1 The PDB debugger

Your goal here is to get to know pdb, a "debugger" that can help you find the problems with your programs. Debuggers can't help you until your program compiles, but once you have eliminated all compiler errors, a debugger can be one of your most valuable tools.

Today, you'll use pdb to (1) watch a variable, (2) single-step through a program, and (3) set breakpoints. All three of these operations should become clear shortly.

1. Go to the department web page, and find the link for `odds.py` for use in the debugging lab. Save it somewhere.
2. Open `odds.py` in your editor, and move the window off to the side.
3. In the terminal window, run the command

   ```
   python -m pdb odds.py
   ```

4. At the pdb prompt, type **list**. You should see the first 11 lines of your program. Type **list** again (or abbreviate with just the letter **l**. You should see the next 11 lines, or in this case, the rest of the program. Next, type **l 8** (that's the letter **l** again, followed by the numeral **8**). You should see the lines of code around line 8.
5. Line 16 should be the one that reads **n = 10**. Set a breakpoint on this line by typing **break 16** at the pdb prompt.
6. Continue the program running (it had stopped running at the first line) by typing **continue** at the pdb prompt. The program will run until your breakpoint and stop there.
7. Type **print n** at the pdb prompt. Also do **print total** at the pdb prompt. What happens? Why? (You can abbreviate **print** with the letter **p**.)
8. At the pdb prompt, type **step**. What happens? What is the value of **n** now? What is the value of **total** now?
9. Keep stepping your code. How do **n** and **total** change as you move through your code? Remember that at any point you can hit the letter **l** to see where in your code the computer is currently working.
10. Run the program again, but this time type **next** instead of "step" to move through your program. What is the difference between "next" and "step"? Why is this useful for debugging?
11. Run the program again. Pdb will automatically start running it again when your program terminates, so type `continue` (or the abbreviation `c` (you might have to do this repeatedly if there are break points along the way), and watch pdb to see when it tells you it is restarting the program. Alternatively, you can quit pdb (**exit**, or `<ctrl>-d`) and start it up again. However you choose to restart your program, this time use `step` again to step into the **sumOfOdds** function. After you have stepped through a couple of iterations of the while loop, type **where** (or **w**) at the pdb prompt. Then try **up** (or **u**) and **down** (or **d**). What do these commands do? How might they be useful?

2 Making and fixing buggy code

Find your DoublyLinkedList class that you used for the history assignment. If you're not working with the same partner that you did at the time you did that assignment, just pick someone's code to use. Introduce subtle and surreptitious bugs into your code that might be hard to find. Perhaps you might make inserting and/or deleting only mess up the pointers in one direction. Maybe your code only fails when the list is empty. Or whatever. Be

creative and devious. Put your names in comments at the top. When you've got something, copy it to the directory `/Accounts/courses/cs201/dmusican/sharedcode` . Then take someone else's code from that same directory, and use pdb to help you debug it. Can you find the bugs?