Assignment 05 (DNSC 6211)

This assignment will require you to use the **`MySQL`** and **`pymongo`** modules.

Your program should do the following

1) Read all `json` files (that contains LP problems) in the current directory[1] and store them in a `pymongo` database. This should be done using a function called **`readAndStore()`**. You don't need to assign your own unique key to the "**`_id`**" field – it is done automatically. You can assign another field to be a unique key if you want to[2].
2) Then the program should read the LP problems from the `pymongo` database, create and solve them (using **`PuLP`**). The creating and solving of the LP problem is available to you from last week. If the "optimal" solution for a problem cannot be found the program should record it as the string "NA".
3) Lastly, the program should provide the output depending on the options given by the user. The program output needs to be sent to either a **text file (-t)**, a table in a **database (-d)**, or to the screen (**default**):
   - the **text file** option requires the problem name and the optimal value to be written to a separate line for each LP problem to a file whose name is provided by the user.
   - the **database** option requires a new row to be added to a table for each LP problem. Each row will contain two attributes: **`problemName`** and the **`optimalValue`**. The program should add the rows to a table whose name is provided by the user. This table should be added to the database called **`LP`**. The program should check if the **`LP`** database exists. If it does not, the program should create it. The program should also check if the table named by the user exists. If it exists, the program should delete the table (i.e., DROP the table) and create another one and name it what the user wanted. Both the fields in the table should be strings of length 20, that is CHAR (20).
   - the **default** output for the program is written to the screen with the problem name and the optimal value written on a separate line for each LP problem

You are required to use a module file and use functions to complete this assignment. I have given yo one useful function below:

```
def getFileNames(fn):
    """
    Input = name of JSON file
    Returns = JSON object
    """
    import os
    included_extenstions = ['json' ] ;
    file_names = [fn for fn in os.listdir(os.getcwd())
        if any([fn.endswith(ext)
        for ext in included_extenstions])];
    return file_names
```

[1] This is a useful reference http://stackoverflow.com/questions/2225564/get-a-filtered-list-of-files-in-a-directory
[2] This is a useful document - http://mycodesite.com/mongodb-basics-and-tips/

Sample outputs are shown below:

```
$ python A05_gwid.py
The optimal value for Whiskas is 0.520
The optimal value for Intro LP is 410.000


$ python A05_gwid.py –d myTable
The optimal value for Whiskas is 0.520
The optimal value for Intro LP is 410.000
Output being sent to table: myTable in the LP database
Output written to table: myTable in the LP database


$ python A05_gwid.py –t myTable.txt
The optimal value for Whiskas is 0.520
The optimal value for Intro LP is 410.000
Output being sent to file: myTable.txt
Output written to file: myTable.txt


$ python A05_gwid.py –t myTable.txt –d myTable
The optimal value for Whiskas is 0.520
The optimal value for Intro LP is 410.000
Output being sent to file: myTable.txt
Output written to file: myTable.txt
Output being sent to table: myTable in the LP database
Output written to table: myTable in the LP database
```

Submit files named: `A05_gwid.py` and `A05Module_Gwid.py`.

How to submit your assignment

1. You need to submit two Python files to Blackboard to the Assignment 05 link.
2. The program should be commented well enough so that the TA or I should not have to struggle with understanding variable names and codes and what statements or code blocks do.
3. The grading rubric is shown on the last page.
4. Name the files `A05_gwid.py` and `A05Module_Gwid.py`.
5. Your program headers for each program should look something like

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 04 09:09:54 2015

@author: kanungo
GWID: G19860011

A brief description of the program not exceeding two lines
"""
```

Rubric for Grading the Programming Assignment

	Unsatisfactory	Satisfactory	Good	Excellent
Delivery	• Completed less than 70% of the requirements. • Not delivered on time or not in correct format (Blackboard or git)	• Completed between 70-80% of the requirements. • Delivered on time, and in correct format (Blackboard or git)	• Completed between 80-90% of the requirements. • Delivered on time, and in correct format (Blackboard or git)	• Completed between 90-100% of the requirements. • Delivered on time, and in correct format (Blackboard or git)
Coding Standards	• No name, date, or assignment title included • Poor use of white space (indentation, blank lines). • Disorganized and messy • Poor use of variables (many global variables, ambiguous naming).	• Includes name, date, and assignment title. • White space makes program fairly easy to read. • Organized work. • Good use of variables (few global variables, unambiguous naming).	• Includes name, date, and assignment title. • Good use of white space. • Organized work. • Good use of variables (no global variables, unambiguous naming)	• Includes name, date, and assignment title. • Excellent use of white space. • Creatively organized work. • Excellent use of variables (no global variables, unambiguous naming).
Documentation	• No documentation included.	• Basic documentation has been completed including descriptions of all variables. • Purpose is noted for each function.	• Clearly documented including descriptions of all variables. • Specific purpose is noted for each function and control structure.	• Clearly and effectively documented including descriptions of all variables. • Specific purpose is noted for each function, control structure, input requirements, and output results.
Runtime	• Does not execute due to errors. • User prompts are misleading or non-existent. • No testing has been completed.	• Executes without errors. • User prompts contain little information, poor design. • Some testing has been completed.	• Executes without errors. • User prompts are understandable, minimum use of symbols or spacing in output. • Thorough testing has been completed	• Executes without errors excellent user prompts, good use of symbols, spacing in output. • Thorough and organized testing has been completed and output from test cases is included.
Efficiency	• A difficult and inefficient solution.	• A logical solution that is easy to follow but it is not the most efficient.	• Solution is efficient and easy to follow (i.e. no confusing tricks).	• Solution is efficient, easy to understand, and maintain.