Loop Patterns

• Validation Loops
• Item Processing
  – Sentinels
• Running Totals
  – Accumulation variables
• Count Controlled Loops
Introduction to Repetition Structures

- Often have to write code that performs the same task multiple times
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places

- Repetition structure: makes computer repeat included code as necessary
  - Includes condition-controlled loops and count-controlled loops
The while Loop: a Condition-Controlled Loop

- **while loop**: while condition is true, do something
  - Two parts:
    - Condition tested for true or false value
    - Statements repeated as long as condition is true
  - In flow chart, line goes back to previous part
  - General format:
    ```
    while condition:
    statements
    ```
The while Loop: a Condition-Controlled Loop (cont’d.)

Figure 5-1  The logic of a while loop
The while Loop: a Condition-Controlled Loop (cont’d.)

• In order for a loop to stop executing, something has to happen inside the loop to make the condition false

• **Iteration**: one execution of the body of a loop

• **while** loop is known as a *pretest* loop
  – Tests condition before performing an iteration
    • Will never execute if condition is false to start with
    • Requires performing some steps prior to the loop
Infinite Loops

• Loops must contain within themselves a way to terminate
  – Something inside a while loop must eventually make the condition false

• Infinite loop: loop that does not have a way of stopping
  – Repeats until program is interrupted
  – Occurs when programmer forgets to include stopping code in the loop
Input Validation Loops

• Computer cannot tell the difference between good data and bad data
  - If user provides bad input, program will produce bad output
  - GIGO: garbage in, garbage out
  - It is important to design program such that bad input is never accepted
Input Validation Loops (cont’d.)

• **Input validation**: inspecting input before it is processed by the program
  – If input is invalid, prompt user to enter correct data
  – Commonly accomplished using a `while` loop which repeats as long as the input is bad
    • If input is bad, display error message and receive another set of data
    • If input is good, continue to process the input
Sentinels

- **Sentinel**: special value that marks the end of a sequence of items
  - When program reaches a sentinel, it knows that the end of the sequence of items was reached, and the loop terminates
  - Must be distinctive enough so as not to be mistaken for a regular value in the sequence
  - Example: when reading an input file, empty line can be used as a sentinel
Calculating a Running Total

- Programs often need to calculate a total of a series of numbers
  - Typically include two elements:
    - A loop that reads each number in series
    - An accumulator variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total
The Augmented Assignment Operators

• In many assignment statements, the variable on the left side of the \( = \) operator also appears on the right side of the \( = \) operator

• Augmented assignment operators: special set of operators designed for this type of job
  – Shorthand operators
The Augmented Assignment Operators (cont’d.)

**Table 5-2** Augmented assignment operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example Usage</th>
<th>Equivalent To</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += 5</td>
<td>x = x + 5</td>
</tr>
<tr>
<td>-=</td>
<td>y -= 2</td>
<td>y = y - 2</td>
</tr>
<tr>
<td>*=</td>
<td>z *= 10</td>
<td>z = z * 10</td>
</tr>
<tr>
<td>/=</td>
<td>a /= b</td>
<td>a = a / b</td>
</tr>
<tr>
<td>%=</td>
<td>c %= 3</td>
<td>c = c % 3</td>
</tr>
</tbody>
</table>
The for Loop: a Count-Controlled Loop

- **Count-Controlled loop**: iterates a specific number of times
  - Use a `for` statement to write count-controlled loop
    - Designed to work with sequence of data items
      -Iterates once for each item in the sequence
    - General format:
      ```
      for variable in [val1, val2, etc]:
        statements
      ```
    - **Target variable**: the variable which is the target of the assignment at the beginning of each iteration
Figure 5-5  The for loop

1st iteration:  
for num in [1, 2, 3, 4, 5]:  
  print(num)

2nd iteration:  
for num in [1, 2, 3, 4, 5]:  
  print(num)

3rd iteration:  
for num in [1, 2, 3, 4, 5]:  
  print(num)

4th iteration:  
for num in [1, 2, 3, 4, 5]:  
  print(num)

5th iteration:  
for num in [1, 2, 3, 4, 5]:  
  print(num)
Generating an Iterable Sequence that Ranges from Highest to Lowest

• The `range` function can be used to generate a sequence with numbers in descending order
  – Make sure starting number is larger than end limit, and step value is negative
  – Example: `range (10, 0, -1)`
Using the `range` Function with the `for` Loop

• The `range` function simplifies the process of writing a `for` loop
  – `range` returns an iterable object
    • `Iterable`: contains a sequence of values that can be iterated over

• `range` characteristics:
  – One argument: used as ending limit
  – Two arguments: starting value and ending limit
  – Three arguments: third argument is step value
Using the Target Variable
Inside the Loop

• Purpose of target variable is to reference each item in a sequence as the loop iterates

• Target variable can be used in calculations or tasks in the body of the loop
  – Example: calculate square root of each number in a range