8. Hand-trace the following code, assuming that a is 2 and n is 4. Then explain what the code does for arbitrary values of a and n.

```
r = 1
i = 1
while i <= n :
   r = r * a
   i = i + 1
```

9. Hand-trace the following code. What error do you observe?

```
n = 1
while n != 50 :
   print(n)
   n = n + 10
```

10. The following pseudocode is intended to count the number of digits in the number *n*:

```
count = 1
temp = n
while temp > 10
   Increment count.
   Divide temp by 10.0.
```

Hand-trace the pseudocode for *n* = 123 and *n* = 100. What error do you find?

Practice It Now you can try these exercises at the end of the chapter: R4.3, R4.6.

4.3 Application: Processing Sentinel Values

In this section, you will learn how to write loops that read and process a sequence of input values.

Whenever you read a sequence of inputs, you need to have some method of indicating the end of the sequence. Sometimes you are lucky and no input value can be zero. Then you can prompt the user to keep entering numbers, or 0 to finish the sequence. If zero is allowed but negative numbers are not, you can use –1 to indicate termination.

Such a value, which is not an actual input, but serves as a signal for termination, is called a **sentinel**.

A sentinel value denotes the end of a data set, but it is not part of the data.

Let's put this technique to work in a program that computes the average of a set of salary values. In our sample program, we will use any negative value as the sentinel. An employee would surely not work for a negative salary, but there may be volunteers who work for free.

Inside the loop, we read an input. If the input is non-negative, we process it. In order to compute the average, we need the total sum of all salaries, and the number of inputs.

In the military, a sentinel guards a border or passage. In computer science, a sentinel value denotes the end of an input sequence or the border between input sequences.

```
while . . . :
   salary = float(input("Enter a salary or -1 to finish: "))
   if salary >= 0.0 :
      total = total + salary
      count = count + 1
```

Any negative number can end the loop, but we prompt for a sentinel of –1 so that the user need not ponder which negative number to enter. Note that we stay in the loop while the sentinel value is *not* detected.

```
while salary >= 0.0 :
   . . .
```

There is just one problem: When the loop is entered for the first time, no data value has been read. We must make sure to initialize `salary` with a value that will satisfy the `while` loop condition so that the loop will be executed at least once.

```
salary = 0.0    # Any non-negative value will do.
```

After the loop has finished, we compute and print the average.

Here is the complete program:

ch04/sentinel.py

```
1  ##
2  #  This program prints the average of salary values that are terminated with a sentinel.
3  #
4
5  # Initialize variables to maintain the running total and count.
6  total = 0.0
7  count = 0
8
9  # Initialize salary to any non-sentinel value.
10 salary = 0.0
11
12 # Process data until the sentinel is entered.
13 while salary >= 0.0 :
14    salary = float(input("Enter a salary or -1 to finish: "))
15    if salary >= 0.0 :
16       total = total + salary
17       count = count + 1
18
19 # Compute and print the average salary.
20 if count > 0 :
21    average = total / count
22    print("Average salary is", average)
23 else :
24    print("No data was entered.")
```

Program Run

```
Enter a salary or -1 to finish: 10000
Enter a salary or -1 to finish: 10000
Enter a salary or -1 to finish: 40000
Enter a salary or -1 to finish: -1
Average salary is 20000.0
```

A pair of input operations, known as the priming and modification reads, can be used to read a sentinel-terminated sequence of values.

Some programmers don't like the "trick" of initializing the input variable with a value other than a sentinel. Although it solves the problem, it requires the use of an `if` statement in the body of the loop to test for the sentinel value. Another approach is to use two input statements, one before the loop to obtain the first value and another at the bottom of the loop to read additional values:

```
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0.0 :
   total = total + salary
   count = count + 1
   salary = float(input("Enter a salary or -1 to finish: "))
```

If the first value entered by the user is the sentinel, then the body of the loop is never executed. Otherwise, the value is processed just as it was in the earlier version of the loop. The input operation before the loop is known as the *priming read*, because it prepares or initializes the loop variable.

The input operation at the bottom of the loop is used to obtain the next input. It is known as the *modification read*, because it modifies the loop variable inside the loop. Note that this is the last statement to be executed before the next iteration of the loop. If the user enters the sentinel value, then the loop terminates. Otherwise, the loop continues, processing the input.

Special Topic 4.1 shows a third approach for processing sentinel values that uses a Boolean variable.

Now consider the case in which any number (positive, negative, or zero) can be an acceptable input. In such a situation, you must use a sentinel that is not a number (such as the letter Q).

Because the `input` function obtains data from the user and returns it as a string, you can examine the string to see if the user entered the letter Q before converting the string to a numeric value for use in the calculations:

```
inputStr = input("Enter a value or Q to quit: ")
while inputStr != "Q" :
   value = float(inputStr)
   Process value.
   inputStr = input("Enter a value or Q to quit: ")
```

Note that the conversion to a floating-point value is performed as the first statement within the loop. By including it as the first statement, it handles the input string for both the priming read and the modification read.

Finally, consider the case where you prompt for multiple strings, for example, a sequence of names. We still need a sentinel to flag the end of the data extraction. Using a string such as Q is not such a good idea because that might be a valid input. You can use the empty string instead. When a user presses the Enter key without pressing any other keys, the `input` function returns the empty string:

```
name = input("Enter a name or press the Enter key to quit: ")
while name != "" :
   Process name.
   inputStr = input("Enter a name or press the Enter key to quit: ")
```

SELF CHECK

11. What does the `sentinel.py` program print when the user immediately types –1 when prompted for a value?

12. Why does the `sentinel.py` program have *two* checks of the form

```
salary >= 0
```

13. What would happen if the initialization of the `salary` variable in `sentinel.py` was changed to

```
salary = -1
```

14. In the second example of this section, we prompt the user "Enter a value or Q to quit." What happens when the user enters a different letter?

Practice It Now you can try these exercises at the end of the chapter: R4.14, P4.28, P4.29.

Special Topic 4.1

Processing Sentinel Values with a Boolean Variable

Sentinel values can also be processed using a Boolean variable for the loop termination:

```
done = False
while not done :
   value = float(input("Enter a salary or -1 to finish: "))
   if value < 0.0 :
      done = True
   else :
      Process value.
```

The actual test for loop termination is in the middle of the loop, not at the top. This is called a **loop and a half** because one must go halfway into the loop before knowing whether one needs to terminate. As an alternative, you can use the `break` statement:

```
while True :
   value = float(input("Enter a salary or -1 to finish: "))
   if value < 0.0 :
      break
   Process value.
```

The `break` statement breaks out of the enclosing loop, independent of the loop condition. When the `break` statement is encountered, the loop is terminated, and the statement following the loop is executed.

In the loop-and-a-half case, `break` statements can be beneficial. But it is difficult to lay down clear rules as to when they are safe and when they should be avoided. We do not use the `break` statement in this book.

Special Topic 4.2

Redirection of Input and Output

Use input redirection to read input from a file. Use output redirection to capture program output in a file.

Consider the `sentinel.py` program that computes the average value of an input sequence. If you use such a program, then it is quite likely that you already have the values in a file, and it seems a shame that you have to type them all in again. The command line interface of your operating system provides a way to link a file to the input of a program, as if all the characters in the file had actually been typed by a user. If you type

```
python sentinel.py < numbers.txt
```

the program is executed, but it no longer expects input from the keyboard. All input commands get their input from the file `numbers.txt`. This process is called *input redirection*.

Input redirection is an excellent tool for testing programs. When you develop a program and fix its bugs, it is boring to keep entering the same input every time you run the program. Spend a few minutes putting the inputs into a file, and use redirection.

You can also redirect output. In this program, that is not terribly useful. If you run

```
python sentinel.py < numbers.txt > output.txt
```

the file `output.txt` contains the input prompts and the output, such as

```
Enter a salary or -1 to finish:
Enter a salary or -1 to finish:
Enter a salary or -1 to finish:
Enter a salary or -1 to finish:
Average salary is 15
```

However, redirecting output is obviously useful for programs that produce lots of output. You can format or print the file containing the output.

4.4 Problem Solving: Storyboards

A storyboard consists of annotated sketches for each step in an action sequence.

When you design a program that interacts with a user, you need to make a plan for that interaction. What information does the user provide, and in which order? What information will your program display, and in which format? What should happen when there is an error? When does the program quit?

This planning is similar to the development of a movie or a computer game, where *storyboards* are used to plan action sequences. A storyboard is made up of panels that show a sketch of each step. Annotations explain what is happening and note any special situations. Storyboards are also used to develop software—see Figure 3.

Making a storyboard is very helpful when you begin designing a program. You need to ask yourself which information you need in order to compute the answers that the program user wants. You need to decide how to present those answers. These

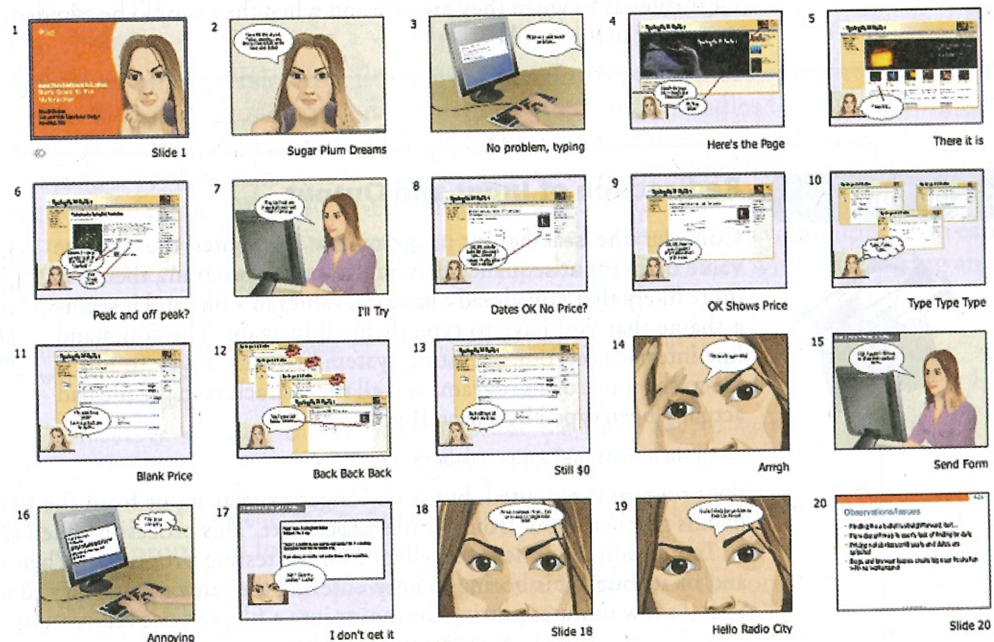

Figure 3 Storyboard for the Design of a Web Application