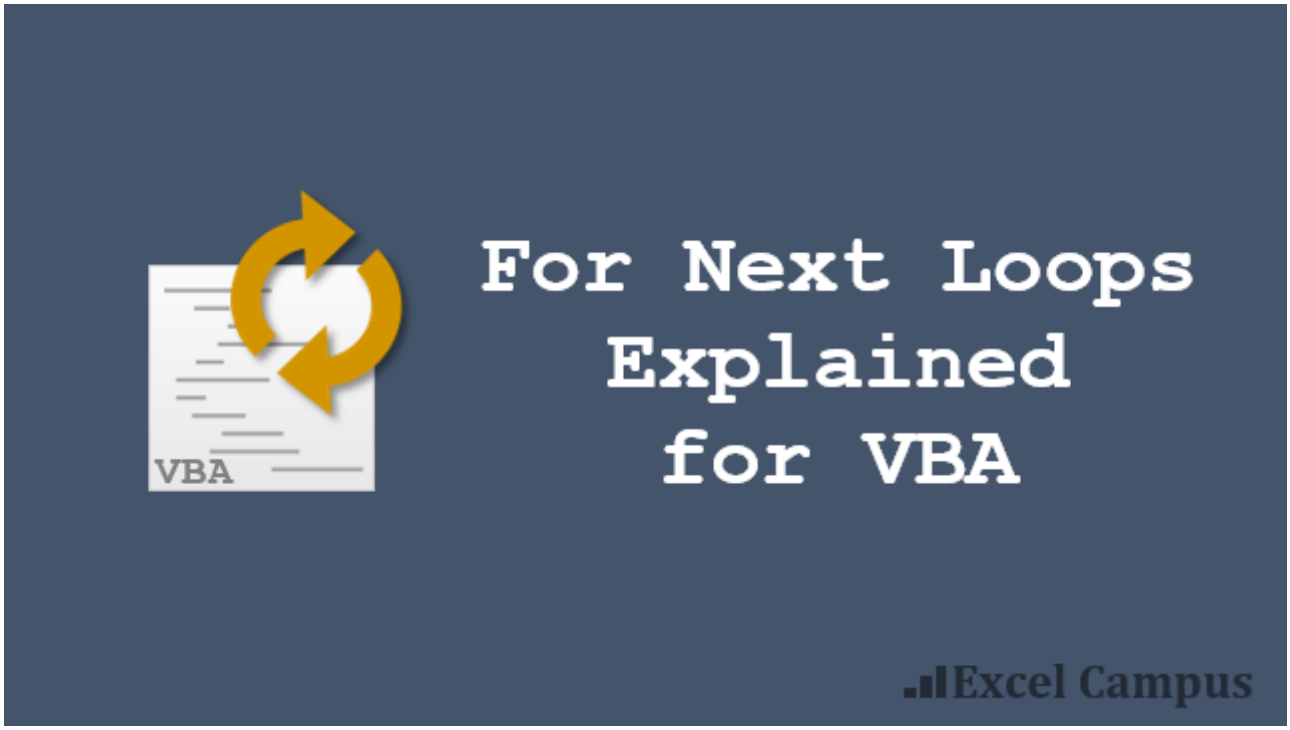
The For Next and For Each Loops Explained for VBA & Excel

excelcampus.com/vba/for-each-next-loop/

16

Bottom line: The For Next Loops are some of the most powerful VBA macro coding techniques for automating common tasks in Excel. This article explains how the loop works to repeat actions on a collection of items, which saves us a ton of time with our jobs.

Skill level: Intermediate



The Power of VBA Loops

In Excel we spend a lot of time repeating simple tasks. This could be actions like: formatting multiple ranges, unhiding multiple sheets, copying and pasting to several workbooks, apply filters to multiple tables or pivot tables, replacing values, updating formulas, etc.

Can you think of a few tasks where you had to repeat the same process over and over again?

These tasks can be extremely time consuming and BORING!

Fortunately, there is a way out. We can use loops in our VBA macros to **repeat actions very quickly**. Tasks that can take hours to do manually can be completed in a matter of seconds with a loop.

The For Next Loop is the most common type of loop that helps us with these repetitive jobs. In this article we will look at the two types of For Next Loops.

Download the Example File

Download the free Excel file that contains examples of macros with the For Next Loop.

For Next Loop VBA Macro Examples.xlsm (75.8 KB)

Download a PDF version of the article for printing.

The For Next And For Each Loops Explained For VBA Excel - Excel Campus.pdf (464.1 KB)

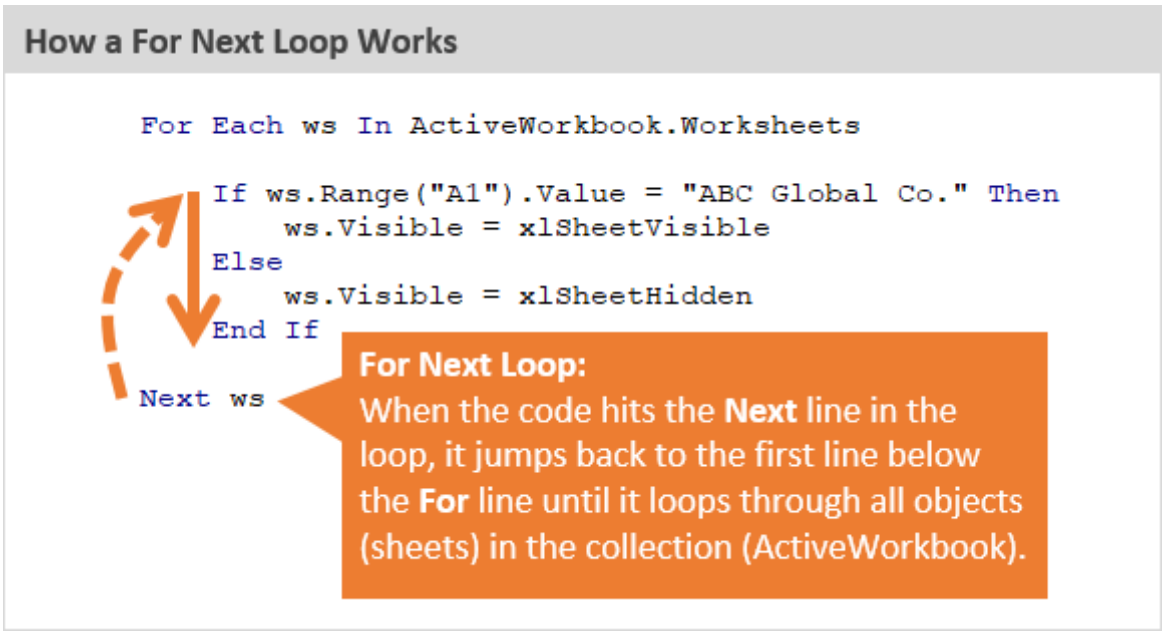
How Does the For Next Loop Work?

The For Next Loop allows us to loop through a collection of items in Excel. The collection can be a collection of objects or a list of numbers.

Examples of collections of objects include:

- Cells in a range.
- Worksheets in a workbook.
- Open workbooks on the computer.
- Pivot tables in a worksheet.
- Pivot fields in a pivot table.
- Shapes on a worksheet.
- And any other object you interact with in Excel.

The job of the For Next Loop is to perform the same actions (lines of code) on each item in the collection.



The example below contains a For Next Loop that loops through each worksheet in the workbook and unhides each sheet. The loop starts at the first item in the collection (the first sheet in the workbook), and performs the line(s) of code between the For and Next lines for each item in the collection (every sheet in the workbook).

```
Sub Unhide_Multiple_Sheets()

Dim ws As Worksheet

    For Each ws In ActiveWorkbook.Worksheets
        ws.Visible = xlSheetVisible
    Next ws

End Sub
```

Of course we can use logical statements like If statements to test properties and conditions before performing the actions. The following macro only unhides sheets that have the phrase “ABC Global Co.” in cell A1 of each sheet, and hides all other sheets.

```
Sub Unhide_Report_Sheets()

Dim ws As Worksheet

    For Each ws In ActiveWorkbook.Worksheets
        If ws.Range("A1").Value = "ABC Global Co." Then
            ws.Visible = xlSheetVisible
        Else
            ws.Visible = xlSheetHidden
        End If
    Next ws

End Sub
```

The Two Types of For Next Loops

There are really two types of For Next Loops.

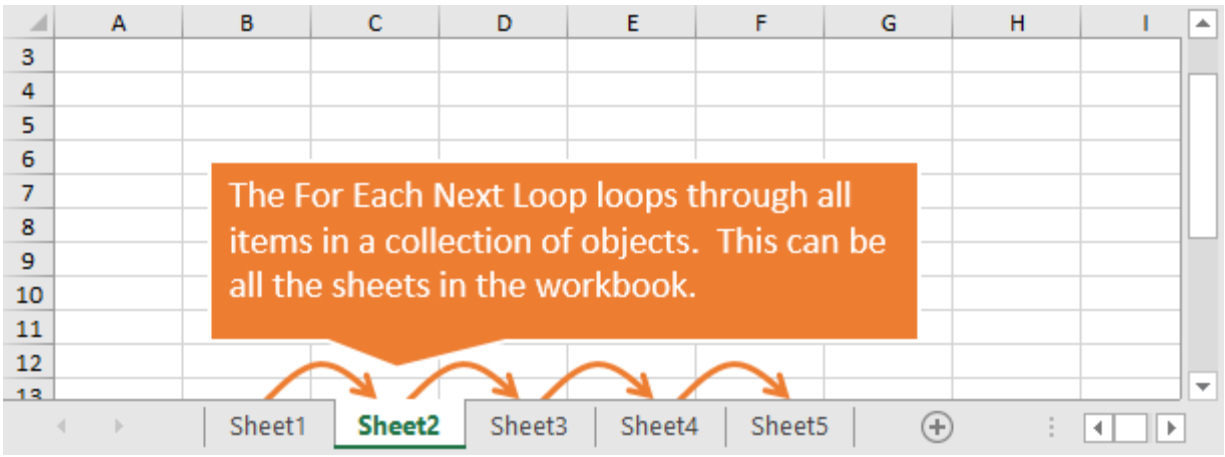
1. For Each Next Loops loop through a collection of items.

2. For Next Loops loop through a set of numbers.

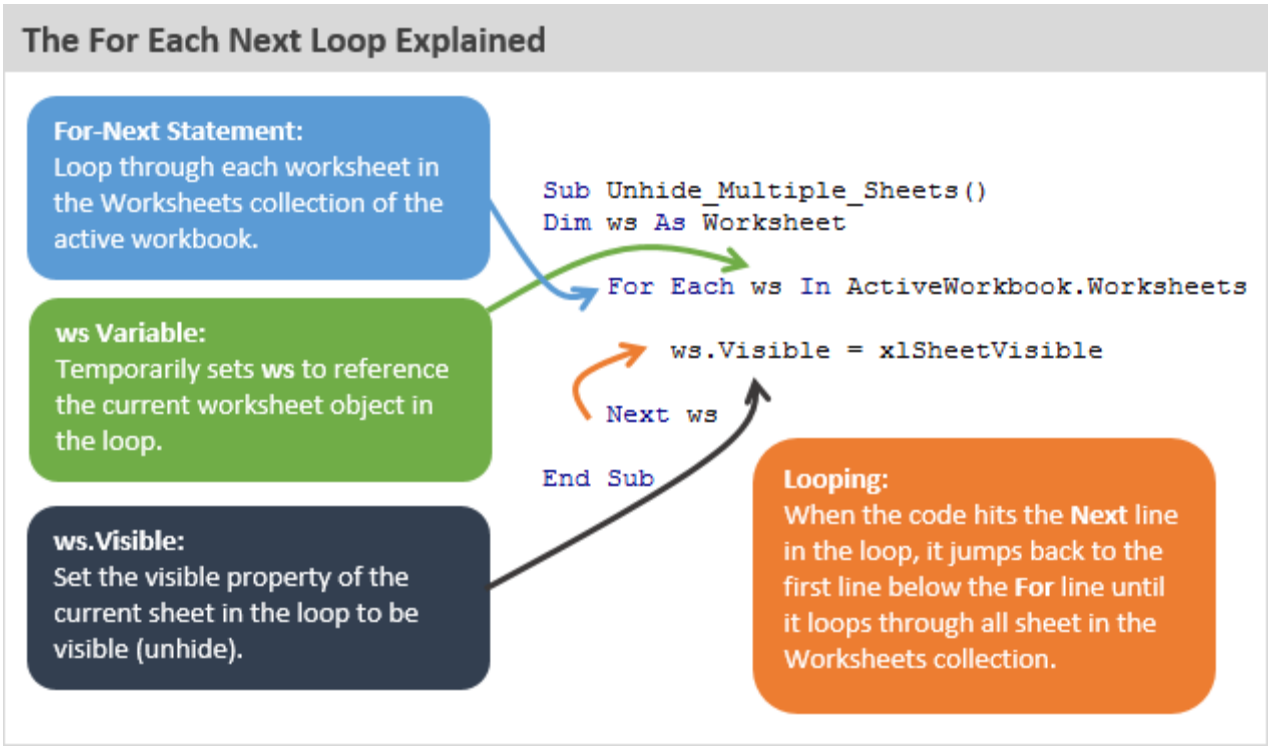
Let's take a look at how each works.

The For Each Next Loop: Loops Through a Collection of Items

As we saw above, the The For Each Next Loop allows us to loop through a collection of items or objects. This is probably the most common loop we use in Excel because we are working with collections of objects. Again, these collections are the cells in a range, worksheets in a workbook, pivot tables in a worksheet, etc.



We will use the example of writing a For Each Next Loop to loop through all the worksheets in a workbook.



There are 4 basic steps to writing a For Each Next Loop in VBA:

1. Declare a variable for an object.
2. Write the For Each Line with the variable and collection references.
3. Add line(s) of code to repeat for each item in the collection.
4. Write the Next line to close the loop.

Let's take a look at each of these steps in detail.

Step 1 – Declare a Variable for an Object

We first need to declare a variable that will temporarily store a reference to the object.

The Dim line at the top of the macro declares a variable as an object. In this case the object is a worksheet. We can create any variable name we want as long as it is not the same as another reference in VBA. "ws" is the most common variable name for a worksheet object, but you can change this.

```
Dim ws As Worksheet
```

Step 2 – The For Each Line

Next we will write the For Each statement. This is the first line of code in the loop.

```
For Each ws In ActiveWorkbook.Worksheets
```

The first two words are **For Each**. Then we type the variable name, followed by the word **In** Finally, we specify where the collection exists. In this case we want to loop through all the worksheets in the ActiveWorkbook. So, we type **ActiveWorkbook.Worksheets**. That line references all the worksheets in the ActiveWorkbook.

If you want to loop through worksheets of a specific workbook, then you can use the Workbooks property to reference that workbook by name.

```
For Each ws In Workbooks("Book2.xlsx").Worksheets
```

Just remember that the workbook you reference has to be open before the For Next line of code runs. Of course, we can use the Workbooks.Open method to open the workbook as well.

Step 3 – Add Code to Repeat for Each Iteration

After the For Each line, we add the line(s) of code that will be performed on each sheet. In this example we just have one line of code that unhides the sheet.

```
ws.Visible = xlSheetVisible
```

In this line of code we are using the **ws** variable to reference the current worksheet in the loop. When the loop runs, it sets a **temporary reference to the ws variable for each iteration in the loop**.

This would be the same as if we were to set the ws variable to a specific sheet using the following line of code.

```
Set ws = Worksheets(1)
```

However, **we do NOT need this line with the For Each Next Loop**. The loop takes care of setting the variable for us for each iteration in the loop.

For the first iteration in the loop, the ws is set to Worksheets(1). In the next iteration, ws is set to Worksheets(2). This continues as the loop iterates through all sheets in the workbook. This is very powerful because we can reuse the variable to reference the worksheet several times within the the loop.

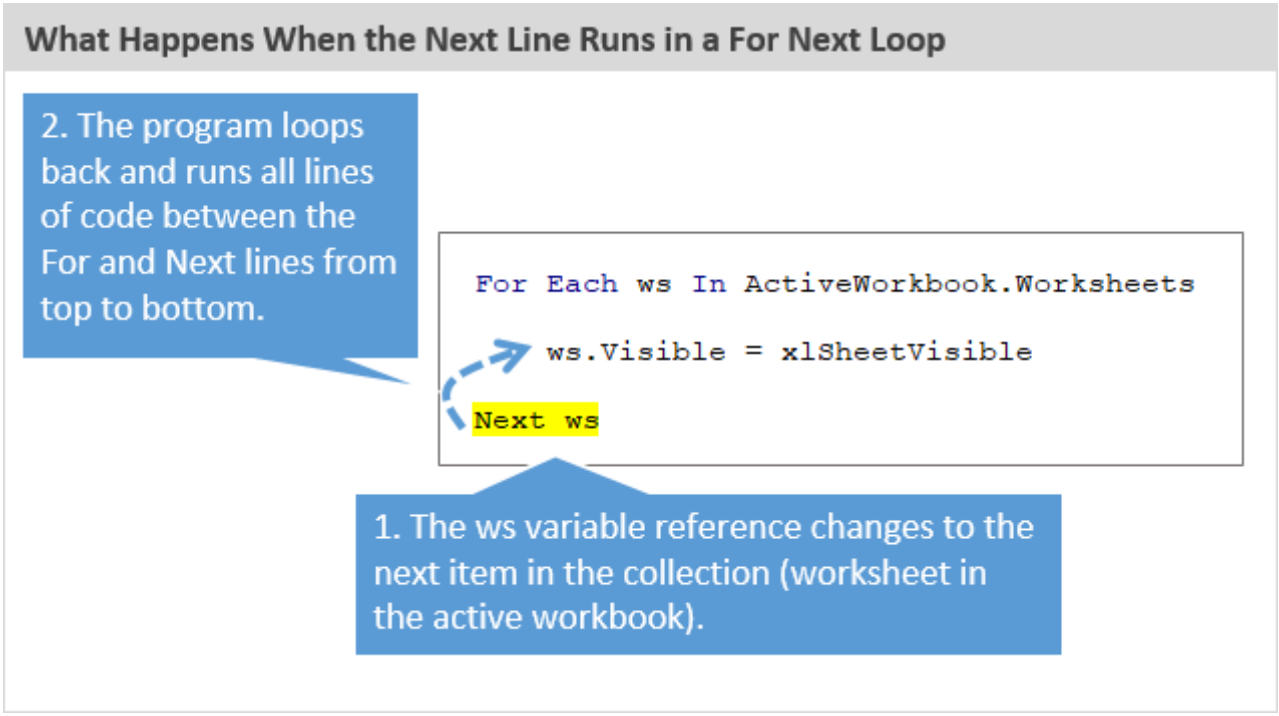
Step 4 – The Next Line Loops Back

The final line of code in the loop is the **Next** line.

```
Next ws
```

When the macro hits this line of code it does two things:

1. First, it changes the variable reference to the next item in the collection. In this example, the ws variable is changed to reference the next sheet in the workbook.
2. Second, it loops back up to the run the line of code directly below the For Each line. It then runs all the lines of code between the For Each and Next lines in top to bottom order.



When the last item in the collection (worksheet in the workbook) is reached, the looping stops, and the macro continues on to the next line of code below the Next line.

What Order Does the For Each Loop Run In?

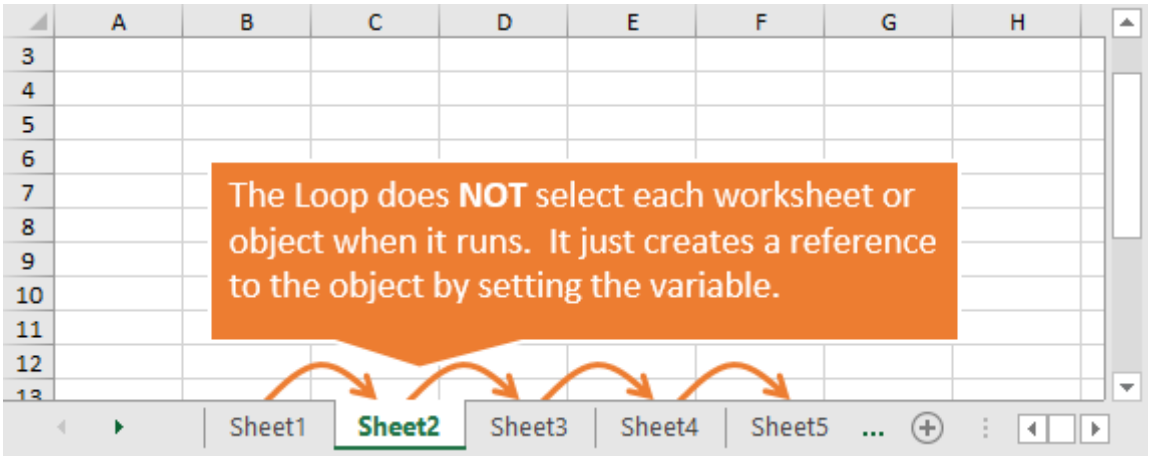
The For Each Loop always starts at the first item in the collection and loops through to the last item in the order they appear in Excel. This is based on the index number of the items in the collection. Here are some examples of the order the loop runs in for common objects.

- **Worksheets:** Starts at the first sheet tab in the workbook and loops to the last in the order the tabs are displayed in the workbook.
- **Workbooks:** Starts at the first workbook that was opened and loops in the order that workbooks were opened. Workbooks are assigned an index number as they are opened.

- **Cells:** Loops left-to-right, then down. Starts in the first cell in the range and loops to the next column in the same row, then goes to the next row.
- **Tables & Pivot Tables:** Starts with the first object that was create in the sheet and loops in the order the objects were created. This same principle is true for other objects you create in sheets like shapes, charts, slicers, etc.

Items/Objects Are NOT Selected in the Loop

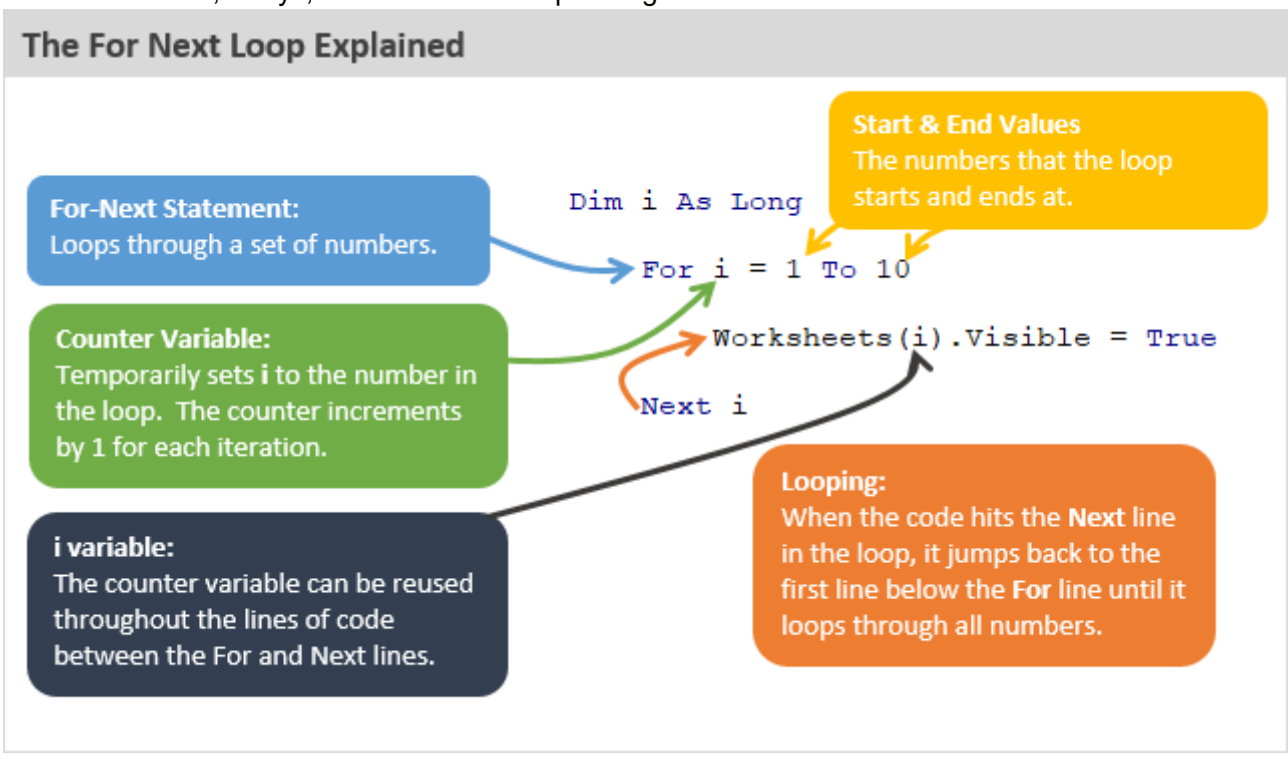
It's important to note that when we loop through a collection of objects, each object is NOT selected in Excel.



The loop creates a reference to the item/object with the variable. The variable is temporarily set to a reference of the object. The object is NOT selected and does NOT necessarily become the active object. To select the object we can use the Select or Activate methods. You would also have to make sure the objects parent object(s) are selected first. Checkout my article on the Excel Object Model and Object Hierarchy in VBA for more on this.

The Next Loop: Loops Through a Set of Numbers

We can also use the For Next Loop to loop through a set of numbers. This can be useful when we are looping through userform controls, arrays, or if we want to loop through a collection backwards.



The basic operation of the For Next Loop is the same as the For Each Loop. The difference is the format of the For line.

Step 1 – Declare a Variable for a Number

To loop through a set of numbers we first need to declare a variable to a whole number data type. We can use Integer or Long integer.

```
Dim i As Long
```

The variable is referred to as the **Counter** because it increments or counts up/down for each iteration in the loop.

A side note on Long: *The Long (integer) data type holds a bigger number than Integer. It takes up more memory, but today's computer have so much memory that it's no longer a problem. We can use Long variables all the time. The letter L looks like the number 1 in VBA, so I'm now using i as the variable name even though I use Long as the data type. This is all a matter of personal preference and you can name the variable whatever you want.*

Step 2 – Write the For Statement

Next we write the For statement. The basic construct is the keyword **For**, followed by the **variable name (counter)**, then **equals sign**, **start value To end value**.

```
For i = 1 To 10
```

The start and end values can be referenced as numbers, or we can use integer/long variables in their place.

```
For i = iStart To iEnd
```

We can also use properties of objects that return a number.

```
For i = 1 To ActiveWorkbook.Worksheets.Count
```

That line of code would return the number of sheets in the active workbook. However, it is NOT looping through each worksheet. The loop is just looping through a set of numbers. We have to create a reference to a worksheet with the counter variable (i) as the index number of the Worksheets property. Step 3 shows this reference.

Step 3 – Add Code that Repeats for Each Iteration

The rest of the loop functions the same as the For Each loop. We can add lines between the For and Next lines that will run for each iteration of the loop. The counter variable can be used multiple times in these lines of code.

```
Worksheets(i).Visible = True
```

Step 4 – The Next Line Increments the Number & Loops Back

Finally, we add the Next line at the bottom.

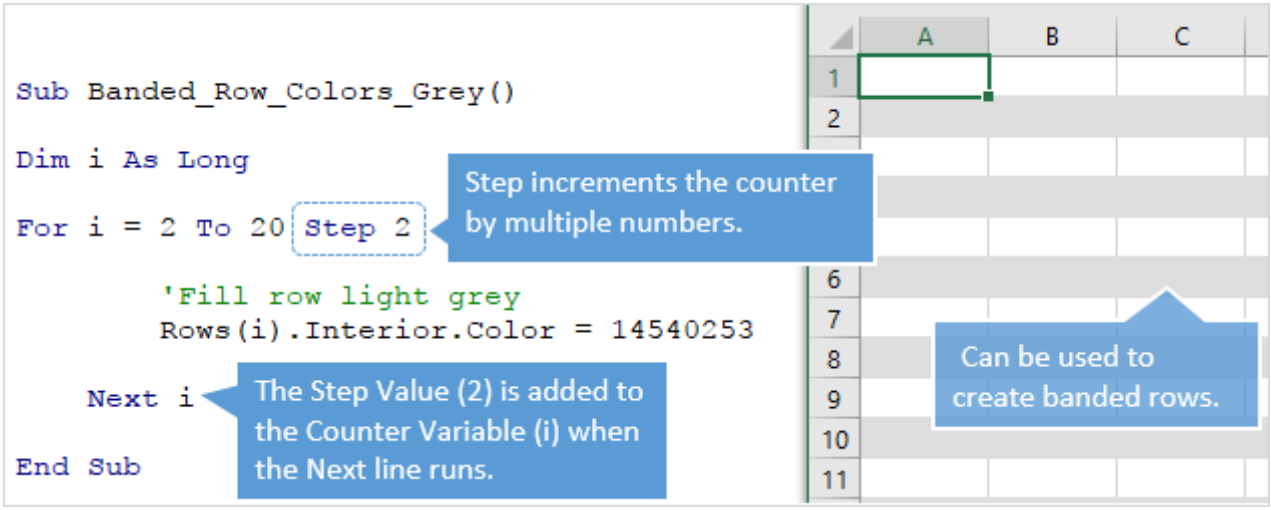
```
Next i
```

When the macro runs it will set the variable equal to the first number in the For line. When the macro hits the Next line, it will add 1 to the value of the variable, or count up. So, i = 2 in the second iteration of the loop. It continues to loop until the last number in the loop is reached.

By default, 1 is added to the variable counter for each iteration in the loop. This is called the **Step Value**, and we can control the value of each step in the counter. The Step value is added to the end of the For line. The following line will add 2 to the counter for each iteration in the loop.

```
For i = 2 To 20 Step 2
```

If you wanted to shade every other row in a sheet, you might use a loop like this.



Looping Backwards

We can also use the Step Value to loop backwards by specifying a negative number.

```
For i = 100 To 1 Step -1
```

Notice that the **Start Value is now the larger number and the End Value is the smaller number**. The loops starts at 100 (Start Value) and subtracts 1 from the counter variable (Step -1) for each iteration in the loop until it gets to 1 (End Value).

The Step keyword is **optional**. If you do not specify it then VBA assumes a Step value of 1.

Looping backwards is great if you are deleting items. I will write a separate post on this, but the general idea is that when we are looping through a collection and deleting items, the size of the collection gets smaller as items are deleted. The loop will typically hit an error once it gets to the 10th item, when there are now only 9 items in the collection. Looping backwards prevents this potential error.

Exiting the Loop Early

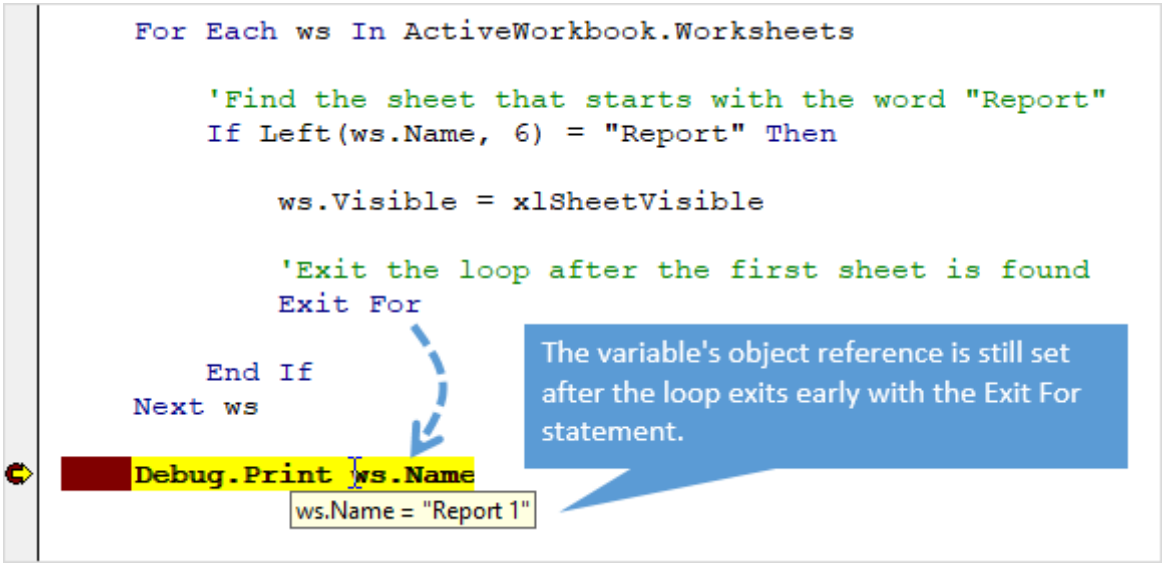
Typically the loop will iterate through all items in the collection, then continue on to the next line of code below the Next line. However, we can stop the loop early with an **Exit For** statement.

```
Exit For
```

The following macro uses the Exit For statement to exit the loop after the first sheet that starts with the word “Report” is found an unhidden.

```
Sub Unhide_First_Sheet_Exit_For()
'Unhides the first sheet that contain a specific phrase
'in the sheet name, then exits the loop.

Dim ws As Worksheet

    For Each ws In ActiveWorkbook.Worksheets

        'Find the sheet that starts with the word "Report"
        If Left(ws.Name, 6) = "Report" Then

            ws.Visible = xlSheetVisible

            'Exit the loop after the first sheet is found
            Exit For

        End If
    Next ws

End Sub
```

The ws variable retains the reference to the worksheet after the loop is exited early, and can be used again in the code below the loop.



Variable Not Required After Next Keyword

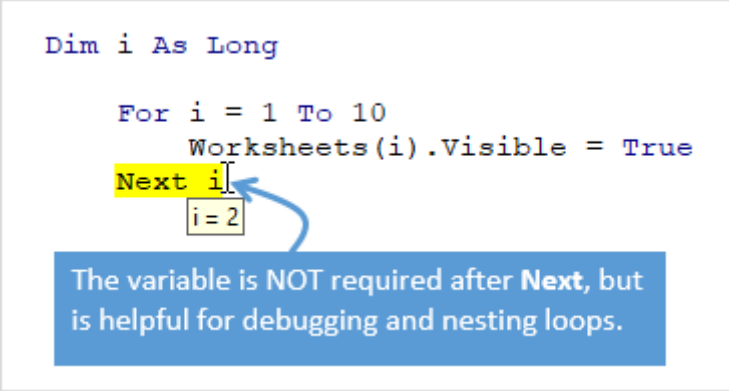
You might have noticed that I added the variable after the Next keyword at the bottom of the loop in the examples above.

```
Next ws Next i
```

This is NOT required, and you might not see it in other examples you find on the web. However, I like to include the variable after Next for two reasons.

1. We can use it when debugging code to see the value of the variable by hovering the mouse over the variable when the code is stopped.
2. It makes it easier to understand which For line the Next line is connected to. This is especially true when you have multiple loops or nested loops in your macros.

Therefore, I recommend adding the variable after the Next keyword as a best practice. A little extra work up front will save time & headache in the future. Trust me! 😉



Macro Code Examples of VBA For Loops

Here are additional articles with macros that use at least one For Next Loop.

3 Ways to Unhide Multiple Sheets in Excel + VBA Tutorial

Automatic Default Number Formatting in Excel Pivot Tables

3 Tips to Save and Close All Open Excel Workbook Files + Macro

The SUBTOTAL Metrics Macro – Create a Summary Table of All Function Types

How to Add a Table of Contents Image Gallery Sheet to Your Excel Files

Hide & Unhide (Filter) Columns with a Slicer or Filter Drop-down Menu

Filter a Pivot Table or Slicer for the Most Recent Date or Period

How to Make Your Excel Dashboards Resize for Different Screen Sizes

Convert Pivot Table to SUMIFS Formulas + Free VBA Macro

VBA Macro to Hide All Columns That Contain a Value in a Cell

How to Repeat Tasks with VBA Code – Looping – Great article with lots of examples from my friend Chris Newman at The Spreadsheet Guru.

What Task Do You Want To Loop?

I hope that article helps get you started with loops. Don't forget to download the free Excel file that contains the code samples.

Loops are definitely an intermediate coding technique that force us to go beyond the macro recorder. Unfortunately, the macro recorded cannot create loops. However, this is a skill that you will be able to use over and over again throughout your career to automate simple and complex tasks. Understanding how to use loops will give you magical powers with Excel.

Please leave a comment below with a task that you want to automate with a loop. Thank you! ☺

Please share