**Portal Data Entry: Sarah Supp**

This project aims to streamline data entry, error checking, and appending to the database for the rodents and weather portions of the Portal Project. More details and project progress can be found on the wiki (Portal TODO List). I have outlined the tasks the program does (or is supposed to do). In red are questions that I have about getting this to work properly. I just got MySQLdb installed, so the MySQL code has not been checked. This project currently includes 4.py files, 1 .DAT file and 1 .xls file.

**Rodents\_python\_sql.py**

1. Takes an excel file with 2 worksheets which have each been entered by a different person.
2. Compares each worksheet line by line to look for typos
   1. Typos discovered are sent back to the user to determine the correct one or if a new line should be entered.
   2. If user types an invalid row, how do I keep the program from kicking the user out so they have to start completely over again? How do I do this throughout the program?
3. The finished data is written to an excel file named by the user and then sent to the database as ‘newdata’.
   1. Should newdata be written to the excel file AFTER the appropriate checks are made? Further changes may occur.
4. Asks for username and password to connect to the server to run queries.
5. ‘newrat’ is compiled from the last 5 years of data (60 periods) in the database. A copy is also sent to python and named ‘newrat’. Comparisons will be made between ‘newdata’ and ‘newrat’.
6. Before any comparisons are made, the computer looks for ‘X’ in all the tags (left and right). X will not natively occur in any alphanumeric tags. It represents a number(s) on an ear tag that could not be read, usually due to a scab. This \*never\* occurs on two eartags, so we can use the non-scabbed one to return and replace the actual eartag.
7. Compares ‘newdata’ tags to ‘newrat’ tags to make sure that no recaptured individuals in ‘newdata’ have a ‘\*’ next to their tag. This indicates a new animal. Returns a problem if the tag HAS an ‘\*’
8. Compares ‘newdata’ tags to ‘newrat’ tags to make sure that all new individuals in ‘newdata’ have a ‘\*’ next to their new tag. Returns a problem if the tag DOES NOT HAVE an ‘\*’.
   1. Looks for similar tags that may indicate a recording error. Looks for tags that match at 4/6 locations or that are the same except for replaced characters (i.e. 8 and B, 0 and D). This function needs work.
9. Looks for cases where ltag is not NULL. When rtag AND ltag exist, the animal has eartags. If both have ‘\*’ or neither have ‘\*’, there is no problem. If only one tag has an ‘\*’, then a tag has changed.
   1. The computer finds the old tag and replaces it in the database and in ‘newrat’ with the new tag. This way individuals can be tracked.
   2. Some of the tag search functions seem redundant because of having to look at both ‘right’ and ‘left’ tags. Is there a better way to structure this?
10. Look for inconsistencies in species and sex. Return rows with the recaptured individual from ‘newrat’ and let the user decide if a solution can be reached and what this solution is.
11. Update database and/or ‘newdata’ with changes.
    1. I’m not sure ‘problem\_solve’ function does all this for me correctly yet.
12. Record errors and solutions (if reached) in an ErrorLog database.
    1. Still working on structure of this function ‘record\_problem’ and its ability to function properly.
13. Append data to database.
14. Let the user know the program is finished and the number of rows appended. This is a good check that things went as planned.

**rodents\_tests.py**

1. Not currently functional, but will be used to write tests for the rodents program.

**Weather\_python\_sql.py**

1. Imports weather data from a .DAT file, pathname given by the user.
   1. Battery readings (short rows) are not included in the import function.
2. Manipulates the file line by line.
3. tempSoil column is added.
   1. None. We don’t actually record this.
4. Year and Julian date are transformed into month and day, and added to the dataline.
5. Columns are rearranged to match the database
6. User inputs username and password to access server
7. Append weather data to the database.
8. Run queries to update the monthly and daily weather tables.
9. Tell the user the program is finished and the number of rows appended to the hourly table.

**weather\_tests.py**

1. Not currently functional, but will be used to write tests for the weather program.

**Met395.DAT**

1. Can be used to play with the weather program.

**NEWDAT397.xls**

1. Can be used to play with the weather program.