4.2 Problem Solving: Hand-Tracing

Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.

In Programming Tip 3.2, you learned about the method of hand-tracing. When you hand-trace code or pseudocode, you write the names of the variables on a sheet of paper, mentally execute each step of the code, and update the variables.

It is best to have the code written or printed on a sheet of paper. Use a marker, such as a paper clip, to mark the current line. Whenever a variable changes, cross out the old value and write the new value below. When a program produces output, also write down the output in another column.

Consider this example. What value is displayed?

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

There are three variables: `n`, `total`, and `digit`.

n	total	digit

The first two variables are initialized with 1729 and 0 before the loop is entered.

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

n	total	digit
1729	0	

Because `n` is greater than zero, enter the loop. The variable `digit` is set to 9 (the remainder of dividing 1729 by 10). The variable `total` is set to 0 + 9 = 9.

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

n	total	digit
1729	~~0~~	
	9	9

Finally, `n` becomes 172. (Recall that the remainder in the division 1729 // 10 is discarded because the // operator performs floor division.)

Cross out the old values and write the new ones under the old ones.

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

n	total	digit
~~1729~~	~~0~~	
172	9	9

Now check the loop condition again.

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

Because `n` is still greater than zero, repeat the loop. Now `digit` becomes 2, `total` is set to 9 + 2 = 11, and `n` is set to 17.

n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
17	11	2

Repeat the loop once again, setting `digit` to 7, `total` to 11 + 7 = 18, and `n` to 1.

n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
1	18	7

Enter the loop for one last time. Now `digit` is set to 1, `total` to 19, and `n` becomes zero.

n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

Because n equals zero, this condition is not true.

The condition `n > 0` is now false. Continue with the statement after the loop.

```
n = 1729
total = 0
while n > 0 :
   digit = n % 10
   total = total + digit
   n = n // 10

print(total)
```

n	total	digit	output
~~1729~~	~~0~~		
~~172~~	~~9~~	~~9~~	
~~17~~	~~11~~	~~2~~	
~~1~~	~~18~~	~~7~~	
0	19	1	19

This statement is an output statement. The value that is output is the value of `total`, which is 19.

Of course, you can get the same answer by just running the code. However, hand-tracing can give you *insight* that you would not get if you simply ran the code. Consider again what happens in each iteration:

- We extract the last digit of `n`.
- We add that digit to `total`.
- We strip the digit off of `n`.

Hand-tracing can help you understand how an unfamiliar algorithm works.

In other words, the loop computes the sum of the digits in `n`. You now know what the loop does for any value of `n`, not just the one in the example. (Why would anyone want to compute the sum of the digits? Operations of this kind are useful for checking the validity of credit card numbers and other forms of ID numbers—see Exercise P4.33.)

Hand-tracing can show errors in code or pseudocode.

Hand-tracing does not just help you understand code that works correctly. It is a powerful technique for finding errors in your code. When a program behaves in a way that you don't expect, get out a sheet of paper and track the values of the variables as you mentally step through the code.

You don't need a working program to do hand-tracing. You can hand-trace pseudocode. In fact, it is an excellent idea to hand-trace your pseudocode before you go to the trouble of translating it into actual code, to confirm that it works correctly.

6. Hand-trace the following code, showing the value of `n` and the output.

```
n = 5
while n >= 0 :
   n = n - 1
   print(n)
```

7. Hand-trace the following code, showing the value of `n` and the output.

```
n = 1
while n <= 3 :
   print(n)
   n = n + 1
```

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.