Loop Practice (Nested and non-nested)

1. Write a program that simulates a stopwatch that records minutes, seconds, and hundredths-of-a-second. This program should start the stopwatch at time 0:00.00 (zero minutes, zero seconds, and zero 1/100 seconds), and stop at 5:59.99. Use three nested loops to print all of these times increasing in order of time. Hint: the inner loop should keep track of the 1/100 seconds part; write this loop first, then add a loop outside of that one, then another one outside of that one. ***Do not worry about formatting the output nicely; just print the three components of the time on a single line***.

   To make this easier and faster to test, once you start experimenting with the nested loops, don't have each time component count to its full upper limit (because you'll have to wait a long time to see all the output). For instance, have hundredths-of-a-second count from 0 to 10 (instead of 100) and seconds count from 0 to 10 (instead of 60). Then once you're convinced everything is working, set the upper limits back to their "correct" values.

2. Write a function called `count_factors` that takes a single parameter called `num`. This function returns the number of positive factors of `num`; this is the number of positive integers between 1 and `num`, inclusive, that divide into `num` evenly. For instance, the number 10 has 4 factors: 1, 2, 5, and 10. So calling `count_factors(10)` should return 4.

   Do this by writing a loop that counts from 1 to `num` and tests the remainder of dividing `num` by whatever the counter variable is.

3. Write a function called `is_prime` that takes a single parameter called `num`. This function returns `True` if `num` is prime; that is, if `num` has only two factors: itself and 1. Do this similarly to `count_factors`, but take advantage of the fact that once you find a single number that divides evenly into `num` (that is not 1 or `num` itself), then you can stop searching for more factors, because `num` can no longer be prime. Hint: use `break` to stop the loop.

4. Write a program that prints all the prime numbers that are less than 100. Hint: the smallest prime number is 2 (1 is not a prime number.)

5. Write a program that prints the first fifty prime numbers. Do not do this by first figuring out what the 50th prime number ahead of time is and printing all prime numbers less than or equal to that number. Do this by generating prime numbers as you go and stopping when you have generated fifty of them.

6. Write a program that starts off asking the user how much money they have in their bank account. Then enter a loop that continuously asks the user to enter an amount of money they want to withdraw from an ATM. Keep looping until the account is empty.

   Next, add a menu to let the user add money, subtract money, or quit the ATM program. Let the user keep using the ATM as long as they want (until they choose to quit). Prevent the user from withdrawing more money than they have in their account. Use input validation to prevent the user from typing in negative amounts of money.

   (turn over)

7. Write a program that lets the user type in a number from the keyboard. The program should print out the pseudo-Roman numeral equivalent of the number. I say "pseudo" because we will simplify Roman numerals a bit by getting rid of the weird subtraction rules for Roman numerals. For example, normally 9 is written as IX = 10 – 1, but your program can print VIIII. (Guide: In Roman numerals, M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, and I = 1.)

   Use a loop that runs until the user's number becomes equal to zero. Inside the loop, write if statements that test how big the number is. If the number is bigger than or equal to one of the exact Roman numerals above, print that numeral, subtract the value from the user's number, and loop again.

   Challenge: make this work with "real" Roman numerals; e.g., for 9 it should print IX, not VIIII. Try this on your own, but I have a hint if you really want it.

8. Write a guess-the-number program. Use `random.randint()` to have the computer pick a random number between 1 and 100. Write a loop that lets the user guess numbers until they guess right --- the computer reports back for each guess whether it was "too low" or "too high."

9. Write a graphical game program, "Find the Hole". The program should use a random number generator to choose a circular "hole", selecting a point on the graphics canvas and a perhaps the radius around that point. These are invisible, however, and are not shown to the player initially. The user is then prompted to click around on the canvas window to "find the hidden hole". You should display the points the user has tried. Once the user selects a point that is within the chosen radius of the mystery point, the mystery circle should appear. There should be a message announcing how many steps it took, and the game should end.

10. Write a program that simulates a graphing calculator for a specific type of function (e.g., parabolas). For instance, let the user type in values for a, b, and c, and graph the equation $y = ax^2 + bx + c$.