**Variables and Data Types**

Before we tell VBA to display a message, we need to give it the exact message we want to display. In programming, all text is just a collection of characters strung together, so they are called **string** data types. This data needs to be stored somewhere with a way for us to reference it. Therefore, we'll tell VBA to clear out some space in the memory for our message, and then give it a name to use in our program. We do this by creating a **variable**.

**Variables** are a fundamental building block of all programming languages. They hold the data in our code. Two important parts of a variable are its **name** and **data type**. Different programming languages have different rules for what you can use for a variable name, but generally speaking, you can name a variable anything that begins with a letter (the first character can't be a number) and isn't already a keyword in the language.

### Data Type Examples

Some common data types are:

* **Integer:** Positive and negative whole numbers between –32,768 and 32,767, stored in 16 bits
* **Long:** Positive and negative whole numbers between –2,147,483,648 and 2,147,483,647, stored in 32 bits
* **Double:** Decimal numbers (i.e., numbers with fractional parts) stored in 64 bits
* **String:** Text
* **Boolean:** True/false values

##### Integer

* Number of trading days in a year
* How many family members a person has
* The floor number in a skyscraper

##### Long

* Number of views for a video on YouTube
* Population sizes of cities

##### Double

* Latitudes and longitudes
* The constant pi
* Interest rates

##### String

* Employee names
* International postal codes
* Movie titles

##### Boolean

* Whether a door is open or closed
* Whether someone is over 18

**If Conditional**

If: must be true in order to enter the block

If: must be true in order to enter the block

Elif: will continue running to see if something else exists

Else: it stops

**Loops** tell a computer to **repeat lines of code over and over** (and over, and over) again. for loops tell the computer to repeat the lines of code a specific number of times. You can think of a for loop like telling the computer to "run this code for as many loops as I tell you to." There are a few different kinds of loops, but for loops are the workhorse of loops. It's entirely possible that you'll never need to use any other kind of loop.

In VBA, the syntax for a for loop has a beginning, middle, and end. The beginning is one line that tells VBA that we're opening a for loop; the middle is the block of code to be repeated; and the end is one line that closes the block of code.

Iterators are named variables that change values over the course of the for loop, usually increasing by 1, thus holding the number of times the loop has repeated. For example, if we wanted a for loop that would loop exactly 3 times, the opening line would look like this:

For i = 1 to 3

Inside the code block, we can use the iterator like any other variable. So, if we want to display 3 message boxes in a row, showing the number of times the loop has repeated, our code block would look like this:

For i = 1 To 3

MsgBox (i)

To tell VBA that the for loop has ended, we use the keyword Next and the iterator we used: i.

For i = 1 To 3

MsgBox (i)

Next i

**Conditionals** tell the computer that certain lines of code should run only under certain conditions. The workhorse of conditionals is an **if-then statement**. The if-then statement checks if a condition is true. If it is, then a block of code will be run.

In VBA, the syntax of an if-then statement also has a beginning, middle, and end. The beginning is one line to open the if-then statement; the middle is a block of code to run if the condition is true; the end is a line that closes the if-then statement.

The opening of an if-then statement uses the keywords if and then, and a condition. The code block can be any number of lines of code. To close the if-then statement in VBA, add the line End If.

The pattern we used follows this general structure:

1. Initialize a variable to hold a sum.
2. Set the variable to zero.
3. Start a for loop.
4. Use a conditional to increase the sum variable by a value.
5. End the loop.

**Logical operators,** also called Boolean operators, link more than one condition together, which allows for more complicated conditional arguments. The logical operators in VBA are And, Or, and Not. That is:

* condition1 And condition2 will only evaluate as true if **both** condition1 and condition2 are true.
* condition1 Or condition2 will evaluate as true if **either** condition1 or condition2 are True.
* Not condition will give the **opposite** value of whatever condition is.

In addition, there is a "not equal to" **comparison operator** that checks whether two values are not equal to each other. In VBA, the "not equal to" operator is two angle brackets:

# **Loop**

[Single Loop](https://www.excel-easy.com/vba/loop.html#single-loop) | [Double Loop](https://www.excel-easy.com/vba/loop.html#double-loop) | [Triple Loop](https://www.excel-easy.com/vba/loop.html#triple-loop) | [Do While Loop](https://www.excel-easy.com/vba/loop.html#do-while-loop)

Looping is one of the most powerful programming techniques. A **loop** in **Excel VBA** enables you to loop through a range of cells with just a few codes lines.

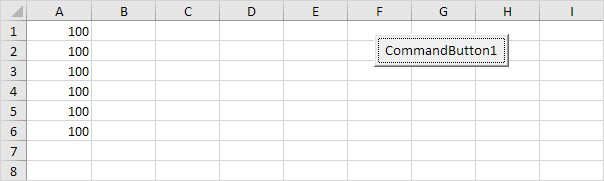
### **Single Loop**

You can use a single loop to loop through a one-dimensional range of cells.

Place a [command button](https://www.excel-easy.com/vba/create-a-macro.html#command-button) on your worksheet and add the following code lines:

Dim i As Integer  
  
For i = 1 To 6  
    Cells(i, 1).Value = 100  
Next i

Result when you click the command button on the sheet:



Explanation: The code lines between For and Next will be executed six times. For i = 1, Excel VBA enters the value 100 into the cell at the intersection of row 1 and column 1. When Excel VBA reaches Next i, it increases i with 1 and jumps back to the For statement. For i = 2, Excel VBA enters the value 100 into the cell at the intersection of row 2 and column 1, etc.

Note: it is good practice to always indent (tab) the code between the words For and Next. This makes your code easier to read.

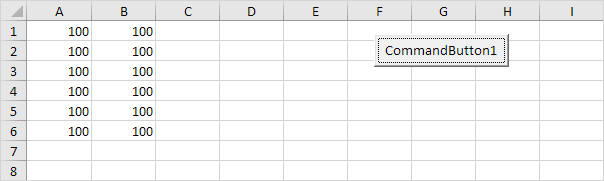
### **Double Loop**

You can use a double loop to loop through a two-dimensional range of cells.

Place a [command button](https://www.excel-easy.com/vba/create-a-macro.html#command-button) on your worksheet and add the following code lines:

Dim i As Integer, j As Integer  
  
For i = 1 To 6  
    For j = 1 To 2  
        Cells(i, j).Value = 100  
    Next j  
Next i

Result when you click the command button on the sheet:



Explanation: For i = 1 and j = 1, Excel VBA enters the value 100 into the cell at the intersection of row 1 and column 1. When Excel VBA reaches Next j, it increases j with 1 and jumps back to the For j statement. For i = 1 and j = 2, Excel VBA enters the value 100 into the cell at the intersection of row 1 and column 2. Next, Excel VBA ignores Next j because j only runs from 1 to 2. When Excel VBA reaches Next i, it increases i with 1 and jumps back to the For i statement. For i = 2 and j = 1, Excel VBA enters the value 100 into the cell at the intersection of row 2 and column 1, etc.

**Arrays in VBA**

In VBA, arrays are initialized with the Dim keyword, but with a couple of key changes:

* Insert a number in parentheses after the array name that represents the number of elements in the array.
* Specify the type of variable for each element in the array.

For example, the code to create an array to hold 12 tickers would be:

Dim tickers(11) As String