**Mastermind**

**Introduction**

In this project, you will program a text-based version of the board game *Mastermind* using all of the following concepts as described in lecture and homework:

* Variables and Functions
* Control Statements
* Strings and String Operations
* Lists and List Operations
* While Loops and For Loops

*Mastermind* is a two player game in which each player is designated either a “Code Maker” or a “Code Breaker”. The Code Maker creates a code which consists of a series of colors. This code is kept secret from the Code Breaker, who tries to guess what the code is. After each guess, the Code Maker gives the Code Breaker feedback on how many colors in the Code Breaker’s attempt is in the right place, how many are in the wrong place, and how many do not exist in the secret code. After ten attempts, if the Code Breaker cannot guess the secret code, the Code Maker wins.

Understanding the rules of the game before you begin programming will ensure that you will not waste any time writing code that will not be used. Please watch this [3-minute video tutorial of *Mastermind*](https://www.youtube.com/watch?v=dMHxyulGrEk). Additionally, you may play *Mastermind* here.

**Logistics**

This project is worth 15% of your grade. We are grading on correctness. However, you are able to gain extra credit by applying the Clean Code Tips that we have been going over in class.

For the functions that we ask you to complete, there may be some initial code that we provide. If you would rather not use that code, feel free to delete it and start from scratch. You may also add new function definitions as you see fit. However, please do **not** modify any function signatures (names, argument order, or number of arguments).

***Phase One: The Code Maker***

**Problem 1: The Secret Code (\_\_\_ points)**

In this version of *Mastermind*, the AI (artificial intelligence, or computer) will **always** be the Code Maker and the human playing will **always** be the Code Breaker. This means we must write the AI to create its own secret code.

Implement the generate\_code function. It does not have any parameters and returns a randomly generated list of the allowed characters ‘1’, ‘2’, ‘3’, and ‘4’. These characters serve the same purpose as the colors in the game described in the introduction.

Important functions to know:

* In your code\_breaker.py file, you may have noticed that we have an import statement at the top: import random. Don’t concern yourself too much with what import means yet, as we’ll go over that more later. All it means is that we can use some special functions for this problem. In particular, random.choice(), which takes a list as an argument and returns a random item from that list.

***Phase Two: Code Breaker***

**Problem 2: Getting User Input (\_\_\_ points)**

Now that the AI has created a secret code, we need a way to ask the player to guess what the secret code is. Implement the function get\_code\_breaker\_attempt. It returns the Code Breaker’s attempt as a list of allowed characters ‘1’, ‘2’, ‘3’, and ‘4’.

Important Functions to Know:

* input(): This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python.
* str(): Notice that the allowed\_characters list has numbers as strings. To convert number input into a string, simply pass it into the str() function which converts its arguments into strings.

**Problem 3: Making Sure Input is Valid (\_\_\_ points)**

Assuming the player inputs a string, we need to check if the input is valid. What criteria should we base this on?

* ***The Code Breaker’s attempt cannot be shorter or longer than the secret code.***Implement wrong\_code\_length, which returns True if the Code Breaker’s attempt does not match the length of the secret code.
* ***The Code Breaker’s attempt must only used the allowed characters ‘1’, ‘2’, ‘3’, and ‘4’.*** Implement wrong\_characters, which returns True if the Code Breaker’s attempt contains any illegal characters.

In addition, inside the function get\_code\_breaker\_attempt, implement code which will **repeatedly ask** the Code Breaker for valid input if they keep putting in invalid code.

***Phase Three: Giving Player Feedback with Key Pegs***

In this phase, we give the AI the tools to give the player feedback. As you saw in the *Mastermind* tutorial, one of the roles of the Code Maker is to place red and white key pegs next to the Code Breaker’s attempt to indicate whether the Code Breaker has colors that are in the right spot (red), or in the wrong spot (white). If a color doesn’t exist in the code, then the Code Maker leaves a spot empty.

Once this phase is finished, the AI will be able to tell the Code Breaker how many red, white, and empty pegs they got with their guess.

**Problem 4: Comparing Codes (\_\_\_ points)**

Implement check\_numbers which returns how many numbers in the Code Breaker’s attempt exist in the secret code. This will be an integer named “numbers\_check”.

Then, implement check\_order which returns how many numbers in the Code Breaker’s attempt are the correct number in the correct position. This will be an integer named “order\_check”.

**Problem 5: Key Pegs (\_\_\_ points)**

Implement get\_key\_pegs which returns a four-element list of the strings, “Red”, “White”, and “Empty”. It indicates how many numbers in the Code Breaker’s attempt are in the right position (“Red”), wrong position (“White”), or don’t exist in the code (“Empty”).

**Problem 6: Feedback (\_\_\_ points)**

Implement give\_player\_feedback, which simply prints the key pegs found from Problem 5, plus some additional words to make it sound like the AI is talking to the Code Breaker. Feel free to be as creative as you want!

***Phase Four: Run the Game!***

**Problem 7: Mastermind (\_\_\_ points)**

Now that we have all these functions at our disposal, implement the function mastermind, which runs through the entire game. The way you implement will be entirely up to you, but these are things to keep in mind as you program:

For this game,

* Tell the player how many attempts they have remaining. If they pass 10 attempts, they lose the game.
* If the key pegs are all “Red”, the player wins.
* When the game is finished, tell the player what the secret code was.
* Have an option to play the game again (hint: use while loops).

We have included a function called continue\_game\_condition. This function returns False if the game is over (because the player has won or lost), and returns True if the player still has remaining attempts. You can use this function if you want to, but it’s completely optional. Remember that you may write your own functions.