Homework 6 **PSY4219/6219**
Due October 26 in class **Fall 2022**
20 points

For this homework assignment, I want you to create/edit/debug a `.py` file using PyCharm or another IDE. Each question / subquestion should be demarcated by `# %%` to create a cell for each part of a question.

To receive full credit, any function definitions should include doc strings. And as usual, do not hard code, use meaningful variable and function names, and use comments.

Q1 (1 point). Include four screen shots showing your use of PyCharm or another IDE.
(a) One should capture a point where you are writing/editing code (within the IDE).
(b) One should capture you running the full code (within the IDE).
(c) One should capture you running a cell (within the IDE).
(d) One should capture you debugging, setting a breakpoint, and stepping through the code (within the IDE).

Q2. Write code that creates a stem-and-leaf plot. Google "stem and leaf plot" (Wikipedia has a fairly clear explanation at https://en.wikipedia.org/wiki/Stem-and-leaf_display).

This is a simple type of data visualization (we will soon talk about other ways of doing plots and visualizing data in Python). Writing this code will give you some more experience with manipulating data in numpy arrays and using other data structures, mathematical operations, control flow elements, and functions.

Assume that you have a one-dimensional numpy array called `data`. For testing purposes, assume it has the following values (but it should work for other data):

```
data = np.array([68, 47, 63, 76, 44, 64, 81, 66, 106, 68, 72, 72, 46, 75,
                 49, 84, 88])
```

The stem and leaf plot for this data should look something like the following (next page):

```
 4 | 4 6 7 9
 5 |
 6 | 3 4 6 8 8
 7 | 2 2 5 6
 8 | 1 4 8
 9 |
10 | 6
```

Note that we will try running your code using different sets of data, so make sure to test it using other data values; definitely make sure it runs with other sets of positive whole numbers.

For full credit, it should run with data sets such as this (note well how single-digit negative and positive numbers appear on the stem and leaf plot):

```
data2 = np.array([-23.75858, -12.45, -3.4, 4.43, 5.5, 5.678, 16.87,
                  24.7, 56.8])
```

(a) (6 points) I want you to first create a function that is passed `data` as a parameter and creates a data structure (as a single variable) that holds the stem and leaf plot and returns that data structure, without printing the stem-and-leaf. You will need to figure out what might be an appropriate data structure for holding the stem and leaf plot (we discussed a possibility in class, but you can choose to use a different data structure); some data structures are better than others for a problem like this.

While conceptually the stem and leaf plot is easy to understand, it is a bit tricky to code this up right. I suggest you try to work through the steps needed to generate a stem and leaf plot by hand to get a sense of the computational steps you will need to carry out in your code. The idea is to take those steps (the algorithm) and turn it into general-purpose code.

Effectively using the techniques available in an IDE like PyCharm (stepping through code, interrogating variable values and evaluating subparts of complicated expressions in the console) should be useful here. Do not try to search online for algorithms describing how to do a stem and leaf plot.

(b) (3 points) I then want you to create a function that prints the stem and leaf plot (using suitably formatted `print` statements). This function should just take the data structure holding the stem and leaf plot as an argument (from part a) and return nothing (it just has the side effect of printing the stem and leaf plot).

Your printed output should look similar to the stem and leaf plots shown on the Wikipedia page.

Q3. Generate a random sequence of trial conditions in a stop signal experiment.

Recall that in a stop signal experiment, a given trial is a GO trial with probability `p` and a STOP trial with probability `1-p` (a weighted coin flip, Bernoulli process).

For STOP trials, there is an additional randomization of the Stop Signal Delay. For this assignment, you can assume you are given an array of stop signal delays (typically, there might be 3, 4, or 5 of them, in milliseconds, for example [50, 100, 150]). You can also assume that each stop signal delay has an equal probability of being selected on a given trial (in some stop signal experiments, certain stop signal delays might be used more frequently than others) (a weighted coin flip, Bernoulli process).

(a) (3 points) Write a function that takes as arguments the probability `p` of each trial being a GO or a STOP trial, an array of Stop Signal Delays on STOP trials, and a number of experimental trials `N`.

The function should return an array of conditions. You can simply code this so that `-1` indicates a GO trial and a (positive) Stop Signal Delay value (selected from the array of Stop Signal Delays) indicates a STOP trial (with the time given by the value of the number).

(b) (2 points) Then I want you to check the randomization (which you should ALWAYS do). Create a new function that takes as input the array of conditions and prints out (1) the proportion of GO vs. STOP trials, and (2) the tally of the number of trials at different Stop Signal Delays. If the code is working properly, for a large `N` (around 100,000, which is way more than you could ever possibly do in an experiment) the proportion of GO vs. STOP trials should be close to `p` and the number of trials at different Stop Signal Delays should be roughly the same.

Q4 (5 points). While the packages available for Python provide a lot of functions for solving a wide range of problems, sometimes you need to implement an explicit algorithm for yourself.

In class, we talked about permutations and how they are done in numpy (using `shuffle()` and `permutation()` functions).

I want you to implement a function that does permutation in a somewhat different way, following a specific algorithm. This function will do a permutation much like the `randperm()` function in Matlab (you do not need to know Matlab to know how to do this, but you can google the Matlab `randperm` function to see examples of how it is used).

If `N` is an integer, then `randperm(N)` returns a random permutation of the integers in the sequence from `1` to `N` (including both `1` and `N`). For example, `randperm(6)` might return `[2 4 5 6 1 3]` and another call to `randperm(6)` might return `[1 6 2 5 4 3]`.

Write Python code that implements a `randperm()` function. The specific algorithm for producing a random permutation is described here:

https://en.wikipedia.org/wiki/Fisher–Yates_shuffle

DO NOT search elsewhere on the internet for code that implements `randperm()`. This Wikipedia page is all that you are permitted to look at.

Of the various algorithms for doing permutations described on this Wikipedia page, I want you to implement the first (and oldest) one called the Fisher–Yates Shuffle. As described on the Wikipedia page, the algorithm consists of the following steps:

1. Write down the numbers from 1 through N.
2. Pick a random number k between one and the number of unstruck numbers remaining (inclusive).
3. Counting from the low end, strike out the kth number not yet struck out, and write it down at the end of a separate list.
4. Repeat from step 2 until all the numbers have been struck out.
5. The sequence of numbers written down in step 3 is now a random permutation of the original numbers.

See the Wikipedia page for illustrations of the method. Try the algorithm by hand with a small value of N before trying to implement the algorithm in Python.

I would suggest you try to literally follow the algorithm as written, taking the words and figuring out how to translate them into code. "Write down" would mean generate a sequence of numbers from 1 through N. "Strike out" could mean delete an item from a list or numpy array. "Until all the numbers have been struck out" might suggest something like a while loop.

REQUIRED FOR GRADUATE STUDENTS
(Extra Credit for Undergraduate Students)

Q5 (2 points). Program up a function that performs a constrained randomization of the order of conditions in an experiment (make sure your function includes a doc string). `Nconds` should be a variable that equals the number of conditions to randomize; set `Nconds = 4` in the code you turn in (but your code should work for any value of `Nconds`). `Nreps` should be a variable that equals the number of repetitions of each condition to use in a block of trials; set `Nreps = 20` in the code you turn in (but your code should work for any value of `Nreps`). This is constrained randomization is that your sequence of conditions on each trial must never include a repeat of the same condition and each condition must appear the same number of times. Your function should take `Nconds` and `Nreps` as arguments and return a numpy array with `Nconds * Nreps` elements, with each element equaling the condition number on a trial.

Write a short function that checks that your constrained randomization function works appropriately: in other words, that there are (1) no repeats, and (2) that each condition is presented an equal number of times.

Unexcused late assignments will be penalized 10% for every 24 hours late, starting from the time class ends, for a maximum of two days, after which they will earn a 0.