Repetition Structures

Chapter 5

Introduction to Repetition Structures

Chapter 5.1

Introduction to Repetition Structures

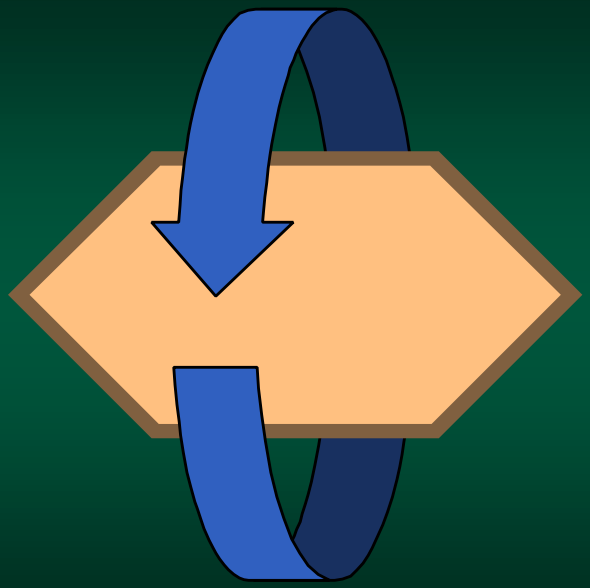
- A repetition structure causes a statement or set of statements to execute *repeatedly*
- It comes in several different forms – which are variations on the same concept

The Problems with Duplicate Code

- Duplicate code makes a program large
- Writing a long sequence of statements is time consuming
- If part of the duplicate code has to be corrected or changed, then the change has to be done many times

How Repetition Solves This

- The same code can be exited multiple times
- Benefits
  - duplicate code is eliminated – smaller size
  - less time consuming to write
  - if the looping code needs to be changed – it only had to be changed in <u>one</u> place

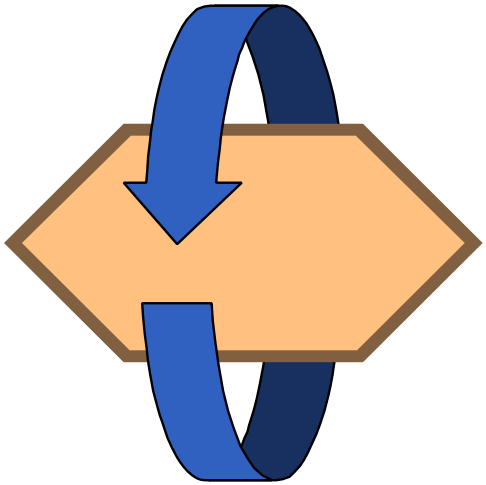


Condition-Controlled Loops

Chapter 5.2

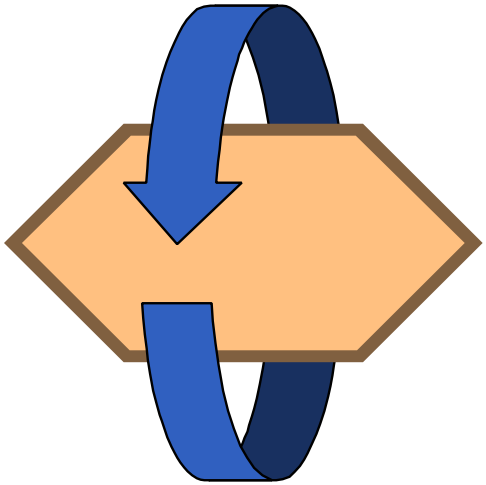
Condition-Controlled Loop

- Loops a group of statements
- Continues to execute…
  - each time the loop completes, the conditional expression is reexamined
  - if True, the loop continues
  - otherwise, it exits



Condition-Controlled Loop

- Think of it as an:
  If Statement that loops
- Be careful not to create *infinite loops* – always provide a way to break out
- Yes, you can do something *forever*



If Statement vs. Conditional Controlled Loop

If Statement	Conditional-Controlled Loop
Uses a conditional expression	Uses a conditional expression
Executes a group of statements	Executes a group of statements
Executes only once	Executes <u>multiple</u> times

Various Forms

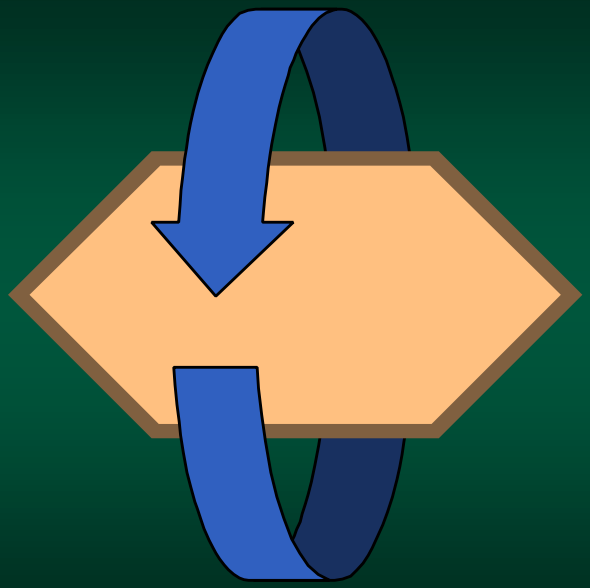
- There are different variations of the condition-controlled loop
- Each variation was created to make it easy on the programmer – so all this helps *you*
- Most languages contain two or three variations - although, you only need one

Common Forms

- While Loop
- Do-While Loop
- Do-Until Loop

Loop Caution

- With all loops, be careful not to create *infinite loops*
- These are loops that will run forever – the condition is always true
- Always provide a way to break out

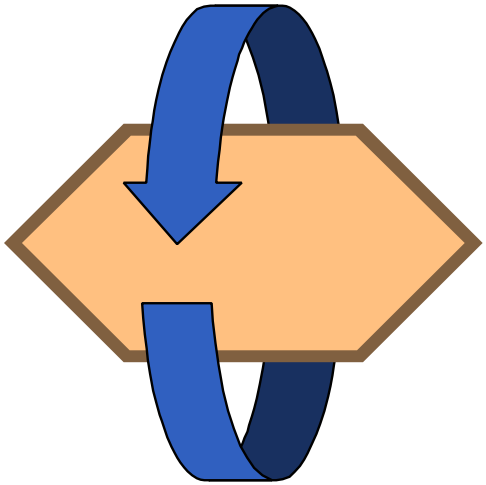


While Loops

Chapter 5.2

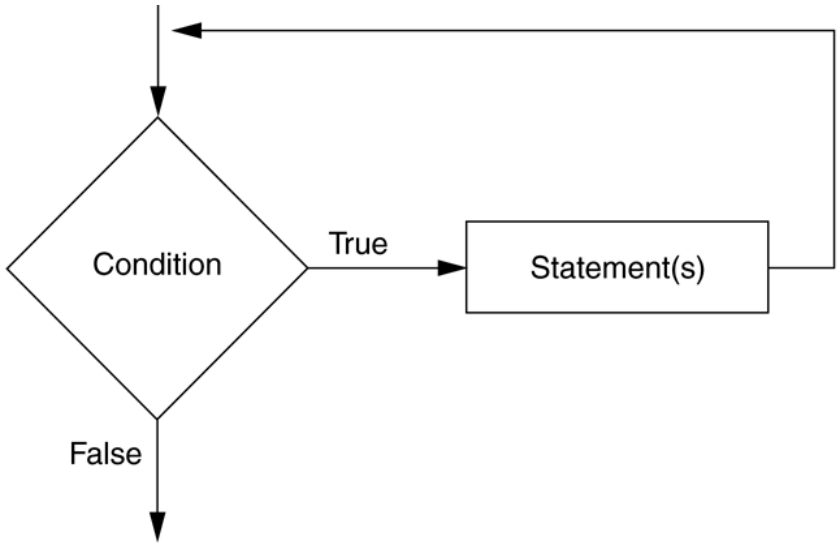
While Loops

- The *While Loop* is almost identical to the If Statement
- The structure is identical except for two properties
  - there is no "else" clause
  - it loops when it reaches the end

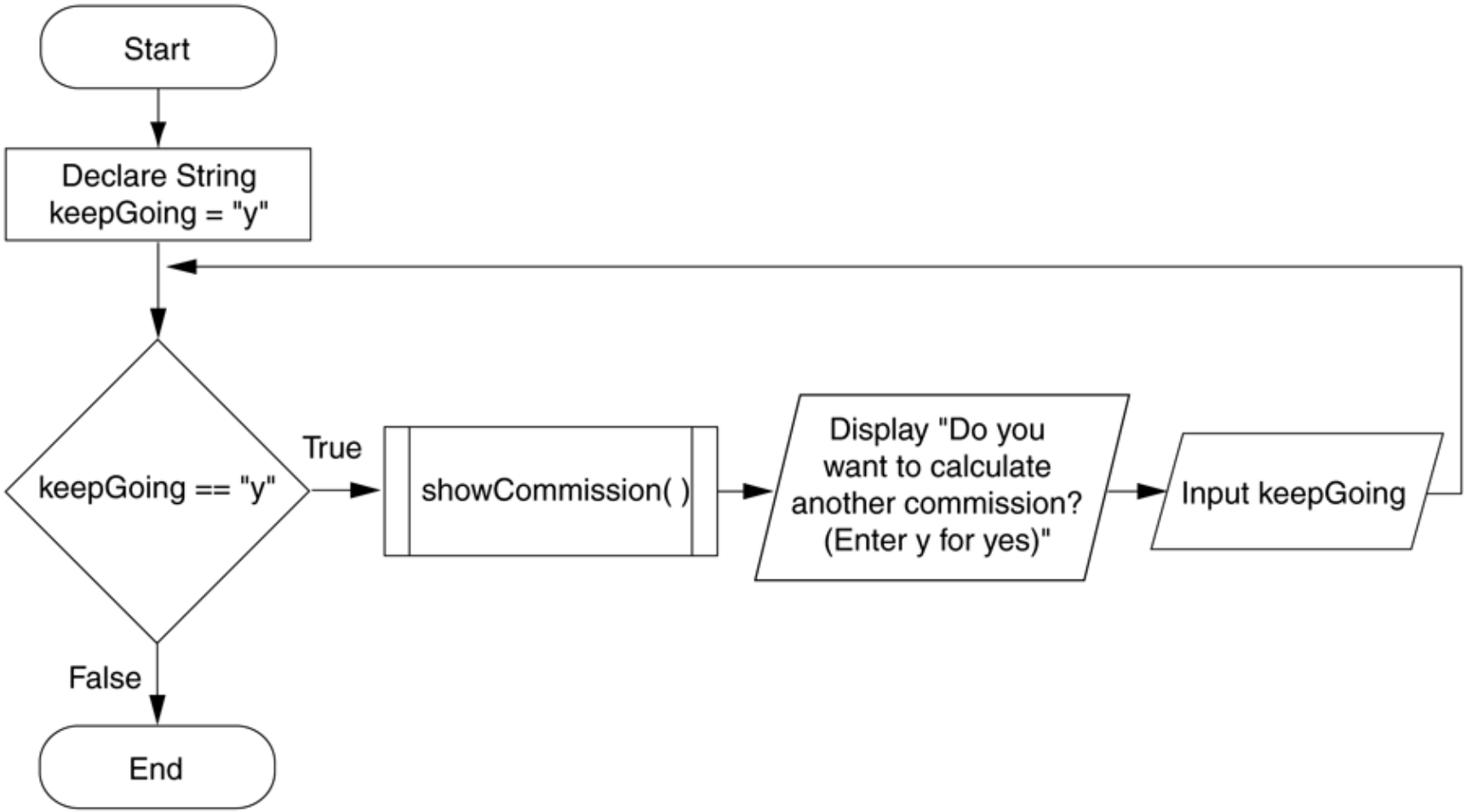


While is a "Pretest"

- While Loop is known as a *pretest loop*
- The conditional expression is checked before (pre) the statements are executed



Example While Loop (Book)



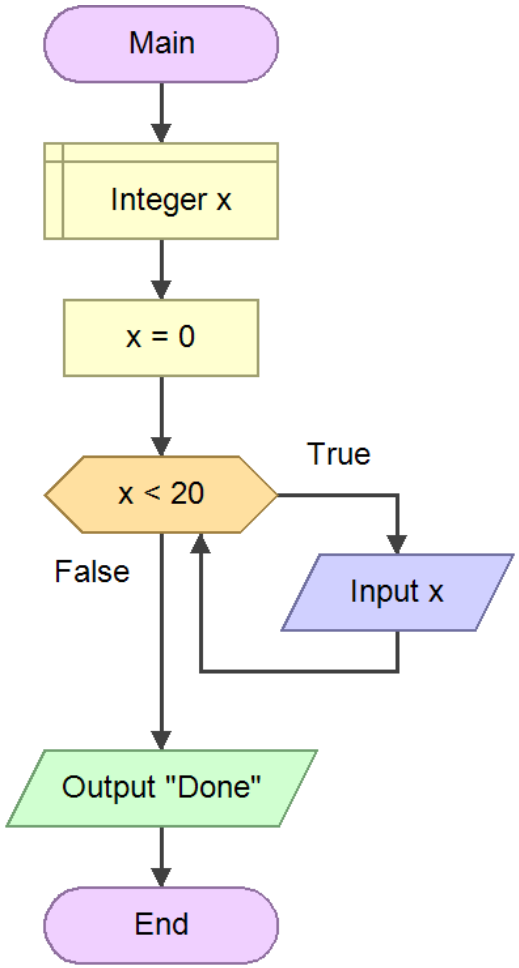
Book Pseudocode: While

```
while condition
    Statements
end while
```

While Loop Example

```
Set x = 0

While x < 20
    Input x
End While

Display "Done"
```

While Loop Example



While Loop Example Output

```
17
12
30
Done
```

While Loop Example 2

```
int x = 22

While x < 20
    Input x
End While

Display "Done"
```

Hmm...

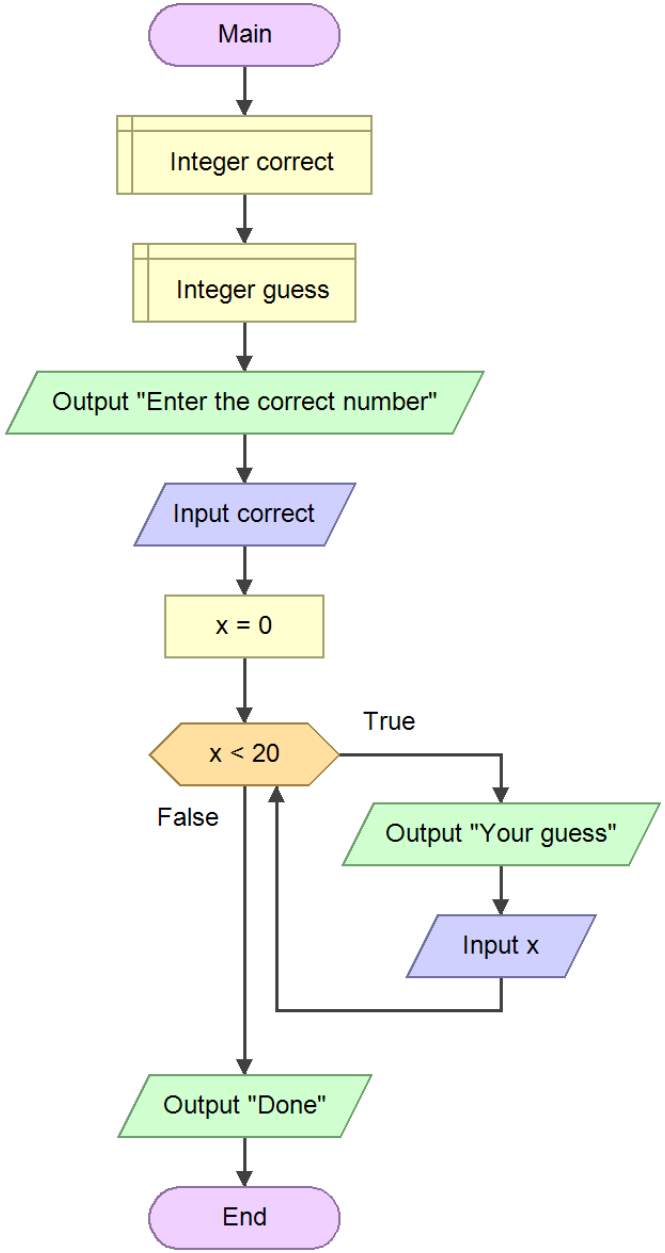
While Loop Example Output 2

```
Done
```

While block never executed

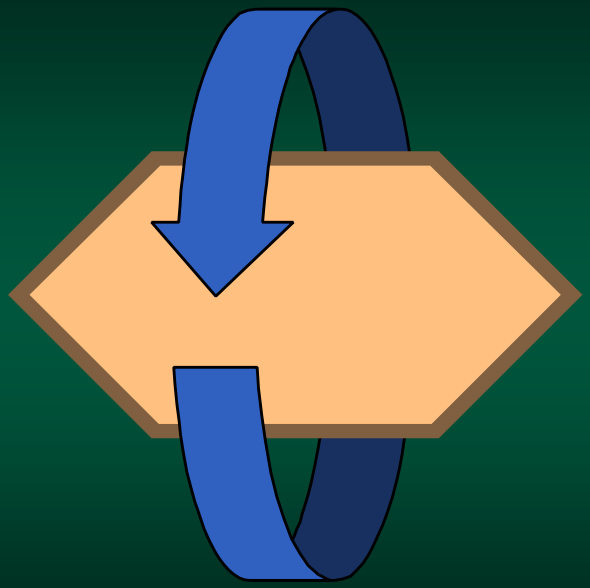
While Loop Example 3

```
Display "Correct: "
Input Correct

Set Guess = 0
While Guess != Correct
   Display "Your guess: "
   Input Guess
End While

Display "Correct!"
```


Main
Integer correct
Integer guess
Output "Enter the correct number"
Input correct
x = 0
x < 20
True
False
Output "Your guess"
Input x
Output "Done"
End

While Loop Example 3 Output

```
Correct: 55
Guess: 32
Guess: 60
Guess: 55
Done!
```

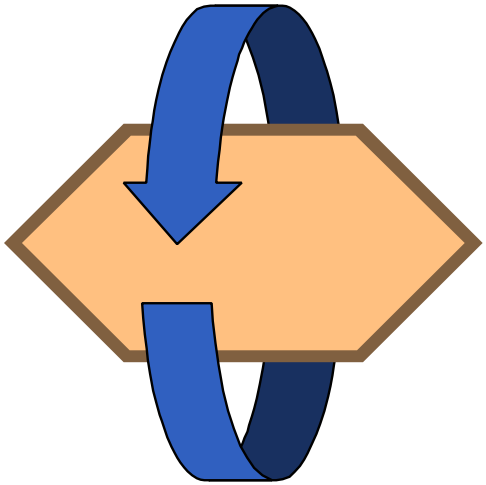


Do Loops

Chapter 5.2

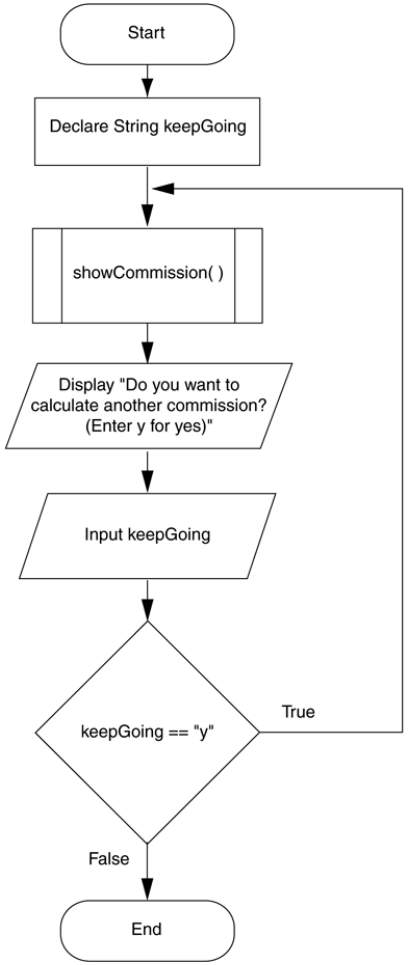
Do Loops

- The *Do Loop* is a variation of the While Loop
- The conditional expression is tested at the end of the loop rather than the beginning
- This takes it a *post-test* loop



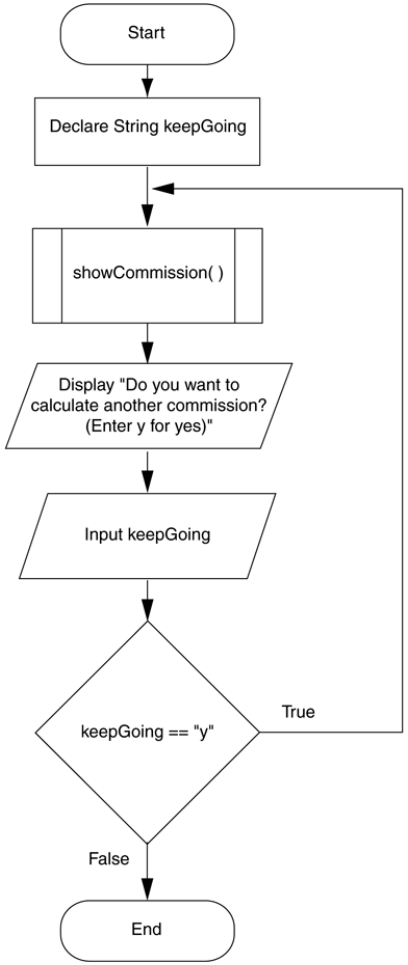
Do Loop

- Basically, the test is conducted after the group of statements
- So, its basically done last
- This minor change has a huge impact



Do Loop

- In a Do Loop, the block of statements is executed *at least* once
- A While Loop may not execute the block at all since the condition is first



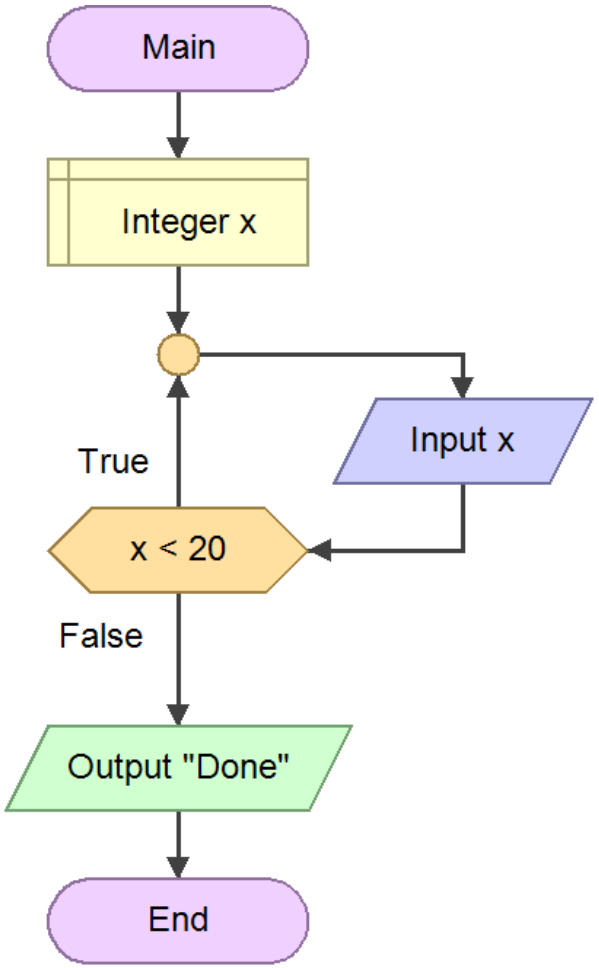
Book Pseudocode: While

```
do
    Statements
while condition
```

Do Loop Example

```
Do
    Input x
While x < 20

Display "Done"
```

Do Loop Example



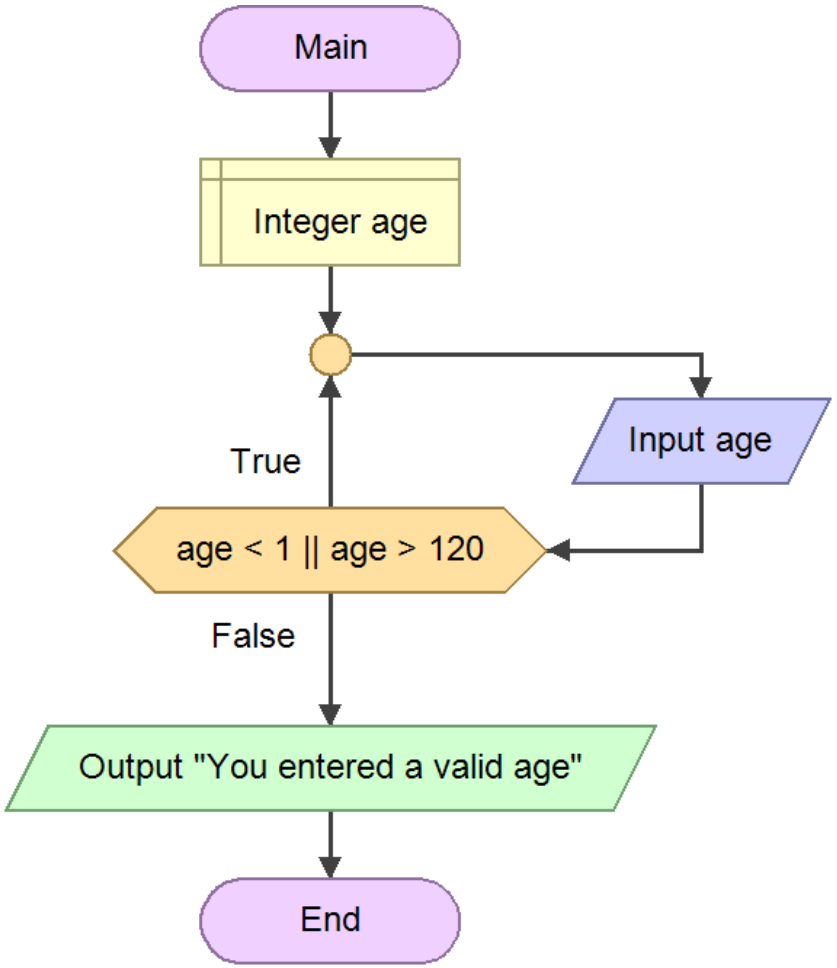
Do Loop Example Output

```
17
12
30
Done
```

Do Loop Example 2

```
Do
    Input age
While age < 1 or age > 120

Display "You entered a valid age"
```

Do Loop Example 2

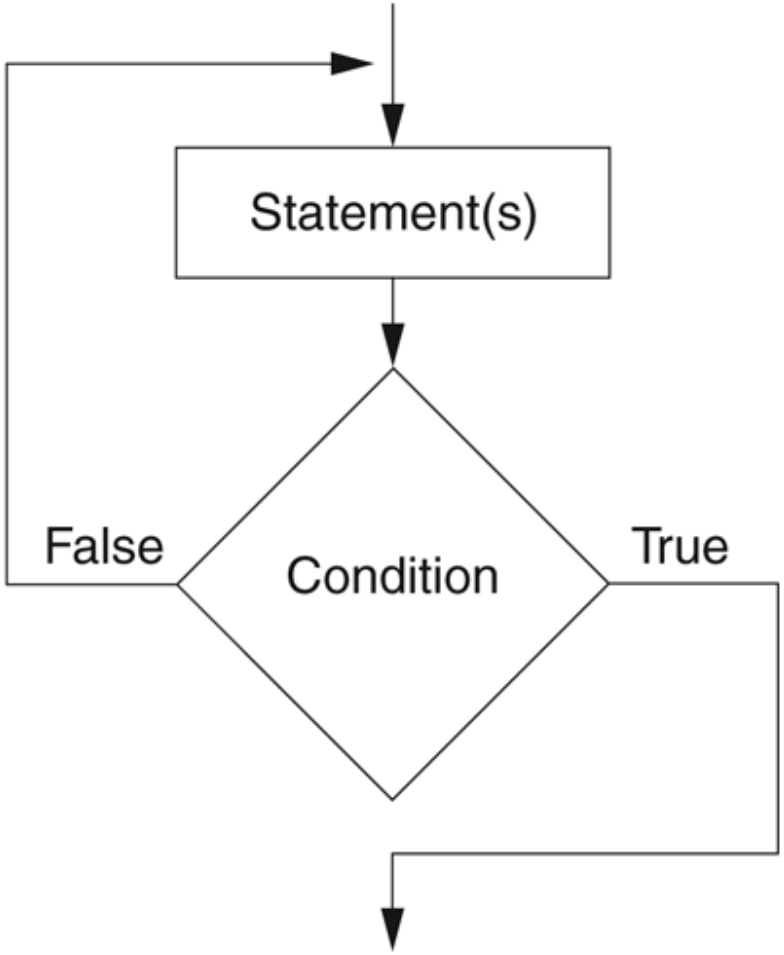


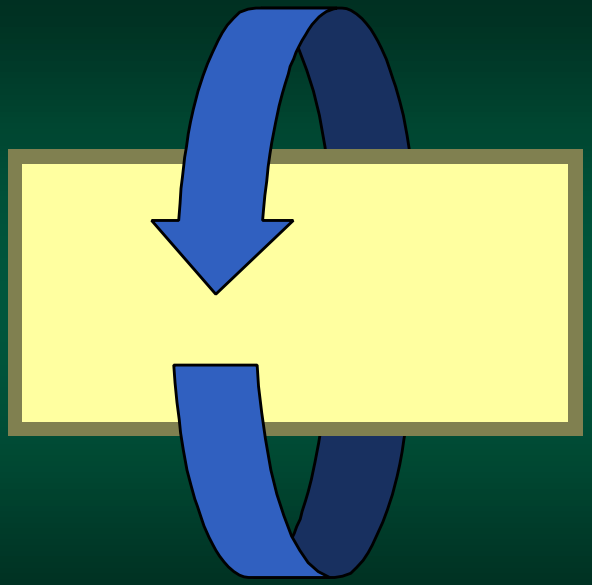
Do-While vs. Do-Until

- The Do-While and Do-Until loops are essential the same
- The only difference is how the condition is treated
- Do-While will loop if the condition is true
- Do-Until will loop if the condition is false

Do-Until is Not Needed

- Programming languages don't need both Do-While and Do-Until
- So, Do-While is usually the only one that is supported



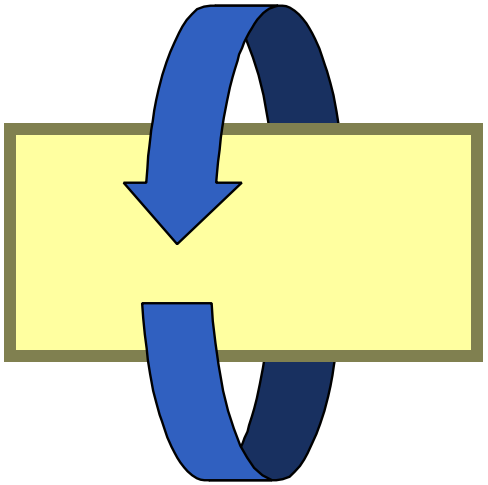


Count-Controlled Loops

Chapter 5.3

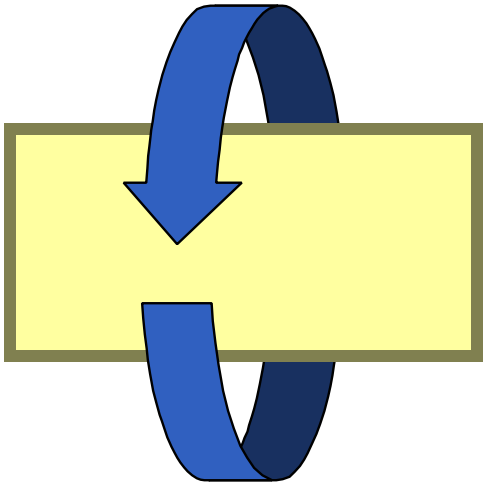
Count-Controlled Loops

- A count-controlled loop runs a *specific number of times*
- When do you use it?
  - execute a block of code a <u>specific</u> number of times
  - iterate a variable through a range of numbers



For Statement

- Most programming languages contain a *For Statement*
- In centralizes 3 expressions
  - starting value of the loop
  - ending value of the loop
  - how to increment the variable



For Statement

- Semantics ...
  - executes the initialization once
  - checks the conditional and executes the block (if true)
  - after each block, it runs the increment
- This special syntax makes it very easy to write and read

Book Pseudocode: For

```
for counter = first to last
    Statements
end for
```

Example

x starts with 1

```
For x = 1 to 5
  Display x
End For
```

Loops for 1 to 5

Example Output

```
1
2
3
4
5
```

Example 2

```
For x = 1 to 5
  Display x, x ^ 2
End For
```

Example 2 Output

```
1     1
2     4
3     9
4    16
5    25
```

Example 3

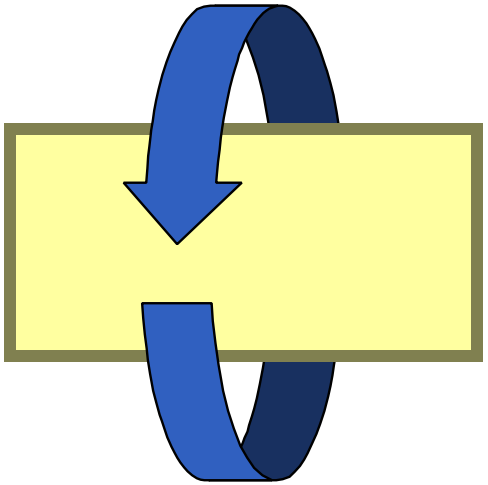
```
For n = -5 to -1
    Display n, n ^ 2
End For
```

Example 3

```
-5   25
-4   16
-3    9
-2    4
-1    1
```

Incrementing by Other Values

- By default, the For Loop will increment by 1
- However, you can change it to anything you need
- The For Loop also allows you to *step* by a different value



Book Pseudocode: For

```
for var = first to last step inc
    Statements
end for
```

Step Example

```
For n = 1 to 10 Step 2
   Display n
End For
```

Step Example Output

```
1
3
5
7
9
```

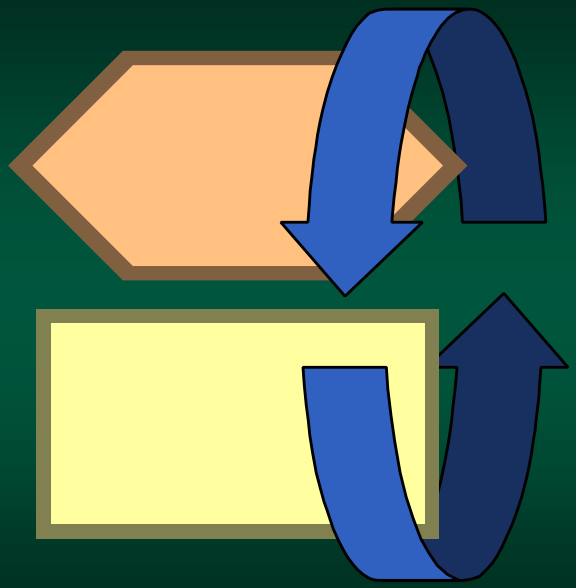
Increment by 2

Negative Step Example

```
For n = 5 to 1 Step -1
   Display n
End For
```

Negative Step Example Output

```
5
4
3
2
1
```

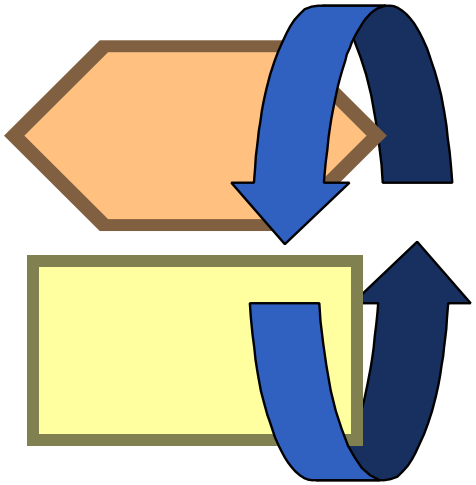


Counter Controlled While Loops

Chapter 5.3

For Loops vs. While Loops

- The For Loops is just a *specialized* version of the While Loop
- So, any For Loop can be implemented with a While Loop



Incrementing Variables

- In real programs, it is very common to increment variable
- So, the value of a variable is changed relative to its current value
- Some languages have a special notation (Java is one of them)

Incrementing Variables

- How?
  - use the variable in an expression
  - then assign result of the expression to the *same* variable
- It's pretty easy to understand, but a little strange

General Format

Same Variable

```
Set var = var + 1
```

Increment Example: What Happens?

```
Declare Integer x
Set x = 5;

Set x = x + 1

Display x
```

6

Increment Example Output

6

Increment Example 2

```
Declare Integer x


Set x = 7
Display "Before: ", x
```

9

```
Set x = x + 2
Display "After: ", x
```

Increment Example 2 Output

```
Before: 7
After : 9
```

Example

Initialize

```
Set x = 1
While x <= 5
   Display x
   Set x = x + 1
End While
```

Loops 1 to 5

Increment x

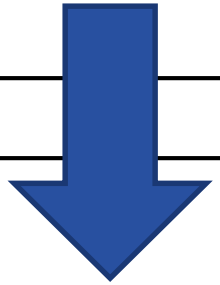
Example Output

```
1
2
3
4
5
```

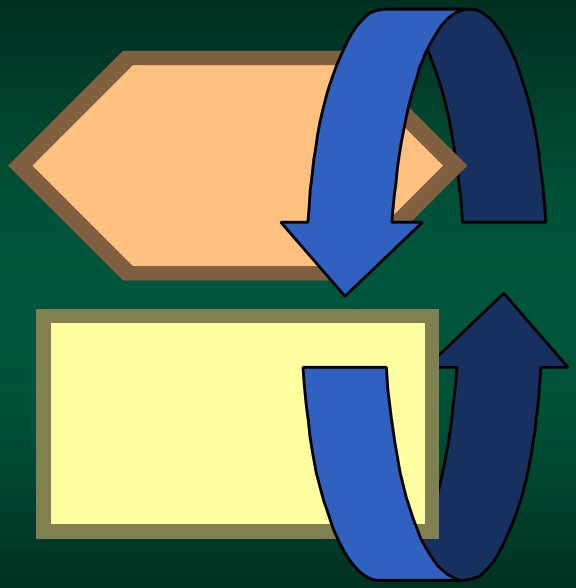
```
For n = start to end step inc
    statements
End For
```

```
Set n = start
While n <= end
    statements
    Set n = n + inc
End While
```

```
For n = 1 to 100 step 2
    Display n
End For
```



```
Set n = 1
While n <= 100
    Display n
    Set n = n + 2
End While
```

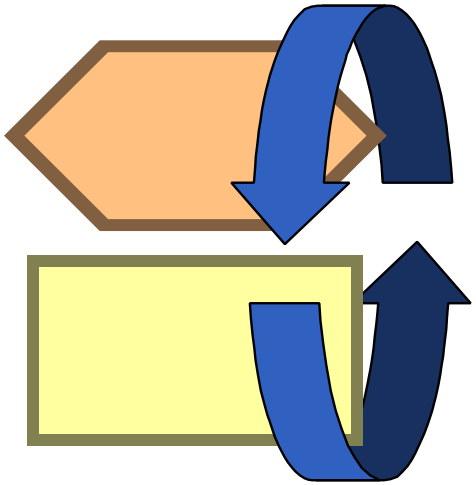


Using the Correct Loop

Chapter 5.3

Why Are For Loops Needed?

- Creating countered-controlled loops can be prone to errors
- While Loop warnings…
  - do not forget to initialize the loop control variable
  - do not forget to modify the loop control variable



Each Loop Has a Purpose

- Use a *While Loop* when…
  - statements may not have to execute
  - so, they will execute **0** or many times
- Use a *Do-While* when…
  - statements have to execute at least once
  - so, they will execute **1** or many times

Each Loop Has a Purpose

- Use a *For Loop* when…
  - statements execute specific number of times
  - or the loop needs to iterate through a range of values (which is a specific number of times)

Calculating a Running Total

Chapter 5.4

Calculating a Running Total

- A *running total* is a sum of number that accumulates with each iteration of a loop
- These are very common in real-World programs
- So, it is important to understand how to create one and the logic behind it

Example Running Totals

- Total bill – adding up each item purchased
- The size of a folder on your computer – adding up all the file sizes
- Grade – adding up the points on each assignment

Bill Example

```
Set total = 0

For n = 1 to 4
    Input cost
    Set total = total + cost
End For

Display "Total bill is ", total
```

Average Example Output

```
12.31
2.55
6.23
23.97
Total bill is 45.06
```

Average Example

```
Set sum = 0

For n = 1 to 4
    Input score
    Set sum = sum + score
End For

Display "Average is", sum / 4
```

Average Example Output

```
97
72
83
75
Average is 81.75
```

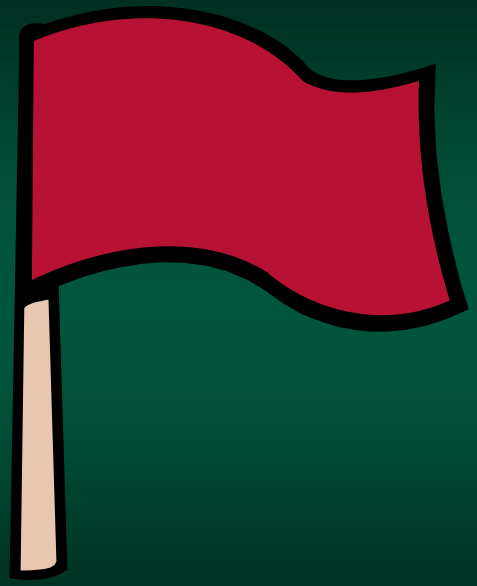
Better Average Example

```
Display "How many numbers?"
Input count

For n = 1 to count
    Input score
    Set sum = sum + score
End For

Set average = sum / count;
Display "Average is", average
```

Average Example Output

```
How many numbers? 4
68
94
83
76
Average is 80.25
```

Count

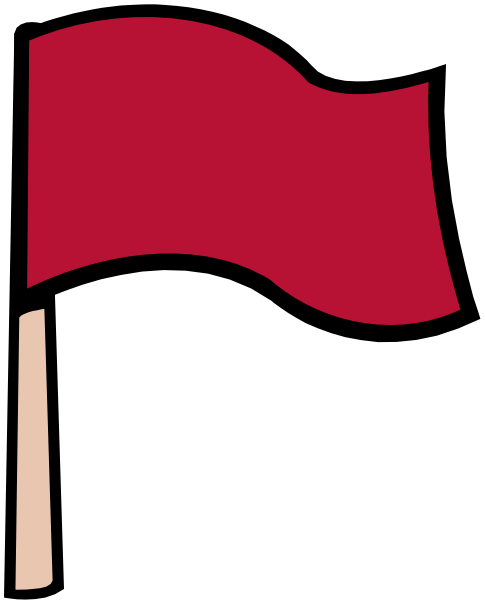


Sentinels

Chapter 5.5

Sentinels

- A *sentinel* is a special value that marks the end of a list of values
- Typically it works as a "flag" that is used to stop loops
- Also can be used to escape



How It Can Be Done

- While or Do Loop
  - ask the user if there is another value to process
  - if so, the loop will continue
- Counter controlled…
  - ask the user at the beginning of the loop, how many times the loop should process
  - use a For Loop to implement

Sentinel Example – Negative Value

```
Declare Integer points = -1
Declare Integer total = 0

Display "Enter 0 to exit"

While points != 0
    Input points
    Set total = total + points
End While

Display "Total is ", total
```

Loop "primed"

Note: this only works since total = total + 0 does nothing

Do-While Example

```
Declare Integer points = -1
Declare Integer total = 0

Display "Enter 0 to exit"

Do
    Input points
    Set total = total + points
While points != 0

Display "Total is ", total
```

Sentinel Example

```
22
120
83
0
Total is 225
```

Average Example

```
Declare Integer points = -1
Declare Integer total = 0
Declare Integer count = 0

Display "Enter 0 to exit"

While points != 0
   Input points
   Set total = total + points
   Set count = count + 1
End While

Display "Average is ", total / count
```

Count number of items

Sentinel Example

```
70
90
62
0
Average is 74
```

Average Program Issue

- What if the student gets a 0 on an assignment?
- Our program, that we wrote, does not allow for this scenario, it ends whenever we enter zero
- So, let's use -1 rather than 0

Average Example – which fails

```
Declare Integer points = 0
Declare Integer total = 0
Declare Integer count = 0

Display "Enter 0 to exit"

While points != -1
    Input points
    Set total = total + points
    Set count = count + 1
End While

Display "Average is ", total / count
```

Prime with different value

When we enter -1, it counts it and subtracts -1 from the total

Example – With an If Statement

```
96
93
0
91
-1
Average is 70
```

Sentinels and Strings

- Often strings can be used as a sentinel value
- In this case, they are often used to prompt the user whether they want to continue or exit the loop

Sentinels and Strings

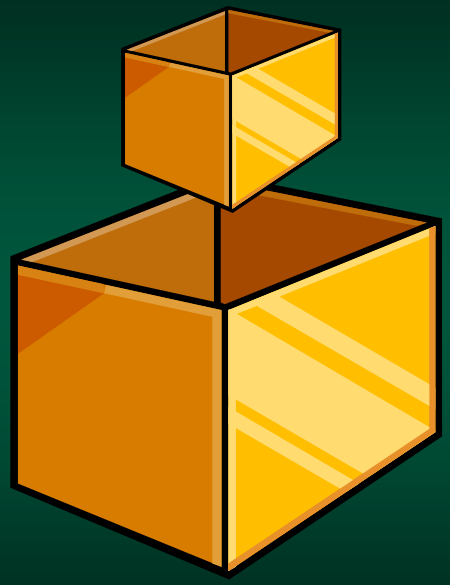
```
Declare String again
Declare Integer points
Declare Integer total = 0

Set again = "y"

While again == "y"
   Input points
   Set total = total + points

   Display "Another? "
   Input again
End While

Display "Total is ", total
```

Loop "primed"

Get sentinel value

Sentinel Example

```
25
Another? y
160
Another? y
110
Another? n
Total is 295
```

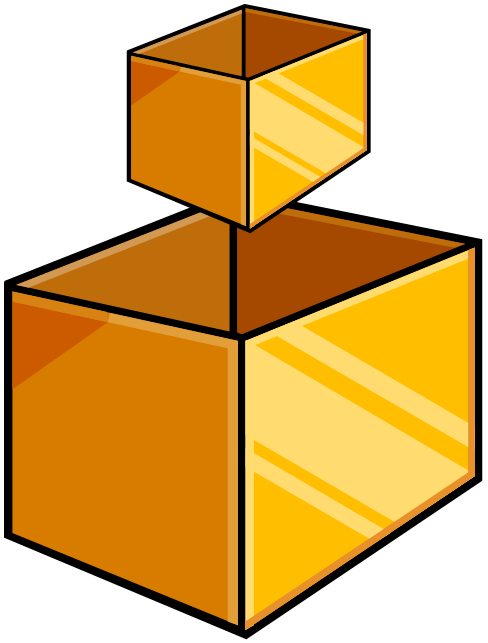


Nested Loops

Chapter 5.6

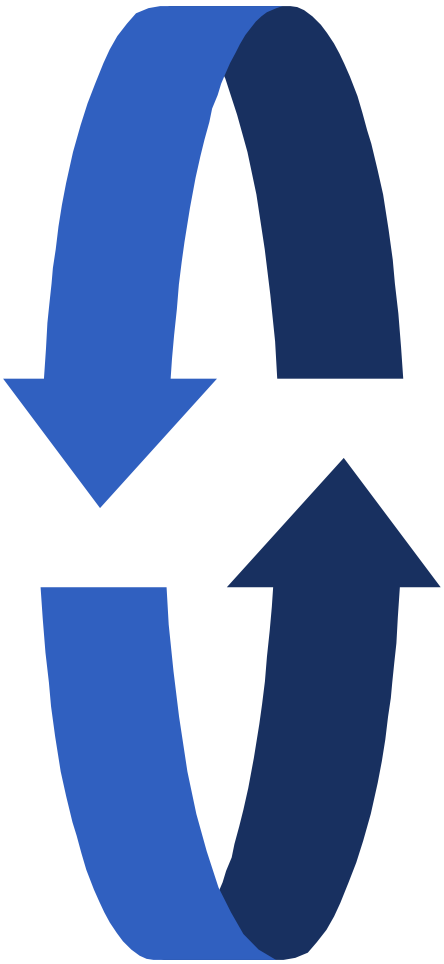
Nested Loops

- All loops can be nested, that is, a loop inside of a loop
- This can be done for any type of loop (for, while, do)
- This is also true of all other structures such as If Statements, etc…



Inner Loops

- The loop in the inner-most block is an *inner loop*
- The loop in the outer-most block is an *outer loop*



Inner Loop Example

```
For x = 100 to 101
    Display "Outer: ", x

    For y = 1 to 2
        Display "  Inner: ", y
    End For
End For
```

Inner Loop Example Output

```
Outer: 100
   Inner: 1
   Inner: 2
Outer: 101
   Inner: 1
   Inner: 2
```