1 Classes, File IO, Datastructures, Imports Warmup

- Submit one file called `warmup.py` .

1. Define a class `SpecialString` that takes the parameter `string` for its constructor. Define a method `print_string` for `SpecialString` that prints the string in a boundary like this

```
**************************
* Hello this is my string *
**************************
```

2. Write a function `class_practice(value1,value2)` that instantiates an object of `SpecialString` with `value1` , then calls `print_string` . Thereafter change the value inside the same object with `value2` and call `print_string`` again.

3. Write a function `record_notes()` that does the following
   A. First asks the user's name.
   B. Asks the user to enter a single-line note. After the user enters the note, it is stored in `USERNAME_notes.txt` on a newline.
   C. Keeps asking the user to enter a note in a loop, or to type "exit" to exit the program.
   D. Make sure that killing the program suddenly doesn't result in losing all the notes entered till that point.
   E. You should be able to call `record_notes()` multiple times without deleting previous notes.
4. Write a function `read_notes()` that does the following
   A. First asks the user's name.
   B. Asks the user to enter a word.
   C. Print all the notes with that word in the user's notes, as created using `record_notes()` .

5. Read the file `passage.txt` and perform the following operations in a function called `process_passage()`
   A. Print the number of characters in the whole file.
   B. Find all the unique words in the file. Remember words may be divided by spaces or punctuation.
   C. Find the list of (start,end) positions of each unique word in terms of characters from the beginning of the file. (Hint: You can find all occurrences of a substring in a string by repeated calls to the find method, increasing the `beg` parameter after each call.)
   D. Write the information regarding unique word vs (start,end) positions as a JSON file `passage_info.json` .
6. Write a function `view_word_examples(word,surrounding_chars,max_examples)` that does the following
   A. Read the information from `passage_info.json` .
   B. Given the word, check if it is one of the unique words, if not print "No examples found".
   C. If yes, print an example of the word from `passage.txt` with `surrounding_chars` number of characters of the surrounding text on both sides.

   > For example, if word is "geological" and surrounding_chars=10, then the first example should be "terval of geological time from" (10 bytes extra on both sides)

   D. Print at most `max_examples` number of examples.

> Do not reiterate over the file passage.txt looking for the words in this question, reuse the preprocessing from the previous step. In real life as well, many times you will want to do this kind of preprocessing so that you can save the time of the user. Do not use seek() method, as len(fi.read(num_bytes))!=num_bytes always (as some characters take multiple bytes to encode)

7. Write a function `random_permuted_list(list_size,low,high,num_permutations)` that
   A. firstly generates a list of `list_size` random integers between `low` and `high` . (hint: `random.randint`)
   B. permute/shuffle the list `num_permutations` times and print each permutation out (hint: `random.shuffle`)
8. Write a function `generate_unique_numbers(num_unique,low,high)` which uses the `random.randint` function to generate random integers between `low` and `high` (both included).

   - Keep calling `random.randint` until `num_unique` unique numbers are generated. Use a list and the membership operator for this step.
   - `generate_unique_numbers` should return a) the list of unique numbers b) the time taken to generate the list (hint : time.time())
   - Add a boolean parameter `use_set` that specifies whether to use a set or a list to keep track of the unique numbers generated so far.
   - Write in the comments which is faster : set or list? Why?