**Chapter 4 – Repetition Structures**

1. Introduction to Repetition Structures – A repetition structure causes a statement or set of statements to execute repeatedly.
   1. Introduction
      1. Code that performs the same task over and over is common
      2. Ex: calculating commission for several salespeople
         1. Can be done by duplicating code repeatedly for each person
            1. Several disadvantages to doing this:

Makes program look large

Long sequences of code can be time consuming

If part of the duplicated code must be changed, then change must be manually reapplied to all to all parts

* 1. Condition-Controlled and Count-Controlled Loops
     1. Condition Controlled
        1. Uses true/false to control the number of times a loop repeats
        2. Use *while* to write the expression
     2. Count Controlled
        1. Repeats a specific number of times
        2. Use *if* to write the expression

1. The *while* Loop: A Condition-Controlled Loop – A condition controlled loop causes a statement or set of statements to repeat as long as a condition is true. In Python, you use the *while* statement to write a condition-controlled loop.
   1. Introduction
      1. While a condition is true, do a task
      2. Two parts
         1. A condition is tested for a true or false value
         2. A statement of set of statements is repeated as long as the condition is true
      3. Each execution if the loop is called an iteration
   2. The *while* Loop Is a Pretest Loop
      1. Pretest loop tests the condition before performing an iteration
      2. Usually have to perform some steps prior to the loop to make sure that the loop executes at least once
   3. Infinite Loops
      1. Loops must contain a way to terminate within themselves
      2. Loops that don’t have a way of ending are called infinite loops
      3. Continues to repeat until the program is interrupted
      4. Usually occurs when the programmer forgets to write code to stop loop
2. The *for* Loop: A Count-Controlled Loop – A count-controlled loop iterates a specific number of times. In Python, you use the *for* statement to write a count-control loop.
   1. Introduction
      1. Iterates a specific number of times
      2. Commonly used in programs
   2. Using the *range* Function with the *for* Loop
      1. Built into Python to simplify writing count-controlled loops
      2. Range function creates an iterable
         1. An object similar to a list
         2. Contains values that can be iterated with a loop
   3. Using the Target Variable Inside the Loop
      1. Purpose of the target variable is to reference each item in a sequence of items as the loop iterates
      2. Helpful to use target variable in a calculation or other task within a loop
      3. Program 4-8, pg. 173 shows how to use it to calculate and displays squares on 1-10
   4. Letting the User Control Loop Iterations
      1. Usually programmer knows the exact number of iterations needed
      2. Sometimes the user needs to control the number of iterations
         1. Program 4-10, pg. 176 shows how this can be done
   5. Generating an Iterable Sequence that Ranges from Highest to Lowest
      1. Use -1 as interval and switch starting and ending numbers to go from highest to lowest
3. Calculating a Running Total – A running total is a sum of numbers that accumulates with each iteration of a loop. The variable used to keep the running total is called an accumulator.
   1. Introduction
      1. Programs that calculate the total of a series of numbers use two elements:
         1. A loop that reads each number
         2. A variable that accumulates the total of the numbers read
      2. *Accumulator*
         1. A variable that totals all the numbers
      3. The loop keeps a running total
         1. Totals all the numbers as they are read
   2. The Augmented Assignment Operators
      1. X=X+1
      2. Table 4-2, pg. 181 shows augmented operators
      3. X=X+1 can be rewritten as X+=1 and still functions the same
4. Sentinels – A sentinel is a special value that marks the end of a sequence of values.
   1. Introduction
      1. Asking to continue at the end of the loops can be cumbersome for user
      2. Asking at the beginning is difficult
         1. Sometimes user isn’t sure how many they will need to input
      3. Adding a sentinel is a better solution
         1. Marks the end of a sequence and terminates loop on its own
      4. Program 4-13, pg. 184
5. Input Validation Loops – Input validation is the process of inspecting data that has been input to a program, to make sure it is valid before it is used in a computation. Input validation is commonly done with a loop that iterates as long as an input variable references bad data.
   1. Introduction
      1. Keeps bad data from being entered
         1. Ex: 400 hours input instead of 40
6. Nested Loops – A loop that is inside another loop is called a nested loop.
   1. Introduction
      1. Loop inside another loop
      2. Clock is a good example
      3. An inner loop goes through all iterations for every iteration of an outer loop
      4. Inner loops complete their iterations faster than outer loops
      5. To get the total number of iterations of a nested loop
         1. Multiply the number of iterations of all loops