T04: Adventures in Gitland

- Recall our discussion on Day 1 about how to work well with your partner.
- Pair programming involves two roles: the driver (who types the code) and the navigator (who reads the instructions).
- If you run into an issue, then work with your partner to solve it.
- If both you and your partner have the same issue, then ask a teaching assistant or instructor.

Learning Objectives

- Refactor existing code into functions.
- Get additional practice with nesting conditionals.
- Learn to use git to collaborate.

How to Start

- 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 addresses of all members in the bottom input field.
- Change the file name of this document to **username1, username2 - T04: Adventures in Gitland** (for example, **pearcej, heggens - T04: Adventures in Gitland**). To do this, click the label in the top left corner of your browser.
- We will **not** be using GitHub Classroom for this assignment. Instead, use PyCharm to clone the code from the T04 master repository. ***Don't start editing code until we instruct you to do so!***
- First, discuss with your team and assign yourselves roles.

Github Repo Link:	https://github.com/Berea-College-CSC-226/t04-main

First, discuss and assign roles. *Try to pick the role you've had the least experience in.*

	Monday:	**Wednesday:**	**Friday:**
Navigator:	Merci	Alpha	Merci
Driver:	Alpha	Merci	Alpha
Quality Control:			

A long time ago…

...in Teamwork T01, remember when we created a text-based adventure game where the user decided what happens in the story?

CSC 226: Software Design & Implementation
T04: Adventures in Gitland

In the T04 repository that you cloned, you will find **t01_final_story.py**, which is the product of your T01 creation. As much as we'd like to enjoy the story now that it is all compiled... it has errors!

One of the many useful skills you'll need to be a programmer is the ability to **refactor code**. Refactoring code is a process where we take code that is already written, and make it better.

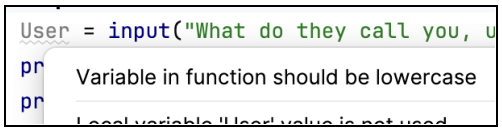
So why do we need to refactor T01? Well, there are a number of reasons:

1. Since T01, you've all become much more knowledgeable coders, and you may notice that your code has some design flaws that you now know how to fix. And bugs!
2. Now that you know about functions, you can see the structure of T01 is all wrong per our new rules (i.e., having a `def main()`; no code at the top level (no indenting); no mental blocks encapsulated into functions, etc.).
3. This code is broken!
4. This code is kind of a mess!

"Better code" can mean a lot of things: easier for the programmer to read; more efficient in terms of lines of code; more efficient in how well it runs; more abstracted, so the code can be reused elsewhere.

You might still be asking, but WHY do we need to refactor T01? Here are two even more compelling reasons:

1. When you leave Berea College and get your fancy job as a programmer, you'll be expected to conform to your company's coding standards (i.e., how your code looks so everyone else in the company can easily read your code). For Python, many companies follow the de facto PEP 8 Standards, and customize them to fit their exact needs. You've likely noticed many gray squiggly lines and warning messages about PEP8 in PyCharm:



   Refactoring code is one method you'll need to be familiar with to be an effective programmer, no matter who you work for in the future. ***Every programmer has to refactor code!***
2. By refactoring the T01 code, you're going to improve your skills with writing functions, writing useful documentation, and also learn how to use git to collaborate with the rest of the class. So, making T01 pretty isn't really the goal of this assignment; learning these other **essential** skills is the ultimate goal.

Your Tasks

For this teamwork, we'll be using git to manage our code. A large part of this teamwork is understanding how git works, and how git facilitates multiple teams of programmers working on one set of code, without clobbering each other's work. At a high level, we will clone the code from a repository on Github, make a new branch, make edits to the code, and push those changes back up to Github. Things should get interesting as multiple teams finish their changes and try to push them to the repository. We'll see what happens…

Branching

So far, we've not used git for collaboration very heavily. In this teamwork, however, everyone in the class will be modifying the same starter code…

1. Right click on the **t04-master** repository folder, then do **Git >> New Branch.** Name the branch you're about to create your username(s) (e.g., **pearcej_heggens**). Make sure that you are working *on your team's branch* before proceeding.
2. On your new branch, make a copy of the **t04_questions.md** file.
3. Paste the file into the **Answers** directory in the repository.
4. Rename the file to **t04_<your group name>_questions.md** (e.g., **t04_tsimalayt_heggens_questions.md**), then:

Inside the **t04_refactored.py** code, you'll notice a function definition for every team:

```
def team_1_adv():
    pass
    # TODO Add your code here
```

5. Copy your team's code from **t01_final_story.py** and paste it into your function. Watch the indentation!
6. Read through your code, and make changes to it when you notice places that could use improvement. Remember, your classmates wrote that code several assignments ago… see how much we've grown as programmers already!
7. Look for gray squiggly lines, indicating places where PEP8 standards aren't being followed. Make the suggested change by PyCharm to remove the squigglies.
8. In your function's docstring, add a link to your team's Google Doc (this document) as well as the names of all partners who worked on that function. For example:

```
def team_1_adv():
    """
    https://docs.google.com/document/...
    Scott Heggen
    Brian Schack
    :return: none
    """
```

Go to **t04_<your group name>_questions.md** and answer **SECTION 1**, then come back to this document to complete the rest of the assignment

Commit and Push your Changes

Test your code and make sure nothing is broken. Be sure you check ALL combinations of inputs and paths through your logic. When you're confident it's ready to go to the repository, **commit the file** (don't forget to write a meaningful commit message!), and **push** the changes to the repository.

Go to GitHub in your browser, and take a look at the repo.

> Go to **t04_questions.md** to **t04_(your group name)_questions.md** and answer **SECTION 2**, then come back to this document to complete the rest of the assignment

Do one more cycle of commit, push, and issue a pull request for the last two files: your **t04_questions.md** file in the **Answers** folder, and your PDF of this Google Doc in the **PDF Submissions** folder.

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.