty*.*ca*/*python*). Με το *scrape*.*py* μπορούμε να ανοίξουμε σελίδες, να ακολουθήσουμε συνδέσμους και να υποβάλουμε φόρμες, όπως κάνουμε και με το mechanize. Ωστόσο, σε αντίθεση με το *BeautifulSoup* που μπορεί να χρησιμοποιηθεί για να κάνει *parse* αρχεία τα οποία είναι κατεβασμένα τοπικά, το *scrape*.*py* λειτουργεί μόνο *on-line*.

Η κύρια διαφορά του scrape.py με άλλος *parsers* είναι ότι δεν αναλύει τη σελίδα σε δέντρο, αλλά τη χωρίζει σε περιφέρειες (*regions*), έτσι ώστε να μπορεί να χειριστεί σελίδες με άσχημα μορφοποιημένη σύνταξη. Μπορούμε να εντοπίσουμε το περιεχόμενο της σελίδας σύμφωνα με κοντινό του κείμενο, ετικέτες, ή ακόμα και σχόλια. Παρακάτω θα δούμε με απλές εντολές πώς μπορούμε να χρησιμοποιήσουμε αυτό το *module*.

```
1: >>> from scrape import *
2: >>> s.go('http://zesty.ca/')
3: <Region 0:17780>
```

Αφού κάνουμε *import* τη βιβλιοθήκη (γραμμή 1) περνούμε την κεντρική σελίδα του ιστοχώρου του δημιουργού της βιβλιοθήκης χρησιμοποιώντας τη συνάρτηση *go*(*url*). Η βιβλιοθήκη παρέχει ένα *default* αντικείμενο συνόδου (*Session*) μέσω της μεταβλητής s. Το αποτέλεσμα είναι ένα αντικείμενο *Region* το οποίο περιέχει το σύνολο των περιεχομένων της σελίδας (17780 bytes).

```
>>> d = s.doc
>>> print d.content[:70]
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
<html
>>> d.text[:30]
u'Ka-Ping Yee\nKa-Ping Yee pingze'
```

Η ιδιότητα doc κρατάει και αυτή το *html* έγγραφο. Κάθε γραμμή του εγγράφου είναι διαθέσιμη μέσω της ιδιότητας *content*, ενώ το κείμενο της σελίδας είναι διαθέσιμο μέσω της ιδιότητας *text.*

```
>>> t = d.first('title')
>>> t
<Region 247:258 title>
>>> t.tagname
'title'
>>> t.text
u'Ka-Ping Yee'
```

Καλέσαμε την μέθοδο *first*(*tag*) για να βρούμε το πρώτο μπλοκ που προσδιορίζεται από τις ετικέτες έναρξης και κλεισίματος της ετικέτας που προσδιορίσαμε μέσω της μεθόδου. Το αποτέλεσμα είναι ένα αντικείμενο *Region,* το οποίο περιέχει πληροφορίες για το συγκεκριμένο *tag*. Για παράδειγμα, η ιδιότητα *tagname* περιέχει το όνομα του *tag,* ενώ η *text,* το κείμενο που βρίσκεται ανάμεσα στο tag ανοίγματος και κλεισίματος. Εκτός από τη μέθοδο *first* μπορούμε να χρησιμοποιήσουμε για τη πλοήγηση από ετικέτα σε ετικέτα τις μεθόδους *last*(), *next*() και *previous*()

```
>>> s = d.first('span')
>>> s.follow('CV')
<Region 0:19843>
>>> s.url
'http://zesty.ca/cv.html'
>>> s.back()
'http://zesty.ca/'
```

Αυτό το παράδειγμα μας δείχνει πως μπορούμε να πλοηγηθούμε από σελίδα σε σελίδα. Για να ακολουθήσουμε έναν σύνδεσμο χρησιμοποιούμε τη μέθοδο *follow*(). Για το προσδιορισμό του συνδέσμου που θέλουμε να ακολουθήσουμε ως όρισμα της μεθόδου *follow*() μπορούμε να χρησιμοποιήσουμε και μια κανονική έκφραση (π.χ. *s.follow(re.compile('…')*). Για να επιστρέψουμε στη σελίδα που βρισκόμασταν χρησιμοποιούμε τη μέθοδο *back()*.

Επίλογος

Η *Python* είναι μια γλώσσα κατανοητή και συνοπτική, με μια πληθώρα βιβλιοθηκών να τη συνοδεύουν. Η απλότητά της αλλά και η χρηστικότητα των βιβλιοθηκών *Beautifulsoup* και *Mechanize* ώθησαν και μένα να τη χρησιμοποιήσω για την ανάπτυξη του προγράμματος εξαγωγής δεδομένων από τον χρυσό οδηγό. Σίγουρα είναι μια εξαιρετική επιλογή για την ανάπτυξη τέτοιων προγραμμάτων.

ΚΕΦΑΛΑΙΟ 7
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ PHP

Εισαγωγή

Η *PHP* είναι μια *scripting* γλώσσα προγραμματισμού που την ενσωματώνουμε σε *HTML* αρχεία. Ένα μεγάλο μέρος της σύνταξής της είναι δανεισμένο από τη *C*, τη *Java* και τη *Perl*. Ο βασικός στόχος αυτής της γλώσσας είναι να επιτρέπει στους προγραμματιστές να γράφουν σελίδες που δημιουργούνται δυναμικά. Μπορούμε όμως να τη χρησιμοποιήσουμε και για τη δημιουργία κάποιον scripts που θα εξάγουν δεδομένα από διάφορες ιστοσελίδες. Τα εργαλεία της *PHP* για να το κάνουμε αυτό αναφέρονται σε αυτό το κεφάλαιο.

7.1 Η συνάρτηση fopen και οι κανονικές εκφράσεις

Η συνάρτηση *fopen* και οι κανονικές εκφράσεις (*regular expressions*) αποτελούν έναν εύκολο τρόπο για την εξόρυξη δεδομένων, αλλά όχι και τον ιδανικότερο, μιας και όπως είδαμε και σε προηγούμενα κεφάλαια η χρήση τον κανονικών εκφράσεων παρουσιάζει κάποια μειονεκτήματα. Δεν υπάρχει καλύτερος τρόπος για να καταλάβουμε πώς μπορούμε να χρησιμοποιήσουμε την *fopen* και τις κανονικές εκφράσεις, από την παρουσίαση ενός απλού παραδείγματος. Ας πούμε, για παράδειγμα, ότι θέλουμε να πάρουμε όλα τα *url* των συνδέσμων που υπάρχουν στη κεντρική σελίδα της *Google*.

```
1: $handle = fopen("http://www.google.com/", "rb");
2: $contents = '';
3: while (!feof($handle)) {
4:     $contents .= fread($handle, 8192);
5: }
6: fclose($handle);
```

Αυτό που κάνει το παράδειγμά μας είναι να ανοίγει τη σελίδα της Google και να αποθηκεύει τον HTML κώδικα στη μεταβλητή '$contents'. Τώρα που έχουμε τον *HTML* κώδικα της σελίδας, το επόμενο βήμα είναι να πάρουμε όλα τα *url* των συνδέσμων. Αυτό μπορούμε να το κάνουμε χρησιμοποιώντας κανονικές εκφράσεις.

```
preg_match_all('/a href\=\"(http\:\/\/.*?)\"/', $contents, $matches);
```

Αυτό που κάνει η παραπάνω συνάρτηση είναι να παίρνει όλα τα *url* που υπάρχουν στη μεταβλητή $*contents* και να τα αποθηκεύει στη μεταβλητή τύπου πίνακα $*matches*. Για να εμφανίσουμε όλα αυτά τα *url* στο παράθυρο ενός *browser,* το μόνο που χρειάζεται είναι να κάνουμε *'$echo'* όλα τα αντικείμενα που υπάρχουν στη μεταβλητή $matches. Όλος ο κώδικας βρίσκεται παρακάτω:

```
1: ?php
2:
3: # Χρησιμοποιούμε τη συνάρτηση fopen για να εξάγουμε
4: # δεδομένα από τη κεντρική σελίδα της Google
5: $handle = fopen("http://www.google.com/", "rb");
6:
7: # Αποθηκεύουμε τον HTML κώδικα στη μεταβλητή $contents
8: $contents = '';
9: while (!feof($handle)) {
10:     $contents .= fread($handle, 8192);
11: }
12: fclose($handle);
13:
14: # Παίρνουμε όλα τα url από τον HTML κώδικα
15: # και τα αποθηκεύουμε στον πίνακα $matches
16: preg_match_all('/a href\=\"(http\:\/\/.*?)\"/', $contents, $matches);
17:
18: # Εκτυπώνουμε όλα τα url.
19: foreach ($matches[1] as $url) {
20:     echo $url . " ";
21: }
22:
23: ?>
```

7.2 Simple HTML DOM Parser

Η ανοιχτού κώδικα βιβλιοθήκη της *PHP Simple HTML DOM Parser* όχι μόνο μας βγάζει από το μπελά των κανονικών εκφράσεων, αλλά κάνει και την εξόρυξη δεδομένων από HTML σελίδες εξαιρετικά απλή διαδικασία. Δείτε, για παράδειγμα, το παρακάτω κομμάτι κώδικα.

```
1: <?php
2: include('simple_html_dom.php');
3: $html = new simple_html_dom();
4: $html->load("<html><body><p>Παράγραφος 1</p><p>Παράγραφος 2</p></body></html>");
5:
6: # βρίσκουμε τα στοιχεία παραγράφους
7: $element = $html->find("p");
8:
9: # τροποποιούμε τη δεύτερη παράγραφο
10: $element[1]->innertext .= "Τροποποιημένη παράγραφος";
11:
12: # το εμφανίζουμε στην οθόνη
13: echo $html->save();
14: ?>
```

Στο παράδειγμά μας αρχικοποιούμε το αντικείμενό μας φορτώνοντας τον *HTML* κώδικα από ένα *string* μέσω της μεθόδου *load*(). Το κώδικα θα μπορούσαμε να τον φορτώσουμε και μέσω της μεθόδου *load_file*() από ένα *url* ή το τοπικό μας σύστημα αρχείων. Για παράδειγμα:

```
$html->load file('http://www.google.com/');
```

Αφού έχουμε το *DOM* αντικείμενό μας, μπορούμε να αρχίσουμε να δουλεύουμε με αυτό χρησιμοποιώντας τη μέθοδο *find*() και δημιουργώντας συλλογές. Μια συλλογή

είναι ένα σύνολο από αντικείμενα, τα οποία τα βρίσκουμε μέσω ενός επιλογέα (CSS *selector*). Στο παράδειγμά μας, χρησιμοποιώντας τη μέθοδο *find*(), δημιουργούμε μια συλλογή από τις παραγράφους του HTML κώδικα.

Στη δέκατη γραμμή του παραδείγματος τροποποιούμε το *innertext* του δεύτερου αντικειμένου της συλλογής μας. Το *innertext* αντιπροσωπεύει τα περιεχόμενα που βρίσκονται ανάμεσα σε δύο ετικέτες, ενώ το *outertext* περιλαμβάνει τα περιεχόμενα συμπεριλαμβανομένων και των δύο ετικετών. Δηλαδή, αν θέλαμε να αλλάξουμε το πρώτο στοιχείο της συλλογής από παράγραφο σε τίτλο, θα αλλάζαμε το *outertext* του αντικειμένου.

Εκτός από τη δυνατότητα να αλλάξουμε το *innertext* ή το *outetext* κάποιου αντικειμένου της λίστας μας έχουμε τη δυνατότητα να προσθέσουμε και κάποια ιδιότητα σε αυτό. Για παράδειγμα, αν θέλουμε να προσθέσουμε την ιδιότητα *class* στη δεύτερη παράγραφο, μπορούμε να το κάνουμε ως εξής:

```
$element[1]->class = "para";
```

Μετά τις αλλαγές το τελικό αποτέλεσμα στον *html* κώδικα θα ήταν το παρακάτω:

```
1: <html>
2: <body>
3:     <p>Παράγραφος 1</p>
4:     <p class="para">Τροποποιημένη παράγραφος</p>
5: </body>
6: </html>
```

Παρακάτω υπάρχουν μερικοί ακόμα τρόποι επιλογής στοιχείων από τον *html* κώδικα για την καλύτερη κατανόηση της χρήσης της μεθόδου *find*().

```
1: # το πρώτο στοιχείο με id="id_name"
2: $single = $html->find('#id_name', 0);
3:
4: # όλα τα στοιχεία με class="class_name"
5: $collection = $html->find('.class_name');
6:
7: # όλοι οι σύνδεσμοι της σελίδας
8: $collection = $html->find('a');
9:
10: # όλοι οι σύνδεσμοι που βρίσκονται μέσα σε H1 tags
11: $collection = $html->find('h1 a');
12:
13: # όλες οι εικόνες με title='image_title'
14: $collection = $html->find('img[title=imagetitle]');
```

7.3 htmlSQL

Η βιβλιοθήκη *simple HTML DOM* είναι σίγουρα πολύ εύχρηστη και λειτουργική. Τώρα, όμως, θα δούμε μια βιβλιοθήκη που ακολουθεί μια διαφορετική προσέγγιση από αυτή της *simple HTML DOM,* αλλά και κάθε άλλης βιβλιοθήκης που έχουμε δει ως τώρα στις προηγούμενες γλώσσες προγραμματισμού. Η *htmlSQL* μας επιτρέπει να έχουμε πρόσβαση στα στοιχεία του *html* εγγράφου με σύνταξη παρόμοια με αυτή των ερωτημάτων *SQL*. Ένα *htmlSQL* ερώτημα μοιάζει με το παρακάτω:

```
SELECT href,title FROM a WHERE $class == "class_name"
```

Στο παραπάνω παράδειγμα επιλέγουμε τις ιδιότητες *href* και *title* από κάθε σύνδεσμο της σελίδας που έχει ιδιότητα *class* ίση με '*class_name*'. Βλέπουμε ότι δεν χρειάζεται να γράψουμε ούτε πολύπλοκες κανονικές εκφράσεις, ούτε *XPath* εκφράσεις. Μετά το *FROM,* αντί κάποιας ετικέτας έχουμε τη δυνατότητα να βάλουμε αστερίσκο (*) για να υποδηλώσουμε όλες τις ετικέτες.

Για να αρχίσουμε να χρησιμοποιούμε τη βιβλιοθήκη htmlSQL, απλά τοποθετούμε τα αρχεία '*snoopy.class.php*' και '*htmlsql.class.php*' στο κατάλογο εργασίας μας και τα κάνουμε '*include*' μέσω του κώδικά μας. Αυτά τα δύο αρχεία μπορείτε να τα βρείτε στην ιστοσελίδα «*http://www.jonasjohn.de/lab/htmlsql.htm*»

7.4 Εξόρυξη δεδομένων με PHP5

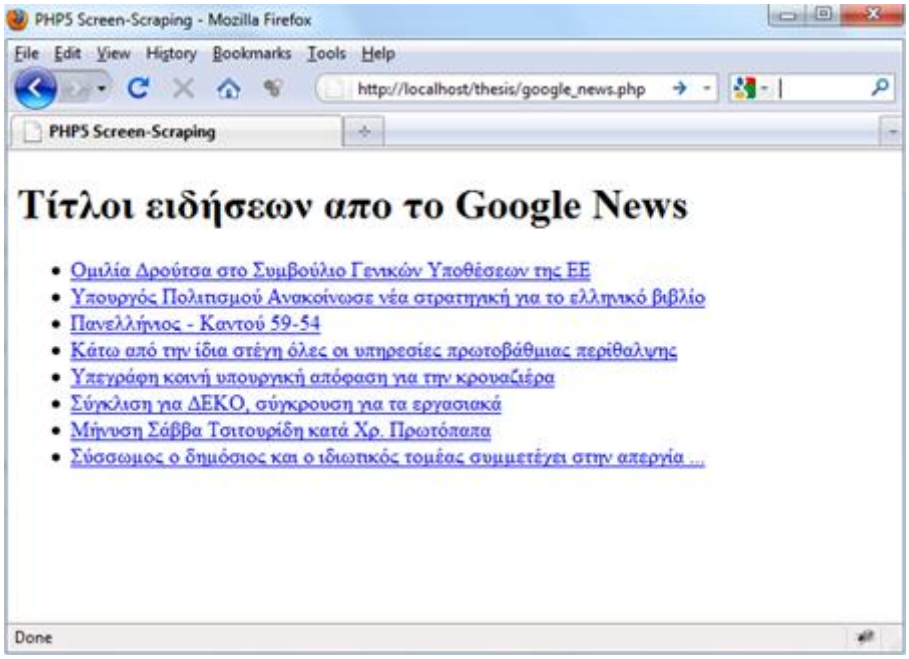
Η έκδοση 5 της *PHP* παρέχει στους προγραμματιστές νέες δυνατότητες όσων αφορά την εξαγωγή δεδομένων από το διαδίκτυο. Στη νέα έκδοση δεν είναι απαραίτητο να κάνουμε χρήση μόνο καλά δομημένων (*well-form*) *XML* εγγράφων αλλά μπορούμε να χρησιμοποιήσουμε οποιοδήποτε *HTML* έγγραφο. Όπως γνωρίζουμε, τα περισσότερα αρχεία *HTML* στο διαδίκτυο δεν είναι σωστά μορφοποιημένα (*not valid HTML*), πόσω μάλλον τα αρχεία *XML*/*HTML*. Αν προσπαθήσουμε να προσπελάσουμε κάποιο άσχημα μορφοποιημένο αρχείο με κάποιον parser, ο οποίος υποθέτει ότι το αρχείο είναι σωστά μορφοποιημένο, τότε σίγουρα θα καταλήξουμε σε λάθη.

Η απλή, αλλά συνάμα και ισχυρή μέθοδος που χρειαζόμαστε είναι η «*loadHTMLFile*», η οποία δέχεται ως παράμετρο οποιοδήποτε *URL*. Παρακάτω είναι ένα παράδειγμα χρήσης της, το οποίο παίρνει από τη σελίδα *Google News* τις πιο πρόσφατες δημοφιλέστερες ειδήσεις και της εμφανίζει στην οθόνη του *browser* σε μορφή συνδέσμου:

```
1: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
2:         1.0 Strict//EN" "DTD/xhtml1-strict.dtd">
3: <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="el" lang="el">
4: <head>
5:     <title>PHP5 Screen-Scraping</title>
6: </head>
7: <body>
8:
9: <?php
10:
11: $dom = new domdocument;
```

```
12:
13: $url = 'http://news.google.com/news?pz=1&cf=all&ned=el_gr&ict=ln';
14:
15: @$dom->loadHTMLFile($url);
16:
17: $xpath = new domxpath($dom);
18: $xNodes = $xpath->query('//div[@class="title"]');
19: echo '<h1>Τίτλοι ειδήσεων απο το Google News</h1>';
20: echo '<ul>';
21: foreach ($xNodes as $xNode)
22: {
23:     $sLinktext = $xNode->firstChild->firstChild->nodeValue;
24:     $sLinkurl = $xNode->firstChild->getAttribute('href');
25:
26:     if ($sLinktext != '' && $sLinkurl != '')
27:     {
28:         echo '<li><a href="' . $sLinkurl . '">' .
29:                 $sLinktext . '</a></li>';
30:     }
31: }
32: echo '</ul>';
33: ?>
34:
35: </body>
36: </html>
```

Βλέπουμε ότι στη γραμμή 18 χρησιμοποιήσαμε μια έκφραση *XPath* για να δημιουργήσουμε μια συλλογή από κόμβους του *DOM* δέντρου. Έχοντας αυτούς τους κόμβους, στις γραμμές 23 και 24 κατεβήκαμε το *DOM* δέντρο, μέσω της μεθόδου *firstChild*, για να δρούμε το τίτλο και τον σύνδεσμο της είδησης. Το αποτέλεσμα φαίνεται στην επόμενη εικόνα :



Εικόνα 4 – Ενδεικτική εφαρμογή εξόρυξης δεδομένων με την PHP

Επίλογος

Όπως έχουμε πει και σε προηγούμενα κεφάλαια, η χρήση κανονικών εκφράσεων μπορεί να μας οδηγήσει σε απροσδόκητα αποτελέσματα, σε περίπτωση έστω και μικρής αλλαγής του κώδικα *HTML* των σελίδων. Αλλά όπως είδαμε υπάρχουν κι άλλες επιλογές.

Η βιβλιοθήκη *htmlSQL* είναι σίγουρα πολύ απλή στη χρήση, όμως υστερεί σε δυνατότητες. Η καλύτερη επιλογή είναι η χρήση του *Simple HTML DOM parser* ή η χρήση *core* μεθόδων της γλώσσας όπως η *loadHTMLFile* με συνδυασμό *XPath* εκφράσεων.

ΚΕΦΑΛΑΙΟ 8
ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ ΜΕ ΑΛΛΕΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Εισαγωγή

Η *Java,* αλλά και οι γλώσσες προγραμματισμού του *.NET* Framework της Microsoft, όπως η *C#* ή η *Visual Basic*, είναι γλώσσες που χρησιμοποιούνται πάρα πολύ για την ανάπτυξη εφαρμογών. Οπότε δεν θα μπορούσαν να λείπουν από αυτή την εργασία.

Σε αυτό το κεφάλαιο περιγράφουμε δύο εξαιρετικούς parsers, τον *JSoup* της Java και τον *HTML Agility Pack* του *.NET*. Και οι δύο παρουσιάζουν πολλές ομοιότητες, οπότε θεώρησα σωστό να συμπεριληφθούν στο ίδιο κεφάλαιο.

8.1 Η βιβλιοθήκη JSoup της Java

Η *jSoup* είναι μια βιβλιοθήκη της *Java* για της εργασία με τα *HTML* αρχεία, η οποία παρέχει ένα API για την εξαγωγή και τον χειρισμό των δεδομένων. Η βιβλιοθήκη *jSoup* είναι ένα έργο ανοιχτού κώδικα, το οποίο διανέμεται δωρεάν. Τον πηγαίο κώδικα μπορούμε να τον βρούμε στη σελίδα: *http://github.com/jhy/jsoup/.*

Ένα βασικό πλεονέκτημα του *JSoup* parser είναι ότι προσπαθεί να δημιουργήσει ένα καθαρό (*well-form*) αντίγραφο του κώδικα *HTML* που θα του παρέχουμε, ανεξάρτητα αν αυτός είναι καλοσχεδιασμένος ή όχι. Για παράδειγμα, σε περίπτωση που ο parser συναντούσε *html* ετικέτες που δεν είχαν κλειστεί σωστά, όπως παρακάτω:

```
<p> Αυτή είναι μια παράγραφος <p> Άλλη μια παράγραφος.
```

Θα δημιουργούσε το παρακάτω κώδικα:

```
<p> Αυτή είναι μια παράγραφος</p> <p> Άλλη μια παράγραφος.</p>
```

Όταν θέλουμε να αναλύσουμε ένα κομμάτι κώδικα από μια μεταβλητή τύπου *string* ή ένα αρχείο, χρησιμοποιούμε τη μέθοδο *parse()*. Για να φορτώσουμε και να διαχειριστούμε ένα αρχείο από το διαδίκτυο, τότε χρησιμοποιούμε τις μεθόδους *connect()* και *get()*.

```
Document doc = Jsoup.connect("http://example.com/").get();
String title = doc.title();
```

Η μέθοδος *connect()* δημιουργεί μια καινούργια σύνδεση, ενώ η μέθοδος *get()* παίρνει τα περιεχόμενα της σελίδας. Αν παρουσιαστεί κάποιο λάθος στο φόρτωμα της σελίδας, τότε δημιουργείται ένα *IOException* το οποίο θα πρέπει να διαχειριστούμε κατάλληλα. Δείτε, για παράδειγμα, το επόμενο κομμάτι κώδικα:

```
1: import org.jsoup.Jsoup;
2: import org.jsoup.nodes.Document;
3: import java.io.IOException
4:
5: public class JSoupExample {
6:
7:     public static void main(String[] args) {
8:
9:         try{
10:             Document doc = Jsoup.connect("http://www.google.com/").get();
11:             String title = doc.title();
12:             System.out.print(title);
13:         }
14:         catch ( IOException e ){
15:             System.out.println("error");
16:         }
17:     }
18: }
```

Αφού εισάγουμε τις κατάλληλες βιβλιοθήκες, δημιουργούμε μια νέα σύνδεση και παίρνουμε τα περιεχόμενα της κεντρικής σελίδας του Google (γραμμή 10). Στην επόμενη γραμμή του κώδικα εκχωρούμε στη μεταβλητή 'title' τον τίτλο της σελίδας του *Google*.

Για την εύρεση των στοιχείων, την εξαγωγή και τη παραποίηση των δεδομένων τους, το JSoup διαθέτει μια πληθώρα μεθόδων οι οποίες είναι παρεμφερείς με αυτές του DOM (Document Object Model).

Εύρεση των στοιχείων:

- `getElementById(String id)`
- `getElementsByTag(String tag)`
- `getElementsByClass(String className)`
- `getElementsByAttribute(String key)`
- `siblingElements(), firstElementSibling(), lastElementSibling(), nextElementSibling(), previousElementSibling()`
- `parent(), children(), child(int index)`

Δεδομένα των στοιχείων:

- `attr(String key), attr(String key, String value)`
- `attributes()`
- `id(), className(), classNames()`
- `text()`
- `html()`
- `outerHtml()`
- `data()`
- `tag(), tagName()`

Παραποίηση των στοιχείων:

- `append(String html), prepend(String html)`

- `appendText(String text), prependText(String text)`
- `appendElement(String tagName), prependElement(String tagName)`
- `html(String value)`

Εκτός από τη χρήση αυτών των μεθόδων θα μπορούσαμε να χρησιμοποιήσουμε και τη μέθοδο *select()*. Στη μέθοδο *select()* βάζουμε ως όρισμα *CSS Selectors*:

```
1: // Όλα τα anchors που έχουν ιδιότητα href
2: Elements links = doc.select("a[href]");
3: // Όλες οι εικόνες με κατάληξη .png
4: Elements pngs = doc.select("img[src$=.png]");
5: // div με class=class_name
6: Element masthead = doc.select("div.class_name ").first();
```

8.2 Το HTML Agility Pack του .NET

Το *.NET Framework* προσφέρει μερικές κλάσεις για τη πρόσβαση σε δεδομένα ενός διαδικτυακού τόπου. Δύο από αυτές τις κλάσεις είναι οι *WebClient* και η *HttpWebRequest*. Αυτές οι κλάσεις είναι χρήσιμες για την υποβολή μια αίτησης HTTP σε μια απομακρυσμένη ιστοσελίδα και το τράβηγμα του πηγαίου κώδικά της, αλλά δεν προσφέρουν τίποτα στην ανάλυση και την εξαγωγή δεδομένων από αυτή. Γι' αυτό το λόγο, πολλές φορές οι προγραμματιστές χρησιμοποιούν μεθόδους επεξεργασίας αλφαριθμητικών ή κανονικές εκφράσεις.

Μια καλύτερη επιλογή για την ανάλυση *HTML* εγγράφων είναι η χρήση του *HTML Agility Pack,* η οποία είναι μια δωρεάν, ανοικτού κώδικα βιβλιοθήκη, σχεδιασμένη για να απλοποιήσει την ανάγνωση και τροποποίηση εγγράφων *HTML*. Όπως και η βιβλιοθήκη *JSoup* της *Java* που είδαμε στη προηγούμενη παράγραφο, η *HTML Agility Pack* κατασκευάζει μια δενδροειδή αναπαράσταση του *HTML* εγγράφου. Με μόνο λίγες γραμμές κώδικα μπορούμε να περιπλανηθούμε στους κόμβους του *DOM* δέντρου και να επιστρέψουμε συγκεκριμένους κόμβους μέσω εκφράσεων *XPath*.

Για την επίδειξη του τρόπου χρήσης του *HTML Agility Pack* θα δημιουργήσουμε ένα απλό πρόγραμμα, το οποίο βρίσκει από μια ιστοσελίδα τις ετικέτες *<meta>*. Το *HTML Agility Pack* περιέχει έναν αριθμό από κλάσεις, όλες στο *HTMLAgilityPack namespace*. Ως εκ τούτου, ξεκινάμε με τη προσθήκη μιας εντολής *using* (ή *Imports*, αν χρησιμοποιούμε *VB*) στη κορυφή του κώδικά μας.

```
1: using HTMLAgilityPack;
```

Για να κατεβάσουμε μια ιστοσελίδα από ένα απομακρυσμένο διακομιστή χρησιμοποιούμε τη μέθοδο *Load* της κλάσης *HtmlWeb* ως εξής:

```
2: var hw = new HTMLWeb();
3: var doc = hw.Load(url);
```

Η μέθοδος *Load* επιστρέφει ένα αντικείμενο *HtmlDocument* το οποίο αποθηκεύουμε στη τοπική μεταβλητή *doc*. Η κλάση *HtmlDocument* αναπαριστά ένα ολοκληρωμένο

έγγραφο HTML και περιέχει μια ιδιότητα *DocumentNode*. Αυτή η ιδιότητα επιστρέφει ένα αντικείμενο *HtmlNode,* το οποίο αναπαριστά τον αρχικό κόμβο του εγγράφου.

Η κλάση HtmlNode περιέχει και αρκετές ιδιότητες άξιες αναφοράς. Είναι ιδιότητες για τη διάσχιση του DOM δέντρου:

- *ChildNodes*
- *ParentNodes*
- *NextSibling*
- *PreviousSibling*

Αλλά και για το καθορισμό πληροφοριών σχετικά με τον κόμβο:

- *Name* – Παίρνει ή θέτει το όνομα του κόμβου.
- Attributes – Επιστρέφει μια συλλογή με τις ιδιότητες του στοιχείου (εάν υπάρχουν)
- *InnerHtml* – Παίρνει ή καθορίζει το περιεχόμενο HMTL στο εσωτερικό του κόμβου.
- *InnerText* – Επιστρέφει το κείμενο εντός του κόμβου.
- *NodeType* – Δηλώνει το είδος του κόμβου. Μπορεί να είναι Document, Element, Comment ή Text.

Λαμβάνοντας υπόψη όλες αυτές τις μεθόδους και τις ιδιότητες, κατανοούμε πως υπάρχουν διαφορετικοί τρόποι με τους οποίους θα μπορούσαμε να πάρουμε μια συλλογή με όλες τις ετικέτες <meta> από το HTML έγγραφο. Στο παράδειγμά μας θα χρησιμοποιήσουμε τη μέθοδο *SelectNodes*. Στη παρακάτω γραμμή κώδικα χρησιμοποιούμε τη μέθοδο *SelectNodes* με όρισμα μια έκφραση *Xpath,* η οποία θα μας επιστρέψει όλες τις ετικέτες *meta* του εγγράφου.

```
4: var metaTags = doc.DocumentNode.SelectNodes("//meta");
```

Εάν δεν υπάρχουν ετικέτες *meta* στο έγγραφο, τότε η μεταβλητή *metaTags* θα έχει τη τιμή *null*. Αλλά εάν υπάρχει μία ή περισσότερες ετικέτες meta, τότε η μεταβλητή *metaTags* θα κρατάει μια συλλογή από αντικείμενα *HtmlNode*.

Αυτό ήταν! Ούτε ακατανόητες κανονικές εκφράσεις, ούτε χρήση μεθόδων επεξεργασίας αλφαριθμητικών, αλλά ευανάγνωστη σύνταξη για την πρόσβαση στο περιεχόμενο του *HMTL* εγγράφου.

Επίλογος

Καί το *HTML Agility Pack* αλλά καί η βιβλιοθήκη *JSoup* της *Java* είναι δυο πολύ ισχυρά εργαλεία που μας βοηθούν να περιηγηθούμε μέσα σε *HTML* αρχεία όπως ακριβώς θα κάναμε καί σε *XML* αρχεία. Βέβαια, τα παραδείγματα που είδαμε σε αυτό το κεφάλαιο είναι πολύ απλά, αλλά σίγουρα μας δίνουν μια γεύση του τί μπορούμε να κάνουμε με τέτοιου είδους *html parsers*.

ΚΕΦΑΛΑΙΟ 9
ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ (ΜΕΡΟΣ Ι) – ΓΙΑΤΙ PYTHON;

Εισαγωγή
Ο κώδικας του προγράμματος για την εξόρυξη δεδομένων από τον χρυσό οδηγό έχει γραφτεί στη γλώσσα προγραμματισμού Python, οπότε η οποιαδήποτε εξοικείωσή σας με αυτή τη γλώσσα θα φανεί χρήσιμη για τη κατανόηση του κώδικα. Κάποιος που δεν έχει ξαναχρησιμοποιήσει τη γλώσσα αυτή, αλλά έχει κάποια εμπειρία με άλλες γλώσσες υψηλού επιπέδου, δεν πρόκειται να συναντήσει σημαντικό πρόβλημα.

9.1 Γιατί Python;
Η Python είναι μια εξαιρετική γλώσσα προγραμματισμού, υψηλού επιπέδου, την οποία επέλεξα για τους παρακάτω λόγους:

1. **Είναι συνοπτική**
   Αυτό σημαίνει ότι ο κώδικας που χρειάζεται να γράψουμε όχι μόνο είναι λιγότερος, αλλά είναι και ευκολότερο να διαβαστεί από κάποιον άλλο.
2. **Εύκολα κατανοήσιμη**
   Κάποιοι αποκαλούν τη Python ως “εκτελέσιμο ψευδοκώδικα”. Αν και απέχει αυτό από τη πραγματικότητα, μας δίνει μια σαφή εικόνα της ευκολίας με την οποία μπορούμε να διαβάσουμε και να κατανοήσουμε το κώδικα που είναι γραμμένος σε Python.
3. **Εύκολα επεκτάσιμη**
   Η Python έρχεται σε συνδυασμό με πολλές βιβλιοθήκες. Μερικές από αυτές τις είδαμε στο κεφάλαιο για την εξόρυξη δεδομένων χρησιμοποιώντας Python. Στην εφαρμογή μας θα χρησιμοποιήσουμε άλλη μία, που δεν έχουμε δει έως τώρα, και η οποία μας βοηθάει να δημιουργήσουμε ένα αρχείο excel. Αυτή είναι η βιβλιοθήκη «xlwt».
4. **Είναι διαδραστική.**
   Όταν δουλεύουμε πάνω σε ένα πρόγραμμα, πολλές φορές είναι χρήσιμο να δοκιμάζουμε τις μεθόδους που φτιάχνουμε, χωρίς να υπάρχει η ανάγκη να δημιουργήσουμε κάποιο άλλο πρόγραμμα για να τις δοκιμάσουμε. Στη Python έχουμε τη δυνατότητα να τρέχουμε τα προγράμματα από τη γραμμή εντολών ή μέσω του προγράμματος «IDLE», το οποίο είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης της Python. Σε αυτό μπορούμε να τρέξουμε εντολές, να καλέσουμε μεθόδους, να δημιουργήσουμε αντικείμενα, κ.α.
5. **Είναι Multiparadigm γλώσσα προγραμματισμού.**
   Μια γλώσσα προγραμματισμού ονομάζεται Multiparadigm όταν υποστηρίζει περισσότερους από έναν τρόπους προγραμματισμού. Συνεπώς, κατά την ανάπτυξη μιας εφαρμογής σε Python μπορούμε να χρησιμοποιήσουμε

διαφορετικά στιλ γραψίματος του κώδικα. Η Python υποστηρίζει και Διαδικασιακό (Procedural) και Συναρτησιακό (Functional) αλλά και Αντικειμενοστρεφή προγραμματισμό.

6. **Είναι Multiplatform και διανέμεται δωρεάν.**
   Ο κώδικας της Python έχει παρόμοια εφαρμογή σε όλες της σημαντικές πλατφόρμες και είναι δωρεάν για όλους. Η εφαρμογή που έχουμε φτιάξει για την εξόρυξη δεδομένων από το χρυσό οδηγό μπορεί να τρέξει εξίσου καλά καί σε Windows, καί Linux, αλλά καί σε Macintosh.

9.2 Χαρακτηριστικά της Python

Εδώ θα αναφέρω μερικά χαρακτηριστικά της γλώσσας, τα οποία διαφέρουν από άλλες γλώσσες προγραμματισμού, γι' αυτούς που δεν έχουν προγραμματίσει ποτέ σε Python.

9.2.1 Στοίχιση του κώδικα

Σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού, η Python χρησιμοποιεί τις εσοχές του κώδικα για το προσδιορισμό των block. Δείτε για παράδειγμα το παρακάτω απόσπασμα:

```
1: if x==1:
2:     print 'x is 1'
3:     print 'Ακόμα μέσα στο block'
4: print 'Έξω από το block'
```

Ο μεταφραστής (interpreter) της Python ξέρει ότι οι δύο πρώτες εντολές *print* εκτελούνται, εάν η συνθήκη είναι αληθής (δηλαδή αν η μεταβλητή *x* είναι ίση με 1) λόγω της εσοχής. Μπορούμε να χρησιμοποιήσουμε οποιονδήποτε αριθμό κενών για την εσοχή, αρκεί να είμαστε συνεπείς σε αυτόν.

9.2.2 Η χρήση των λιστών

Οι **λίστες (*lists*)** είναι μια δομή δεδομένων, την οποία θα χρησιμοποιήσουμε κατά κόρον στην εφαρμογή μας, οπότε θα ήταν καλό να δούμε μερικά πράγματα για αυτές προτού προχωρήσουμε στο κώδικα της εφαρμογής.

Οι δομές δεδομένων είναι αυτό που λέει το όνομά τους, δηλαδή είναι *δομές* που μπορούν να κρατήσουν μαζί μερικά *δεδομένα*. Με άλλα λόγια, χρησιμοποιούνται για να αποθηκεύουν δεδομένα που έχουν σχέση μεταξύ τους. Εκτός από της λίστες στη *Python,* υπάρχουν άλλες τρεις τέτοιες δομές ενσωματωμένες στη γλώσσα. Αυτές είναι οι **πλειάδες *(tuples)***, τα **λεξικά *(dictionaries)*** και τα **σύνολα (*sets*)**.

Μια λίστα είναι μια δομή δεδομένων που συγκρατεί μια διατεταγμένη συλλογή στοιχείων. δηλαδή μιά λίστα μας επιτρέπει να αποθηκεύσουμε μια *ακολουθία* (*sequence*) αντικειμένων μέσα της. Αυτό είναι εύκολο να το φανταστείτε αν σκεφτείτε μια λίστα για αγορές, μέσα στην οποία έχετε καταγεγραμμένα τα αντικείμενα που πρέπει να αγοράσετε. Στη λίστα των αγορών σας είναι πολύ πιθανόν να έχετε το κάθε αντικείμενο που θέλετε να αγοράσετε σε διαφορετική γραμμή. Με παρόμοιο τρόπο δημιουργούμε μια λίστα στη *Python*, μόνο που τοποθετούμε κόμματα ανάμεσα στα

στοιχεία και όλα αυτά τα στοιχεία τα κλείνουμε μέσα σε αγκύλες (δηλαδή [και]), έτσι ώστε να καταλαβαίνει η *Python* ότι καθορίζετε μια λίστα.

Αφού έχουμε δημιουργήσει μια λίστα, μπορούμε να προσθέσουμε, να μετακινήσουμε ή να αναζητήσουμε ένα στοιχείο σε αυτή. Από τη στιγμή που μπορούμε να προσθέσουμε και να μετακινήσουμε στοιχεία, λέμε ότι η λίστα είναι ένας *μεταβλητός* τύπος δεδομένων (*mutable data type*), δηλαδή αυτός ο τύπος μπορεί να αλλαχθεί.

9.2.3 Κλάσεις, αντικείμενα και μέθοδοι

Μιας και μιλήσαμε για τις λίστες, καλό είναι να κάνουμε μια γρήγορη αναφορά και στα **αντικείμενα (*objects*)** και **τις κλάσεις (*classes*)**. Η λίστα είναι ένα παράδειγμα χρήσης αντικειμένων και κλάσεων. Όταν χρησιμοποιούμε τη μεταβλητή `i` και εκχωρούμε σ' αυτή μια τιμή, ας πούμε τον ακέραιο αριθμό `1`, αυτό μπορούμε να το δούμε σαν τη δημιουργία ενός αντικειμένου (δηλ. υπόστασης (*instance*)) `i` της κλάσης (δηλ. τύπου (*type*)) *int*.

Μια κλάση μπορεί επίσης να έχει **μεθόδους (*methods*)**, δηλαδή συναρτήσεις που ορίστηκαν για χρήση που έχει σχέση μόνο με αυτή την κλάση. Μπορείτε να χρησιμοποιήσετε αυτά τα μέρη λειτουργικότητας μόνο όταν έχετε ένα αντικείμενο για αυτή τη κλάση. Για παράδειγμα η Python παρέχει μια μέθοδο *append* για την κλάση *list*, που μας επιτρέπει να προσθέσουμε ένα αντικείμενο στο τέλος της λίστας. Έτσι, το `mylist.append('item')`, παραδείγματος χάριν, θα προσθέσει τη συμβολοσειρά *item* στο τέλος της λίστας *mylist*.

Μια κλάση μπορεί επίσης να έχει **πεδία (*fields*)**, που δεν είναι τίποτα άλλο παρά μεταβλητές που ορίστηκαν για χρήση που έχει σχέση μόνο με αυτή τη κλάση. Μπορείτε να χρησιμοποιήσετε αυτές τις μεταβλητές μόνο όταν έχετε ένα αντικείμενο αυτής της κλάσης. Όπως και με τις μεθόδους, τα πεδία εισάγονται με τη χρήση της τελείας, για παράδειγμα, `mylist.field`. Ακολουθεί ένα παράδειγμα:

```
 1: #!/usr/bin/python
 2: # -*- coding: utf-8 -*-
 3:
 4: shoplist = ['milk', 'cereals', 'bread']
 5:
 6: shoplist.append('cheese')
 7: print('My shoping list:', shoplist)
 8:
 9: shoplist.sort()
10: print('Ordered list:', shoplist)
11:
12: del shoplist[0]
13: print('The list after delete of the first item:', shoplist)
```

Το αποτέλεσμα της εκτέλεσης του παραπάνω προγράμματος είναι το εξής:

```
('My shoping list:', ['milk', 'cereals', 'bread', 'cheese'])
('Ordered list:', ['bread', 'cereals', 'cheese', 'milk'])
('The list after delete of the first item:', ['cereals', 'cheese', 'milk'])
```

Επίλογος

Όπως είδαμε, η Python είναι μια εύκολα κατανοήσιμη και απλή στη χρήση γλώσσα προγραμματισμού. Αυτό βοηθάει τον οποιοδήποτε να κατανοήσει τη λειτουργία της εφαρμογής μας. Επιπλέον, η βιβλιοθήκες που διαθέτει μας βοηθούν σε πολύ μεγάλο βαθμό στην ανάπτυξη του προγράμματος που θέλουμε να δημιουργήσουμε. Οπότε, θεωρώ ότι η *Python* ήταν η καλύτερη επιλογή για το συγκεκριμένο πείραμα. Οι βιβλιοθήκες που θα μας χρειαστούν κατά την ανάπτυξη του προγράμματος περιγράφονται στο επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 10

ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ (ΜΕΡΟΣ ΙΙ) – ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΧΡΗΣΗ ΤΩΝ ΒΙΒΛΙΟΘΗΚΩΝ

Εισαγωγή

Στην εφαρμογή μας θα χρησιμοποιήσουμε διάφορες βιβλιοθήκες, καθεμία για διαφορετικό σκοπό. Σε αυτό το κεφάλαιο θα κάνουμε μια εισαγωγή σε αυτές τις βιβλιοθήκες και θα δούμε πώς μπορούμε να τις εγκαταστήσουμε και να τις χρησιμοποιήσουμε σε windows και linux συστήματα. Πρώτα, όμως, θα δούμε πώς μπορούμε να εγκαταστήσουμε το περιβάλλον της Python.

10.1 Εγκατάσταση της Python

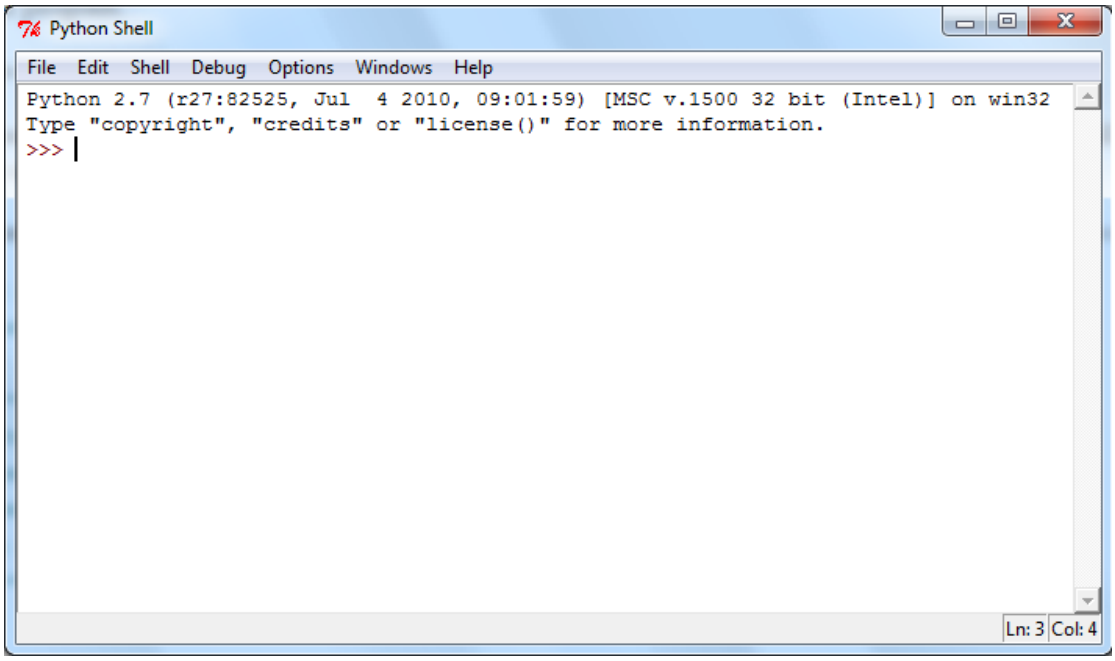
Για την ανάπτυξη της εφαρμογής μας θα χρησιμοποιήσουμε την έκδοση 2.7 της Python. Αυτή δεν είναι η τελευταία έκδοση της γλώσσας. Ο λόγος που δεν θα χρησιμοποιήσουμε την 3.1, η οποία αυτή τη στιγμή είναι η τελευταία έκδοση της γλώσσας, είναι ότι η βιβλιοθήκη BeautifulSoup παρουσιάζει κάποια προβλήματα με αυτή την έκδοση.

10.1.1 Windows

Για να εγκαταστήσετε την Python στα windows μηχανήματα, ακολουθήστε αυτά τα βήματα:

1. Ανοίξτε έναν browser και πηγαίνετε στη σελίδα *http://www.python.org*
2. Κάντε κλικ στο σύνδεσμο *Download*
3. Θα δείτε διάφορους συνδέσμους εκεί. Κάντε κλικ στο *Python 2.7.1 Windows Installer* για να κατεβάσετε το αρχείο εγκατάστασης.
4. Αφού κατεβάσετε το αρχείο, κάντε διπλό κλικ σε αυτό και θα εμφανιστεί ο οδηγός εγκατάστασης. Απλά αποδεχτείτε τις προεπιλεγμένες ρυθμίσεις και περιμένετε μέχρι να ολοκληρωθεί η εγκατάσταση.

Υποθέτοντας ότι η εγκατάσταση πήγε καλά, εκτελέστε το Python Integrated Development Environment (IDLE) επιλέγοντας Έναρξη→ Όλα τα προγράμματα → Python 2.7 → IDLE (Python GUI), θα εμφανιστεί στην οθόνη σας κάτι σαν το παρακάτω:



Εικόνα 5 – Το ολοκληρωμένο περιβάλλον ανάπτυξης της Python IDLE

10.1.2 Linux (Ubuntu)

Στα *linux* συνήθως η Python υπάρχει προ-εγκατεστημένη. Για να δούμε πια έκδοση χρησιμοποιούμε πληκτρολογούμε στο *terminal* τη λέξη python. Για να εγκαταστήσουμε την έκδοση 2.7 μπορούμε να χρησιμοποιήσουμε τις εξής εντολές:

```
$ wget http://www.python.org/ftp/python/2.7/Python-2.7.tgz
$ tar xzf Python-2.7.tgz
$ cd Python-2.7
$ ./configure
$ make
$ sudo make altinstall
```

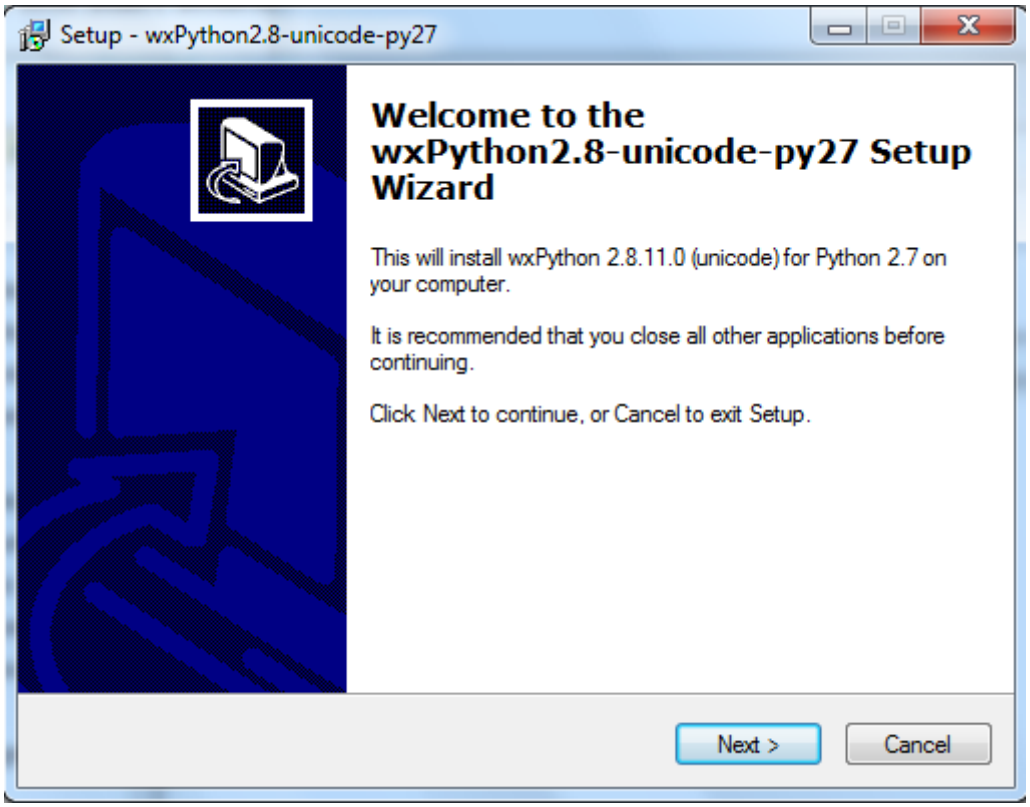
10.2 wxPython

Η *wxPython* είναι μια *cross platform* εργαλειοθήκη για τη δημιουργία εφαρμογών με γραφικό περιβάλλον (*GUI applications*). Με το *wxPython* οι προγραμματιστές μπορούν να δημιουργήσουν γραφικά περιβάλλοντα για εφαρμογές σε *Windows*, *Mac* και *Linux*.

10.2.1 Windows

Μπορούμε να κατεβάσουμε το *wxPython* από τη σελίδα *http://www.wxpython.org/*. Πρέπει να επιλέξουμε το σωστό αρχείο σύμφωνα με την έκδοση της *Python* που χρησιμοποιούμε. Αφού εμείς χρησιμοποιούμε την έκδοση 2.7 θα επιλέξουμε τον σύνδεσμο wxPython2.8-win32-unicode-py27. Εκτός από την επιλογή *win32-unicode,* υπάρχει και η *win32-ansi,* αλλά η επιλογή συνήθως είναι η πρώτη γιατί υποστηρίζει και άλλες γλώσσες εκτός από τα αγγλικά.

Αφού κατεβάσουμε το αρχείο εγκατάστασης, το εκτελούμε και εμφανίζεται ο οδηγός εγκατάστασης. Το μόνο πράγμα που πρέπει να κάνουμε είναι να αποδεχτούμε τους όρους χρήσης.



Εικόνα 6 – Εγκατάσταση του wxPython

10.2.2 Linux (Ubuntu)

Εδώ η εγκατάσταση είναι ακόμα πιο απλή. Μπορούμε να πραγματοποιήσουμε την εγκατάσταση μέσω του *Synaptic Package Manager*. Το πακέτο του *wxPython* ονομάζεται *python-wxgtkx.x*. Τσεκάρουμε το πακέτο για εγκατάσταση και κάνουμε κλικ στο *Apply*. Ο *Package Manager* αυτόματα κάνει όλες τις απαραίτητες διαδικασίες για την εγκατάσταση του πακέτου.

10.3 BeautifulSoup

Η *BeautifulSoup* είναι μια μικρή βιβλιοθήκη κώδικα για τη *Python* την οποία περιγράψαμε στην ενότητα 5.3. Για να τη χρησιμοποιήσουμε δεν έχουμε τίποτα άλλο να κάνουμε, παρά να πάμε στη σελίδα *http://crummy.com/software/BeautifulSoup* και να κατεβάσουμε το αρχείο *BeautifulSoup.py*. Αφού το κατεβάσουμε το τοποθετούμε στο path της Python (Για παράδειγμα *C:\Python27\Lib\site-packages* στα *Windows* ή *usr\lib\python2.7\dist-packages* στα *Ubuntu*)

10.4 Mechanize

Καί τη βιβλιοθήκη *mechanize* τη περιγράψαμε σε προηγούμενο κεφάλαιο (κεφάλαιο 5, ενότητα 5). Για να τη χρησιμοποιήσουμε στο πρόγραμμά μας ακολουθούμε την ίδια διαδικασία που κάναμε και για τη βιβλιοθήκη *BeautifulSoup*. Αφού επισκεφθούμε τη σελίδα *http://wwwsearch.sourceforge.net/mechanize/* κάνουμε κλικ στο σύνδεσμο *download* και κατεβάζουμε τη τελευταία έκδοση. Το αρχείο που θα κατεβάσουμε είναι σε συμπιεσμένη μορφή, οπότε το αποσυμπιέζουμε και τοποθετούμε ολόκληρο το φάκελο *'mechanize'* στο *path* της *Python*.

10.5 xlwt

Το *xlwt* είναι μια βιβλιοθήκη για τη δημιουργία αρχείων *excel*. Για να το εγκαταστήσουμε στα *windows* κατεβάζουμε τον *Windows Installer* από τη σελίδα *http://pypi.python.org/pypi/xlwt* και τον εκτελούμε. Σε *Linux* συστήματα μπορούμε να εκτελέσουμε τις παρακάτω εντολές στο terminal:

```
$ tar xzf xlwt.tgz
$ cd xlwt-0.7.2
$ python setup.py install
```

10.6 py2exe

Για να τρέξουμε την εφαρμογή μας σε *Windows* συστήματα που δεν έχουν εγκατεστημένη την *Python,* θα χρειαστεί να δημιουργήσουμε ένα εκτελέσιμο αρχείο (*executable file - .exe).* Για τη δημιουργία του εκτελέσιμου θα χρησιμοποιήσουμε τη βιβλιοθήκη *py2exe* της *Python*. Τον τρόπο που θα το κάνουμε αυτό θα τον δούμε στο επόμενο κεφάλαιο. Για να κατεβάσουμε και να εγκαταστήσουμε τη βιβλιοθήκη, πάμε στη σελίδα *http://sourceforge.net/projects/py2exe/files/* και προσέχουμε έτσι ώστε να κατεβάσουμε το αρχείο που αντιστοιχεί στην έκδοση της *Python* που χρησιμοποιούμε. Για παράδειγμα, κατεβάζουμε το αρχείο *py2exe-0.6.9.win32-py2.7.exe* εάν έχουμε εγκαταστήσει την έκδοση 2.7 της *Python* στον υπολογιστή μας.

Επίλογος

Το κατέβασμα και η εγκατάσταση όλων των εργαλείων που θα μας βοηθήσουν στην ανάπτυξη της εφαρμογής ήταν απαραίτητη. Το επόμενο στάδιο είναι να αρχίσουμε να γράφουμε το κώδικα της εφαρμογής μας. Όλη η πορεία ανάπτυξης περιγράφεται στο επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 11
ΔΗΜΙΟΥΡΓΙΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ (ΜΕΡΟΣ ΙΙΙ) – ΑΝΑΠΤΥΞΗ ΤΟΥ ΚΩΔΙΚΑ

Εισαγωγή

Αφού αναφέραμε μερικά χαρακτηριστικά της *Python* και είδαμε πως θα εγκαταστήσουμε τη γλώσσα αλλά και τις εξωτερικές βιβλιοθήκες που θα χρησιμοποιήσουμε στην εφαρμογή, ήρθε η ώρα να αρχίσουμε να προγραμματίζουμε. Σε αυτό το κεφάλαιο θα δούμε και θα επεξηγήσουμε τα βασικότερα κομμάτια κώδικα της εφαρμογής. Ολόκληρος ο κώδικας υπάρχει στο συνοδευτικό CD της πτυχιακής εργασίας.

Στην ενότητα 2.3 είδαμε ότι τα τρία βασικά βήματα δημιουργίας ενός προγράμματος εξόρυξης δεδομένων μετά την εξερεύνηση του τρόπου δομής του ιστότοπου είναι η λήψη, η εξαγωγή και η αποθήκευση των δεδομένων. Αυτήν ακριβώς τη πορεία θα ακολουθήσουμε για τη δημιουργία της εφαρμογής.

11.1 Λήψη των δεδομένων

Το πρώτο βήμα που πρέπει να ακολουθήσουμε είναι να βρούμε ένα τρόπο να κατεβάσουμε τις *HTML* σελίδες του χρυσού οδηγού από τις οποίες θέλουμε να εξάγουμε τα δεδομένα. Το ποιες θα είναι αυτές οι σελίδες θα το ορίζει ο χρήστης μέσω του γραφικού περιβάλλοντος που θα δούμε στο τέλος του κεφαλαίου. Αυτό που μας ενδιαφέρει αυτή τη στιγμή είναι να δημιουργήσουμε τις μεθόδους για να περιηγηθούμε από σελίδα σε σελίδα και να ανακτήσουμε τον HTML κώδικα. Αυτές τις μεθόδους θα τις βάλουμε σε ένα αρχείο με όνομα '*Browser.py*'.

Οι δυο βιβλιοθήκες που θα χρησιμοποιήσουμε είναι η *BeautifulSoup* και η *Mechanize*, οπότε πρέπει να τις κάνουμε *import* στην αρχή του κώδικά μας.

```
1: import BeautifoulSoup
2: import mechanize
```

Ο βασικός στόχος μας είναι να μπαίνουμε στη σελίδα και να αποθηκεύουμε το κώδικά της σε μια μεταβλητή. Αυτό θα το κάνουμε μέσω δυο μεθόδων· και οι δυο με την ίδια λειτουργικότητα μόνο που η μια θα παίρνει ως όρισμα το *url* της σελίδας που θέλουμε να κατεβάσουμε και η άλλη μια κανονική έκφραση για το προσδιορισμό της σελίδα που επιθυμούμε.

```
4: def open by url(url):
5:     html = br.open(url).read()
6:     soup = BeautifulSoup(html)
7:     return soup
8:
9: def open by regex(re):
10:    html = br.follow link(text regex=re).read()
11:    soup = BeautifulSoup(html)
12:    return soup
```

Όπως βλέπουμε, οι δυο μέθοδοι ανοίγουν μια νέα σελίδα με τη βοήθεια είτε της μεθόδου *open(),* είτε της μεθόδου *follow_link()* της βιβλιοθήκης Mechanize και δίνουν μια νέα τιμή στο στιγμιότυπο *soup* του αντικειμένου *BeautifulSoup*. Το στιγμιότυπο soup είναι αυτό που θα μας βοηθήσει να εξάγουμε τα δεδομένα μέσα από τον *HTML* κώδικα, οπότε είναι αυτό που επιστρέφουμε στο τέλος της μεθόδου.

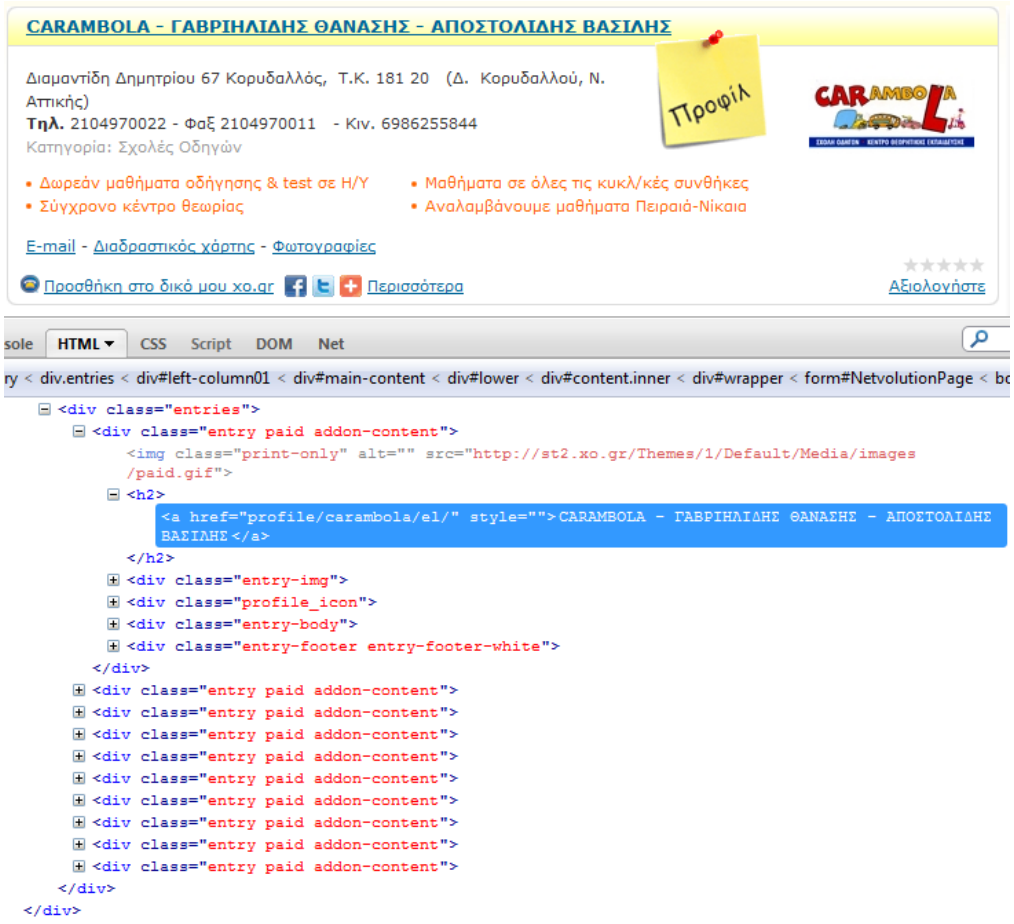
11.2 Εξόρυξη των δεδομένων

Αφού φτιάξουμε τις μεθόδους για την ανάκτηση των ιστοσελίδων, θα δημιουργήσουμε ένα αρχείο με όνομα *'Parser.py',* στο οποίο θα προσθέσουμε τις μεθόδους για την εξαγωγή των δεδομένων που χρειαζόμαστε από τον *HTML* κώδικα των σελίδων. Για την κατανόηση της διαδικασίας που ακολουθήσαμε θα δούμε τη δημιουργία μιας απλής μεθόδου για την εξαγωγή των επωνυμιών των καταχωρίσεων του χρυσού οδηγού.

Πριν δημιουργήσουμε τη μέθοδο, πρέπει να δούμε τη δομή του κώδικα του χρυσού οδηγού. Ας κάνουμε για παράδειγμα μια αναζήτηση για σχολές οδηγών:

Εικόνα 7 – Αναζήτηση κατηγοριών στο Χρυσό Οδηγό

Παρατηρούμε ότι υπάρχουν δέκα καταχωρίσεις ανά σελίδα. Ας δούμε τώρα και των κώδικα :



Εικόνα 8 – Η δομή του κώδικα του Χρυσού Οδηγού

Η επωνυμία των καταχωρίσεων είναι το κείμενο ενός στοιχείου *anchor,* το οποίο είναι παιδί ενός στοιχείου *h2,* το οποίο με τη σειρά του είναι παιδί ενός στοιχείου *div*. Και οι δέκα καταχωρίσεις που βρίσκονται στη σελίδα είναι παιδιά του *div* με ιδιότητα *class='entries'*. Αυτό που έχουμε να κάνουμε λοιπόν είναι να πάρουμε με κάποιο τρόπο το κείμενο των συνδέσμων που περιέχουν την ονομασία της καταχώρισης:

```
1: def parse_title(soup):
2:     tags = soup.find("div", { "class" : "entries" }).findAll("h2")
3:     data=[]
4:     for text in tags:
5:         stuff = text.contents[0].string
6:         data.append(stuff)
7:     return data
```

Αρχικά, περάσαμε στη μέθοδό μας ως παράμετρο το στιγμιότυπο 'soup' που είδαμε στην προηγούμενη παράγραφο. Στην επόμενη γραμμή, με τη βοήθεια της μεθόδου *find* του *BeautifulSoup,* βρήκαμε όλες τις επικεφαλίδες *h2* που βρίσκονται μέσα στο *div* με ιδιότητα *class* ίση με *entries*. Ορίσαμε μια λίστα, και μέσα από έναν επαναληπτικό βρόγχο, προσθέσαμε το κείμενο του πρώτου παιδιού κάθε επικεφαλίδας (δηλαδή το *text* του στοιχείου *anchor*) στη λίστα μας.

11.3 Αποθήκευση των δεδομένων

Τα δεδομένα που εξάγαμε μέσω της προηγούμενης διαδικασίας στην εφαρμογή μας, ήταν για κάθε καταχώριση η επωνυμία, η διεύθυνση, το τηλέφωνο/φαξ/κινητό, ο ταχυδρομικός κώδικας, η ιστοσελίδα και το e-*mail*. Όλα αυτά τα δεδομένα τα αποθηκεύσαμε προσωρινά σε δομές δεδομένων τύπου λίστας. Όμως για να τα διαχειριστούμε σωστότερα, αλλά και για να κάνουμε νέες αναζητήσεις για άλλα δεδομένα, πρέπει να τα αποθηκεύσουμε σε κάποια βάση δεδομένων ή κάποιο υπολογιστικό φύλλο.

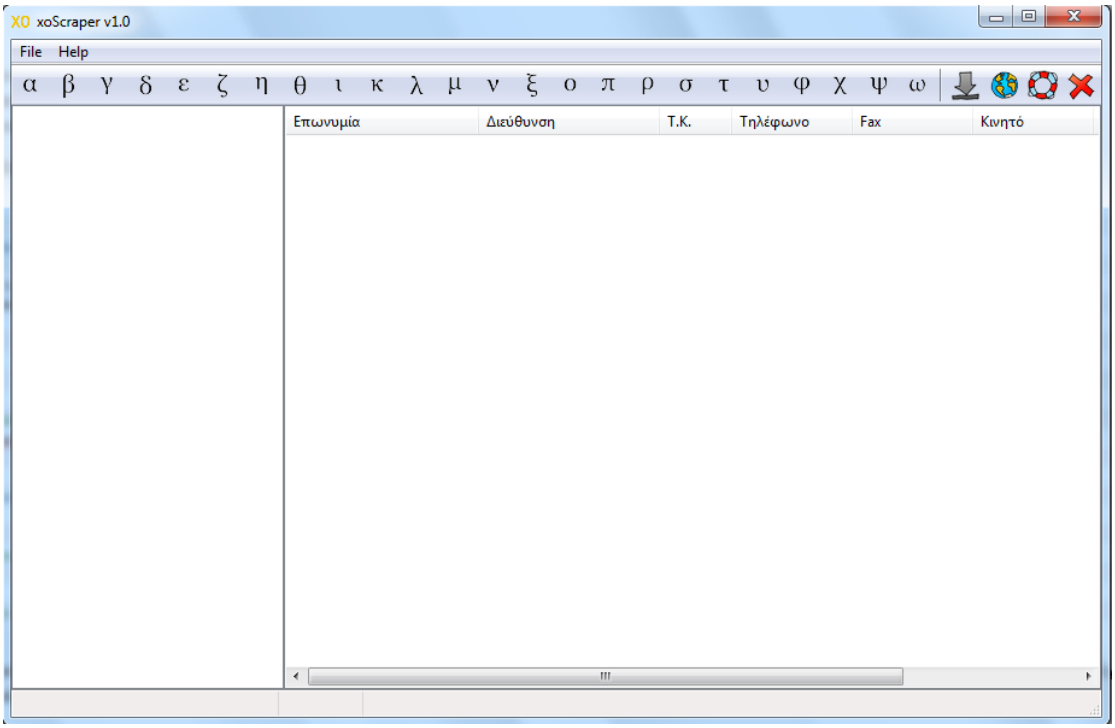
Στην εφαρμογή μας θα δημιουργήσουμε ένα υπολογιστικό φύλο χρησιμοποιώντας τη βιβλιοθήκη *xlwt*. Οι βασικές λειτουργίες, δηλαδή δημιουργία υπολογιστικού φύλλου και προσθήκη δεδομένων σε αυτό, θα βρίσκονται σε ένα αρχείο το οποίο θα ονομάσουμε '*FileCreator.py*'.

```
 1: import xlwt
 2:
 3: xlwt.Workbook
 4: book = Workbook(encoding='utf-8')
 5:
 6: sheet = book.add sheet('sheet 1')
 7:
 8: def write to sheet(row num, col num, data):
 9:         sheet.write(row num, col num, data)
10:
11: def save workbook(w name):
12:         book.save(w name)
```

Όπως παρατηρούμε ο κώδικας είναι πολύ απλός. Το πρώτο πράγμα που έχουμε να κάνουμε είναι να εισάγουμε τη βιβλιοθήκη *xlwt* στο κώδικά μας μέσω της εντολής *import*. Στη τρίτη γραμμή δημιουργούμε ένα νέο έγγραφο excel αρχικοποιώντας ένα αντικείμενο *xlwt.Workbook*. Αφού το κάνουμε αυτό, δημιουργούμε ένα νέο υπολογιστικό φύλλο με τη μέθοδο *add_sheet()* της κλάσης *Workbook*. Η μέθοδος *write_to_sheet*() εισάγει δεδομένα στο υπολογιστικό φύλλο με βάση τον αριθμό της στήλης και της γραμμής που παίρνει ως παράμετρο, ενώ η *save_workbook()* αποθηκεύει το υπολογιστικό φύλλο με το όνομα που περνάμε ως παράμετρο σε αυτήν.

11.4 Δημιουργία του γραφικού περιβάλλοντος

Σε αυτή την ενότητα θα δούμε τον τρόπο δημιουργίας του βασικού παραθύρου της εφαρμογής μας (ολόκληρος ο κώδικας υπάρχει στο συνοδευτικό CD της πτυχιακής εργασίας, ενώ τα υπόλοιπα στοιχεία της εφαρμογής μπορείτε να βρείτε και στην ενότητα 'Οδηγός Χρήσης Λογισμικού'). Το βασικό παράθυρο είναι αυτό που βλέπουμε όταν ανοίγουμε την εφαρμογή.



Εικόνα 9 – Βασικό παράθυρο της εφαρμογής

Όπως παρατηρούμε, το παράθυρο αποτελείται από πέντε βασικά μέρη. Αυτά είναι τα εξής:

1. Η κεντρική μπάρα μενού
2. Η μπάρα εργαλείων ή συντομεύσεων
3. Η αριστερή λίστα, όπου εμφανίζονται η κατηγορίες του Χρυσού Οδηγού
4. Η δεξιά λίστα ,όπου εμφανίζονται οι καταχωρίσεις των κατηγοριών
5. Η μπάρα κατάστασης στο κάτω μέρος του παραθύρου

Για τη δημιουργία του παραθύρου κατασκευάζουμε μια κλάση που θα ονομάσουμε *Window* και η οποία θα περιέχει μια μέθοδο *__init__* για την αρχικοποίηση όλων αυτών των στοιχείων:

```
1: class Window(wx.Frame):
2:     """
3:     Δημιουργία του κεντρικού παραθύρου της εφαρμογής
4:     """
5:     def   init  (self, parent, id, title):
6:         wx.Frame.  init  (self, parent, id, title)
7:
8:         self. init menubar()
9:         self. init toolbar()
10:        self. init lists()
11:
12:        #Δημιουργούμε μια μπάρα κατάστασης
13:        self.statusbar = self.CreateStatusBar()
14:
15:        #Ορίζουμε ένα εικονίδιο
16:        icon = wx.Icon("icons/favicon.ico", wx.BITMAP TYPE ICO)
17:        self.SetIcon(icon)
```

```
18:
19:         #Ορίζουμε το μέγεθος της εφαρμογής μας
20:         self.SetSize((890,600))
21:
22:         #Τοποθετούμε το παράθυρο στο κέντρο της οθόνης
23:         self.Center()
```

Παρατηρούμε ότι για την κατασκευή της κεντρικής μπάρας, της μπάρας εργαλείων και των δύο λιστών καλούμε τρεις μεθόδους (*_init_menubar*, *_init_toolbar* και *_init_lists*). Αυτό τι κάναμε απλά για τη δημιουργία πιο ευανάγνωστου κώδικα. Παρακάτω βλέπουμε το κώδικα της μεθόδου *_init_lists*:

```
1: def _init_lists(self):
2:     """
3:     Δημιουργία ενός listbox όπου θα μπαίνουν οι κατηγορίες
4:     και ενός list όπου θα μπαίνουν οι καταχωρήσεις των κατηγοριών.
5:     """
6:     panel = wx.Panel(self, -1)
7:
8:     #Δημιουργία ενός Listbox
9:     self.listbox = wx.ListBox(panel, 25)
10:    self.Bind(wx.EVT_LISTBOX, self.OnSelect, id=25)
11:
12:    #Δημιουργία ενός List
13:    self.list = wx.ListCtrl(panel, 26, style=wx.LC_REPORT)
14:    self.list.InsertColumn(27, 'Επωνυμία', width=140)
15:    self.list.InsertColumn(28, 'Διεύθυνση', width=150)
16:    self.list.InsertColumn(29, 'Τ.Κ.', width=60)
17:    self.list.InsertColumn(30, 'Τηλέφωνο', width=100)
18:    self.list.InsertColumn(31, 'Ιστοσελίδα', width=100)
19:    self.list.InsertColumn(32, 'E-mail', width=100)
20:
21:    #Τοποθέτηση των λιστών σε οριζόντια διάταξη
22:    hbox = wx.BoxSizer(wx.HORIZONTAL)
23:    hbox.Add(self.listbox, 1, wx.EXPAND)
24:    hbox.Add(self.list, 3, wx.EXPAND)
25:
26:    panel.SetSizer(hbox)
```

Για τη δημιουργία του *listbox* χρησιμοποιήσαμε τη κλάση *ListBox*, ενώ για τη δημιουργία της λίστας μας χρησιμοποιήσαμε τη κλάση *ListCtrl* (καί οι δύο κλάσεις ανήκουν στο πακέτο *wx* της βιβλιοθήκης *wxPython*). Στη λίστα προσθέσαμε τις στήλες Επωνυμία, Διεύθυνση, Τ.Κ., Τηλέφωνο, Ιστοσελίδα και E-mail. Τέλος, χρησιμοποιήσαμε την κλάση *BoxSizer* για την οριζόντια στοίχιση των δυο λιστών.

11.5 Δημιουργία του εκτελέσιμου αρχείου

Την εφαρμογή που δημιουργήσαμε μπορεί να θέλουμε να την εκτελέσουμε σε windows συστήματα στα οποία δεν υπάρχει εγκατεστημένο το περιβάλλον της *Python*. Για να το κάνουμε αυτό θα πρέπει να μεταγλωττίσουμε τα *.py* αρχεία μας σε ένα εκτελέσιμο αρχείο (*.exe*) των windows. Η μεταγλώττιση του πηγαίου κώδικα είναι η μετατροπή του σε γλώσσα μηχανής, δηλαδή τη γλώσσα που κατανοεί η υπολογιστής μας.

Το πρώτο βήμα είναι να κατεβάσουμε και να εγκαταστήσουμε τη βιβλιοθήκη *py2exe (βλέπε ενότητα 10.6).* Αφού εγκαταστήσουμε τη βιβλιοθήκη θα πρέπει να δημιουργήσουμε ένα python αρχείο με το όνομα *setup.py*. Το περιεχόμενο αυτού του αρχείου θα πρέπει να είναι το εξής:

```
1: from distutils.core import setup
2: import py2exe
3: setup(console=['xoScraper.py'])
```

Αποθηκεύουμε το αρχείο *setup.py* στον ίδιο φάκελο με το *xoScraper.py* και ανοίγουμε μια γραμμή εντολών (δεν μπορούμε να τρέξουμε το *setup.py* με το *IDLE*, πρέπει να χρησιμοποιήσουμε τη γραμμή εντολών των windows). Πληκτρολογούμε "*cd C:\MyFolder*" (Πρέπει να αλλάξουμε το *MyFolder* με τη διαδρομή που βρίσκεται ο φάκελος στον οποίο έχουμε τα αρχεία της εφαρμογής μας). Από εκεί εκτελούμε την εξής εντολή:

`python.exe setup.py py2exe`

Όταν η μεταγλώττιση ολοκληρωθεί θα έχουν δημιουργηθεί δύο νέοι φάκελοι, που ονομάζονται *build* και *dist*. Μπορείτε να διαγράψετε το φάκελο build γιατί περιέχει μόνο τα αρχεία που χρησιμοποιήθηκαν κατά τη μεταγλώττιση. Ο φάκελος *dist* είναι αυτός που περιέχει τα χρήσιμα αρχεία, συμπεριλαμβανομένου και του *xoScraper.exe* που είναι το εκτελέσιμο αρχείο μας.

Επίλογος

Το πιο δύσκολο μέρος κατά την ανάπτυξη της εφαρμογής ήταν σίγουρα η δημιουργία των μεθόδων για την εξαγωγή των δεδομένων από τις HTML σελίδες. Αυτό συμβαίνει γιατί από σελίδα σε σελίδα ο κώδικας μπορεί να έχει κάποιες αλλαγές. Για παράδειγμα, στις σελίδες του χρυσού οδηγού πολλές καταχωρίσεις είχαν διαφορετικές ιδιότητες class, ενώ από πολλές έλειπαν και στοιχεία όπως η διεύθυνση και ο Τ.Κ. Αυτό έκανε την αποσφαλμάτωση (debugging) του κώδικα μια δύσκολη διαδικασία. Γενικά, αυτό είναι ένα πρόβλημα που συναντάμε σε όλα τα προγράμματα αυτού του είδους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Segaran, T. (2007), **Programming Collective Intelligence**, Sebastopol: O'Reilly

Calishain, T., & Hemenway, K. (2004), **Spidering Hacks**, Sebastopol: O'Reilly

Schrenk, M. (2007), **Webbots, Spiders, and Screen Scrapers**, San Francisco: No Starch

Hetland, L. M. (2008), **Beginning Python: From Novice to Professional, Second Edition**, New York: Apress

Watson, M. (2009), **Scripting Intelligence: Web 3.0 Information Gathering and Processing**, New York: Apress

Rappin, N. & Dunn, R. (2006), **wxPython in Action**, Greenwich: Manning

Turland, Matthew, (2010), **php|architect's Guide to Web Scraping**, Toronto: Marco Tabini

Keith, Jeremy, (2005), **DOM Scripting**, New York: Friendsof

Goyvaerts, Jan & Levithan, Steven, (2009), **Regular Expressions Cookbook**, Sebastopol: O'Reilly

Payne, James, (2010), **Beginning Python: Using Python 2.6 and Python 3.1**, Indianapolis: Wiley

Baird, Kevin, (2007), **Ruby By Example: Concepts and Code**, San Francisco: No Starch

Orchard, Leslie, (2005), **Hacking RSS and Atom**, Indianapolis: Wiley

ΔΙΑΔΙΚΤΥΑΚΕΣ ΠΗΓΕΣ

Ruby
HPricot: http://hpricot.com/
Nokogiri: http://nokogiri.org/
RubyfulSoup: http://www.crummy.com/software/RubyfulSoup/
Mechanize: http://mechanize.rubyforge.org/mechanize/
ScRUBYt!: http://scrubyt.rubyforge.org/
Watir: http://watir.com/

Perl
Τα documentations για όλες τις βιβλιοθήκες της Perl βρίσκονται στη σελίδα http://search.cpan.org/

Python
BeautifulSoup: http://www.crummy.com/software/BeautifulSoup/
Mechanize: http://wwwsearch.sourceforge.net/mechanize/
lxml: http://lxml.de/
scrape.py: http://zesty.ca/scrape/
wxPython: http://www.wxpython.org/
xlwt: http://pypi.python.org/pypi/xlwt

PHP
Simple HTML DOM parser: http://simplehtmldom.sourceforge.net/
htmlSQL: http://www.jonasjohn.de/lab/htmlsql.htm

Java
JSoup: http://jsoup.org/

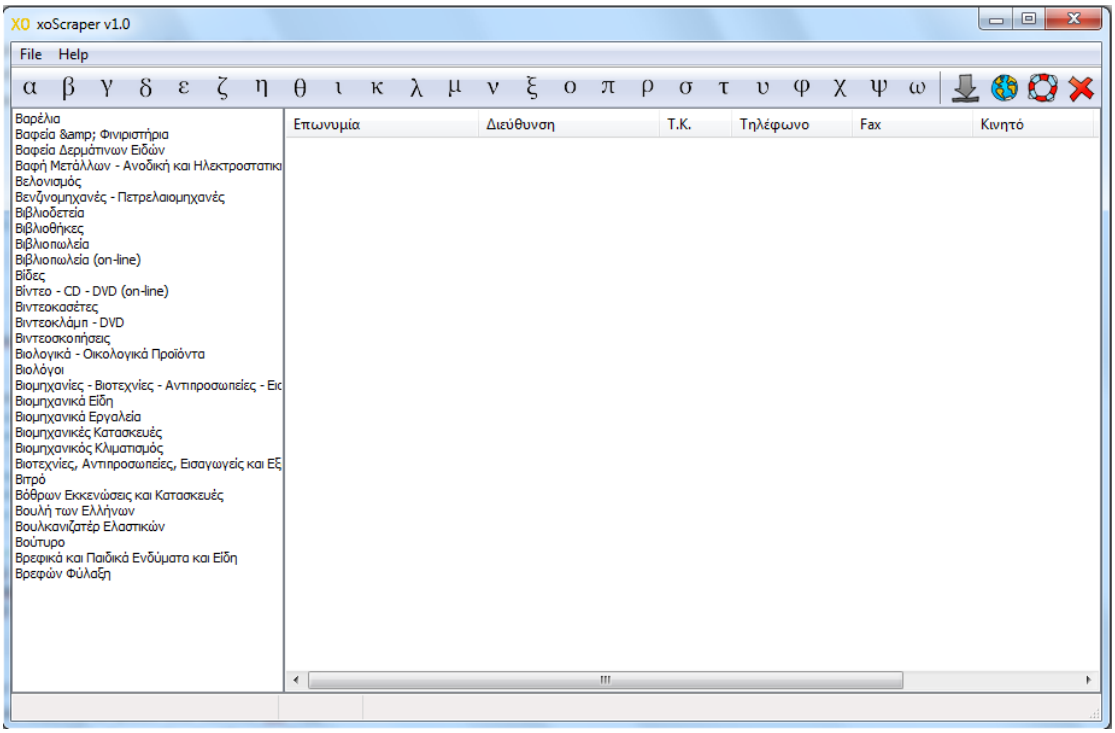
.NET
HTML Agility Pack: http://htmlagilitypack.codeplex.com/

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Το κεντρικό παράθυρο της εφαρμογής το είδαμε στην ενότητα 11.4 (εικόνα 9). Εδώ θα δούμε βασικές οδηγίες χρήσης της εφαρμογής.

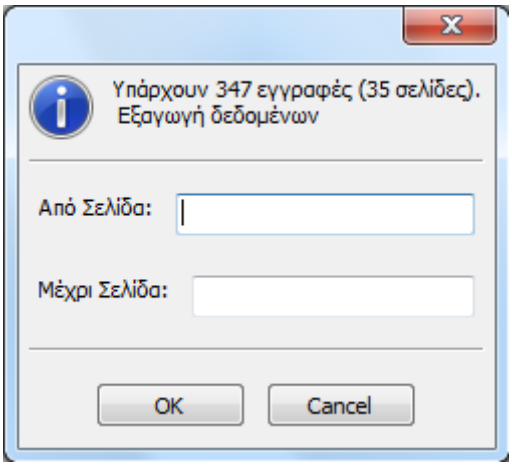
i. Επιλογή επαγγελματικής κατηγορίας βάσει αλφαβητικής σειράς

Για την επιλογή μιας επαγγελματικής κατηγορίας βάσει αλφαβητικής σειράς επιλέγουμε το αρχικό γράμμα της κατηγορίας από τη μπάρα εργαλείων. Για παράδειγμα, επιλέγοντας το γράμμα 'β' θα εμφανιστούν όλες οι επαγγελματικές κατηγορίες οι οποίες ξεκινούν με αυτό το γράμμα:



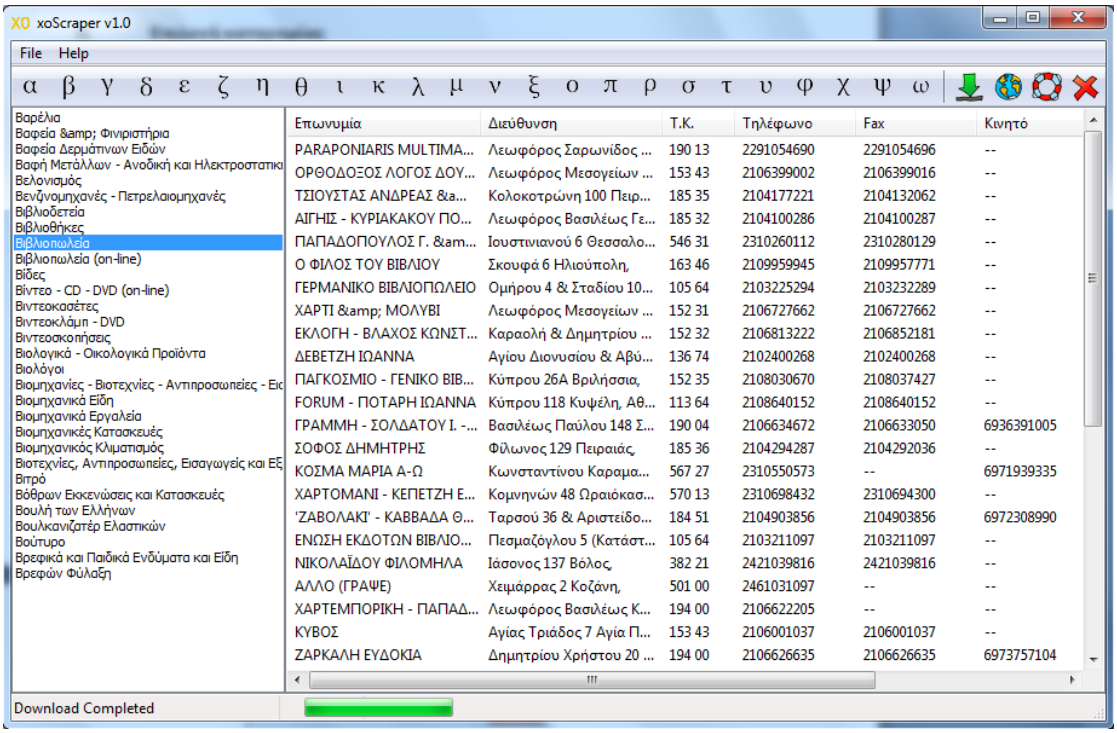
ii. Επιλογή κατηγορίας

Αφού επιλέξουμε επαγγελματική κατηγορία από την αριστερή λίστα θα εμφανιστεί ένα παράθυρο διαλόγου που θα μας πληροφορεί για το πλήθος των καταχωρίσεων στη συγκεκριμένη κατηγορία και θα μας ζητάει να εισάγουμε από ποιες σελίδες θέλουμε να εξάγουμε τα δεδομένα. Για παράδειγμα, εάν επιλέξουμε την κατηγορία 'Βιβλιοπωλεία' θα εμφανιστεί το παρακάτω παράθυρο διαλόγου:



iii. Εμφάνιση καταχωρίσεων

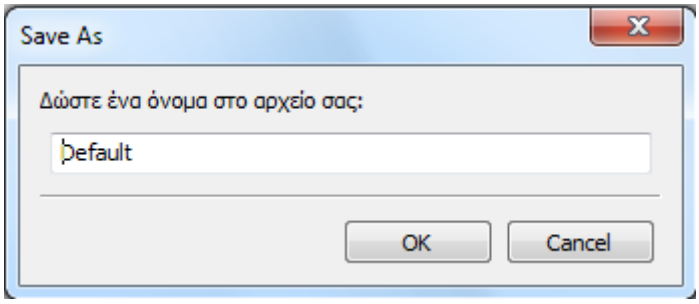
Από τη στιγμή που θα εισάγουμε από ποια έως ποια σελίδα θέλουμε τα εξάγουμε τις πληροφορίες, κάνουμε κλικ στο κουμπί OK και εμφανίζονται όλες οι καταχωρίσεις που ανήκουν στη κατηγορία που έχουμε επιλέξει. Στη παρακάτω εικόνα έχουμε επιλέξει να εμφανιστούν οι καταχωρίσεις για τη κατηγορία 'Βιβλιοπωλεία':



Κατά τη διάρκεια κατεβάσματος των ιστοσελίδων ενημερωνόμαστε για το ποια σελίδα κατεβαίνει κάθε στιγμή αλλά και τη συνολική πρόοδο όλης διαδικασίας μέσω της γραμμής κατάστασης (statusbar).

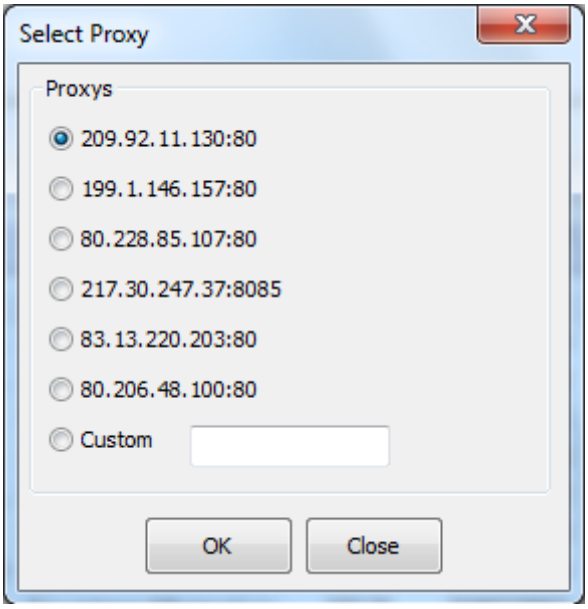
iv. Αποθήκευση των δεδομένων

Την αποθήκευση των δεδομένων μπορούμε να τη πραγματοποιήσουμε μέσω της επιλογής File → Save (συντόμευση Ctrl+S) από τη κεντρική μπάρα ή πατώντας το πράσινο κουμπί που βρίσκετε στη μπάρα εργαλείων. Το κουμπί στη μπάρα εργαλείων ενεργοποιείτε (αποκτά πράσινο χρώμα) μόνο όταν έχουμε εμφανίσει κάποιες καταχωρίσεις στη λίστα μας. Σε διαφορετική περίπτωση μένει απενεργοποιημένο. Αφού επιλέξουμε έναν από τους δυο τρόπους, εμφανίζετε στην οθόνη μας το παρακάτω παράθυρο διαλόγου που μας προτρέπει να δώσουμε ένα όνομα στο αρχείο xls που θα αποθηκεύσουμε.



v. Επιλογή Proxy Server

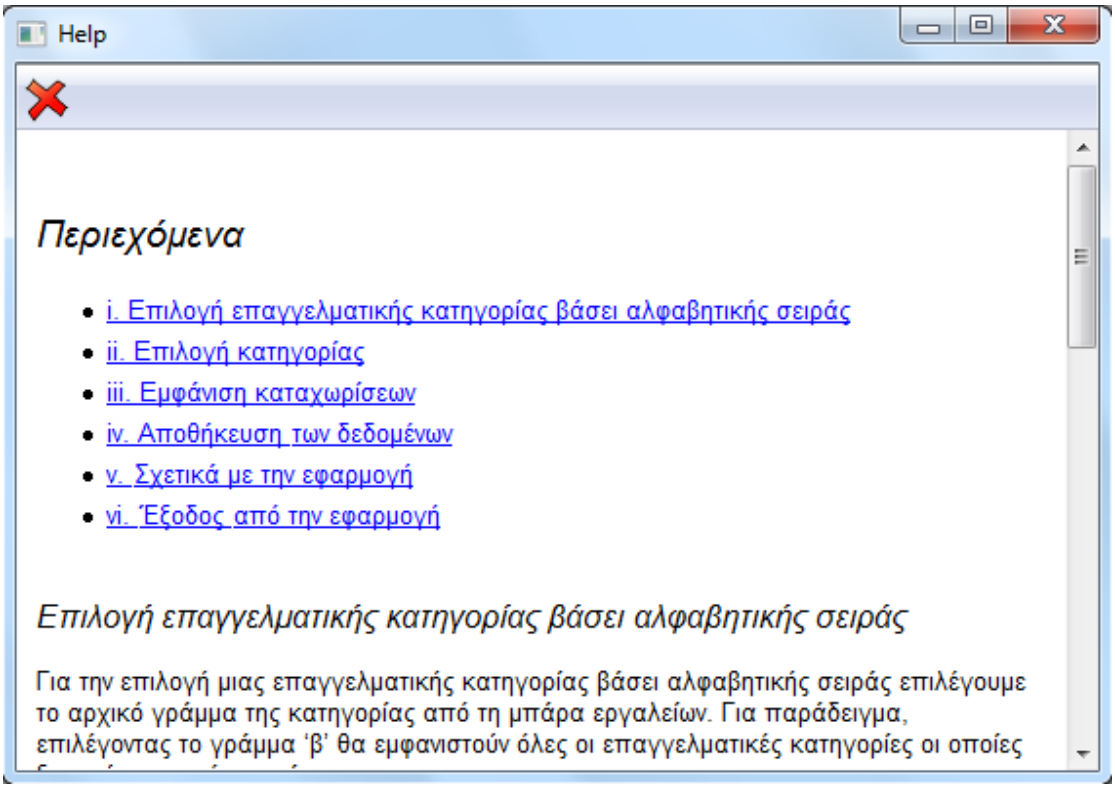
Εάν θέλουμε να χρησιμοποιήσουμε κάποιον proxy server για την σύνδεσή μας με τον χρυσό οδηγό μπορούμε να το κάνουμε πατώντας το κουμπί με το σχήμα της υδρογείου στη μπάρα εργαλείων. Επιλέγοντας αυτό το κουμπί θα εμφανιστή το παρακάτω παράθυρο διαλόγου:



Από αυτό το παράθυρο μπορούμε να επιλέξουμε ανάμεσα στους ήδη υπάρχοντες proxy servers της λίστας ή να εισάγουμε κάποιον δικό μας μέσω της επιλογής *'Custom'*.

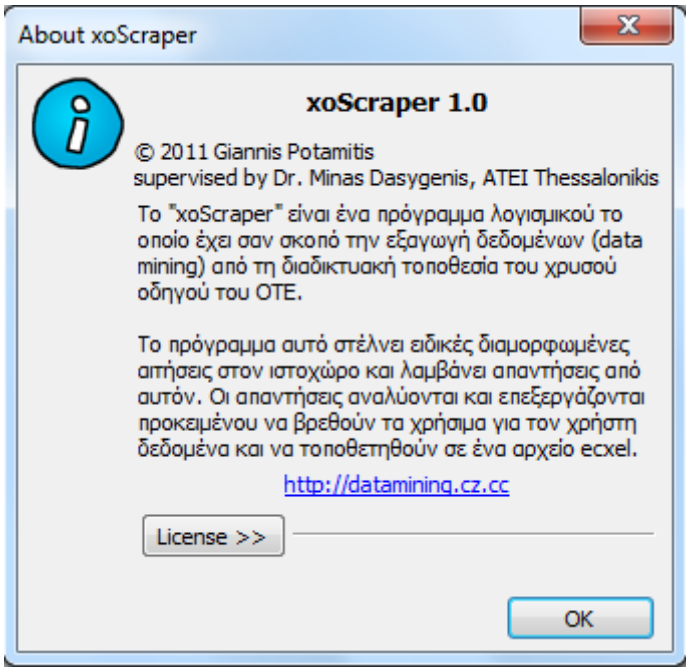
vi. Επιλογή βοήθειας

Για την εμφάνιση του παραθύρου βοήθειας έχουμε ξανά δυο επιλογές: Πρώτων, μέσω της επιλογής Help → xoScraper Help (συντόμευση Ctrl+H) και δεύτερων, κάνοντας κλικ στο κουμπί με το σχήμα σωσίβιου στη μπάρα εργαλείων. Το παράθυρο βοήθειας είναι το παρακάτω:



vii. Σχετικά με την εφαρμογή

Επιλέγοντας Help → About (συντόμευση Ctrl+A) θα εμφανιστεί ένα παράθυρο με πληροφορίες για την εφαρμογή όπως το έτος κατασκευής της, το όνομά της, την έκδοσή της, το όνομα του δημιουργού, μερικές πληροφορίες για τη λειτουργία της και την ιστοσελίδα στην οποία μπορείτε να βρείτε περισσότερες πληροφορίες για αυτήν.



viii. Έξοδος από την εφαρμογή

Η εφαρμογή κλείνει επιλέγοντας File → Quit (συντόμευση Ctrl+Q) ή κάνοντας κλικ στο κουμπί με το σύμβολο 'Χ' στη μπάρα εργαλείων. Πριν κλείσει η εφαρμογή θα εμφανιστεί ένα παράθυρο διαλόγου για την επικύρωση του τερματισμού της εφαρμογής.