Dario Bonino, Luigi De Russis

LAB 2 – PYTHON SOFTWARE

EXERCISE – SIMPLE MP3 LIST

Write a program that, given an initial “root folder”, scans the file system (including sub-folders and links) and indexes music files (consider .flac and .mp3 files, only) extracting the associated metadata (e.g., ID3 tags). Given the resulting “list” of music *tracks*, the program should provide simple search based on tags' content. All functions shall be accessible through a command line interface supporting the following commands, commands marked as “optional” may be skipped:

Command	Description	
`index <folder_name>`	Indexes the given folder; may be called more than on time for different directories	
`search <tag_name> <text>`	Search for tracks having the given <text> in the given <tag>	(optional)
`list`	Displays the list of tracks currently indexed (titles only) and the corresponding id. Provides a final summary reporting the number of tracks and the number of .mp3 and .flac files.	
`show <track_id>`	Show the details (i.e., the indexed information) about the music file having the given <track_id>	
`exit`	Quit the program	

Suggestions:

- *Exploit the mutagen python module to extract track tags*
- *Only consider title, album, genre, artist tags (might also be empty for some files)*
- *Use a scalable approach defining re-usable objects, e.g., a Track class for modeling track information, Tracks class for handling track collections, etc.*

EXERCISE – SIMPLE MP3 PLAYER

Extend the program developed in the first exercise to allow playing selected tracks. For the sake of simplicity, assume that tracks are uniquely identified by a numeric id, which is part of the search result. Assume simple command-line interface, allowed commands include:

Command	**Description**
`play <track_id>`	Plays the file corresponding to the given track index

`Stop`	Stops the player

Suggestions:

- *Exploit the MPlayer process to actually play files, look at MPlayer man pages for command-line options*
- *The output and input streams of an external process may be captured using the following line of python code:*
  - `player = Popen("mplayer -slave -quiet -nolirc -msglevel all=-1 -idle", stdin=PIPE, stdout=PIPE, shell=True)`, *then the input stream will be available as the* `stdin` *file and the output stream as the* `stdout` *file.*ù

EXERCISE - TWEET A PYTHON

Write a simple Twitter monitor that, given a Twitter user(name), provides a vocal summary of the latest two tweets for each user's followers.

Optionally, if you have your own Twitter account, write a function for getting a vocal alert every time that the Twitter status of your friends changes.

If you don't have a Twitter account, you can exploit the AmI course account (@AmICourse2015) with the following authentication parameters:

- consumer key: wIDHvofdfV2QO94s1bjebQ
- consumer secret: nO0q0Ko8EBQ6Lb8FNLwEsT3r2QLkjWsO02dr9uegU
- access token: 2408639030-691GXH8B4aQt2JgXN05uSkWAJyywmds6OeLCaI4
- access token secret: I7lxOY8wSBf0bKlWKJ5UlI3tVRoSaYUeiUseRLo9VBoky

To create your own credentials, log in at http://dev.twitter.com and create a new application. Then, generate the access token.

You can exploit the python-twitter library at http://github.com/bear/python-twitter, the available tutorials, the infos reported in the repository (README.md), and any portion of code published in this course. python-twitter can be installed using easy_install or pip.

Suggestion: to get the tweets of a selected user, you can call the following method: api.getUserTimeline(screen_name='@username'), where username is the Twitter screen name.