New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Introduction to Python
Homework Discussion, Session 2

2.1 Count from 0 up to 100 by 2's (or 3's, or 4's). Our first step is to take user input to take a step value from the user (i.e., 2 for counting by 2's, etc.). A while True loop should be used to keep looping until the user's input is correct. Inside the loop we'll call input() and ask for an integer, then we'll use the string isdigit() method to test to see whether the input is all digits (i.e., an integer value). If isdigit() returns True, we can break out of the while loop. Otherwise program flow will continue, go back to the top of the while block, and we'll hit the input() line again.

You should begin by writing this part of the program and testing it thoroughly before moving on. Make sure that you're always getting an integer by trying out some non-digit input and seeing that the program continues to ask for input until all digits are received. Test this loop with numbers, letters, and also special characters.

I want to strongly recommend that you "keep it flat" (see the Code Quality document) by exiting the while loop at this point. Don't set up your logic so that you need to stay in the while loop for the rest of the program. The reason I recommend this is that this while loop is doing a particular thing (taking and validating using input) that is a different step than the counting loop. So when we've got the input we want and we know it's correct, we should drop out of this loop, return to the left margin, and be done with the first step -- and then proceed with the next step -- the counting loop. Otherwise the logic of the two steps gets confused, and so often do we.

The value from input() is a str, so you must convert it to an integer for it to work for us numerically. We use the int() function to do this. I also recommend that this be done before you start calculations. Just convert it once and you won't have to remember to convert it later. As with the while True step, you want to take care of this step and then move on, not have to think about doing it later.

Next, the counting loop. The most intuitive approach is to set the counter to 0 before the loop, and then inside the loop print the counter value, and then add the user's integer to it. This way when the while block ends and Python jumps back up to the top of the while loop to test that the counter is less than or equal to 100, it will be testing the incremented value, which it can then print.

In fact you can probably just start with a program that counts from 0 and up to 100 by 1's, and then replace the counter = counter + 1 line (whatever your count variable is named) with counter = counter + user_number (whatever you have named your user's variable).

Alternate approach: incremental loop with filter

Another approach is to loop through every digit between 0 and (up to) 100 and print only some of them. In other words, we aren't really counting by 2's or 3's, we're simply counting by 1's and printing only every 2nd, 3rd, etc. This solution would involve the modulus operator (%).

You can begin building this loop by simply creating a while loop that counts from 1 to 100 and prints out all the numbers in between. That's a good way to begin because it establishes the entirety of what the loop

does before the filtering (counting by 2's, etc.) is added. You should test this step before adding in the filtering test. Establish a counting variable (you can call it count) by assigning 0 to count. Then your while test should ask whether count has reached 100 or not; of course this means that we need to add 1 to count inside the loop so that it will eventually reach 100 and then exit.

Once you've thoroughly tested both your while True: loop for accepting an integer, as well as your while loop for counting from 1 to 100 and printing each one out, the final step will be to place the print count statement inside an if block, and have the if test check to see if this is one of the values to print.

How do we test to see which number to print? We have a step integer that we took from the user (in the first while loop). We can see if the number is evenly divisible into the count by using the modulus operator (%). See the slides or quicksheet for details on this operator.

Here is a basic outline of the program:

Pseudocode:

```
# establish a while True loop

    # take user input
    # if the input is all digits:
        # break
    # otherwise loop will automatically return to the top of the block

# convert user input to an integer

# set a count to 1

# establish another while loop, this one testing to see if count is less than or equal to 100

    # add the user input integer to count and assign back to count

    # (while loop automatically returns to top of loop)
```

2.2 Sum a sequence of integers. This program will also begin with a while True: loop to take user input. This time we will break out of the loop only if two conditions are True: that the input is all digits, and that the input value is greater than 0. Of course to test to see if the value from input() is > 0, we'll need to convert the user's input to int before testing. (In fact, the "all digits" test and the "greater than 0" test can be done in one if test, using the compound statement and.

As always, make sure to test this part before moving on to further functionality. You should make a habit of testing couple of lines that you write; please don't write 4 or 5 lines of code and then allow yourself to get confused by the errors.

Next, we have to consider the approach: we want to sum up all values from 0 to a given maximum value. How do we do this? Well, we definitely need to count from 0 to a particular value, so let's begin by doing that. Before the loop we'll set an integer count to 0. This while loop will be very similar to the counting loop from the previous homework assignment; the only difference is that our while test will check to see whether count is less than or equal to the user's input rather than 100.

So, write the while loop so that it counts from 0 to the user's value; you'll need to make sure the user's input is converted to an integer before the loop begins. Now test this by printing each value of count inside the loop; run the program several times with different values to see that this works correctly. It's imperative to test this part before moving on because later you'll only see the sum and it will be harder to see what's happening inside the loop. You could even keep this print statement in your program while you're testing so you can continue to have visibility. Later when you've confirmed the program is working properly, you can remove the print statement so the only output is the sum.

Next, we need to actually sum up each of the values of count. In other words, it will run like this: before the loop, set a "summing" integer (you can name it my_sum) to 0; also set another integer named count to 0; loop through the values of count, in which your while loop test to see if the count is still less than the user's value; inside the loop, add the value of count to my_sum; increment count; then allow the while loop to loop back. So in the first iteration of the loop my_sum will be 0; in the next it will be 1 (0+1); in the next it will be 3 (1+2); then 6 (3+3), 10 (6+4), etc.

After the loop is done, add a print statement that reports (prints) the value of the summing integer.

If you're not getting the same result as that shown in the sample program runs, you need to debug. First make sure that you understand the algorithm and how it is supposed to work. In other words, in your head work through the values of count and my_sum and what they are supposed to be in each iteration:

```
first iteration:  count is 0 and my_sum is 0
second iteration:  count is 1 and my_sum is 1
third iteration:  count is 2 and my_sum is 3
fourth iteration:  count is 3 and my_sum is 6
```

If this doesn't make sense, you should email me. One of my most important tasks is to reinforce in you the concept that you need to understand everything about how the program is supposed to work and what values the variables should hold at any given time. I realilze that this can be a challenge on the first try (and second and third tries), but this is part of the essence of learning how to code: modeling the program's dynamics in your head. This means that you literally have to keep in mind the value of the two variables and "watch" them change in your head as you "run" the program mentally.

Basic outline:

```
# establish a while True loop

    # take user input
    # if the input is all digits and int(input) > 0:
        # break
    # otherwise loop will automatically return to the top of the block

# convert user input to an integer

# set a counting variable to 0 (this will go up by 1's as the loop progresses)
# set a summing variable to 0 (this will receive the sum of integers (1 + 2 + 3 etc.)

# while counting variable <= the user's input variable:
    # add the count value to the summing value and assign back to the summing variable
    # add 1 to the counting variable

# print the summing value
```

2.3 Text replacement utility. The first thing to make clear is that the script can replace text in the sample text (stored as sample_text), or it can replace the text contained in any file (to be returned from the read_file() function), but that the file text replacement is optional and only intended to give you a sense of this program as a practical solution. You don't need to read from a file to complete this assignment; you only need to perform replacements on the text found sample_text. If you want to make it work with a file, enjoy - it can be fun to see your program work in a practical way.

We're going to need two calls to input() to get both the "search text" and the "replacement text". We should be able to use the str.replace() method to do the replacement. As far as the count goes, you'll need to count the number of occurrences of the "search text" in the original text, rather than the number of occurrences of the replacement text in the replaced text -- because what if the replacement text already existed in the original? Your count wouldn't be accurate.

Basic outline:

```
# take user input:  the text to be replaced
# take user input:  the replacement text

# caling replace() on the text on which to perform the replacement, pass the two user input values.
replace() returns the replaced text.

# call count() on the original text, passing the "text to be replaced" value to see how many
occurrences of that value exist in the original text.  This could be done before the
replacement action above, obviously, because we are searching on the original text.
```

2.4 Reverse Number Guessor. There isn't much more to this solution (once the random number is chosen) than a while True: loop that keeps asking the user to guess, and then using an if/elif/else compound block to select whether to report higher, lower or correct.

2.5 Number Guessor. The key here is keeping track of two numbers: the "maximum value" that the user's number could be (initially, 100) and the "minimum value" that the user's number could be (initially, 1). The guess is made halfway between the two numbers (initially, 50). Once the user identifies that his/her number is higher or lower than the guess, one of the two values can be changed to reflect this information. For example, if the program guesses 50 and the user indicates that his/her number is higher than 50, then the "lowest value" is changed to 50. Now the "halfway" guess will be halfway between 50 and 100.

So after the user hits enter for the first time, the program will enter a while True loop so it can allow the program to guess the number repeatedly. Inside the same loop the program also asks whether or not the number is correct and if not, whether the number is greater or lesser. An input of 'quit' can simply result in a call to exit().

The only other helpful feature is the round() function, which will round float division. This is probably important to do over regular integer division, which also results in an integer but might lead the system to the wrong number.

2.6 Find prime numbers in a range of numbers. This program is somewhat more complex than the others -- it requires that we employ a while loop inside a while loop. (A for loop over a range() works as well, but it is recommended to stick with this week's features, particularly while, so they can be thoroughly explored.)

After taking user input for the "max" integer, the "outer" while loop loops over every integer between 2 and the max. This while loop is identifying each "test" number to test for prime-ness.

Then inside this while loop, another while loop loops over the values of a second integer between 2 and the "test" number. The idea is that for any given "test" number, we need to check to see whether it is evenly divisible by any integer below it. In other words, for 3 we should check to see whether 2 divides evenly into it (we use the modulus operator for this purpose). For 4 we should check to see whether 2 or 3 divides evenly into it; for 5 we should check 2, 3 and 4. However, if any number divides evenly into the "test" number, we know that the number is not prime and should not be printed. But if we get all the way from 2 to the "test" number without seeing an even division, we know the number is prime and should be printed.

2.7 Chicken, Fox and Grain game. Using just the features previously learned, this solution came out to quite a few lines in my solution. (Features discussed in upcoming sessions would make this game much easier to code.)

The game must of course keep track of the position (west bank or east bank) of each of: the chicken, the fox, the grain, and the player. For each of these I chose to use an integer variable with a value of 1 or -1, basically because if I needed to "flip" the position of any of the actors, I needed only to multiply its value by -1.

A while True loop is used to allow repetitive "turns" by the player. During a turn, the following happens:

1. we are told the position of each of the actors
2. we are asked what we want to take in the boat for the next trip
3. we see an error if we ask to take an actor that is not currently on the same bank as we are
4. we change the position value of the selected actor, as well as the player's position (since the player goes back and forth with each turn)
5. we check to see whether any of the actors will die as a result of the new positions -- i.e., if the chicken and fox are together in a different position as the player, or the chicken and the grain are together in a different position as the player, the error is printed and the game ends

The calculations are of course simple (multiplying the selected actors' values by -1), and it requires only a series of compound tests to detect an error, if any. Finally the program must also be able to detect a successful result, i.e., all four actors' values should indicate their presence on the far bank.