CSCI 141 - Spring 2021
Lab 7: Strings
Due Date: See Canvas

Introduction

In this lab you'll gain practice using and manipulating strings. In lecture you've seen how to iterate over the characters in a string, access individual characters by index, and how to slice out substrings. We also touched on a few of the multitude of methods that can be called on string objects, including `upper`, `lower`, `find`, and `replace`.

Suppose you are an employee at a software company called Lotter.io. You have been tasked with writing a Python program that will simulate the drawing of a winning lottery pick, then prompt a user to guess the lottery pick, and inform the user if their pick is the winning pick.

Your company's secret sauce that will guarantee its explosive growth into a billion-dollar ("unicorn") startup is that the lottery doesn't just involve guessing numbers: Instead, this lottery involves both words *and* digits. A lottery pick is made up of one of three words (e.g., `baker`, `shucksan`, or `glacier`), and one of three digits (e.g., 1, 2, or 3), arranged either with the word first or the number first. For example, `shucksan1`, and `2baker` are both possible winning picks. The user only wins if the correct word and the correct digit are guessed in the correct order.

The program consists of four subtasks:

0. Determine the game mode.
1. Based on the game mode, either generate a winning pick randomly or use the user-supplied pick.
2. Get a valid guess from the user.
3. Check the user's guess against the winning pick and tell them how they did.

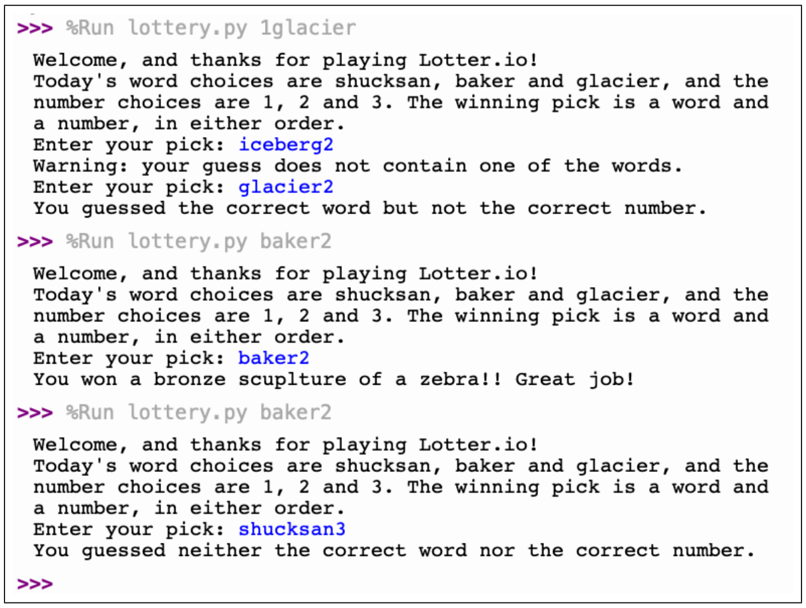
Three sample invocations are shown in Figure 1.

0 Determine the Game Mode

The main program is implemented inside the `main` function, which is called at the bottom of the program inside a main guard. Your first (`0`th) task is to determine the game mode based on the command line arguments.

As in the Guessing Game problem from A3, the lottery program will have a debugging mode which allows the person running the program to specify the winning pick. The game mode is determined as follows:

- If the program is run with no command line arguments, the game runs in its regular (player) mode, and the winning pick is randomly generated.



```
>>> %Run lottery.py 1glacier
 Welcome, and thanks for playing Lotter.io!
 Today's word choices are shucksan, baker and glacier, and the
 number choices are 1, 2 and 3. The winning pick is a word and
 a number, in either order.
 Enter your pick: iceberg2
 Warning: your guess does not contain one of the words.
 Enter your pick: glacier2
 You guessed the correct word but not the correct number.
>>> %Run lottery.py baker2
 Welcome, and thanks for playing Lotter.io!
 Today's word choices are shucksan, baker and glacier, and the
 number choices are 1, 2 and 3. The winning pick is a word and
 a number, in either order.
 Enter your pick: baker2
 You won a bronze scuplture of a zebra!! Great job!
>>> %Run lottery.py baker2
 Welcome, and thanks for playing Lotter.io!
 Today's word choices are shucksan, baker and glacier, and the
 number choices are 1, 2 and 3. The winning pick is a word and
 a number, in either order.
 Enter your pick: shucksan3
 You guessed neither the correct word nor the correct number.
>>>
```

Figure 1: Three sample invocations

- If the program has at least one command line argument, the winning pick is set to the first command line argument. You should assume that the command line argument is a valid pick; you do **not** need to verify this. This feature will be useful for you in debugging (and for us in rading!).

We've been using command line arguments since A2, but at this point we've actually seen everything we need to understand the syntax. Recall that we access command line arguments inside our programs by first importing the `sys` module:

```
import sys
```

and then accessing the individual arguments as `sys.argv[i]` to get the `i`th command line argument. We can now see that what's actually happening here is that inside the `sys` module lives a special variable called `argv`, which points to a `list` of command line arguments, and we're simply *indexing* into that list to get access to each argument.

There's one more quirk: lists indices start at zero, but the command line argument indices start at 1. This is because of a convention in Python that `sys.argv[0]`, the first element of the arguments list, stores the name of the program that was run. You can see this for yourself by printing `sys.argv[0]` in any program: you'll find that it contains the name of the file containing the program.

A consequence of `sys.argv` being a `list` is that you can have your program's behavior depend on the number of arguments given, as prescribed above, by simply checking the value of of `len(sys.argv)`. Just keep in mind that a length of 1 means there were no arguments, a length of 2 means there was one argument, and so on.

1 Generate a winning pick

The `random_pick` function is responsible for choosing a random winning pick. The function header, specification, and some basic pseudocode comments have been written for you. The random pick is a randomly chosen word from the `words` list and a randomly chosen number from the `numbers` list, concatenated in a randomly chosen order. So in the case of our program, the following are all valid picks:

- baker3
- 1baker
- shucksan2
- 2glacier

You may find the `random` module's `choice` function useful.

2 Get a valid pick from the user.

Implement the `get_valid_guess` function to prompt the user, repeatedly if necessary, until they enter a guess that contains one of the words and begins or ends with one of the numbers. Notice that these criteria do make it possible to enter an invalid pick (e.g., `mountbakery3` contains one of the words and ends with one of the numbers. This is fine: cases such as these will be handled later in the program. The function header, specification, and pseudocode for this part are provided in the skeleton code. *Hint:* to check whether the guess contains a valid word, you might be tempted to check whether the guess is a member of the list of words; this won't work because the guess also has a number. Instead, I recommend checking whether each of the valid words is a substring of the guess string.

3 Check the user's pick

Now, we need to tell the user whether any or all aspects of their pick were correct. Here's what the program should do once it has determined the winning pick and the user's pick. Your messages should convey the same content but otherwise you can be creative with them. Feel free to decide on your own prize for guessing the winning pick.

- If the user inputs a pick that exactly matches the winning pick character for character, the program outputs a message telling the user they've won.
- If the user's word and number both match the winning pick, but their pick doesn't match the winning pick exactly, print *You guessed the correct number and word in the wrong order!*
- If the number matches but the word doesn't, print *You guessed the correct number but not the correct word.*

- If the word matches but the digit isn't correct: print *You guessed the correct word but not the correct number.*
- If neither the word nor number are correct, print *You guessed neither the correct word nor the correct number.*

The `main` method includes calls to the two functions you completed in the prior tasks, followed by pseudocode for this step. You may want to, but are not required to, create a separate function to accomplish this task.

Getting Started

Download the skeleton code `lottery.py` from the course webpage. Open up and read through the skeleton code, making sure that you understand the provided code and function specifications. We recommend completing the three tasks in the order described above.

You may find the `in` and `not in` operators useful on strings, as well as indexing individual characters of a string and slicing substrings. Review the lecture slides for details on the syntax for these operators. Don't forget that when a program's logic gets complicated, it can be helpful to define boolean variables to keep the conditions of `if/elif` statements short and easy to read.

Submission

Submit your completed program `lottery.py` file via Canvas.

Rubric

Comment at the top of program specifies author, date, and description	1
Command line argument is used as the winning pick if specified	3
Random pick is generated if there are no command line arguments	3
User is prompted again if guess doesn't contain one of the words	3
User is prompted again if guess doesn't begin or end with one of the numbers	3
Message printed for correct pick	3
Message printed for correct word and number but incorrect order	3
Message printed for correct word but not number, and vice versa	3
Message printed for neither correct	3
Coding style (commenting, variable names, etc.)	5
Total	30 points