Homework 3

Due: Wednesday Oct 09, 2019, 11:59 pm

To submit:

- Save your work as a .ipynb file. Use the filename: **LASTNAME_HW3.ipynb**
- Upload the file to the *Homework 3* Activity in Moodle

Collaboration:

You may work with other students, but each student should write their own code for the assignment separately. In a comment at the top of your homework please indicate the people with whom you worked on the assignment.

Assignment goals:

In this assignment you will put together many of the ideas that we have covered so far in class (defining and modifying variables, looping, control flow) by writing code to read in raw data from a memory experiment. You'll format this data in a way that will facilitate doing some simple analysis on the data using built-in Python functions.

Problem 1: Reading in a raw data file (20 pts)

Download `hw3_data.txt` from Moodle. This file contains data from a single subject who participated in several sessions of a free recall experiment. Each line/row of the file contains information about a word that was studied during the task, and whether or not the subject recalled the word. Use the following code to (1) read the contents of the file into a list, `full_data`, and (2) assign the first line of the file to a new variable, `line1`:

```
f = open('hw3_data.txt','r')
full_data = f.readlines()
line1 = full_data[0]
```

This file *is tab-delimited*, meaning that the values of interest in each row are separated by tab (`\t`) characters. Each line also ends in a newline (`\n`) character. You can see this for `line1` by displaying its contents:

```
In [32]: line1
Out[32]: 'R1065J\t0\t1\t1\tSEAT\t1\n'
```

Each row contains six pieces of data for a given trial, separated by `\t` characters: subject, session, list, serial position, the word that was presented, and whether or not the word was recalled (0 or 1). In the case of line1, these values are:

Subject:	R1065J	
Session:	0	(first session completed)
List:	1	(first list of the session)
SerialPos:	1	(first item in the list)
Word:	SEAT	
Recalled:	1	(successfully recalled)

Your first job is to write code to extract the contents of line1 into a set of individual variables that correspond to subject, session, list, serial position, word, and recalled. This will involve removing the `\n` character from the end of the line, and splitting the rest of the line into its separate parts.

Some hints:

- The command `type(line1)` will let you check what type of object `line1` is. You'll see that it's a string--so to remove the `\n` from the end of the line you can use the string method `.strip()`.
- You can also use the string method `.split()` to split a string into parts. The default behavior (if you call `.split()` with no input) is to split at whitespace characters. By using `'\t'` as the input, however, you can split the line at the tab characters.

Use these hints to strip and split `line1` and assign the output to a new variable `line1_split`. This should be of `type` list.

Although it's nice to have the data split and stored in a list, it would probably be better to store each piece of data in its own variable. Now, instead of assigning the output of `line1.split('\t')` to a single variable, assign it to a tuple so that each piece of data (subject / session / list / serial position / word / recalled) is stored in its own variable. You'll have to choose variable names (e.g. `subject, session, list_num, serialpos, word, recalled`).

Now, convert the variables that contain numbers to integer type.

Problem 2: Build lists for each variable and display the overall recall performance (30 pts)
Now that you've stripped, split, and converted the data from a **single line** of the file, write code that performs these operations for all lines and stores all the values for a given type of data (subject, session, list, etc…) in a list, with separate lists for each type of data. You'll probably make use of the `.append()` list method.

Your lists should have 1800 elements each, which is the number of trials this subject performed across several sessions.

Once you have your lists, execute the following command to import Python's function for computing the average of a list of numbers:

```
from statistics import mean
```

Then, use the `mean()` function to compute and display the proportion of recalled words for this subject (i.e. the average of the values in the `recalled` list). Store this mean in a variable `prec_overall`.

Problem 3: Compute the proportion recall individually for each session (30 pts)
What if we want to know the subject's proportion recall for each session individually? One way to do this is to identify the identities of the unique sessions in the data. Then, for each unique session number, you loop through the `recalled` list. For each element of the `recalled` list, you check whether it corresponds to the current session number for which you are trying to compute proportion recall. If so, add the recalled value for the current trial to a list where you store the recalled/not recalled data only for the current session. Then, once you've grabbed all the recalled/not recalled data for the current session, compute the mean and store it in a list of session averages (e.g. `prec_by_sess`). Then, iterate back through and do the same for trials from the next unique session.

The first step is to identify the unique names (i.e. the numbers) of the sessions in the file. Luckily, you have all your data about the session number of each trial stored in a list (`session`) from Problem 2. You can then use the `set()` command to create a set of the unique values--by wrapping it in a `list()` command you convert the set to a list:

```
sessions_unique = list(set(session))
```

At the end of computing proportion recall for each session, print the values to the output.

Problem 4: Compute the proportion recall individually for each serial position (20 pts)
Now, update your code from Problem 3 so that instead of computing the proportion recall for each session individually, you do so for each unique serial position. There are 12 unique serial positions (one for each word position on each list).

Bonus Problem: PsychoPy (20 pts)

In class we started programming a recognition memory experiment with individual words presented one at a time. Update your code from Problem 5 of the In Class work (Sept 26) to allow the user to specify the number of words that are used in the experiment. Your code should ask the user for input, check whether it will work, and then execute the experiment using that number of words. You should also write your code so that the user can run it in either ‘test’ or ‘experimental’ modes, such that the words are presented either for 0.25 or waiting for a keypress.