Teamwork T07: Mad Libs

- Do this teamwork assignment with a partner.

Learning Objectives

- Additional practice breaking a larger problem down into smaller "pieces" using functions.
- Gain practice using strings.

How to Get Started

- To begin, make a copy of this document by going to File >> Make a Copy...
- Share the copied document with all members of your team. You can share this document by hitting the blue button in the top right of the document, then entering the email address of all members in the bottom input field.
- Change the file name of this document to **username1, username2 - T07: Mad Libs** (for example, **shepherdp - T07: Mad Libs**). To do this, click the label in the top left corner of your browser.
- Navigate to the **GitHub Classroom for Teamwork T07: Mad Libs.**
- You will be asked to create a team:
    - One of you will create the team
    - The other(s) will join the team when prompted.
- Paste the link to your team's GitHub repository here:

GitHub Repo Link:	

First, discuss with your team and assign yourselves roles. Pick the role you've done the least.

Driver[1]:	
Navigator[2]:	
Quality Control[3] (if the class is odd numbered):	

[1]The driver will be doing the majority of the typing in PyCharm. Your job is to solve the problem given to you by the Navigator.

[2] The navigator gives directions to the driver and helps the driver catch syntax and logic errors as he or she creates the code. The navigator should keep track of time and make sure progress is being made.

[3] The quality control specialist ensures rules are followed, in the code (adding comments, fixing typos, etc.) and in the document (questions are answered, fixing typos, etc.) In a group of two, everyone is responsible for quality control.

Palindromes

A **palindrome** is a word, line, verse, number, or a sentence that reads the same backward as forward (ignoring punctuation and spacing). Examples include "*Madam, I'm Adam*" and "*Poor Dan is in a droop*". Some people love making palindromes, as the Palindrome List demonstrates.

The **t09_palindrome.py** code in your repo is an example of a program in Python that uses several of the features of the string class to check for palindromes. While we won't be working with palindromes in this assignment, it might be a useful starting point for this assignment.

Mad Libs

A Mad Lib is a game in which words are substituted for blanks in a story. Each blank is specified by a generic category like "noun", "verb", "adjective", "place", etc. The words are chosen by the category and substituted into blanks in the story. This process often produces funny results. You can try some Mad Libs on the Mad Takes website. If you have a Google Home smart speaker, then you can even play Mad Libs on it.

Your primary challenge in this assignment is a **design** problem. Before coding, you must figure out how to design a program which will generate a Mad Lib story when run by the user. As you hopefully learned from previous assignments, without a solid plan, coding can become unruly and messy.

Your task is to create a program to accomplish the following sub-tasks:

1. Create a story using a triple-quoted string, which will allow it to span multiple lines.
2. This story string will contain the unchanging parts of the story (the **template**) as well as the blanks that will be replaced by words from categories input by the user. Use the {#} format as the blanks (e.g., `'''The cat in the {0} knows a lot about {1}'''`)
3. Your story can say whatever you want it to say, but it must have at least five of these fields which will be delineated by place holders ({0}, {1}, ... {n}). At least one of these fields must appear in the story more than once, which works just as you might hope using the Python string format method.
4. For example, your story string might look like something like:

```
'''Be kind to your {0}-footed {1},
For a {2} may be somebody's mother.
Be kind to your {1} in the {3},
Where the weather is always {4}.'''
```

5. When the program runs, it should ask the user to input various words for the categories that will replace the blanks in the template. The program can prompt for that information with questions or commands like:

```
Enter your choice of noun:
```

```
Enter your choice of noun:
Enter your choice of noun:
Enter your choice of place:
Enter your choice of adjective:
```

6. After the user has entered all of the words, the program should then display the completed story on the screen. For example, suppose the user entered:

```
Enter your choice of noun: dog
Enter your choice of noun: table
Enter your choice of noun: lamp
Enter your choice of place: phone booth
Enter your choice of adjective: tall
```

7. The completed story becomes:

```
Be kind to your dog-footed table,
For a lamp may be somebody's mother.
Be kind to your table in the phone booth,
Where the weather is always tall.
```

8. **Hint:** The various words that are input from the user for each category might best be stored in a **list** of **strings**, so you can have access to them. For example, once the data is input by the user, the list in the example above will look like:

```
["dog", "table", "lamp", "phone booth", "tall"]
```

9. One might find the `append()` method of lists useful.
10. You may design your code however makes sense to you, but realize that there are much easier ways and much harder ways to design this program, so please design before you implement! A healthy discussion between partners should prove fruitful.

Space for Annotating Your Design:

Usual Disclaimer About Good Coding Practices:

- At the start of each coding session, remember to create a new Git branch before you start. Remember, branches are cheap!
- The file **t07_stubs.py** may be useful as starter code for when you start developing your solution.
- Include comments for any parts of the code that are non-intuitive to let the next coder or the grader know what that portion does.

- Make sure that each function has parameters that make sense. If a function does not need a parameter, do not include one.
- Use meaningful variable names.
- Include a descriptive header as a comment at the top of your source code.
- Include a `main()` function.
- The highest level of your program (i.e., no indenting) should **only** contain the following:
    - the header
    - any `import` statements
    - function definitions
    - a call to the `main()` function
- Use functions with useful docstrings. Docstrings must include a description of the main purpose of the function, and descriptions of all input parameters, as well as what is returned by the function.

Submission Instructions

1. Edit the **README.md** file in your repository. Replace the lines with the correct information for your name(s), the link the repository, and the link to this document, which you can get via the blue Share button 

in the top right of this window.
2. Check the **Share** settings for this document (top right). Set them to **"Anyone with the link can view"**. That is how we will be able to access this document to grade you:

3. **Merge** the master branch into your branch (if any changes were made to master).
4. **Add, Commit,** and **Push** any additional changes to your repository.
    a. Right click any new files (they'll be green) in the Project pane of PyCharm (far left), and click **Git** >> **Add.**
    b. Right click the project directory in the Project pane of PyCharm, and click **Commit File**...:
    c. Add a meaningful commit message, and click "**Commit and Push**":
    d. Click the "**Push…**" button on the screen that follows.
5. Issue a pull request in GitHub. If you worked with a partner, review each other's code to ensure no conflicts were created.
6. Accept the pull request once all issues are resolved.
7. **Check your repository in GitHub to ensure everything was submitted to the master branch.** You can view the updated repository at the link you pasted in the "How to Get Started" section.