Indefinite Iteration

The loops we looked at in last week's lab are examples of *definite* iteration, where we know how often the loop would have to repeat, based on the definition of what the function did. There are also cases where we do not know how often a loop may have to repeat before a desired value or condition is obtained. These case use *indefinite* iteration, usually implemented in Python with the `while` statement.

Using the while Statement

We will start with an example from the lectures that could have been written with a loop based on the `for` statement, since we know from the beginning how often the loop will repeat, (an example of definite iteration).

```
def countdown(start):
    t = start
    while  t  >  0:
        print(t)
        t = t - 1
    print('Blast  Off!')
```

Given a positive integer for `start`, it prints a sequence of decreasing numbers until zero is reached and then prints a final `Blast Off!` message.

Try this function a few times and make sure you understand what the `while` statement is doing and how it works. There is no need to add printed test cases to your program for this function.

The Collatz Conjecture

Start with any positive integer, n. Generate a sequence of numbers where the next number in the sequence is half of current value of n, if n is even, and 3 times n plus 1 if n is odd. The Collatz conjecture claims that this sequence will eventually reach 1 for all positive integers.

Write a function, `collatz(n)`, which will print out this sequence for a given value of n. If n is not positive, your function should do nothing. For example, for an original value of 3 your function should print something like the following. (The # comments in the output below were added only to clarify how a new value in the sequence was calculated and should not be printed by your function.)

```
In [1]: collatz(3)
3        # original n
10       # 3 times 3 plus 1
5        # half of 10
16       # 3 times 5 plus 1
8        # half of 16
4        # half of 8
2        # half of 4
1        # done
```

Recall that we can use the remainder operator, %, to determine whether or not an integer is even. As well, note that halving the even values is best done using //, the integer division operator.

Make sure that your function prints the correct sequence for n equal to 3, as in the example above. Work through at least one other value of n by hand to and confirm that your function works for that value as well.

Find a value for n (that is less than 100) where the length of the sequence, (i.e. the number of steps to reach 1), is more than 50.

Try to find the smallest value for n where the sequence length is more than 100.

If you feel you have to, you can find these values for n by brute force, counting the number of lines of output by hand each time you call the function. However, it's strongly suggested that it would be simpler to find them by modifying `collatz()` or writing additional functions. For instance, it may be helpful to count the number of steps as you print out the sequence and have `collatz()` return this value.

Submitting your work

Once you have completed your `collatz()` function, and tried to find the values of n described above, you are ready to submit your work.

Check that your file is named correctly, `lab7.py`, and that both your and your partner's names are included in the header comment.

Upload your file to the Lab 7 page on Moodle. Make sure that both partners will have access to the work you did today for when you each want to review the lab material.

Have one of the TAs in the lab come over and briefly evaluate your work.