1 Mastermind

Today we will implement the game of mastermind. An online version is here:

`http://www.web-games-online.com/mastermind/`

Here is a high level description of it.

```
# generate secret code (4 letters from R,G,B,Y,O,P)

# repeat the following steps...

    # ask user for guess (only accept valid guesses)
    # report number of red and white pins

    # stop when user has guessed 10 times, or
    # they guess the code!

# print final result (win, or loss and reveal secret)
```

2 Top down design

Top Down Design is a problem solving technique where you:

1. Start with general description of problem
2. Break it into several high-level steps
3. Iteratively break the steps into smaller steps until you have steps that are easy to solve.

It is similar to writing a paper where you start with an outline, then fill in the second-level details, and so on until you can start writing each section/function.

Whether or not you practice top down design, your code should look like it was written using top down design. In other words, the code should have **structure**. Almost always, that structure is **hierarchical**: some functions carry out "high level" tasks and other functions are helper functions that take care of "low level" details.

The same is true for writing. You may or may not write your essay by starting with an outline. However, given the final essay, it should be easy to extract an outline from it.

Programs that are written using top-down design tend to have functions that are SOFA and DRY (which stands for Don't repeat yourself). If you don't use top-down design, you still must meet these criteria!

Some material adapted from Wexler and Knerr.

3 SOFA

In well-designed programs, each function meets the SOFA criteria:

- **S**hort: more than 10-15 lines is probably too much
- does **O**ne thing: if the function does more than one task, break it down into smaller functions.
- takes **F**ew parameters: Alan Perlis, a famous computer scientist said, "If you have a [function] with 10 parameters, you probably missed some."
- maintains a single level of **A**bstraction: a function should focus on either high-level or low-level, and not try to do both.

The first three criteria are straightforward. The A needs some additional explanation. Here are some examples from mastermind.

- The `play_game` function is **high level**. It focuses on game logic. It's quite easy even for a non-programmer to read it and understand exactly what is happening.
- The `count_red` function is **low level**. The function's purpose is narrow: to count matches between two strings. It would be incomprehensible to someone who hasn't taken 101. However, the docstring should be clear enough that someone can understand *what* your program does without having to read the body of `count_red`.

4 One more design tip

Test your code often. There are two ways to test it.

- Run the entire main program and check that the latest feature you added works correctly. For example, with mastermind, to test `count_red` I could play the *entire* game, trying different guesses and making sure the red pins were calculated correctly.
- Comment out the call to the main program and "Run Module" in IDLE. This will effectively import your code. Try out individual functions in the interpreter. For example, I can focus exclusively on `count_red` by trying it out in the interpreter.

```
>>> count_red('RGYY', 'PGOY')
2
```

5 Exercises

Solutions are presented in class and also included in the moodle version of this handout.

1. Use top-down design to write a super bare bones version of mastermind. Focus on writing a function that covers the high-level game logic (asking the user for input, keeping track of number of guesses, and announcing the game outcome). Leave low level details (generating a random code, counting red/white pins, etc.) for later.