The While Loop and the Do Loop are two powerful concepts that allow your program to do the same task multiple times.

However, these are not the only loop structures that professional programmers use on a regular basis. There is a third structure called the For Loop. It's actually not necessary, but makes a very common task very easy.

### Part 1 – What is a Counter-Controlled Loop

Before we cover counter-controlled loops, it is important to understand the different types of things a typical program will do. One of the most common tasks is to simply go through all the items in a list or range of values. For example, this can be the values in a table, the files in a folder, a list of numbers, etc.... All these tasks start with one value (or item) and proceeds until the last value (or item).

The process of moving through a range of values is called iteration. It always starts with an initial value and proceeds until it reaches the end value. The loop typically jumps by values of 1, but this is not always the case. It all depends on what the programmer needs the program to do.

Let's try implementing a simple iterative counter with a While Loop. The pseudocode below (written in the textbook's format) will display the numbers 1 to 100.

```plaintext
Declare Integer n
Set n = 1
While n <= 100
    Display n
    Set n = n + 1
End While
```

Notice that it assigns the variable `n` to 1 before the loop begins. At the bottom of the loop, it assigns `n` to its value plus 1. This will increment the value from 1 to 2. Then, after that, from 2 to 3, from 3 to 4, etc... The loop continues while `n` is ≤ 100.

In this case, the variable `n` is used as a counter for the values from 1 to 100. At the same time, `n` is used to control whether the loop continues or exits. This logic is known as a **counter-controlled loop**.

1. Start a new Flowgorithm Flow Chart.
2. Implement the pseudocode above.
3. Execute your flowchart
Ah! It printed the numbers from 1 to 100 in the console. That was easy! But, it does have its dangers. Try removing the \( n = n + 1 \) assignment statement. What happens?

**Part 2 – For Loops**

Counter-Controlled Loops are very common in real programming. They are so common, most programming languages (and Flowgorithm) have "shorthand" syntax for such loops.

A **For Loop** implements the logic of a counter-controlled loop but makes it easy to read, easier to understand, and safer than using a While Loop. The most important thing to understand is that this is just shorthand for the counter-controlled loop you learned in the previous exercise. It works exactly the same way -- it just saves you some typing.

The pseudocode below (written in the textbook's format) will display the numbers 1 to 100. It does exactly the same thing you did before, but uses a For Loop.

```plaintext
Declare Integer n
For n = 1 to 100
   Display n
End For
```

There are a number of really nice advantages to this notation.

First, you can see the entire range of values in the start of the loop. You can tell it will iterate from 1 to 100 at a glance. Before, you had to read and carefully analyze the code to see the logic. Now? Poof! Its right there!

Second, the "Set \( n = n + 1 \)" statement is gone! **The For Loop increments the loop variable for you. I would make this sentence flash in obnoxious colors and dance a jig on the screen if I could, because this is important.** So, that dreaded infinite loop will no longer be an issue.

Finally, the initialization "Assign \( n = 1 \)" is also gone. The For Loop handles all of this!
Let's replace your While Loop with one of these handy-dandy For Loops.

1. Add a For Loop to your flowchart. Put it above or below your current While Loop.

2. Select your Output Shape, cut it, and paste it into your new For Loop.

3. Now, delete that old While Loop (including that Assignment Shape).

4. Double-click on the For Loop.

The For Properties window should be open.

Let's look at the different fields here:

The Variable is changed each time the loop executed. It is initially set to the Start Value and continues until it reaches the End Value.

The Direction determines if the loop increments from a small value to a large value or decrements from a large value to a small one. The Step By value is used to indicate how much to increment (or decrement) the value each time in the loop. Practically all loops increment (or decrement) by one, so Flowgorithm sets this to 1 by default. (Also, Flowgorithm doesn't bother to show the step on the flowchart if it is simply 1)

5. Fill in all the fields.

6. Execute your flowchart.

Did you get the same results? You should have it. If not, please recheck your work.

Just so you can see the difference between and a While Loop and For Loop graphically, the following is a side-by-side comparison.
Which is easier to read? Which is smaller? Yes, the For Loop is indeed helpful!

**Part 3 – Controlling the End Value**

Your For Loop is quite impressive, but it is a bit limited. It always counts to 100. Is there a way to control how many times it executes?

1. Declare a new variable to represent the number of times you want it to loop. You can name it whatever you want. So possible names are `max`, `count`, etc...

2. Input this variable before the loop. Give it a nice prompt output shape.

3. Double-click to edit the For Loop.

4. What value should be changed? Think about it for a sec. Change the correct one.

5. Execute and test the results.

If everything worked correctly, then you can control how many times the For Loop will execute. You can make it loop a single time or up to a million!

**Part 4 – Computing an Average**

Simply printing a bunch of numbers to the console is not all that useful. So, let's make something a bit more helpful. One common computing task is to compute the sum of a series of numbers.

This type of logic is known as a **running total.** As a program runs, it will continuously add values to an existing variable. You have done this before. Yes, you have! Anytime you added a bunch of values on the calculator, you were creating a running total.

1. Delete the Output Shape that displayed the counter. We don't need it anymore.

2. Declare a variable for the running total. Make sure to put this at the above the loop

3. Declare a variable for the value that the user enters (it will be added to the total)
4. Inside the For Loop's body, add an input for the value.

5. Add an Assignment Shape that adds the inputted value to the running total.

6. Add an Assignment shape that sets your running total to 0. (Where should you put it?)

So, at this point, your program should: (1) input for the number of times to execute the For Loop, (2) each time in the loop you input a value and add it to the total.

---

**EXPLORE: Variable Watch Window**

Flowgorithm can graphically show you the current value of your variables. This works anytime you are stepping through your program. Select the "Variable Watch Window" from the Tools menu. There is also an icon on the main toolbar.

Different data types are displayed in different colors: integers are blue, reals are purple & strings are red.

---

7. Execute and observe the results. The Variable Watch Window is great for this.

Your program lets you select how many numbers to add up and does exactly that. Now that you can create a total, the next step is to create an average.

8. Declare a variable for the average. Make it a real.

9. After your For Loop completes, compute the average and store it in your new variable.

10. Output the average to the screen with a nice caption.

11. Execute and check the results.

---

**Part 5 – Hokey-Pokey Dance-a-rama**

News about your incredible programming skills has spread. As a result, you were hired for the exciting (and mildly frightening) International Hokey Pokey Dance-a-rama!

In case you don't remember, the Hokey Pokey has the following lyrics:

> You put your right foot in,  
> You put your right foot out;  
> You put your right foot in,  
> And you shake it all about.  
> You do the Hokey-Pokey,  
> And you turn yourself around.  
> That's what it's all about!

*(Editor's note: don't ask. -C)*

The dancers get to choose how many attempts they get to make. All these attempts are averaged together. Then, based on their average, they know if they qualified or were eliminated. Each dancer receives scores from 0 to 100. They need a 70 to pass!
1. Add an If Shape after the average is outputted
2. Add the proper condition logic so the *true branch* is followed if they qualify
3. Add Output Shapes letting them know if they qualified or not.
4. Execute and check the results.

If you did everything correctly, your flowchart should display the average and then a nice "congrats" or "sorry" message.

5. Update all your prompts so they are consistent with this wonderful contest!

**Part 5 – Big Problem**

I think the Hokey Pokey Dance-a-rama officials are happy with your program! Everything is great! You are going to get a huge paycheck! Everything is awesome…. uh oh…. one of the officials looks rather confused.

It turns out that your program has a bit of a flaw. What happens if one of the contestants simply drops out? Perhaps, it's their nerves. Perhaps, they have sore feet. Regardless, they didn't make any attempts!

1. Execute the flowchart one more time. This time, enter 0 attempts.

Yep. It's the good old divide-by-zero error. You learned how to fix this in your last lab. Add an If-then statement around the division. Print an error message if the dancer has 0 attempts. Otherwise, print their average.

**Part 6 – Multiple Dancers**

Ah, the Hokey Pokey Dance-a-rama officials are once again happy. But, they want to add one final feature to the program. Currently, your program only handles a single dancer. However, this contest has dozens of contestants from all over the World. Yes, it's that popular!

So, the new addition will allow the program to handle multiple dancers. How? Well, with another For Loop! Your current For Loop – that reads all the attempts and gets an average – can be put *inside* another For Loop. Just like If Statements, For Loops can also be embedded within each other.

This might get scary dangerous fun. So, please save your program if you haven't already.

1. Declare a variable for the total number of dancers. This should be an integer.
2. Declare a variable for the current dancer. This will be used to iterate through your new For Loop. Make it an integer.
3. Input the total number of dancers. Put it right at the top of your program (below the declarations and above your current loop)
4. Add a nice prompt output shape.
5. Add a new For Loop. Put it right above current For Loop.
6. Double-click on this new For Loop.
7. Just like you did with your first loop, make it so it loops a fixed number of times – depending on the value the user entered for the number of dancers.
8. Here's the fun part. Highlight your "attempts" For Loop. Yes, the whole thing. You might have to zoom out to see all the shapes.
9. Cut and paste it into the new "dancer" For Loop. Whew!

10. Execute your flowchart.

Did it work correctly? If it did, you should be prompted for the number of dancers. And, for each dancer, your program should run like before. You are prompted for the number of attempts, it calculates the average, and tells you the results.

One final touch... Let's let the user know which dancer they are entering attempts for.

11. Add an Output Statement inside your new "dancer" For Loop. Put it right at the top – above the prompt for the number of attempts.

12. Make it output something like "Dancer #2" where the 2 is the current dancer (not just the value).

Here is a screenshot from one possible solution. Your output doesn't have to look exactly like this. What is important is the logic!

Ta da! You have made incredible program. I wonder if they will need your help with the Chicken-Dance-a-rama! One can only hope.