Chapter 5
Repetition Structures: Looping

5.1 An Introduction to Repetition Structures: Computers Never Get Bored!

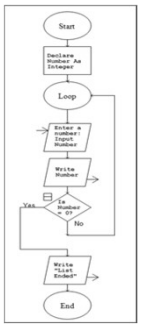
A **loop** is a block of code that, under certain conditions, will be executed repeatedly.

```
Declare Number As Integer
Repeat
      Write "Please enter a number: "
      Input Number
      Write Number
Until Number == 0
Write "List Ended"
```

The **body** of the loop is executed repeatedly until the user enters a **0**. At that point the loop is exited, and the statement that follows the loop is executed.

Note the indentation.

Flowchart:



Iterations

```
Declare Name As String
Repeat
      Write "Enter the name of your
          brother or sister: "
      Input Name
      Write Name
Until Name == "Done"
```

How many times will the loop run? Each time the loop runs it is called an **iteration**.

Jim has 2 brothers & 1 sister: 4 iterations

Marie has 1 sister: 2 iterations

Can we create a loop that doesn't use the test condition as one of the iterations? Yes … we'll see how later.

Beware the Infinite Loop

The following loop will repeat continually. Can you see why?

```
Declare ComputerNumber As Integer
Declare Number As Integer
Repeat
        Write "Please enter a number: "
        Input Number
        Set ComputerNumber = Number + 1
        Write Number
Until Number > ComputerNumber
Write "The End"
```

This is called an **infinite loop**.

Don't Let the User Get Trapped in the Loop

Be sure to tell the user how to leave the loop.

Example: Add a line, as shown, so the user knows how to stop entering numbers!

```
Repeat
      Write "Enter a number"
      Write "Enter 0 to end the program"
      Input Number
      Write Number
Until Number == 0
Write "Done"
```

Review: Comparison and Assignment Operators

As an **assignment operator**, the equals sign sets the value of an expression on the right side to the variable on the left side.

As a **comparison operator**, the double equals sign asks the question, "Is the value of the variable on the left side the same as the value of the expression, number, or variable on the right side?"

a single equals sign (**=**) signifies the assignment operator

a double equals sign (**==**) signifies the comparison operator

Review: Relational and Logical Operators and Compound Conditions

Relational operators are the symbols used in the condition to be evaluated:

- **==** is the same as (the comparison operator)
- **!=** is not the same as (not equal to)
- **<** less than
- **>** greater than
- **<=** less than or equal to
- **>=** greater than or equal to

Logical operators are used to connect simple conditions into a more complex condition called a **compound condition**.

The simple conditions each contain one relational operator.

Review: **AND**, **OR**, and **NOT** Logical Operators

- A compound condition joined by **AND** is `true` only if both simple conditions are `true`. It is `false` otherwise.

```
If (X > 5) AND (X < 10) Then...
```

is `true` if **X** is 6, 7, 8, or 9, both greater than 5 and less than 10.

- A compound condition joined by an **OR** is `true` if one of the simple conditions is `true`. It is `false` only if both are `false`.

```
If (Response ="Y") OR (Response ="y") Then...
```

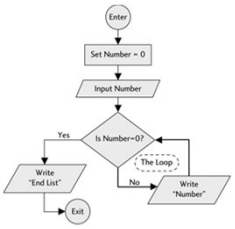
is `true` if **Response** is uppercase ("Y") or lower case ("y").

- **NOT** affects only one condition. A condition with the **NOT** operator is `true` only if the condition is `false`.

```
(X > 100) AND NOT(X == Y)
```

is `true` only if **X** is greater than 100 but not equal to the value of **Y**.

Flowchart for a Simple Repetition Structure (a Loop):



5.2 Types of Loops

- Loops are one of the most indispensable tools of any programmer.
- Loops are used to load data, manipulate data, interact with the user, and much more.
- Loops come in various types. One type may work well for one specific program's need while another type fits a different program design.
- A programmer needs to understand be able to use the different types of loops.
- Loops can be divided into two fundamental types: **pre-test loops** and **post-test loops**.

post-test loops

The test condition occurs after the body of the loop is executed.

```
Repeat
    Write "Enter a number: "
    Input Number
    Write Number
Until Number == 0
Write "Done"
```

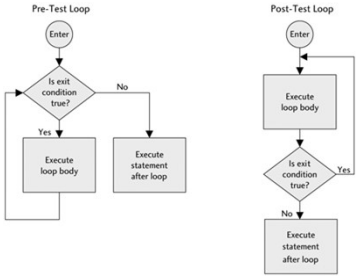
```
Do
    Write "Enter a number: "
    Input Number
    Write Number
While Number != 0
Write "Done"
```

pre-test loops

The test condition appears at the top, before the body of the loop is executed the first time.

```
Write "Enter a number: "
Input Number
While Number != 0
      Write Number
      Input Number
End While
Write "Done"
```

Flowcharts for Pre-test and Post-Test Loops



Trace a Loop: Walk through the loop manually

It is very hard to see what a loop is doing without tracing it to see how it works.
Suppose the user enters: **3**, **1**, **−1**.
Trace the code with this input.

```
Declare Number As Integer
Write "Enter a number or 0 to quit:"
Input Number
While Number > 0
      Write Number^2
      Input Number
End While
```

Number	Output
3	___
1	___
-1	___

Using a Pre-test Loop Wisely

Previously the program to list the brothers and sisters of a user always ended with the output "**Done**" (i.e, the test condition).

A **pre-test loop** can be used to avoid having unwanted data:

```
Declare Name As String
Write "Enter the name of your brother or sister: "
Input Name
While Name != "Done"
   Write Name
   Write "Enter the name of your brother or sister: "
   Input Name
End While
```

Counter-controlled Loops

- **Define** a counter: the counter is a variable.
- It is always an **integer**
- Common variable names are: **`counter`**, **`Count`**, **`I`**, or **`j`**.
- **Initialize the counter**: set the counter to a beginning value.
- To count by ones, the computer takes what it had before and adds one.

  The code for a computer to count by ones looks like:

  `Count` + 1
- **Store the new value**: store it where the old value was.
- If we are counting up by ones, we store the new value by the statement:

  `Count` = **`Count`** + 1

Example: Use a Counter to Display the Squares of Numbers

```
Declare PositiveInteger As Integer
Declare Count As Integer
Write "Please enter a positive integer:"
Input PositiveInteger
Set Count = 1
While Count <= PositiveInteger
      Write Count + " " + Count^2
      Set Count = Count + 1
End While
```

Output if the user enters 5 for **`PositiveInteger`**:

Count	Output
1	1 1
2	__ __
3	__ __
4	__ __
5	__ __

Counters Count Up and Down

Here is an example of using a counter in a loop to count down.

Countdown to Blastoff:

```
Declare Count As Integer
Set Count = 100
Write "Countdown in ..."
While Count > 0
     Write Count + " seconds"
     Set Count = Count – 1
End While
```

5.3 The **For** Loop

- Most languages have a shortened method, called a **For loop**, to initialize the counter; increase or decrease the counter; and to tell the computer when to stop.
- We use the following pseudocode in our `For` loop:

```
For (Counter = InitialValue; TestCondition; Counter++)
      body of the loop
End For
```

- **Counter** equals the specified **InitialValue**
- **Counter** increments by `1` (in this example) on each pass through the loop.
  - **Counter**`++` acts just like the statement **Counter** `=` **Counter** `+ 1`
- This continues until the **TestCondition** is met

The Initial Value in a **For** Loop

- The **initial value** can be:
  - any integer constant, such as `1`, `0`, `23`, or `-4`
  - another numeric variable
  - set equal to an expression containing a numeric variable and a number
    - such as **Count** `=` `(`**LowNumber** `+ 3)`
- The **counter** itself must be a variable and the **initial value** must be an integer.

Examples:

- **Count** `=` `5` is valid
- **Count** `=` **NewNumber** is valid if **NewNumber** is an integer variable
- **Count** `=` `(`**NewNumber** `* 2)` is valid
- **Count** `=` `(5/2)` is not valid because `5/2` is not an integer
- `23 =` **Count** is not valid

The Test Condition in a **For** Loop

- The **test condition** asks, "Is the counter within the range specified by this condition?"
  - If the test condition is **Count** `< 10`:
    - If the answer is "`yes`" then the loop executes again.
    - If the answer is "`no`" then the loop is exited.
    - When **Count** is equal to `10`, the loop will be exited.
  - However, if the test condition is **Count** `<=10`:
    - The question is, "Is the value of **Count** less than or equal to `10`?"
    - The loop will not be exited until **Count** is at least `11`.
- The test condition is checked at the end of a loop in a post-test loop and at the beginning in a pre-test loop.
- In a **For** loop, the test condition is checked at the beginning. If the initial value of the counter passes the test condition, the loop is entered once. After the loop body executes once, the counter is then either incremented or decremented and the test condition is checked again.

More About the Test Condition

The test condition can also be a number, another variable with a numeric value, or an expression containing variables and numbers. For example:

- **Count** < 5
  - is valid and will execute until **Count** has the value of 5 or more
- **Count** >= 6
  - is valid and will execute until **Count** becomes 5 or less
- **Count** >= **NewNumber**
  - is valid and will execute until **Count** becomes less than the value of **NewNumber**
- **Count** < (**NewNumber** + 5)
  - is valid and will execute until **Count** becomes greater than or equal to the value of **NewNumber** + 5

The Increment/Decrement Statement

- The **Increment** or **Decrement** statement is like:

  Count = **Count** + 1 or **Count** = **Count** – 1
- Many programming languages use a shorthand method. We will use:
  - **Count**++ increments the variable named **Count** by 1 (i.e., counts up)
  - **Count**-- decrements the variable named **Count** by 1 (i.e., counts down)
- To increase or decrease a counter by any integer other than **1**, we will use the shorthand:

  Count+2 increments **Count** by 2

This shorthand is comparable to **Count** = **Count** + 2

Count-3 decrements **Count** by 3

This shorthand is comparable to **Count** = **Count** - 3

- **Count**+**X** will increase **Count** by the value of **X**
- **Count**-**X** will decrease **Count** by the value of **X**

Examples

1.
```
For (Count = 0; Count <= 15; Count+5)
      Write Count
  End For
```

2.
```
For (Count = 15; Count >= 0; Count-5)
      Write Count
  End For
```

3.
```
Set MyNumber = 7
 For (Count = 1; Count <= (MyNumber + 1); Count+3)
      Write Count
 End For
```

4.
```
For (Count = 1; Count < 15; Count+2)
      Write Count
 End For
```

Results:

1.
```
0
5
10
15
```

2.
```
15
10
5
0
```

3.
```
1
4
7
```

4.
```
1
3
5
7
9
11
13
```

The Prisoner in the Loop

If the loop increment is positive and the initial value is greater than the limiting value, then the body of the loop is skipped as follows:

```
Write "Hello"
Declare Count As Integer
For (Count = 5; Count < 4; Count++)
      Write "Help, I'm a prisoner in a For loop!"
End For
Write "Goodbye"
```


5.4 Applications of Repetition Structures

Loops have many purposes. This section will cover a few:

- Use to input data
- Data validation
  - Using the **`Int()`** or **`Floor()`** and **`Ceiling()`** functions
- Computing sums and averages

Sentinel Controlled Loops

- Loops are often used to input large amounts of data.
- On each pass through the loop, one item of data (or one set of data) is entered.
- The test condition must cause the loop to be exited after all data has been input.
- Often the way to force a loop to end is to have the user enter a **sentinel value** to signal that input is complete.
- The sentinel item (or **`end-of-data`** marker) should be chosen so that it cannot possibly be mistaken for actual input data.
- A **sentinel-controlled loop** uses a sentinel value to determine whether or not the loop is to be exited.

Sentinel Controlled Loop Example

```
1)  Declare Hours, Rate, Salary As Float
2)  Write "Enter the number of hours this employee worked: "
3)  Write "Enter -1 when you are done."
4)  Input Hours
5)  While Hours != -1
6)        Write "Enter this employee's rate of pay: "
7)        Input Rate
8)        Set Salary = Hours * Rate
9)        Write "An employee who worked " + Hours
10)       Write "at the rate of " + Rate + " per hour"
11)       Write "receives a salary of $ " + Salary
12)       Write "Enter the number of hours this employee worked: "
13)       Write "Enter -1 when you are done."
14)       Input Hours
15) End While
```


Data Validation: Loops are often used to validate data entered by a user.

Example: to ensure that a user enters a positive number:

```
Declare WidgetsOrdered As Integer
Write "How many widgets do you want to order? "
Input WidgetsOrdered
While WidgetsOrdered < 0
      Write "You can't order a negative number of widgets."
      Write "Please enter a positive number or zero: "
      Input WidgetsOrdered
End While
```


The **`Int()`** Function

The `Int()` function takes any number and turns it into an integer. It is often used to ensure that data is an integer.

Examples:

1) `Int(53) = 53` → the integer value of an integer is just that integer.

2) `Int(53.195) = 53` → the integer value of a floating point number is just the integer part, with the fractional part discarded.

3) `Int(53.987) = 53` → note that since the fractional part is discarded, `53.0001` is the same, after implementing the `Int()` function, as `53.9999`.

Given: **`Number1`** `= 15.25` and **`Number2`** `= −4.5`, then:

4) `Int(`**`Number1`**`) = 15` → **`Number1`** represents `15.25`, `Int()` function turns the value of **`Number1`** into its integer part.

5) `Int(`**`Number2`**`) = −4` → **`Number2`** represents `−4.5` and the integer part of this is `−4`.

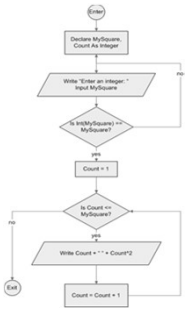
6) `Int(4*2.4) = 9` → first the `Int()` function does the math inside the parentheses and then returns the value as an integer.

Using the `Int()` Function for Data Validation

```
Declare MySquare As Integer
Declare Count As Integer
Repeat
      Write "Enter an integer: "
      Input MySquare
Until Int(MySquare) == MySquare
For (Count = 1; Count <= MySquare; Count++)
      Write Count + " " + Count^2
End For
```

This program segment first validates the input to make sure it is an integer. Note how it uses the `Int()` function and a loop to do this. Once the valid input has been entered, a `For` loop is used to display the squares of the numbers from 1 up to and including the number input.

Flowchart for using the `Int()` function for data validation



The `Floor()` and `Ceiling()` Functions

- `Floor()` takes any number and discards the decimal part (i.e., it "rounds down")
- `Ceiling()` takes any number and "rounds up" to the next integer
- Like `Int()`, `Floor()` and `Ceiling()` accept numbers, variables, and expressions

Examples:

```
Floor(62) = 62                  Ceiling(62) = 62
Floor(62.34) = 62               Ceiling(62.34) = 63
```

Given: **NumberOne** = 12.2 and **NumberTwo** = 3.8

```
Floor(NumberOne) = 12           Ceiling(NumberOne) = 13
Floor(NumberTwo) = 3            Ceiling(NumberTwo) = 4
Floor(NumberOne * 4) = 48       Ceiling(NumberOne * 4) = 49
```

If **Number** = 7.83, the following statements are valid:

```
Write Floor(Number) displays 7
Write Ceiling(Number) displays 8
Set Y = Floor(Number) assigns the value of 7 to Y
Set Y = Ceiling(Number) assigns the value of 8 to Y
Set X = Floor(Number/2) assigns the value of 3 to X
Set X = Ceiling(Number/2) assigns the value of 4 to X
```

Using a Loop to Compute a Sum

Note: the variable **Sum** is known as the **accumulator** because it accumulates the values of the inputs.

Example:

```
1       Declare Sum, Number As Integer
2       Set Sum = 0
3       Write "Enter a positive whole number or 0 when done."
4       Input Number
5       While Number > 0
6               Set Sum = Sum + Number
7               Write "Enter a positive whole number or 0 when done."
9               Input Number
9       End While
10      Write "The sum of the numbers input is " + Sum
```


Computing Averages

Example:

```
1       Declare CountGrades As Integer
2       Declare SumGrades, Grade, ExamAverage As Float
3       Set CountGrades = 0
4       Set SumGrades = 0
5       Write "Enter one exam grade. Enter 0 when done."
6       Input Grade
7       While Grade > 0
8               Set CountGrades = CountGrades + 1
9               Set SumGrades = SumGrades + Grade
10              Write "Enter an exam grade. Enter 0 when done."
11              Input Grade
12      End While
13      Set ExamAverage = SumGrades/CountGrades
14      Write "Your exam average is " + ExamAverage
```