CHAPTER 5:
LOOP STRUCTURES

While Loops

A loop performs an iteration or repetition
A while loop is the simplest form of a loop
  Occurs when a condition is true

Figure 5.1:

```ruby
1 while (condition)
  2  # statement 1
  3  # statement 2
  4  # ...
  5  # statement n
  6 end
```

Control flow enters the while loop at the instructions while (condition)
The condition is evaluated
  Each of the statements within the loop are executed if the condition is true
  Otherwise, the control flow skips the loop
The while (condition) is reevaluated after the control flow reaches the end
Control flow will repeat the loop from the first to the last statement if the condition evaluates to true
This will continue until the condition evaluates to false
Infinite Loops

- Every while loop must lead to the condition eventually becoming false; otherwise, there will an infinite loop.
- An infinite loop is a loop that does not terminate.
- Small mistakes can cause infinite loops.

The program will not terminate.
To terminate the program, hold the control key (CTRL) and then press C.
This sequence means cancel.

Figure 5.6:
1 puts "Count up from 0 to ?"
2 n = gets.to_i
3 i = 5
4 while (i > 0) # always true
5   i = i + 2
6   # no provision to change the condition to false
7 end

The program will not terminate.
To terminate the program, hold the control key (CTRL) and then press C.
This sequence means cancel.

Figure 5.3
1 until (condition)
2 # statement 1
3 # statement 2
4 # ...
5 # statement n
6 end

Until Loops

- Until loops are the opposite of while loops.
- The until loop executes until a condition is true.
- In other words, until loops execute while a condition is false.
- Until loops execute similarly to while loops.
- Until loop conditionals map to logical opposites of while loop conditionals (Table 5.1).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Opposite Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>!=</td>
</tr>
<tr>
<td>&gt;</td>
<td>&lt;=</td>
</tr>
<tr>
<td>&lt;</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>
While and until loops are interchangeable. These two loops should be used based on clarity. Use “until this is true” instead of “while this is not true” and vice versa.

Based of Figures 5.1 and 5.3

- While and until loops are interchangeable
- These two loops should be used based on clarity
  - Use “until this is true” instead of “while this is not true” and vice versa

**Figure 5.5: For Loops and Nested Loops**

```
1 for i in 1..3
2   puts "Outer loop: i = " + i.to_s
3   for k in 1..4
4     puts "Inner loop: k = " + k.to_s
5   end # for k
6 end # for i
```

Note: Indentation and end labeling by comments [For clarity and documentation only]

**For Loops**
- Execute the statement or statements in the loop once for each iteration element

**Nested Loops**
- A loop inside a loop
- For loops are most commonly used in nested loops

**Figure 5.5: For Loops and Nested Loops**

The first loop will initialize, then the second will follow. The second loop will finish before the first goes on to the next iteration.
A prime number can only be divided by one and itself. This program determines whether a number is prime or not. Only numbers less than half the given value need to be checked. This reduces the number of possible loop iterations by half.

```ruby
# Initialize our counter
i = 1

# i: [0, 100]
while (i <= 100)
    # Initialize prime flag
    prime_flag = true
    j = 2

    # Test divisibility of i from [0, i/2]
    while (j <= i / 2)
        # puts " i ==> " + i.to_s + " j ==> " + j.to_s
        if (i % j == 0)
            prime_flag = false
            # break
            end
        j = j + 1
    end
    # We found a prime!
    if prime_flag
        puts "Prime ==> " + i.to_s
    end
    # Increment the counter
    i += 1
end
```

(c) 2012 Ophir Frieder et al
Example: Finding Prime Numbers

- Good debugging statements show the most information with the least output statements
- Having many statements can make it harder to find errors
- Sometimes, debugging statements are commented out or disabled by Boolean conditions, but not deleted
- Could be used later on for debugging or other code might accidentally get deleted with it

(c) 2012 Ophir Frieder et al

Example: Finding Prime Numbers

- The syntax for the shorthand notation can be used with "+", "-", "*", or "/" operators
- Stores the result of the operation in the variable used

- Output for the program: modified to check numbers up to 25
  - Prime 1
  - Prime 2
  - Prime 3
  - Prime 5
  - Prime 7
  - Prime 11
  - Prime 13
  - Prime 17
  - Prime 19
  - Prime 23

(c) 2012 Ophir Frieder et al
**Summary**

- **Loop structures** instruct the computer to repeat a set of steps until a condition is met.
- **While loops, until loops, and for loops** can be used to create a **loop structure**.
- **Nested loops** are loops within loops.