Programming Project 2: MasterMind

CS-106 Fall 2020

Overview

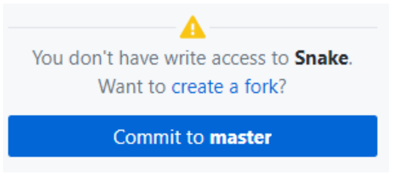
GitHub Repository: https://github.com/Qu-CMU/ACES_CS106_F20_MasterMind_Lab

This lab will ask you to make choices about which data structure to use to for different variables.

1. Clone the Repository
2. Background Information
3. Generating a Random Code
4. Counting Digits
5. Checking the Guess

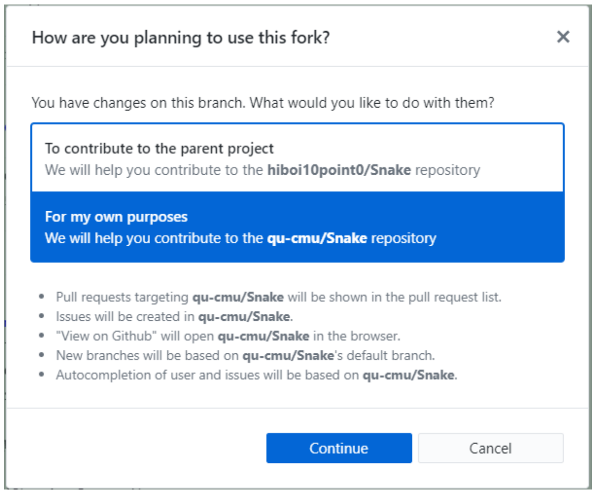
Clone and Fork the Repository

- Clone the project repository from https://github.com/Qu-CMU/ACES_CS106_F20_Snake_Lab
- Rename '**MasterMind_StarterCode.py**' to '**MasterMind.py**'
- Go to the GitHub Desktop app and choose to '**create a fork**'



You don't have write access to **Snake**.
Want to create a fork?

- When it asks, 'How are you planning to use this fork?'
  Choose '**For my own purposes**'



How are you planning to use this fork?

You have changes on this branch. What would you like to do with them?

To contribute to the parent project
We will help you contribute to the **hiboi10point0/Snake** repository

For my own purposes
We will help you contribute to the **qu-cmu/Snake** repository

- Pull requests targeting **qu-cmu/Snake** will be shown in the pull request list.
- Issues will be created in **qu-cmu/Snake**.
- "View on Github" will open **qu-cmu/Snake** in the browser.
- New branches will be based on **qu-cmu/Snake**'s default branch.
- Autocompletion of user and issues will be based on **qu-cmu/Snake**.

- Click 'Continue' and commit the changes

Background Information

MasterMind is a board game where two players compete against each other. One player creates a code and the second player tries to guess the code. We will be playing a variation where we use numbers instead of colors.

For this lab, we are programming the computer to be the code maker. At the start of the game, the program should generate a completely random code. After, every time the player makes a guess, the program should tell the player how many numbers they guessed correctly and how many numbers are in the correct spot.

For example, pretend the code is **0023**.

- If you guess 0000, then 2 numbers are correct, and 2 numbers are in the correct spot
- If you guess 1234, then 2 numbers are correct, and 0 numbers are in the correct spot
- If you guess 0203, then 4 numbers are correct, and 2 numbers are in the correct spot
- If you guess 0023, then 4 numbers are correct, and 4 numbers are in the correct spot
- If all numbers are correct and in the correct spot, the player wins.

Generating a Random Code

- Implement the function **generate_code**()
- If you are not sure which random function to call, try seeing what this one does:
  - https://www.w3schools.com/python/ref_random_randint.asp

Some examples:

- **length = 4, possibilities = 6**
  - Valid Codes: 0000, 0123, 1234, 5122
  - Invalid Codes: 12311, 6666, 4672, 231
- **length = 6, possibilities = 2**
  - Valid Codes: 000000, 111111, 011001
  - Invalid Codes: 000, 1111111, 012012

Counting Digits

- Implement the function **count_digits(code)**
- This can be done with a list or a dictionary, your choice.

Some examples:

- **Code = 0203**
  - In this code, ‘0’ appears 2 times, ‘1’ appears 0 times, ‘2’ appears 1 times … and so on.
- **Code = 2212**
  - In this code, ‘0’ appears 0 times, ‘1’ appears 1 times, ‘2’ appears 3 times … and so on.

Check the Guess

- Implement **check_guess(guess)**
- We want this to check how correct the player’s guess is.

An example:

- **Code = 0023**
  - If you guess 0000, then 2 numbers are correct, and 2 numbers are in the correct position
  - If you guess 1234, then 2 numbers are correct, and 0 numbers are in the correct position
  - If you guess 0203, then 4 numbers are correct, and 2 numbers are in the correct position
  - If you guess 0023, then 4 numbers are correct, and 4 numbers are in the correct position
  - If all numbers are correct and in the correct spot, the player wins.

Getting the Input

- Implement **get_guess()**
- We need to read in the player’s guess from the shell.
- If you don’t know what function reads inputs, try seeing what this function does:
  - https://www.w3schools.com/python/ref_func_input.asp
- The input format can be whatever you want, as long as you print the instruction for the player.
- If the player types in something that doesn’t match the format, ask them for another input.
- Example formats:
  - 0123
  - 0 1 2 3
  - [0, 1, 2, 3]

Input Validation

- Implement **is_valid_guess(string)**
- This function checks if the string input by the player is valid.
- It should only return **True** if the string is a valid input.
- Some things that could make an input invalid:
    - It contains too many numbers
    - It is too short
    - It has letters and other symbols that are not numbers
    - It has a number that is not a possible digit value

Putting it All Together

- Implement **start()**
- Using all the functions that you have written, put them together to make the game.
- The process should look something like this:
    1. The computer needs to come up with a code
    2. The computer needs to ask the player for a guess
    3. The computer needs to tell the player how correct the guess is
        - How correct is the guess if the player guessed the code exactly?
    4. Repeat steps 2 and 3 until the player runs out of guesses or guesses correctly

Optional Challenges

- So, you thought the lab was too easy?
- Make a pretty print game screen using strings and the string format function.
    - https://www.w3schools.com/python/ref_string_format.asp