# Lab 3: Basic Recursive Design

This lab will give you a chance to practice basic recursive design (as discussed in class and the course notes). Note that some of these still have interesting "corner cases", so you should remain alert to the various kinds of legal questions that could be given to your algorithm. Furthermore, you will be graded on both your design comments (Questions 1 and 2, part (a)), so give some thought to them as well as the Python functions you write.

**Pre-Lab Work:**

Mark up a copy of a test suite for `is_palindrome` (you may print it on paper or open it in an editor in which you can make markups by circling things or permantly highlighting text). Your test suite may be from your answer to the previous lab, or an answer key for this if one has been made available. In your markup, make a preliminary identification of each test as either "base case" or "recursive"; for each of the latter, circle/highlight the parts of the parameters that would constitute a smaller instance of the is_palindrome problem whose answer would help you to find the final answer for that test.

**Lab Work:**

1. In the file `palindrome.py` of the `basic_recursive_design` project, enter design comments and a Python function for the `is_palindrome` problem.
   a) Give, in a comment at the top of `palindrome.py`, a description of the five steps of a basic recursive design for the palindrome problem. If you wish to refer to the reversal of a string, you must also give a self-referential definition of reversal or a basic recursive design for a reversal algorithm — do **not** use built-in features of Python to reverse a string in your algorithm (e.g., with `[::-1]`), though you may use them in your postcondition.
   b) Give a straightforward translation of your design into a Python function. If you are already familiar with Python's `while` and `for` statements, *do not* use them yet, but instead do a simple translation like the ones we've done in lecture for exponentiation and other problems — we'll get to loops later in the semester.
   c) Test and debug your function. If you think of new important tests, add them to your test suite; if you must revise your basic approach, revise your design comment as well as your Python function.
   Debugging `is_palindrome` may be a bit trickier than your debugging for previous labs. Before "playing computer", you may find it helpful to find the smallest example that demonstrates a problem: Choose a test case that shows a problem, and add simplified versions of it (possibly simplifying in the way your recursive design does), until you find a simplified version that works. At this point, "play computer" on the simplest test that was still problematic — if it calls is_palindrome recursively for the simplified problem that you now know works, just insert the answer you've confirmed in your test suite.
2. In the file `fibonacci.py` of the `basic_recursive_design` project, enter design comments and a Python function for the `fib` function you described last week.

a) Provide design notes in a comment before the function `fib`. These notes may be *either*
   - a self-referential mathematical definition of the Fibonacci sequence (either based on the text from last weeks lab or other references (you should cite, in the comment, any other reference you use)), or
   - a description of all five of the "basic recursive design" steps from the Basic Recursive Design section of the course notes.

b) Write a function that is a straightforward translation of your design that uses the language features we have covered in class. Python's `while` and `for` could be useful here, but using them will not serve as a good preparation for more advanced labs. Also, do not worry about the processing speed of your function — the fact that this function is very slow for large $n$ will be the motivation for revisiting this problem later.

c) Test and debug your function as necessary. If you think of new important tests, add them to your test suite; if you must revise your basic approach, revise your design comment as well as your Python function.

3. Obtain the files for the `graph_coloring` project and create a *test suite only* for the `is_a_legal_coloring` function in the `is_legal.py` file. This problem is described in Exercise 4.9 of the course notes; this week you are doing only 4.9.a.i. For now, use only the colors r, g, and b (for red, green, and blue). You can experiment with examples by running the program in `A_graphical_user_interface.py`. As you add tests, watch to see how many of our sample answers you can break.

We will continue to work on the `graph_coloring` project over the coming weeks, creating the function to test for a valid coloring, and adding an enumeration function, eventually completing a full enumerate and test algorithm.

Remember to use "Team→Commit" as you make progress and when you are done (but let us know, via the comment that you can enter as you commit, whether you have finished).