## ***3.2 for loop***

*Counting Loop*

A while loop is a general purpose loop that can run either a specified number of times using a counting variable, or an unknown number of times, if it's controlled by some other condition. If the number of loop iterations is known, then a while loop is sometimes clunky, because the information needed to determine how many times it will run is located in three separate places in the code - where the counting variable is initialized, where the Boolean condition is checked, and where the counting variable is incremented.

The for loop provides a syntax where all of this information is provided within the same line, so that a programmer can easily see exactly how the loop will be controlled without scrolling all over the code.

The basic syntax of a for loop is as follows:

```
for <variable> in <range>:
    stmt1
    stmt2
    ...
    stmtn
```

A range can be specified in three different ways. The most simple way to specify a range is to provide a single number. For example, `range(10),` represents the set of integers {0, 1, 2, ..., 9}. In general, `range(n),` represents the set of integers {0, 1, 2, ..., n-1}.

The way this loop works is as follows:

The variable is set to the first number specified in the range. Then, the statements stmt1 through stmtn are executed. Next, the variable is set to the second number in the range and the statements stmt1 through stmtn are executed again. The loop continues in this manner until the variable has been set to each number in the range, in order, and all the statements in the loop have been run each time. Here is the most simple for loop:

```
for i in range(10):
    print("I love programming in Python!")
```

If we wanted a numbered list starting at 1, we could do the following:

```
for i in range(10):
    print((i+1),". I love programming in Python!", sep="")
```

Now, we can express our sum program succinctly as follows:

```
def main():

    MAX_TERM = 100

    sum = 0
    for count in range(MAX_TERM):
        sum = sum + (count + 1)

    print("The total is ",sum,".", sep="")

main()
```

All the key logic is the same here as the while loop version, but the code is more succinct, since the counting information is represented all on the same line, more compactly.

Obviously, it is a bit limiting that so far we have been forced to start all of our loops at 0. But, we can specify ranges that don't start at 0 by adding a second parameter. For example, the expression `range(a,b)`, represents the set of numbers {a, a+1, a+2, ..., b-1}. Thus, `range(2, 6)` represents the set of values {2, 3, 4, 5}. Notice that the first value is inclusive, meaning it's included in the range and the second value is exclusive, meaning it's not included in the range. While this seems strange, it's consistent with our intuitive notion that the number of values represented is `b - a`. In the given example, we have 6 - 2 = 4, and there are exactly 4 values in the set {2, 3, 4, 5}, but 5 values in the set {2, 3, 4, 5, 6}. Note that this specification is inconsistent with the specification of the `random.randint` method which allows both endpoints to be generated as random numbers. With this added flexibility in the range function, we can now write our sum program a bit more naturally, as follows:

```
def main():

    MAX_TERM = 100
    sum = 0
    for count in range(1, MAX_TERM+1):
        sum = sum + count

    print("The total is ",sum,".", sep="")

main()
```

Finally, consider the situation where we want to add the odd integers from 1 to 99, inclusive. In our previous solution, we simply added 2 to our counting variable, each time through the loop. The range function allows us to specify integers that are equally spaced out as well with the addition of a third parameter.

The function call `range(a, b, c)` specifies the numbers starting at a, with a step size of c, that are less than b. The step size simply specifies what to add to get to the next number. For example, `range(12, 30, 4)` represents the set {12, 16, 20, 24, 28} and `range(30, 40, 2)` represents the set {30, 32, 34, 36, 38}. Using this incarnation of the range function, we can very easily add the odd numbers less than 100:

```
def main():

    MAX_TERM = 100
    sum = 0
    for count in range(1, MAX_TERM, 2):
        sum = sum + count

    print("The total is ",sum,".", sep="")

main()
```

Now that we have specified the most general range function, we can write a short program that allows the user to enter these three parameters in their sum:

```
def main():

    sum = 0
    start = int(input("Please the starting integer."))
    end = int(input("Please the end integer."))
    step = int(input("What is your step?"))

    for i in list(range(start,end+1,step)):
        sum = sum + i

    print("Total is",sum)

main()
```

Note that because of how the range function works and how users will intuitively understand a range, we can adapt bu simply passing in end+1 to the range function, to ensure that the value end will be included in the sum.

*Diamond example*

Consider the problem of printing out a diamond. Let's define diamonds for the positive odd integers, where the integer repesents the number of rows and the number of stars on tin the middle row. Here are diamonds of size 1, 3, 5 and 7:

```
*          *          *          *
         ***        ***        ***
          *        *****      *****
                    ***     *******
                     *       *****
                              ***
                               *
```

The key here is that we realize that there isn't one single pattern in play. The first "half" of the design follows a simple pattern while the second half of the design follows a different pattern. Thus, we should separate our work into two loops. For example, if n is 7, we want our first loop to print the first four lines of the design and our second loop to print the last three lines of the design.

We further notice that the number of spaces decreases by 1 for the first half of the design and increases by 1 in the second half of the design. Secondly, the number of stars increases by 2 in the first half of the design and decreases by 2 in the second half of the design.

Though there are many ways to write this program, we'll use our for loop counting variable to represent the number of spaces on each line. Thus, the first loop will "count down" while the second loop will "count up." Also, we could create a formula for the number of stars on the basis of the number of spaces, but it's easier to just create a separate variable that stores the number of stars and update that each time in our loop accordingly.

In addition, we'll add a minimal bit of error checking in our program. If the user enters an even number, we'll simply tell them that we only make odd sized diamonds.

Note that we use integer division to ensure the accurate answers for the number of spaces and stars, since the range function needs integers to operate properly. Also note that the step in a range can be negative. This is useful for us since we want to go through the space values in backwards order for the top half of the design. When n = 7, we want our first line to have 3 spaces, our second line to have 2 spaces, our third line to have one space and our fourth line to have zero spaces.

A good exercise to solidify your understanding of loops would be to write this program without using the '*' operator for repeatedly printing out a string.

```
# Arup Guha
# 6/27/2012
# Prints a diamond of star characters.

def main():

    carats = int(input("How many carats is your diamond?\n"))

    if carats%2 == 0:
        print("Sorry, we only make odd carat diamonds.")
    else:

        spaces = carats//2
        stars = 1

        # Print top part of diamond here
        for spaces in range(carats//2, -1, -1):

            # Print out spaces number of spaces.
            print(" "*spaces + "*"*stars)
            stars = stars + 2

        # New starting values for rest of the sequence.
        stars = carats - 2

         # Print bottom part of the diamond.
        for spaces in range(1,carats//2+1):

            # Print out spaces number of spaces.
            print(" "*spaces + "*"*stars)
            stars = stars - 2

main()
```