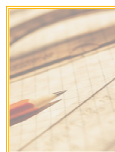
(c) 2012 Ophir Frieder et al

# CHAPTER 5:
## LOOP STRUCTURES

Introduction to Computer Science Using Ruby

## While Loops



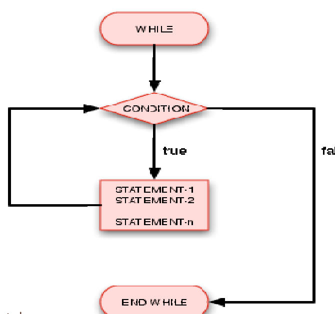
- A loop performs an **iteration** or **repetition**
- A while loop is the **simplest form** of a loop
  - Occurs when a condition is **true**

(c) 2012 Ophir Frieder et al

## While Loops

Figure 5.1:

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



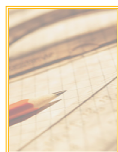

(c) 2012 Ophir Frieder et al

## While Loops

- Control flow enters the while loop at the instruction: **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**

(c) 2012 Ophir Frieder et al

## 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

(c) 2012 Ophir Frieder et al

## Infinite Loops

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**

(c) 2012 Ophir Frieder et al

## 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)

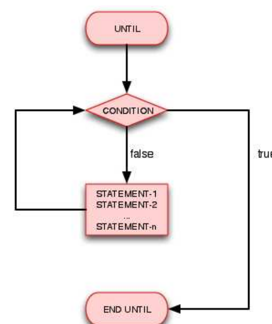
| Operator | Opposite Operator |
|---|---|
| == | != |
| > | <= |
| < | >= |

(c) 2012 Ophir Frieder et al

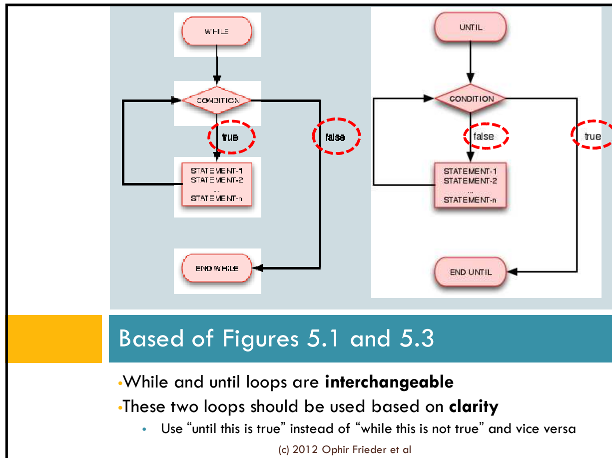
## Until Loops

Figure 5.3

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




(c) 2012 Ophir Frieder et al




## 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

(c) 2012 Ophir Frieder et al

## For Loops and Nested Loops

### For Loops

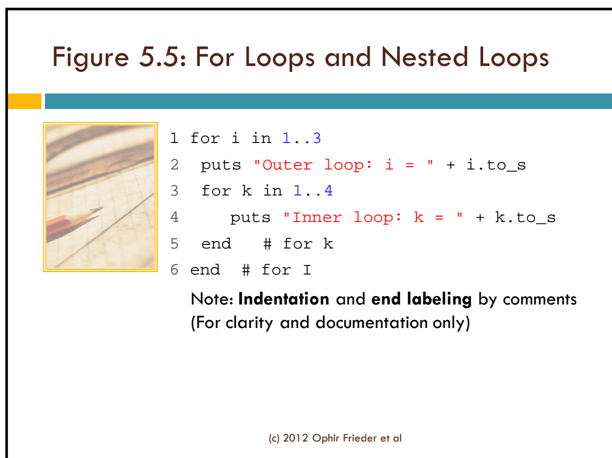
- Execute the statement or statements in the loop once for each iteration element
- Figure 5.4:

```
1 for num in 0..5
2 puts num
3 end
```

### Nested Loops

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

(c) 2012 Ophir Frieder et al

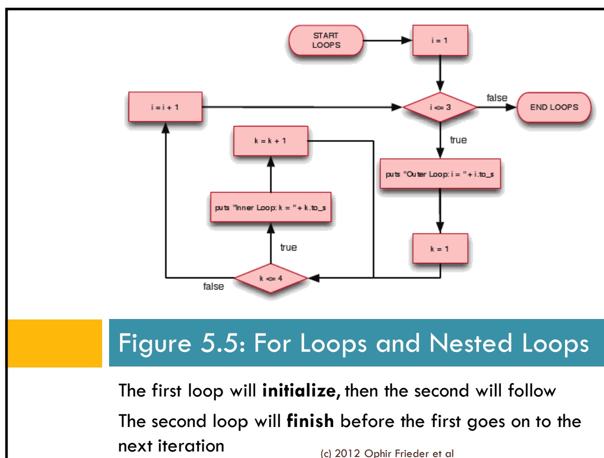


## Figure 5.5: For Loops and Nested Loops

```
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)

(c) 2012 Ophir Frieder et al



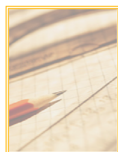

## Figure 5.5: For Loops and Nested Loops

The first loop will **initialize,** then the second will follow

The second loop will **finish** before the first goes on to the next iteration

(c) 2012 Ophir Frieder et al

## Example: Finding Prime Numbers



- 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

(c) 2012 Ophir Frieder et al

## Figure 5.7: Finding Prime Numbers

```
1 # Initialize our counter
2 i = 1
3
4 # i: [0, 100]
5 while (i <= 100)
6   # Initialize prime flag
7   prime_flag = true
8   j = 2
9   # Test divisibility of i from [0, i/2]
10  while (j <= i / 2)
11    # puts " i ==> " + i.to_s + " j ==> " + j.to_s
12    if (i % j == 0)
13      prime_flag = false
14      # break
15    end
16    j = j + 1
17  end
```

(c) 2012 Ophir Frieder et al

## Figure 5.7 Cont'd: Finding Prime Numbers

```
18  # We found a prime!
19  if prime_flag
20    puts "Prime ==> " + i.to_s
21  end
22  # Increment the counter
23  i += 1
24 end
```

(c) 2012 Ophir Frieder et al

```
1 # Initialize our counter
2 i = 1
3
4 # i: [0, 100]
5 while (i <= 100)
6   # Initialize prime flag
7   prime_flag = true
8   j = 2
9   # Test divisibility of
10  while (j <= i / 2)
11    # puts " i ==> " + i.to_s + " j ==> " +
j.to_s
12    if (i % j == 0)
13      prime_flag = false
14      # break
15    end
16    j = j + 1
17  end
```

Starts the outer loop for searching

Uncomment for debugging

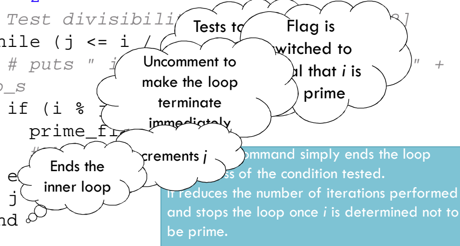
(c) 2012 Ophir Frieder et al

## 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

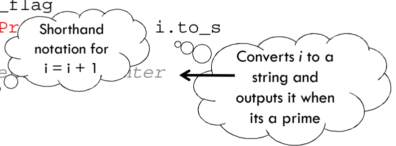
(c) 2012 Ophir Frieder et al

```
1 # Initialize our counter
2 i = 1
3
4 # i: [0, 100]
5 while (i <= 100)
6   # Initialize prime flag
7   prime_flag = true
8   j = 2
9   # Test divisibili
10  while (j <= i /
11    # puts " i                    " +
  j.to_s
12    if (i %
13      prime_fl
14      #
15    e
16    j
17  end
```




(c) 2012 Ophir Frieder et al

```
18   # We found a prime!
19   if prime_flag
20     puts "Pr              i.to_s
21   end
22   # Increme          ter
23   i += 1
24 end
```




(c) 2012 Ophir Frieder et al

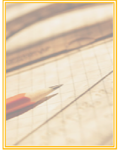
## Example: Finding Prime Numbers

- 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

(c) 2012 Ophir Frieder et al