Log Puzzle Python Exercise

For the Log Puzzle exercise, you'll use Python code to solve two puzzles. This exercise uses the urllib module, as shown in the Python Utilities section. The files for this exercise are in the "logpuzzle" directory inside google-python-exercises (download the google-python-exercises.zip if you have not already, see Set Up for details). Add your code to the "logpuzzle.py" file.

An image of an animal has been broken it into many narrow vertical stripe images. The stripe images are on the internet somewhere, each with its own url. The urls are hidden in a web server log file. Your mission is to find the urls and download all image stripes to re-create the original image.

The slice urls are hidden inside apache log files (the open source apache web server is the most widely used server on the internet). Each log file is from some server, and the desired slice urls are hidden within the logs. The log file encodes what server it comes from like this: the log file animal_code.google.com is from the code.google.com server (formally, we'll say that the server name is whatever follows the first underbar). The animial_code.google.com log file contains the data for the "animal" puzzle image. Although the data in the log files has the syntax of a real apache web server, the data beyond what's needed for the puzzle is randomized data from a real log file.

Here is what a single line from the log file looks like (this really is what apache log files look like):

```
10.254.254.28 - - [06/Aug/2007:00:14:08 -0700] "GET /foo/talks/
HTTP/1.1"

200 5910 "-" "Mozilla/5.0 (X11; U; Linux i686 (x86_64); en-US;
rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4"
```

The first few numbers are the address of the requesting browser. The most interesting part is the "GET *path* HTTP" showing the path of a web request received by the server. The path itself never contain spaces, and is separated from the GET and HTTP by spaces (regex suggestion: \S (upper case S) matches any non-space char). Find the lines in the log where the string "puzzle" appears inside the path, ignoring the many other lines in the log.

Part A - Log File To Urls

Complete the read_urls(filename) function that extracts the puzzle urls from inside a logfile. Find all the "puzzle" path urls in the logfile. Combine the path from each url with the server name from the filename to form a full url, e.g. "http://www.example.com/path/puzzle/from/inside/file". Screen out urls that appear more than once. The read_urls() function should return the list of full urls, sorted into

alphabetical order and without duplicates. Taking the urls in alphabetical order will yield the image slices in the correct left-to-right order to re-create the original animal image. In the simplest case, main() should just print the urls, one per line.

```
$ ./logpuzzle.py animal_code.google.com

http://code.google.com/something/puzzle-animal-baaa.jpg

http://code.google.com/something/puzzle-animal-baab.jpg

...
```

Part B - Download Images Puzzle

Complete the download_images() function which takes a sorted list of urls and a directory. Download the image from each url into the given directory, creating the directory first if necessary (see the "os" module to create a directory, and "urllib.urlretrieve()" for downloading a url). Name the local image files with a simple scheme like "img0", "img1", "img2", and so on. You may wish to print a little "Retrieving..." status output line while downloading each image since it can be slow and its nice to have some indication that the program is working. Each image is a little vertical slice from the original. How to put the slices together to re-create the original? It can be solved nicely with a little html (knowledge of HTML is not required).

The download_images() function should also create an index.html file in the directory with an *img* tag to show each local image file. The img tags should all be on one line together without separation. In this way, the browser displays all the slices together seamlessly. You do not need knowledge of HTML to do this; just create an index.html file that looks like this:

```
<verbatim>

<html>

<body>

<img src="/edu/python/exercises/img0"><img
src="/edu/python/exercises/img1"><img
src="/edu/python/exercises/img2">...

</body>

</html>
```

Here's what it should look like when you can download the animal puzzle:

```
$ ./logpuzzle.py --todir animaldir animal_code.google.com

$ ls animaldir

img0  img1  img2  img3  img4  img5  img6  img7  img8  img9  index.html
```

When it's all working, opening the index.html in a browser should reveal the original animal image. What is the animal in the image?

Part C - Image Slice Descrambling

The second puzzle involves an image of a very famous place, but depends on some custom sorting. For the first puzzle, the urls can be sorted alphabetically to order the images correctly. In the sort, the whole url is used. However, we'll say that if the url ends in the pattern "-*wordchars*-*wordchars*.jpg", e.g. "http://example.com/foo/puzzle/bar-abab-baaa.jpg", then the url should be represented by the **second** word in the sort (e.g. "baaa"). So sorting a list of urls each ending with the word-word.jpg pattern should order the urls by the second word.

Extend your code to order such urls properly, and then you should be able to decode the second place_code.google.com puzzle which shows a famous place. What place does it show?

CC Attribution: the images used in this puzzle were made available by their owners under the Creative Commons Attribution 2.5 license, which generously encourages remixes of the content such as this one. The animal image is from the user zappowbang at flickr and the place image is from the user booleansplit at flickr.

Last updated December 13, 2012.