T08: 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 - T08: Mad Libs** (for example, **heggens, wilborned - T08: Mad Libs**). To do this, click the label in the top left corner of your browser.
- Next, go to the Github classroom for T08.
- 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 repo here:

Github Repo Link:	

- Open PyCharm. If you are not at the Welcome to PyCharm page, go to File >> Close project.
- Using the **Get from VCS** button, open your repository using the URL above.
- In the PyCharm menu bar, go to **Git >> New Branch...**
- Name the branch your usernames (e.g., **heggens_wilborned**). Click **Create**.

Go to t08_questions.md and answer the **Section 1** questions. Then, return to this document to continue the assignment.

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 **t08_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 task is to create a program to accomplish the following sub-tasks:

1. Create a story in a text file (e.g., story.txt).
    a. 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. For example, your file might start like this:

```
The cat
in the {0}
knows a lot
about {1}
```

    b. Your story can say whatever you want it to say, but it must have at least five 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. For example, your story string might look something like this, where `{1}` is replaced multiple times:

```
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}.
```

2. Create a second file for the prompts. For example, for the story in 1.a. the prompts might be:

```
Give me a noun:
Next, another noun:
```

3. Your program should open the prompts file, and for each prompt, capture the user's input. Save each response to a list.
4. Next, your program should read the story file into a string, and then replace all placeholders with the prompts captured in the list above. A handy way to format a string when you have a list is the following (note the asterisks (`*`) before the list `answers`):

```
story.format(*answers)
```

5. Finally, write the final output to a new file to save the result.
6. When the program runs, it should look similar to the following flow:

```
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:
```

7. 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
```

8. 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.
```

9. **Hints:**
    a. 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"]
```

    b. One might find the `append()` method of lists useful.
    c. Don't forget to `close()` files when you're done working with them!
    d. 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**.

Go to **t08_questions.md** and answer the **Section 2** questions. Then, return to this document to continue the assignment.

The 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 **t08_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.

CSC226: Software Design and Implementation
T08: Mad Libs

Submission Instructions

At the end of every assignment, I will include these instructions. They do change on occasion, so be sure you check them each assignment to ensure no special instructions were added.

Follow the submission instructions by Friday at 11:55PM.