Lab Exercise 2: Seconds Converter

You may use whatever method you wish to write and run a Python program for this lab. If you prefer using IDLE or some other IDE (integrated development environment) you may do so. If not, you can simply create a file named `lab2_<your Pitt account>.py` using your favorite text editor and run it using the Python interpreter from the command line. You will only need to submit your Python source code to get credit for this lab.

Goal:

Write a program that asks the user to enter a number of seconds and then prints the same amount of time in days, hours, minutes, and seconds.

For example, 3667 seconds is equivalent to 0 days, 1 hour, 1 minute, and 7 seconds.

Print out the result in the format:

"0 day(s), 1 hour(s), 1 minute(s), and 7 second(s)."

Procedure:

1. First, we need to read in and store an int variable representing the number of seconds. Add the following line to your program.

   ```
   seconds = int(input('Enter number of seconds to convert: '))
   ```

2. We're going to want to use a format string for our output that includes a placeholder for days, hours, minutes, and seconds. Build that string (using new style string formatting placeholders) and store it in a variable named output as follows:

   ```
   output = '{} day(s), {} hour(s), {} minute(s), and {} second(s).'
   ```

3. Now we need to think about calculations. How many seconds are in a day, hour, and minute? Let's store some additional variables with these values to make them easy to reference when we need to do comparisons and calculations.

   ```
   secs_in_min = 60

   secs_in_hour = secs_in_min * 60  # 60 minutes per hour

   secs_in_day = secs_in_hour * 24 # 24 hours per day
   ```

4. We're also going to need some other variables to store the results of calculations and to provide them to the .format() method of our `output` variable. Create the following three variables.

```
days = 0

hours = 0

minutes = 0
```

5. Now for the fun part, how exactly do we go about determining the number of days, hours, minutes, and seconds from a given number of seconds? Let's think of some examples and work through them as we design the algorithm

    a. In a given day there are 24 hours. If there are 60 minutes per hour and 60 seconds per minute than 1 day contains 24 * 60 * 60 or 86,400 seconds. If the user typed in something greater than or equal to 86,400 seconds, we would expect the output to start with at least 1 day(s). If they typed in less than 86,400 seconds, we would expect the output to start with 0 day(s).

    b. If the value is >= 86400, we know we'll have at least 1 whole day. We can divide the total figure by the value we calculated and stored in `secs_in_day.` The result of the integer division is the number of days, and the fractional remainder that is left over will then need to be subdivided to determine the number of hours, minutes, and seconds it represents.

6. Add an if statement to check whether the `seconds` variable provided by the user's input is greater than or equal to `secs_in_day`. If it is, we'll perform some calculations to get the number of days the figure represents, and then assign the remainder of the operation to seconds to continue our conversion.

```
if seconds >= secs_in_day:

      days = seconds // secs_in_day  # Integer division

      seconds = seconds % secs_in_day  # Remainder operation
```

7. Ok, we've only solved part of the problem but let's add a print statement so we can begin to test the output. Add this to the bottom of your program, save it, and then run it. If you're using IDLE, you can run the program by hitting F5 or by selecting Run from the Run menu. If you're working from a terminal, you can invoke the python3 interpreter and pass it the name of the file you created.

```
print(output.format(days, hours, minutes, seconds))
```

8. Let's try supplying some inputs that we know what the output should be.

    a. Type 86400 at the program's input prompt. You should receive an output string that says "1 day(s), 0 hour(s), 0 minute(s), and 0 second(s).".

    b. Type 432000 at the program's input prompt. That's 86400 * 5. You should receive an output string that says "5 day(s), 0 hour(s), 0 minute(s), and 0 second(s).".
    c. Type 120 at the program's input prompt. You should receive an output string that says "0 day(s), 0 hour(s), 0 minute(s), and 120 second(s).". That's not correct yet, indicating that we still have work to do.
    d. Type 86410 at the program's input prompt. You should receive an output string that says "1 day(s), 0 hour(s), 0 minute(s), and 10 second(s).". By now, you can see that our program is calculating days and seconds correctly, but we need to be able to also account for the hours and minutes conversions.
9. After our first if statement has been evaluated, we know that the value stored in the seconds variable is an integer somewhere between 0 and 86399. Why? Because, if the if statement's condition was false, we know that seconds was less than 86400 to begin with. If the condition was true, the % (remainder) operation would ensure that we end up with a number less than 86400 since the remainder of a division operation can never be larger than denominator (bottom part of a fraction or division operation).
10. We need to repeat this type of calculation again to figure out the number of hours reflect in the seconds figure. It should look very similar to how we calculated days. Add these lines after (and outside of) your first if statement, but before your final print statement added in step 8.

```
if seconds >= secs_in_hour:

    hours = seconds // secs_in_hour  # Integer division

    seconds = seconds % secs_in_hour  # Remainder operation
```

11. Let's save our program and test it again by supplying some known inputs.
    a. Type 90000 at the program's input prompt. This is the same as 86400 (the seconds in a day) + 3600 (the seconds in an hour). You should receive an output string that says "1 day(s), 1 hour(s), 0 minute(s), and 0 second(s).".
    b. Type 7200 at the program's input prompt. You should receive an output string that says "0 day(s), 2 hour(s), 0 minute(s), and 0 second(s).".
    c. Type 8100 at the program's input prompt. You should receive an output string that says "0 day(s), 2 hour(s), 0 minute(s), and 900 second(s).". We're still not there yet since 900 seconds needs to be converted to minutes.
12. Now that we have factored out the days and hours from our user's input, we need to do the same procedure one final time to get the number of minutes. Like we saw in step 9, after we've completed the evaluation of the if seconds >= secs_in_hour statement, we are ensured to have a value stored in seconds that is less than one whole hour in length. Add these final lines before your print statement to perform the final calculation step.

```
if seconds >= secs_in_min:

    minutes = seconds // secs_in_min  # Integer division

    seconds = seconds % secs_in_min  # Remainder operation
```

13. Save your program one last time and test it again with the following inputs. This time, our calculations should be right on for each placeholder.
    a. 129: "0 day(s), 0 hour(s), 2 minute(s), and 9 second(s)."
    b. 180122: "2 day(s), 2 hour(s), 2 minute(s), and 2 second(s)."
    c. 31556939: "365 day(s), 5 hour(s), 48 minute(s), and 59 second(s)."
    d. 604800: "7 day(s), 0 hour(s), 0 minute(s), and 0 second(s)."
    e. 604799: "6 day(s), 23 hour(s), 59 minute(s), and 59 second(s)."
14. When you are done, show your lab assistant your code, save a copy of it (one for each of you if you are working in pairs), and submit them to CourseWeb.