COMP 202

Winter 2022

Assignment 3

Due: Sunday, March 27th, 11:59 p.m.

Please read the entire PDF before starting. You must do this assignment individually.

Part 1:	20 points
Part 2:	40 points
Part 3:	30 points
Part 4:	10 points
Part 5:	6 bonus points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, allowing the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must follow all directions below:

- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty per question will be applied.
- Make sure that your code **runs without errors**. Code with errors will receive a very low mark.
- Write your name and student ID in a comment at the top of all `.py` files you hand in.
- Name your variables appropriately. The purpose of each variable should be obvious from the name.
- **Comment your code.** A comment every line is not needed, but there should be enough comments to fully understand your program. (See the discussion later in this PDF.)
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing.
- Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.
- **Up to 30% can be removed for bad indentation of your code, omission of comments, and/or poor coding style (as discussed in class).**

Hints & tips

- **Start early.** Programming projects always take more time than you estimate!
- Do not wait until the last minute to submit your code. **Submit early and often**—a good rule of thumb is to submit every time you finish writing and testing a function.

- Write your code **incrementally**. Don't try to write everything at once. That never works well. Start off with something small and make sure that it works, then add to it gradually, making sure that it works every step of the way.
- Read these instructions and make sure you understand them thoroughly before you start. Ask questions if anything is unclear!
- Seek help when you get stuck! Check our discussion board first to see if your question has already been asked and answered. Ask your question on the discussion board if it hasn't been asked already. Talk to your TA during office hours if you are having difficulties with programming. Go to an instructor's office hours if you need extra help with understanding a part of the course content.
  - At the same time, beware not to post anything on the discussion board that might give away any part of your solution—this would constitute plagiarism, and the consequences would be unpleasant for everyone involved. If you cannot think of a way to ask your question without giving away part of your solution, then please drop by our office hours.
- If you come to see us in office hours, please do not ask "Here is my program. What's wrong with it?" We expect you to at least make an effort to start to debug your own code, a skill which you are meant to learn as part of this course. And as you will discover for yourself, reading through someone else's code is a difficult process—we just don't have the time to read through and understand even a fraction of everyone's code in detail.
  - However, if you show us the work that you've done to narrow down the problem to a specific section of the code, why you think it doesn't work, and what you've tried to fix it, it will be much easier to provide you with the specific help you require and we will be happy to do so.

Revisions

None yet!

Warm-up questions (0 points)

*Do **NOT** submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of the actual assignment questions, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a function `same_elements` which takes as input a two dimensional list and returns true if all the elements in each sublist are the same, false otherwise. For example,

```
>>> same_elements([[1, 1, 1], ['a', 'a'], [6]])
True
>>> same_elements([[1, 6, 1], [6, 6]])
False
```

Warm-up Question 2 (0 points)

Write a function `flatten_list` which takes as input a two dimensional list and returns a one dimensional list containing all the elements of the sublists. For example,

```
>>> flatten_list([[1, 2], [3], ['a', 'b', 'c']])
[1, 2, 3, 'a', 'b', 'c']
>>> flatten_list([[]])
[]
```

Warm-up Question 3 (0 points)

Write a function `get_most_valuable_key` which takes as input a dictionary mapping strings to integers. The function returns the key which is mapped to the largest value. For example,

```
>>> get_most_valuable_key({'a' : 3, 'b' : 6, 'g' : 0, 'q' : 9})
'q'
```

Warm-up Question 4 (0 points)

Write a function `add_dicts` which takes as input two dictionaries mapping strings to integers. The function returns a dictionary which is a result of merging the two input dictionary, that is if a key is in both dictionaries then add the two values.

```
>>> d1 = {'a':5, 'b':2, 'd':-1}
>>> d2 = {'a':7, 'b':1, 'c':5}
>>> add_dicts(d1, d2) == {'a': 12, 'b': 3, 'c': 5, 'd': -1}
True
```

Warm-up Question 5 (0 points)

Create a function `reverse_dict` which takes as input a dictionary `d` and returns a dictionary where the values in `d` are now keys mapping to a list containing all the keys in `d` which mapped to them. For example,

```
>>> a = reverse_dict({'a': 3, 'b': 2, 'c': 3, 'd': 5, 'e': 2, 'f': 3})
>>> a == {3 : ['a', 'c', 'f'], 2 : ['b', 'e'], 5 : ['d']}
True
```

Note that the order of the elements in the list might not be the same, and that's ok!

Assignment 3

The questions in this part of the assignment will be graded.

The main learning objectives for this assignment are:

- Apply what you have learned about lists.
- Apply what you have learned about dictionaries.
- Understand how to test functions that return dictionaries.
- Solidify your understanding of working with loops and strings.
- Understand how to write a docstring and use doctest when working with dictionaries.
- Learn to identify when using the built-in function `enumerate` can help you write a cleaner code.
- Apply what you have learned about file IO and string manipulation.

Note that this assignment is designed for you to be practicing what you have learned in the videos up to and including Lecture 33 (Function objects). For this reason, you are NOT allowed to use anything seen after Lecture 33 or not seen in class at all. You will be heavily penalized if you do so.

For full marks, in addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least three (3) examples of calls to the function. You are allowed to use *at most one* example per function from this PDF.

Examples

For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to test that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that will use inputs not seen in the examples. You should make sure that your functions work for all the different possible scenarios. Also, when testing your code, know that mindlessly plugging in various different inputs is not enough—it's not the quantity of tests that matters, it's having tests that cover all of the possible scenarios, and that requires thinking about possible scenarios.

Furthermore, please note that your code files **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that contains function calls in the main body **will automatically fail the tests on codePost and thus be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. You can also test your functions by calling them from the shell. Please review what you have learned in Lecture 14 (Modules) if you'd like to add code to your modules which executes only when you run your files.

Safe Assumptions

For all questions in this assignment, you can safely assume that the **type** of the inputs (both to the functions and those provided to the program by the user) will always be correct. For example, if a function takes as input a string, you can assume that a string will always be provided. At times you will be required to do some input validation, but this requirement will always be clearly stated. Otherwise, your functions should work with any possible input that respect the function's description. For example, if the description says that the function takes as input a positive integer, then it should work with all integers greater than 0. If it mentions an integer, then it should work for *any* integer. Make sure to test your functions for edge cases!

Code Repetition

One of the main principles of software development is DRY: Don't Repeat Yourself. One of the main ways we can avoid repeating ourselves in code is by writing functions, then calling the functions when necessary, instead of repeating the code contained within them. Please pay careful attention in the questions of this assignment to not repeat yourself, and instead call previously-defined functions whenever appropriate. As always, you can also add your own helper functions if need be, with the intention of reducing code repetition as much as possible.

Comments

For this assignment and future work for this class, we ask that you adhere closely to the following guidelines for comments and docstrings.

Docstrings: A docstring must contain the type contract, description, and three (or more) examples. Each example must show off a different case handled by the function (by varying the inputs). The **description** should indicate in plain English what the function does (i.e., the logic of the function). It should discuss how the different parameters are used in the function, and include the **exact parameter names** when doing so. It should not go into specifics (e.g., do not write that a for loop is used, or that other functions are called, etc.).

Comments: Comments in a function should be included **sparingly**. They should not be used to describe basic features of the Python language nor when the line itself can read as an English statement. For instance, when using a for loop to go over the lines of a file (`for line in f:`), a comment is not needed since reading the line out loud already indicates exactly what is going on (we are going through each line in the file).

A good rule of thumb is to assume that the reader of the comments is experienced with the Python language. So there is no need to write a comment about creating a variable, using a loop or opening a file. Instead, comments should be used when something logically complicated is going on. For instance, if there is a complicated conditional expression that involves some real-world idea (like the conditions in the `get_cheapest_trip_cost` function of Assignment 1), then it may be helpful to have a comment to explain the logic of what is going on (but even in this case, it may be better to include this logic in the docstring instead of having a comment).

We can also avoid writing comments by using descriptive and informative names for our variables; instead of `x`, we can write `rectangle_area`, and then there is no need to have a comment explaining the purpose of that variable.

Failure to adhere to these new guidelines will incur style penalties.

With the digital revolution has come an array of wondrous inventions that have changed forever our daily lives. But with any new power there is also a dark side. The past decade has seen the rise in prominence of digital disinformation to the highest heights and the lowest lows.

Disinformation is a kind of propaganda. It is a false statement spread deliberately to influence public opinion on an issue. Disinformation has been used as a tactic for centuries, but lately has increased in application through the Internet and social media platforms such as Facebook and Twitter. The Internet by design is (mostly) anonymous, making it a prolific den for falsehoods.

Though you may have heard the term in the popular news, what you may not know is how easy it is to apply the tactic on a large scale, even being only a single individual. In this assignment, we will see just how easy it is!

Part 1: The Reddit API (20 points)

Reddit is a popular social media service used by many students and the occasional instructor. Content such as links, text posts, images and videos can be submitted to the site under different user-created subsections of the site (known as 'subreddits'). Each piece of posted content is known as a 'topic' and has its own webpage on the site. Each topic can be voted on and discussed by the community in a series of replies on its particular page.

Reddit also provides an API, Application Programming Interface, to users. An API allows a computer program to interact with an application (or website) instead of a user interacting with it directly.

An API, especially an API for a website, is often designed to be language-agnostic. That is, it can be used by programs written in many different languages. Helper code (as part of a library) is then written in a particular language to serve as a bridge between the API and code written in that language. Reddit has an API which can be used in Python through the use of a special library called `PRAW`.

To begin this assignment, you must install the `praw` library into Thonny, using the same steps as done for the libraries needed for Assignment 2.

`PRAW` lets us access the Reddit API and do such things as list the topics in a subreddit, list the replies for a topic, and even make new replies to posts, all from within our Python code. To use `PRAW`, you must first create a new Reddit account at `https://www.reddit.com`. **Do not use an existing account.**

Your choice of Reddit username is entirely up to you and does not have to include your real name or any personal information, but as part of your username **you must include the word 'bot'.**

After creating an account, we must obtain some extra information in order to use the API. Visit `https://www.reddit.com/prefs/apps` and click the 'are you a developer? create an app...' button in the middle of the page. Select the 'script' radio button, and enter a name (could be anything). For the redirect URI, use 'http://localhost:8080'. Click the button to create the app, and the page will refresh. Make a note of the text next to 'secret' (we will call this the client secret), and also the text under 'personal use script' (we will call this the client ID).

After obtaining the information, create a file called `praw.ini` in a new folder in which you will store your files for this assignment. Include the following in this file. After each equals sign, include the relevant piece of information, including the username and password for your new Reddit account and the client ID and secret. The User Agent should be of the form "<your OS>:<client ID>:1.0 (by /u/<your reddit username>)". The information stored in this file will let `PRAW` connect to the Reddit API using your new account credentials.

```
[bot]
client_id=
client_secret=
username=
password=
user_agent=
```

Note: Do **not** submit `praw.ini` to codePost. You will be not be marked for this file, but it will be required for later parts of the assignment.

We can now begin to use the functions and methods available in the `PRAW` library to directly communicate with Reddit from within our Python code.

We will start by creating a file `reddit.py` and importing the `praw` module.

First, we need to call the `Reddit` function from the `praw` module. It will authenticate us on the website using our credentials and return to us an object of type `Reddit` on which we can then call methods. (It is similar to how we called the `Turtle` function from the `turtle` module to obtain a `Turtle` object, and then called methods on that object, to draw things for Assignment 1).

To create the `Reddit` object, you can copy the following line of code into your program. The function should be called only **once** per program execution, and should not be called when we import your file to run our tests; so, place it at the bottom of the file in a `if __name__ == 'main'` block.

```
reddit = praw.Reddit('bot', config_interpolation="basic")
```

`PRAW` objects and methods

Once we have an object of type `Reddit`, we can call the following methods on it:

- `submission(url='https://www.reddit.com/r/.../...')`: takes a URL (string) that corresponds to a Reddit topic, and returns an object of type `Submission` (see below).
- `subreddit`: takes the name of a subreddit (string), and returns an object of type `Subreddit` (see below).

We can call the following methods on an object of type `Subreddit`:

- `top`: Returns a list-like object containing the top-voted submissions in the subreddit (as `Submission` objects).

We can call the following methods on an object of type `Submission`:

- `reply`: takes a string as input, and posts a reply to the submission on Reddit, using the credentials specified in `praw.ini`.

We can also access certain **attributes** of a Submission object. An attribute is a piece of data stored inside an object under a fixed name and is specific to that particular object. For instance, the `Submission` object has an attribute called `created_utc`, which stores the time at which the submission was first posted to the Reddit site (in Unix time). Each `Submission` object will have its own value for the `created_utc` object, because each submission was posted at a different time.

We will learn a lot more about attributes towards the end of our class. For now, all that you need to know is that you may access the value of an attribute by using the dot notation on an object, much like calling a method, except without putting parentheses afterwards. E.g., we may write the following to print out the creation time of a particular submission object called `sub1`:

```
print(sub1.created_utc)
```

An important attribute of a `Submission` object is `comments`, which is an object of type `CommentForest`, a list-like object which you can iterate over using a for loop to obtain the comments (replies) on the submission. The comments are given as `Comment` objects.

A big part of programming in any language, including Python, is being able to work with libraries (such as `PRAW`) by reading their online documentation. We have listed some of the types of objects that you will need for this assignment above, but there are others (e.g., `Comment` and `Redditor`) that we intentionally omit from this PDF. To understand the methods and attributes those kinds of objects have, and how you can use them, we ask you to consult the `PRAW` documentation yourself. You can access the documentation at `https://praw.readthedocs.io/en/stable/` and `https://praw.readthedocs.io/en/stable/code_overview/praw_models.html`. Note that it can be challenging at first to understand the documentation, but many of the methods have example code which can be helpful to look at. (Note that there are many methods and attributes available, but you will end up needing to use very, very few of them to complete this assignment.)

Part 1 of this assignment involves writing some basic functions to gain familiarity with `PRAW`. Include the following functions in a file `reddit.py`:

- **`get_topic_comments`**: takes a `Submission` object as input, and returns a list of `Comment` objects contained within the submission. Comments (replies) at all levels should be included in the returned list, not just top-level comments. That is, replies to comments, and replies to replies, and so on, should all be included. Note that each `Comment` object, as well as the `Submission` object, has a `replies` attribute containing a `CommentForest` of the replies to that comment/submission.

```
>>> url = 'https://www.reddit.com/r/mcgill/comments/eay2ne/mcgill_subreddit_bingo_finals_edition/'
>>> submission = reddit.submission(url=url)
>>> get_topic_comments(submission)
[Comment(id='fb0vh26'), Comment(id='fb0l4dk'), Comment(id='fb15bvy'),
 Comment(id='fb1pwq8'), Comment(id='fb26drr'), Comment(id='fj2wd6x'),
 Comment(id='fb1spzv'), Comment(id='fb1td2g'), Comment(id='fb1trul')]
```

- **`filter_comments_from_authors`**: takes a list of `Comment` objects and a list of author names (strings) as input, and returns a list containing the `Comment` objects that were written by any of the given authors.

```
>>> url = 'https://www.reddit.com/r/mcgill/comments/paf85s/the_only_society_we_deserve/'
>>> submission = reddit.submission(url=url)
>>> comments = get_topic_comments(submission)
>>> filter_comments_from_authors(comments, ['Juan_Carl0s', 'Chicken_Nugget31'])
[Comment(id='ha4piat'), Comment(id='ha4j1r7')]
```

- **`filter_out_comments_replied_to_by_authors`**: takes a list of `Comment` objects and a list of author names (strings) as input, and returns the same list of `Comment` objects except for comments which have been replied to by any of the authors in the given list of authors, as well as comments written by those authors themselves.

```
>>> url = 'https://www.reddit.com/r/mcgill/comments/pa6ntd/does_mcgill_have_a_taylor_swift_society/'
>>> submission = reddit.submission(url=url)
>>> comments = get_topic_comments(submission)
>>> filter_out_comments_replied_to_by_authors(comments, ['basicbitch122'])
[Comment(id='ha33z5m'), Comment(id='ha2sq62'), Comment(id='ha3d39f'),
 Comment(id='ha2s4lw'), Comment(id='ha3mrwm'), Comment(id='ha3m2kv'),
 Comment(id='ha5okfd'), Comment(id='ha7e0ei'), Comment(id='hbpxpi1'),
 Comment(id='ha4e526'), Comment(id='ha3837c'), Comment(id='hdo2kmm'),
 Comment(id='ha3f5q2'), Comment(id='hdof500'), Comment(id='hdol6rn'),
 Comment(id='hcrklp6')]
```

When testing your functions using `doctest`, make sure to place the line `doctest.testmod()` directly after the line that creates your `Reddit` object (at the bottom of your file), so that all your docstring examples will be able to access it.

Part 2: Mad Mad Libs (40 points)

We now know how to programmatically retrieve and reply to messages on Reddit. That is the first step in a disinformation campaign. The second step is to produce the messages to be posted.

Typically, a state actor will hire individuals to write normal-sounding messages as part of the campaign. However, in certain cases, the false content is produced itself by an algorithm.

For instance, when the Federal Communications Commission (FCC) of the United States opened up a public online discussion forum to discuss net neutrality rules in 2017, 18 of the 22 million comments submitted were later determined to be inauthentic. You can read the conclusions of the report into this influence campaign here. Of note, the report found that 9 of the 18 million were generated by a group of internet service providers (ISPs), while a further 7.7 million were generated by a single 19-year-old college student pursuing a degree in computer science. Over 9.3 million comments were submitted using automated software (p.27), and one of the recommendations of the report was to introduce safeguards to the FCC website to prevent people from using automated software to submit comments (p.35).

An earlier analysis of the comments found that over 1.3 million comments contained very similar textual patterns. Observe the following three comments that were submitted to the FCC at different times and by supposedly different people and see if you can spot the similarities:

- Americans, as opposed to Washington bureaucrats, deserve to enjoy the services they desire.
- Citizens, rather than the FCC, deserve to use whichever services we prefer.
- People like me, as opposed to so-called experts, should be free to buy whatever products they choose.

Notice anything similar in these comments? All three follow an identical sentence structure and express the same idea. The only difference is in the word choice. In fact, these at a glance different pieces of text could have all been generated through the use of a Mad Libs algorithm. You can read the full analysis here: `https://hackernoon.com/more-than-a-million-pro-repeal-net-neutrality-comments-were-likely-faked-e9f0e3ed36a6`.

Mad Libs is a word game where a player comes up with a story (a piece of text) that contains some blank words, and asks others for suggestions to fill in each word. Each blank space is annotated by a part of speech (adjective, noun, verb, etc.). The word that fills in a blank space must correspond to the annotated part of speech. For instance, a (short) Mad Lib story may read as follows:

"[EXCLAMATION]!" she uttered [ADVERB], while jumping into her [COLOR] [VEHICLE].

There are four different words in this story that must each be filled in with a word of the appropriate part of speech. A filled-in story might be:

"Smell ya later!" she uttered comically, while jumping into her glowing green spaceship.

Although Mad Libs originated as a party game, as shown above it can have uses far beyond what its creators intended. Part 2 of this assignment asks you to write the following functions in a file `madlibs.py` to implement a Mad Libs-based text generation approach.

- **`capitalize_sentences`**: takes a string as input containing a number of sentences, and returns the string with the first letter of each sentence capitalized. You may assume that each sentence ends with a period, exclamation point or question mark, each followed by a space.

```
>>> capitalize_sentences("hello. hello! hello???? HI!")
'Hello. Hello! Hello???? HI!'

>>> capitalize_sentences("...hey")
'...hey'
```

- **`capitalize_sentence_grid`**: takes a nested list as input, where each inner list contains a series of strings. The nested list may not be a matrix (each inner list may have a different number of strings).

Each string in the inner lists will consist of one word with possible punctuation at the end. (There will be no whitespace.)

The return value for the function should be the same nested list, but with the first letter of the first word of each new sentence capitalized. You may assume that each sentence ends with a period, exclamation point or question mark. However, a sentence may start and stop in different inner lists. That is, a sentence may begin in one of the inner lists, and continue to the middle of the next inner list (or even further).

Your function should **not** modify the input list.

```
>>> grid = [["you", "might", "think"], ["these", "are", "separate", "sentences"], \
            ["but", "they", "are", "not!", "ok,", "this"], ["one", "is."]]
>>> capitalize_sentence_grid(grid)
[['You', 'might', 'think'], ['these', 'are', 'separate', 'sentences'], ['but', 'they', \
'are', 'not!', 'Ok,', 'this'], ['one', 'is.']]
```

- **`fill_in_madlib`**: takes a madlib string and a dictionary of word replacements, and returns the filled in madlib string. A madlib string is a piece of text that contains one or more annotated blank spaces. An annotated blank space is a set of square brackets containing a part of speech in all capitals, e.g., [ADJECTIVE]. The word may also optionally be followed by a underscore and integer, e.g., [ADJECTIVE_1].

  A dictionary of word replacements is one where the keys are parts of speech in all capitals, e.g., ADJECTIVE, VERB, etc., and the value is a list of (lowercase) words (or phrases) of the given part of speech. E.g., the value for the key ADJECTIVE may be a list containing the words good, bad, bright, dark, red, bright green, etc.

  The function should fill in all of the blank spaces with words of the appropriate part of speech taken from the dictionary. Blank spaces for the same part of speech should all be replaced by the same randomly-chosen word. For instance, given the madlib string `'I feel [ADJECTIVE]. Do you feel [ADJECTIVE]?'` and given the replacement dictionary `{'ADJECTIVE': ['bright green', 'good', 'happy']}`, the returned string should replace all occurrences of [ADJECTIVE] by one of the three words for that language part in the dictionary (choosing randomly amongst them), not a different word for each blank space. It may be helpful here to use the `choice` function from the `random` module: it takes a list as input, and returns a random element from the list. (The `choice` function may also be useful at later points in this assignment.)

  As stated, it is also possible for a part of speech in the madlib string to be followed by an underscore and an integer, e.g., [ADJECTIVE_1], [ADJECTIVE_2], etc. In this case, the words should still be randomly chosen from the ADJECTIVE list, but a different one should be chosen each time per sentence; that is, each time a word is chosen to be used, it should not be re-used in the same string. But, if there are two or more of the same part of speech and integer, e.g., two or more [ADJECTIVE_1], then all those should be replaced by the same word.

  Finally, the first letter of the first word of each sentence in the returned string should be capitalized.

  Your function should **not** modify the input dictionary. Also, for this function, you may **not** assume anything about the inputs. That is, the types and/or values of the inputs may be invalid. In such case, raise an `AssertionError` with an appropriate error message tailored for each case. Your function should never raise any exception other than `AssertionError` no matter what inputs are given. You will have to think about the possible invalid inputs could be invalid and protect your function against them.

  Show an example in your docstring for each different `AssertionError` that could be raised. These examples are in addition to the three examples that your function must normally have.

```
>>> random.seed(9004)
>>> d = {'COLOR': ['yellow', 'glowing green', 'red'], 'VEHICLE': ['hoverboard', \
         'sportscar', 'electric bike', 'starship']}
>>> fill_in_madlib("Wow! Is that a [COLOR] [VEHICLE]?", d)
'Wow! Is that a glowing green starship?'

>>> random.seed(2022)
>>> d = {'PAST-TENSE-VERB': ['pondered', 'scribbled', 'snoozled', 'studied'], \
         'ADJECTIVE': ['dreamy', 'weak', 'weary', 'starry', 'lazy']}
>>> fill_in_madlib("Once upon a midnight [ADJECTIVE_1], while I [PAST-TENSE-VERB], \
                    [ADJECTIVE_2] and [ADJECTIVE_3],", d)
'Once upon a midnight weary, while I snoozled, lazy and dreamy,'
```

- `load_and_process_madlib`: takes as input a filename corresponding to a file that contains a madlib string and returns nothing. Reads in the madlib string, and reads in a dictionary of word replacements from a Pickled file `word_dict.pkl`, then fills in the madlib string, and saves it back to a new file with `'_filled'` appended to the end of the original filename (but before the `.txt` extension). Do not forget to close the file.

  The following example assumes that `madlib1.txt` contains the madlib string `Once upon a midnight [ADJECTIVE_1], while I [PAST-TENSE-VERB], [ADJECTIVE_2] and [ADJECTIVE_3],`, and that `word_dict.pkl` contains the Pickled dictionary `{'PAST-TENSE-VERB': ['pondered', 'scribbled', 'snoozled', 'studied'], 'ADJECTIVE': ['dreamy', 'weak', 'weary', 'starry', 'lazy']}`.

```
>>> random.seed(2022)
>>> load_and_process_madlib('madlib1.txt')
>>> f = open('madlib1_filled.txt', 'r')
>>> s = f.read()
>>> s
'Once upon a midnight weary, while I snoozled, lazy and dreamy,'
```

Part 3: Putting it all together (30 points)

We all know about the SSMU – the Students' Society of McGill University. You may have also heard about CSUS, the Computer Science Undergraduate Society. But have you heard about the CSSSMU? The Computer Science Students' Society of McGill University? It aims to be an alternative student governing body for students taking computer sciences courses (such as you!).

The CSSSMU is currently holding elections for their 2022-2023 Executive Council. The current President, Boole Ian, is facing against a fearsome list of contenders, including Alex Thonny, Elsa Ifstatement and Dee Buh-Ger.

The CSSSMU has recently opened a subreddit at `https://reddit.com/r/csssmu`. We will test our Mad Libs code here and create chaos.

The Reddit website encourages users to make use of its API to create 'bots', or automated software programs. In this part of the assignment, we will write code to create a bot that not only posts on the aforementioned subreddit, but also talks with the bots of other students in our class, to create a totally inauthentic conversation surrounding the CSSSMU elections.

Before beginning, you must familiarize yourself with the Reddit API Rules at `https://github.com/reddit-archive/reddit/wiki/API` and the Reddit Botiquette at `https://www.reddit.com/wiki/bottiquette`. Here are some very important points, some of which go further than those mentioned in the linked pages:

- Your bot may make up to 60 requests per minute. A request means the use of the API to fetch information about a subreddit or submission or post a reply.
- Your bot's User Agent (specified in `praw.ini`) must be of the format given earlier in this PDF.

- Your bot must not make a series of infinite posts to the website.
- Your bot may only operate in subreddits which **explicitly** allow bots. For this assignment, your bot may post in the `csssmu` subreddit **only**.
- Your bot must not vote. Votes must be cast by humans.
- Your bot must comply with all the regular rules of the website that humans must adhere to.

Failure to observe these rules may lead to your user account being banned by Reddit.

Note that many 'benevolent' bots roam Reddit (and other social media platforms) regularly. Check here for some examples of such bots:

- Good Reddit bots
- 7 of the best Twitter bots in journalism
- Student's Twitter tracking Russian oligarchs' yachts and jets

We are going to make a bot that posts about the CSSSMU elections using text generated using our Mad Libs algorithm. First, create ten madlib strings related to the elections, and save each in a file `madlibk.txt`, where `k` is a number between 1 and 10.

For instance, here is a sample story (do not use this one):

I [SUPPORT-ADJECTIVE] [GOOD-CANDIDATE]! All other candidates can [FAILURE-EUPHEMISM]!

Some of your madlib strings can be just one sentence, but at least half of them should have two or three sentences which each form a consistent paragraph.

Then, create a word replacement dictionary that contains all of your parts of speech as keys, and a list for each of them containing at least five words each, and pickle it to the file `word_dict.pkl`. We will use this file for Part 4. (You will submit these files to codePost.)

Next, implement this function at the end of your `madlibs.py` file:

- **`generate_comment`**: takes no inputs. Chooses a random number `k` between 1 and 10, and reads in the madlib string contained in the file at `madlibk.txt`, fills in the madlib string using the word dictionary at `word_dict.pkl`, and then returns the string that was saved to the file `madlibk_filled.txt`.

```
>>> random.seed(9001)
>>> d = {'PAST-TENSE-VERB': ['pondered', 'scribbled', 'snoozled', 'studied'], \
         'ADJECTIVE': ['dreamy', 'weak', 'weary', 'starry', 'lazy']}
>>> generate_comment()
'Once upon a midnight dreamy, while I snoozled, lazy and starry,'
```

Finally, implement these functions at the end of your `reddit.py` file:

- **`get_authors_from_topic`**: takes a `Submission` object as input and returns a dictionary where the keys are authors of the comments in the submission, and the value of a key is the number of comments that author has made in the submission (as an integer). All the comments of a submission should be counted, including replies to replies and so on, except for authors whose names have been deleted.

```
>>> url = 'https://www.reddit.com/r/mcgill/comments/pa6ntd/does_mcgill_have_a_taylor_swift_society/'
>>> submission = reddit.submission(url=url)
>>> num_comments_per_author = get_authors_from_topic(submission)
>>> len(num_comments_per_author)
23
>>> num_comments_per_author['basicbitch122']
12
```

- `select_random_submission_url`: takes a `Reddit` object, a topic URL (string), a subreddit name (string) and a replace limit (integer). Roll a six-sided die. If the result is 1 or 2, returns the `Submission` object for the given topic URL, after first loading the given number of extra comments (see below). Otherwise, returns a `Submission` object corresponding to a random submission from the top submissions of the given subreddit name.

  Do **not** write examples in your docstring for this function. Instead, test your function in the shell.

  Note that when a Reddit submission has many comments, or when there is a long chain of replies to a comment, Reddit by default hides the remainder of the comments, or the remainder of the chain of replies. If the user wants to see them, they must manually click a "load more comments" or "continue this thread" link to load more. Depending on the number of comments, the link may have to be clicked many times in order for the entire page of comments to be loaded. `PRAW` handles this issue through a method called `replace_more` which can be called on a `CommentForest` object. The method takes one argument, an integer, corresponding to how many times to click the "load more comments" or "continue this thread" link. Loading more comments takes time, so when debugging, you may like to keep the integer small (1 or 2), but when you are more certain of your code, you may like to increase it (or pass the value `None` to load all comments on the page).

  You can read more about `replace_more` here: `https://praw.readthedocs.io/en/stable/tutorials/comments.html`.

- `post_reply`: takes as input a `Submission` object and a string corresponding to your Reddit username. If your bot has not made any replies in the given submission, then create a new top-level comment with text given by your `generate_comment` function. Otherwise, choose a random comment from all comments in the submission that you have not already replied to, and reply to it with text given by your `generate_comment` function.

  Do **not** write examples in your docstring for this function. Instead, test your function in the shell.

- `bot_daemon`: takes the following as input: a `Reddit` object, a starting topic URL (string), a replace limit (integer) (for the `replace_more` method), a subreddit name (string) and the name of your new Reddit username (string). In an infinite loop, calls the following three functions: `select_random_submission_url` to get a `Submission` object, then `post_reply` to reply to that submission, and finally `time.sleep(60)`, to make your program sleep for 60 seconds before repeating the loop (make sure to import the `time` module).

  Do **not** write examples in your docstring for this function. Instead, test your function in the shell.

Part 4: Chaos (10 points)

You will receive ten points on this assignment for contributing at minimum 100 automatically generated comments to the `csssmu` subreddit under your Reddit bot username. The comments must be divided up amongst at least four different topics (submissions), and they must be submitted at minimum 1 minute apart from each other to count.

Attach a file to your codePost submission called `username.txt` which contains your username to be graded on this part.

Part 5: Extra Credit (6 bonus points)

1 point: Victory!

As you may know, the CSSSMU election voting period takes place from March 21 to 25. Through the comments posted by your Reddit bot, you may influence the minds of voters. You may decide, for example, to create your madlib strings such that they all support one candidate over the others.

You will have the option during this assignment to side with one candidate in particular. You can declare your allegiance by attaching a file to your codePost submission called `my_choice.txt` containing the full name (with no typos) of the candidate you are supporting, before the end of the voting period (at 3 p.m. on Friday, March 25). If the candidate you have supported wins the election, you will get 1 bonus point on this assignment.

Note that you may switch your allegiance to another candidate at any time during the campaign and voting period by simply re-uploading your submission with a new `my_choice.txt` file, but such files submitted after the end of the voting period (3 p.m. on Friday, March 25) will be ignored.

Be warned: if any particular candidate gets *too* much support, Elections CSSSMU may begin to investigate them for possible campaign violations, and possibly disqualify them during *or* after the voting period!

5 points: More advanced text generation!

There are many more sophisticated approaches to text generation. If you can write your own code to generate text in a more sophisticated fashion, post at least 50 comments to the `csssmu` subreddit using this approach, **and** can demonstrate your knowledge of your code and approach to a satisfactory level to the TA during your grading meeting, then you can be awarded up to 5 bonus points. Make sure to submit any additional code files for this task to codePost so that the TA may examine and run them.

You may use third-party libraries to help you with this task, as long as you write your own code that calls on them. Here are a couple of ideas for you:

- Markov chain generator
- GPT-2 text generation AI model

***** Important Note *****

As a final note to cap this assignment, we want to state the importance of election integrity. Voters in an election should be able to cast their vote based on the proper issues and correct information. Disinformation in an election manipulates the mind of a voter and undercuts the principles of a democracy. We have prepared this assignment to show you just how easy it is for disinformation to be spread online, even just by one individual sitting alone at their desk. Imagine how much more powerful and insidious such tactics could be if employed by one with much more resources, such as a state actor. Be on your guard when consulting anything online – there is always someone with an incentive to influence you in some way.

Finally, although the CSSSMU is a fiction, we do have two student bodies that do represent your interests as students at McGill: the SSMU (Students' Society of McGill University) and, in our department, CSUS (Computer Science Undergraduate Society). Both of these societies are holding or will soon hold elections, so please consider making yourself aware of the candidates' platforms and casting an informed ballot.

Special Thanks

With thanks to Mike Izbicki, Assistant Professor at Claremont McKenna College, who dreamed up the original Reddit Bot assignment which was selected as a Nifty Assignment at SIGCSE'22.

What To Submit

You must submit all your files on codePost (https://codepost.io/). The files you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`reddit.py`
`madlibs.py`
`word_dict.pkl`
`madlib1.txt`
`madlib2.txt`
`madlib3.txt`
`madlib4.txt`
`madlib5.txt`
`madlib6.txt`
`madlib7.txt`
`madlib8.txt`
`madlib9.txt`
`madlib10.txt`
`username.txt`
`my_choice.txt` (optional)
`README.txt` In this file, you can tell the TA about any issues you ran into while doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.

You may make as many submissions as you like, but we will only grade your final submission (all prior ones are automatically deleted).

Note: If you are having trouble, make sure the names of your files are exactly as written above.

Assignment debriefing

In the week following the due date for this assignment, you will be asked to meet with a TA for a 10 minute meeting. In this meeting, the TA will grade your submission and discuss with you what you should improve for future assignments.

Only your code will determine your grade. You will not be able to provide any clarifications or extra information in order to improve your grade. However, you will have the opportunity to ask for clarifications regarding your grade.

You may also be asked during the meeting to explain portions of your code. Answers to these questions will again not be used to determine your grade, but inability to explain your code may be used as evidence to support a charge of plagiarism later in the term.

Details on how to schedule a meeting with the TA will be shared with you close to the due date of the assignment.

If you do not attend the meeting, you will not receive a grade for your assignment.