Lab 5 Undead Book

Brigham Ray

Taylor Canada

Section 4

October 20, 2016

Post Lab

1. Understand the Problem:

The test code is already defined in the lab directions but an assumption is that it is mutable!! HAH!!! The test code was changed to fit certain needs. Another assumption is that the book is a dictionary of dictionaries. Another assumption made is that even though there are inappropriate names in the test code, the grade of the lab will not be affected because of it.

The code from main must run through the user defined functions and output the time of the post, the groups of the post, the amount of likes of the post, and what the post says from each character.

2. Plan the Solution:

In terms of the book the structure must be a dictionary of dictionaries with the higher order key values as the post id. The lower order key values represent the posts' contents including likes, text of post and groups, and the id of the poster.

book = {

"barnabas\_one": {"likes": set(), "groups": [], "userid": ["BarnabasCollins"],

"text": [], "post\_time": []},

"barnabas\_two": {"likes": set(), "groups": [], "userid": ["BarnabasCollins"],

"text": [], "post\_time": []},

"barnabas\_three": {"likes": set(), "groups": [], "userid": ["BarnabasCollins"],

"text": [], "post\_time": []},

"casper\_one": {"likes": set(), "groups": [], "userid": ["Casper"],

"text": [], "post\_time": []}

}

The update function outputs the time and userid of the poster, it must be unique for each poster and so that is important. The update also appends the text and groups of the poster to their respective dictionary keys in conjunction with the userid.

def update(book, status, audience, id):

posting\_time = str(time.time())

status = str(status)

audience = str(audience)

userid = str(id)

book[userid]["post\_time"].append(posting\_time)

book[userid]["text"].append(status)

book[userid]["groups"].append(audience)

print("Post made by " + str(book[userid]["userid"]) + " at " + posting\_time)

return userid + " " + posting\_time

The like function takes in the id of the post and who liked the post and updates the key value “likes” which is a set (to avoid duplicates) so that we can count the number of values in the set when we display the number of likes for each post.

def like(book, id, userid):

userid = str(userid)

ids = str(id)

book[ids]["likes"].update([userid])

The unlike function uses the same parameters as the like function but instead of adding to the set of the key value “likes” in each post id, it removes the value which is the userid of the person liking the post.

def unlike(book, id, userid):

userid = str(userid)

ids = str(id)

book[ids]["likes"].discard(userid)

The update function displays the time each post was posted, the id of the person posting, the groups involved in the post, the text of the post, and the amount of the likes in each post.

def display(book, id):

ids = str(id)

print("Time: " + str(book[ids]["post\_time"]))

print("Groups: " + str(book[ids]["groups"]))

likes = book[ids]["likes"]

num\_of\_likes = len(likes)

print("Likes: " + str(num\_of\_likes))

print(str(book[ids]["userid"]) + " says " + str(book[ids]["text"]))

Then run main at the end of the file to see if the function works

3. Implementation and Testing:

The code runs according to plan, the output is not very pretty though, but it outputs what it is supposed to which is what the plan was in the first place.

4. Reflect and Refactor:

The code runs about as well as it needs to but does not resemble the example output. This is troubling as it will affect the grade of this lab but to be quite honest I'm ready to be done with this crap. Refactoring the code to be more compatible with certain logistics would allow it to be pep8 compliant and have a more organized feel. The user id was not factored into the solution of the problem because it was very complicated to implement with the solution of the problem. To make it better, the variables need to be defined properly to run the original test code.