



Power Spreadsheets

Become an Excel Power User

Excel VBA Tutorial For Beginners: 16 Essential Terms You Must Know To Learn VBA Programming

By Jorge A. Gomez

So you've created your first (or your first few) Excel macro(s), perhaps by following these [7 easy steps to create a macro](#). By now, your colleagues are already looking at you like you're a wizard.

That is a great sign that you're on a good way to learning macros and Visual Basic for Applications (VBA).

However...

Being able to create a basic macro in Excel is only the beginning in the process to become a really efficient and productive user of macros and VBA. If you really want to unleash the power of these tools, **you must learn VBA** due to the fact that, among others, [recording a macro sometimes simply doesn't "cut it"](#).

The bad news (at least for some of you) is that using VBA requires that you learn programming.

The good news is that programming in Excel is not as difficult as it may sound at first.

If this doesn't sound that convincing, wait...

In order to help you during the process of learning Visual Basic for Applications, I have created this Excel VBA tutorial for beginners where I explain, in detail, 16 (actually you'll probably learn even more)



By the time you're done reading this beginners guide, you will understand the basics of at least 16 essential terms you need to know to learn Visual Basic for Applications. The following is the outline for this Excel tutorial:

Table of Contents [\[hide\]](#)

What Is VBA?

What Is A Macro? The Difference Between Macros And VBA?

What Is VBA Code?

What Is A Module?

What Are Procedures And Routines?

What Is A Statement?

What Are Objects?

What Are Classes?

What Are Collections?

How Are Objects Related To Each Other?

What Is A Property?

What Are Methods?

How Do Properties And Methods Look Like In Excel?

What Are Variables And Arrays?

What Is A Condition?

What Is A Loop?

Macro Example: Horse_Sugar_Cubes

Conclusion

Which other terms do you consider essential for purposes of learning VBA programming?

Books Referenced In This Excel Tutorial

I have prepared two sample Excel workbooks which I use for purposes of explaining these 16 essential terms you need to know to learn VBA programming. **You can get both Excel workbooks for free by [clicking here](#).**

If some (or all) of the terms above sound completely strange to you, don't worry. Before learning VBA, I studied and worked in 2 careers that are infamous for using specialized jargon that is completely incomprehensible to outsiders: law and finance. To make a long story short...Visual Basic for Applications still seemed to me like a completely different language. But then I realized something:

VBA is indeed a different language and, after some time, I realized is not that hard to learn. This brings me to the first question I cover in this Excel VBA tutorial for beginners...

What Is VBA?

VBA is a programming language which was developed by Microsoft and is included in most products that are part of Microsoft Office.

What does this mean exactly?

You can think of a programming language just as you would think of pretty much any other language: English, Spanish, German, French, Italian, Portuguese, Hindi, Mandarin Chinese, Korean, etc. A language has several functions but, for purposes of this Excel VBA tutorial for beginners, I focus on one aspect. As explained in [this SlideShare presentation](#) by Aneshia Beach:

“ Language affords human beings the ability to communicate anything they can imagine.

Programming languages are slightly different in that you generally don't use them to communicate with a human being. You use a programming language, such as VBA, to communicate with a computer. More particularly, you communicate instructions to the computer.

In other words, **Visual Basic for Applications is the language that allows you and me to communicate instructions to Excel.**

So there is no way around it:

In order to be able to automate tasks in Excel and unleash its power, you must learn VBA. However, as you may expect, VBA is different from regular human languages.

More generally, the codes you can use to communicate with your computer are slightly different from those you use to communicate with other people. One of the main reasons for this, as explained by Charles Petzold (one of the seven [Windows Pioneers](#)) in *Code: The Hidden Language of Computer Hardware and Software*, is that despite the recent advances computers can't fully “deal with human codes directly because computers can't duplicate the ways in which human beings use their eyes, ears, mouths and fingers.”

Imagine you are learning a new human language (such as French, Italian or Spanish). One of the subject you need to study is its structure and, quite likely, you come across terms such as nouns, pronouns, verbs, adjectives, and the like. If you hadn't heard those terms before or you're like me and forgot their precise meanings, learning a new language could be quite challenging.

Well...

same as that of human languages, you encounter some special (and different) terms that you must learn.

And **this is where this Excel VBA tutorial for beginners comes in to help. In the following sections, I explain to you some of the most important terms that you need to know to learn VBA programming.** By the end of this guide, you will understand the basic building blocks of Visual Basic for Applications, which will allow you to learn this programming language and become fluent much faster.

As a side note: you can, to a certain extent, make an analogy between some of the word classes (nouns, verbs, adverbs, etc.) in English and some of the components of VBA to help you understand these elements better. If this sounds interesting, you may want to check out *Excel 2013 VBA and Macros*, written by Excel MVP Bill Jelen (Mr. Excel) and Excel consultant Tracy Syrstad, where you can find a concrete example about how the parts of speech of a regular human language are comparable to certain VBA components.

In this particular Excel VBA tutorial for beginners, I use the analogy between VBA components and parts of speech to a certain extent but, where appropriate, I use different illustrations.

What Is A Macro? The Difference Between Macros And VBA?

Excel macros and Visual Basic for Applications are not exactly the same thing, although they are closely related and, sometimes, people use them interchangeably.

As explained above, VBA is a programming language which can be used in several programs that are part of Microsoft Office such as Excel, Power Point, Word and Access.

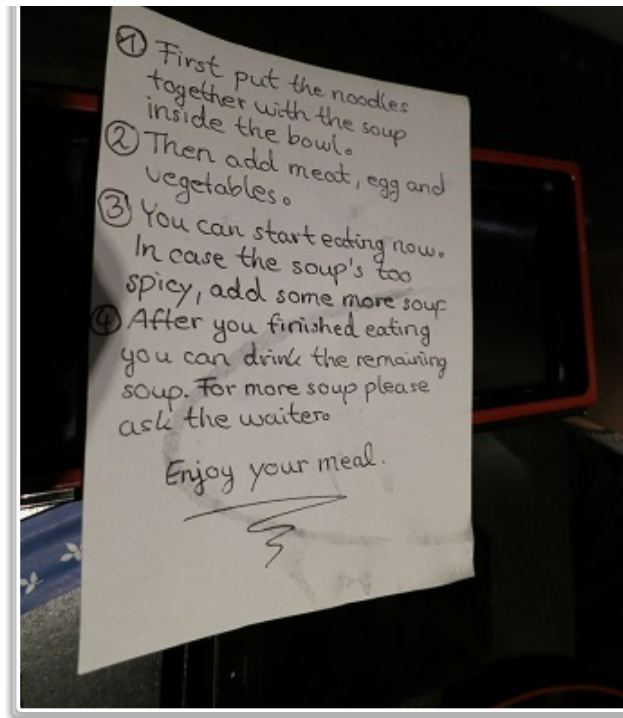
A macro is not a programming language. As explained by John Walkenbach (one of the most outstanding authors on the topic of Excel) in the *Excel 2013 Bible*:

“ A macro is a sequence of instructions that automates some aspect of Excel.

In other words:

- **A macro is the sequence of instructions that you want Excel to follow in order to achieve a particular purpose.**
- **Visual Basic for Applications is a programming language you can use to create macros.**

Take, for example, the following instructions:



The whole set of instructions is the equivalent of an Excel macro. It's a sequence of instructions that should be followed in order to achieve a purpose which, in this case, is enjoying a tsukemen meal.

The language in which the instructions in the image above are written is English. This is the equivalent of Visual Basic for Applications.

So... as you can see: VBA and Excel macros have a very close relationship but, strictly speaking, they are not the same.

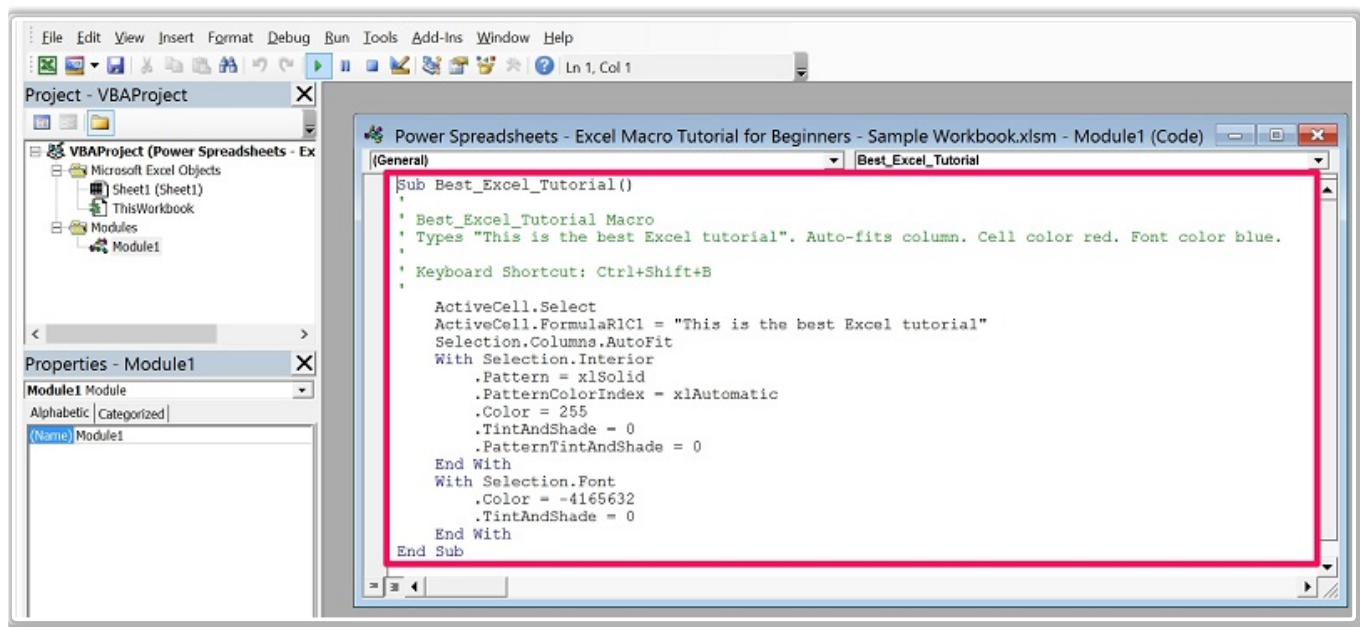
However, as you will see below, it is not uncommon to use certain terms interchangeably with the word macro.

What Is VBA Code?

This is where it may start to get a little bit confusing, but please bear with me...

As explained by John Walkenbach in the *Excel 2013 Bible*, "you perform actions in VBA by executing VBA code." You can generate VBA code in one of the following 2 ways:

- By recording certain actions you perform in an Excel workbook by using the macro recorder.
- By writing the VBA code in the **Visual Basic Editor (VBE)**.



Now, you may wonder... *what is the difference between a macro and VBA code?*

I don't go into too many details (it deviates from the purpose of this Excel VBA tutorial for beginners) but, if you are interested, you can [check out this discussion](#) about the difference between macro code and VBA code in Mr. Excel's forum. For purposes of this guide, there is no essential difference. In fact, as [Microsoft explains here](#), **several programs that are part of Microsoft Office "use the term 'macro' to refer to VBA code."**

Some very prominent Excel professionals and authors do make a distinction between code and macros. For example, when providing some key definitions in the *Excel 2013 Bible*, John Walkenbach distinguishes between code ("VBA instructions that are produced" when recording a macro or entered manually) and macro ("a set of VBA instructions performed automatically").

Now that the definitions of macros, VBA and VBA code are clear (or at least clearer than they were at the beginning), let's start looking at the different components of Visual Basic for Applications.

What Is A Module?

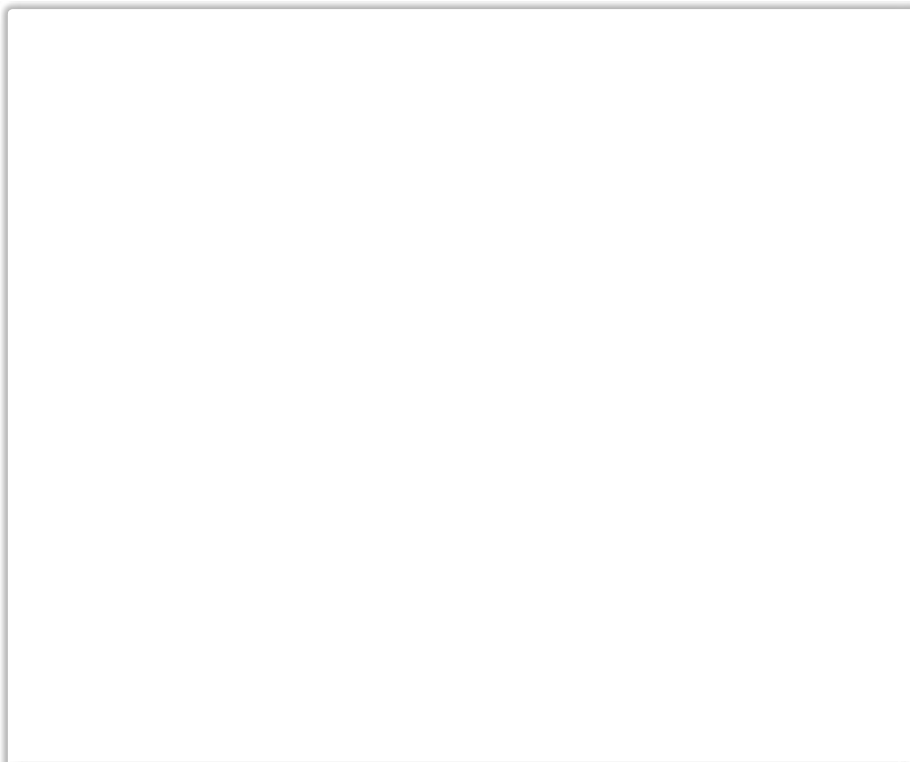
In broad terms, a module is the equivalent of a VBA container. In other words, it is **where Excel actually stores the VBA code**.

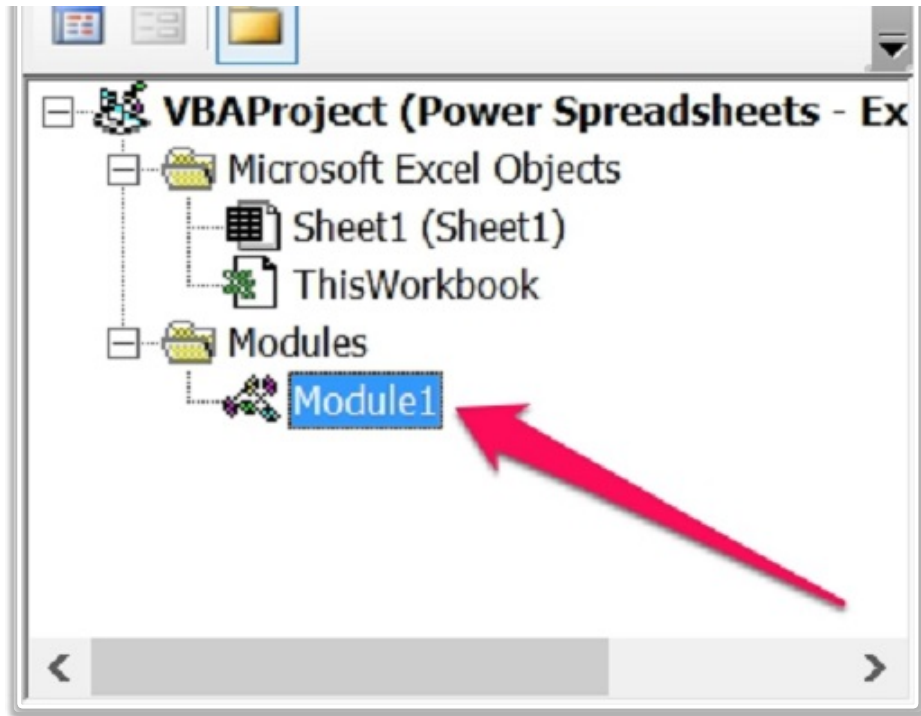
If you have seen a cargo ship or port, or if you have ever been involved in shipping, you may have seen intermodal containers (such as the ones in the image below). These containers are used for, among other purposes, storing goods.



In Excel, the equivalent of the intermodal containers are modules and the goods that are stored are the pieces of VBA code.

You can check out which modules are stored in the Excel workbook you're currently working on in the Project Explorer (which is one of the sections of the Visual Basic Editor). The following screenshot shows an example of the Project Explorer, where there is only 1 standard module (called "Module1").





There are other types of modules in addition to standard modules but I'll cover this topic in future tutorials. Note however that, as explained by [Chip Pearson](#), standard modules are also referred to simply as modules.

Modules are made out of procedures so, as you can imagine, the next logical question is...

What Are Procedures And Routines?

A **procedure** is, basically, the **part of a computer program that performs a particular task or action**.

In more technical terms, a procedure is a block of statements that is enclosed by a particular declaration statement and an End declaration. In *Excel 2013 Power Programming with VBA*, John Walkenbach explains how VBA supports two types of procedures:

- Sub procedures, which perform an action in Excel.

The declaration statement that begins a Sub procedure is "Sub".

For example, the following piece of VBA code (the comments near the top of the image describe its basic purpose) is a Sub procedure. Notice the opening declaration statement, the matching End declaration and how the block of statements is enclosed by these two declarations.




```

' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

```

Declaration statement

End Sub statement

- Function procedures, which **carry out calculations** and return a value.

Sub procedures do not return a value but function procedures can perform certain activities before returning a value.

As explained by Walkenbach in *Excel VBA Programming for Dummies*, the **use of terms such as sub procedure, routine, program, procedure and macro can be a little bit confusing and, generally, they're used interchangeably**. Perhaps the most important distinction you should be able to make is between Sub and Function procedures, as explained above. You can refer to these tutorials on [Sub procedures](#) and [Function procedures](#) for further information..

I imagine that you may have several questions related to the (more technical) definition of procedure given above, so let's take a look at the meaning of one of its key words...

What Is A Statement?

A statement is an instruction. In some contexts, you can distinguish **2 main types of statements**:

1. Declaration statements are, as implied by their name, used to declare something such as a variable or a constant.

When defining what a Sub procedure is, I showed you an example of a declaration statement. In this particular case, this declaration statement is the opening Sub statement which declares the Sub procedure named Best_Excel_Tutorial.

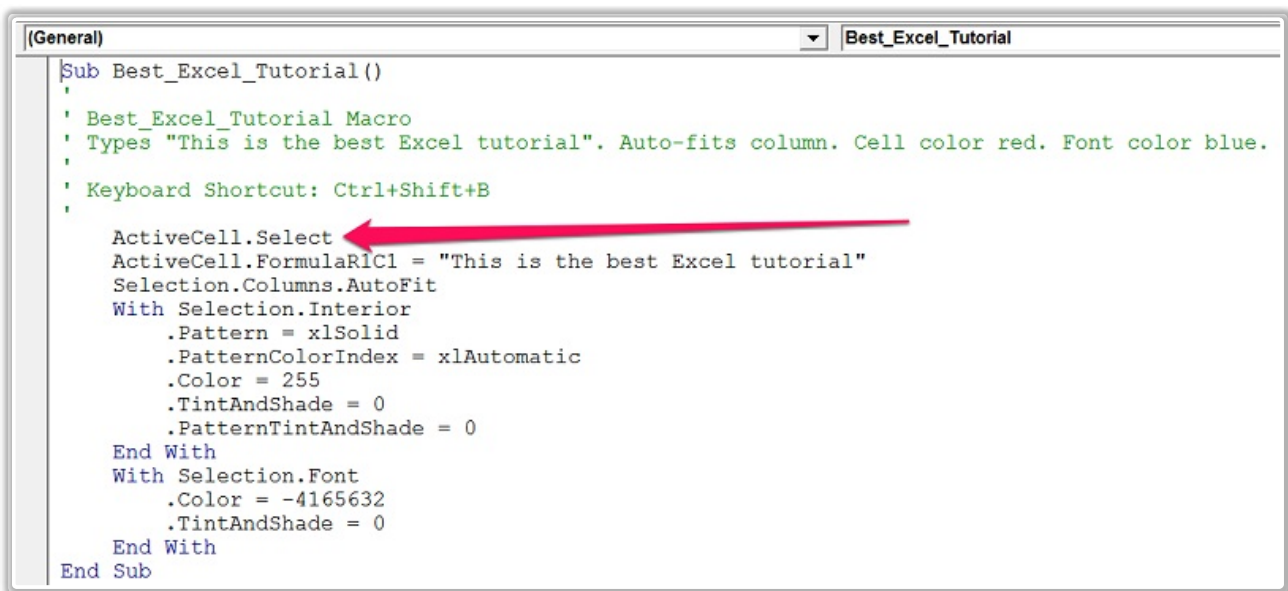
```

' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

```

2. Executable statements are statements that specify that a particular action should be taken.

The Best_Excel_Tutorial macro used as an example above has several executable statements. For example, the statement "ActiveCell.Select" specifies that Excel should select the current active cell.



```

Sub Best_Excel_Tutorial()
' Best_Excel_Tutorial Macro
' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

```

A special type of executable statements are assignment statements. These type of statements, as you can imagine from their name, assign a particular "value or expression to a variable or constant."

What Are Objects?

As you've seen above, procedures perform tasks or actions.

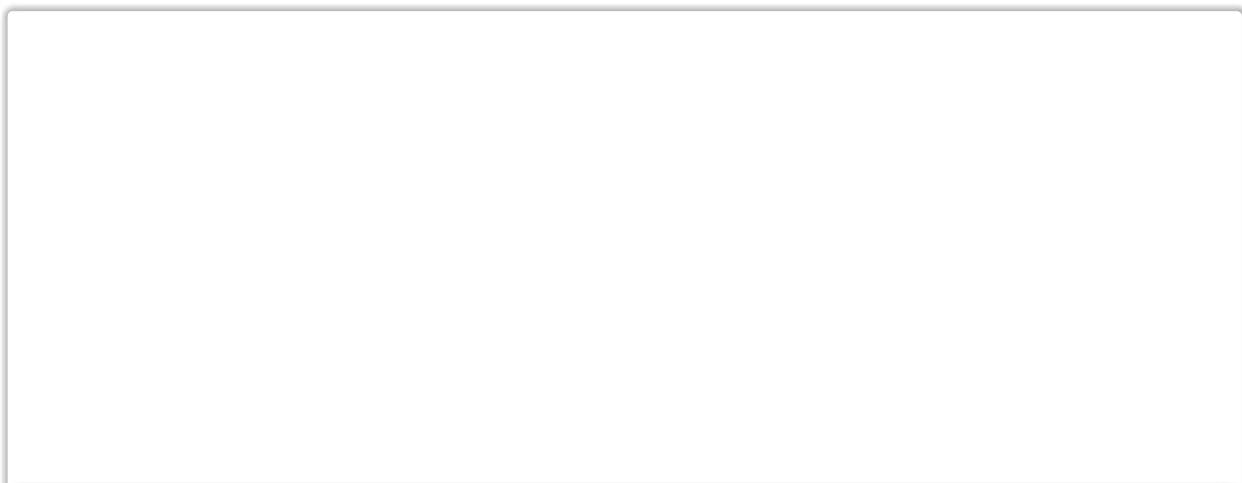
The answer is **objects**. And this is where the analogies between parts of the VBA programming language and English parts of speech used in *Excel 2013 VBA and Macros* comes in handy.

So let's change topics for a second and focus on English grammar. As explained by **Grammar Girl**, in regular English **an object "is having something done to it"**.

In real life, you can find objects anywhere, including the laptop that you use to work on Excel.



Horses are another example of objects.





Actually, since I used to be a real big fan of horses when I was a child, I use them for explanation purposes throughout this Excel VBA tutorial for beginners and have some more horse pictures throughout this guide.

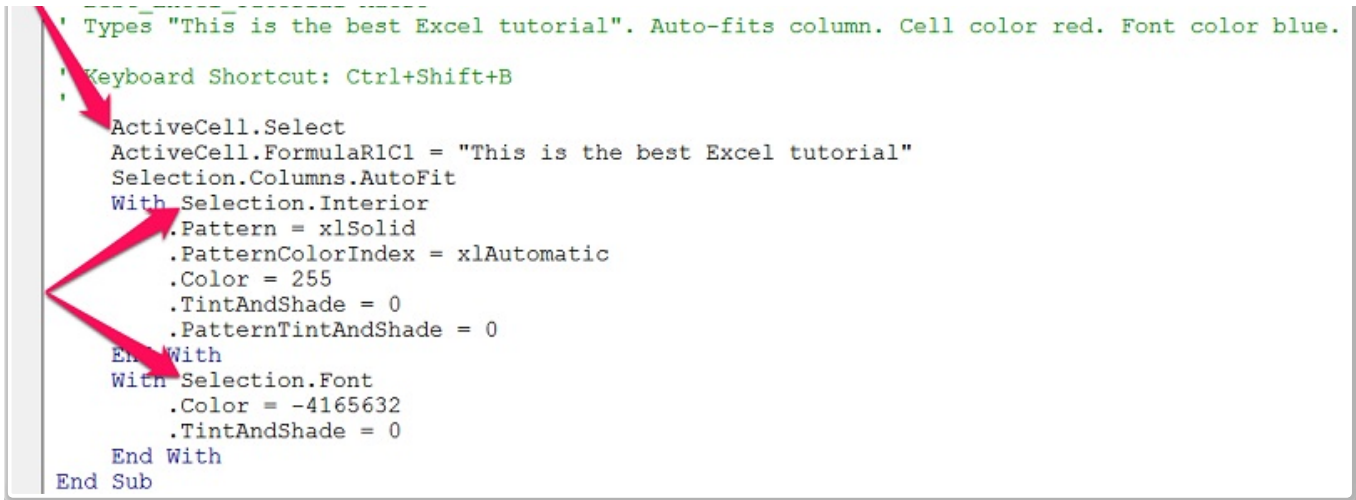
In Visual Basic for Applications things are not very different. This is because, as explained by John Walkenbach in *Excel VBA Programming for Dummies*, **“VBA manipulates objects”**. According to Walkenbach, there are more than 100 classes of objects that can be manipulated using Visual Basic for Applications.

The following are some examples of objects in Visual Basic for Applications:

- Workbooks.
- Worksheets.
- Cell ranges.
- Cells.
- Cell fonts.

Can you spot 2 objects in the Best_Excel_Tutorial macro that has been used as an example?

If you haven't, don't worry; I point them out for you in the following screenshot.

A screenshot of a VBA code editor window. The code is as follows:

```
' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.  
' Keyboard Shortcut: Ctrl+Shift+B  
'  
ActiveCell.Select  
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"  
Selection.Columns.AutoFit  
With Selection.Interior  
    .Pattern = xlSolid  
    .PatternColorIndex = xlAutomatic  
    .Color = 255  
    .TintAndShade = 0  
    .PatternTintAndShade = 0  
End With  
With Selection.Font  
    .Color = -4165632  
    .TintAndShade = 0  
End With  
End Sub
```

Three red arrows originate from the left margin. The first arrow points to the line 'ActiveCell.Select'. The second arrow points to the line 'With Selection.Interior'. The third arrow points to the line 'With Selection.Font'.

As explained by Chandoo (probably the most prominent Excel blogger) [here](#), the ActiveCell and Selection objects which I highlight above are among the most common.

- ActiveCell refers to the current active cell in the current active Excel workbook.
- Selection refers to the currently selected object which, in the example above, is a cell.

Objects are defined by classes, so the next question is...

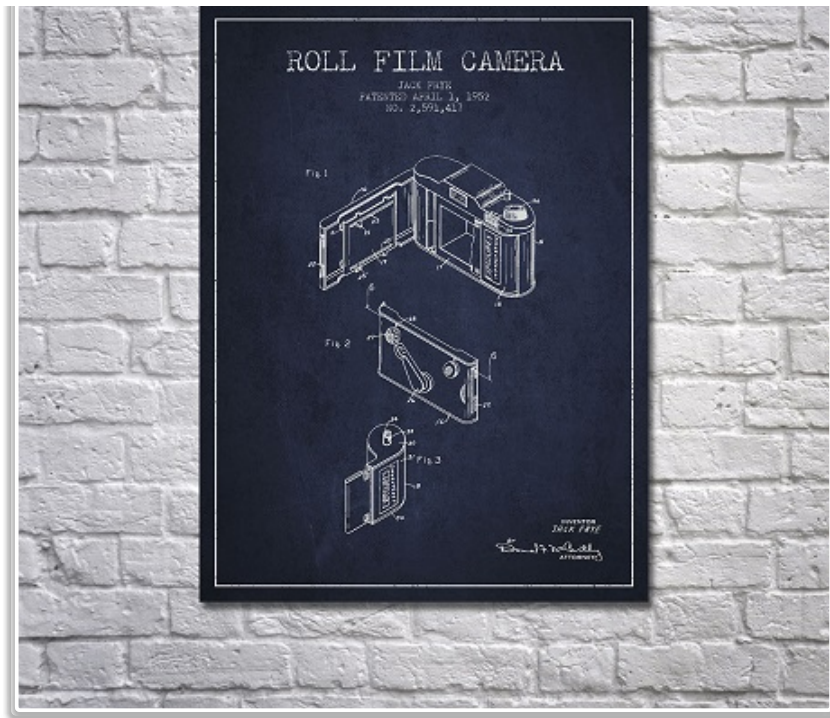
What Are Classes?

As I mention above, classes define objects and, more particularly, classes define the following aspects of an object:

- Variables.
- Properties.
- Procedures.
- Events.

As a consequence of the above, you can think of **objects as instances of classes** or, similarly, you can think about classes as blueprints.

For example, let's assume you used to run a company that produced roll film cameras. That company had a basic blueprint or technical drawing such as the following:



This blueprint defines the characteristics of each of the roll film cameras to be produced and is, therefore, **the equivalent of a VBA class**. Once the company has the blueprint, it can produce the actual cameras.



The actual produced cameras are the equivalent of a VBA object.

object.

What Are Collections?

In Visual Basic for Applications, the word collection refers to collections of objects.

At the basic level, the general use of the word collection doesn't differ too much from the use given to it in VBA. As you'd expect, in very general terms, **a collection is a group of objects, more precisely a group of related objects.**

Therefore, you can use collections to group and manage objects that are related between them.

At a basic level, the concept of collections is relatively simple (I will cover more complicated collection topics in future tutorials) but, if you want a more graphical illustration, check out the following Dr. Seuss collection.



As explained by John Walkenbach in *Excel 2013 Power Programming with VBA*, **collections themselves are objects**, although there is also a collection class.

If collections group objects that have some relationship between them, you may be wondering...

As explained by Microsoft, objects can be related to each other in several ways but **the main type of relationship is of containment**.

Containment relationships are present when objects are put within a container object. This means that objects can contain other objects within them, such as the plastic container (an object) that holds the Dr. Seuss books (other objects) in the image above.

A very good example of a containment relationship is a collection of objects.

Another important type of relationship is hierarchical, which is mostly applicable to classes. Hierarchical relationships occur when a class is derived from a more fundamental class.

What Is A Property?

Objects have properties. These are **the attributes, characteristics or qualities that can be used to describe the object**. I cover the topic of properties in [this Excel tutorial](#).

As explained by John Walkenbach in *Excel VBA Programming for Dummies*, Visual Basic for Applications allows you to both determine and change the properties of a particular object.

Let's take, for example, this horse:



You get the idea.

In addition to having properties, objects have methods. As you may expect, the next question I answer is...

What Are Methods?

To understand what methods are, let's go back to English grammar.

As I explain above when defining the term "object", an object has something done to it. **The method is the "something" which is done to the object.** In the words of Walkenbach in *VBA Programming for Dummies*:

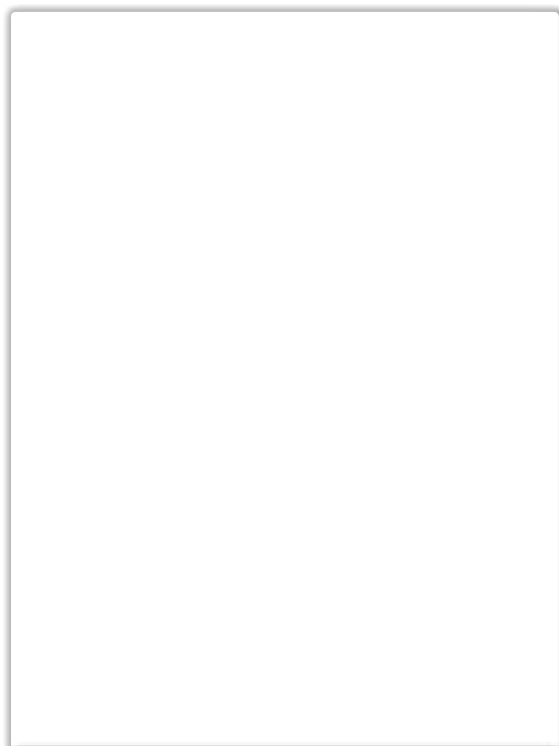
" A method is an action Excel performs with an object.

In grammatical terms, and as illustrated by Bill Jelen and Tracy Syrstad in *Excel 2013 VBA and Macros*, a method (in VBA) is roughly the equivalent of a verb. **I cover the topic of methods in more detail [here](#).**

Let's continue using horses to illustrate the meaning of these Visual Basic for Applications components:

What is an example of a method (verb) that could be applied to a horse?

How about horseback riding?

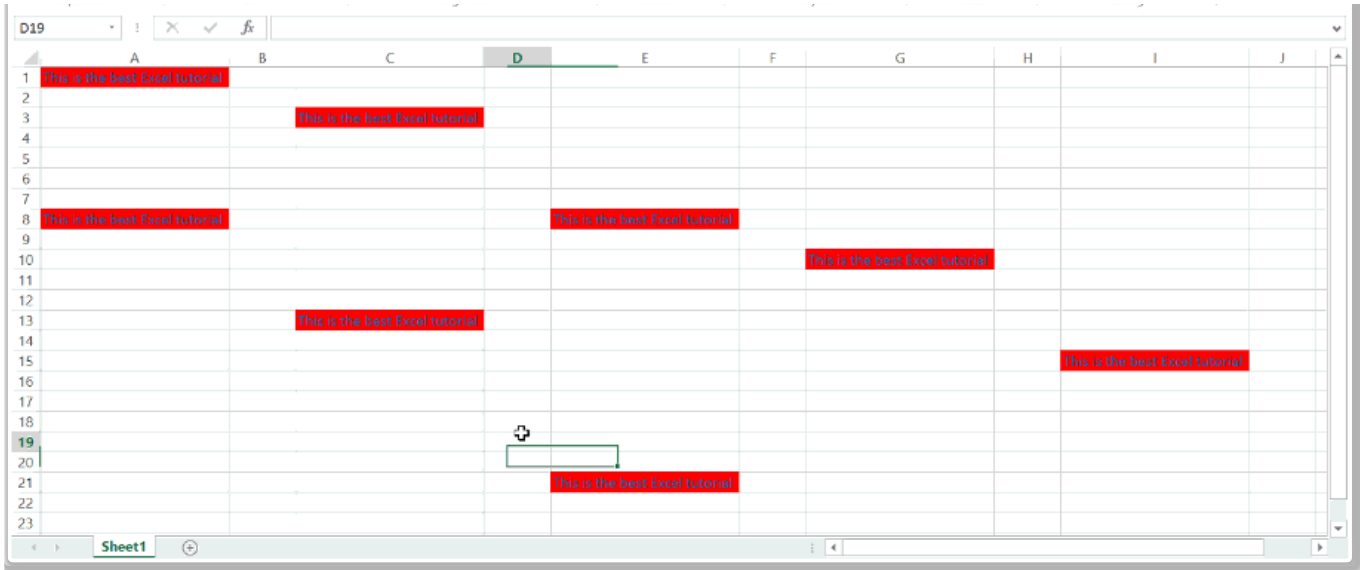




How Do Properties And Methods Look Like In Excel?

Let's go back to the Best_Excel_Tutorial macro that I have been using as an example in this particular guide for beginners. If you run that macro, Excel does the following:

- Types "This is the best Excel tutorial" into the active cell.
- Auto-fits the column width of the active cell.
- Colors the active cell red.
- Changes the font color of the active cell to blue.



Now that you know what is a property and what is a method:

Would you be able to distinguish which of the above make reference to a property and which to a method?

To answer this question, let's take a look again at the VBA code:

```

(General) | Best_Excel_Tutorial
Sub Best_Excel_Tutorial()
'
' Best_Excel_Tutorial Macro
' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
'
' Keyboard Shortcut: Ctrl+Shift+B
'
    ActiveCell.Select
    ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
    Selection.Columns.AutoFit
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .Color = 255
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
    With Selection.Font
        .Color = -4165632
        .TintAndShade = 0
    End With
End Sub

```

And let's take a closer look at the relevant lines to determine whether they make reference to a property or a method:

- "ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"" tells Excel to write "This is the best Excel tutorial" in the active cell.

```

' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

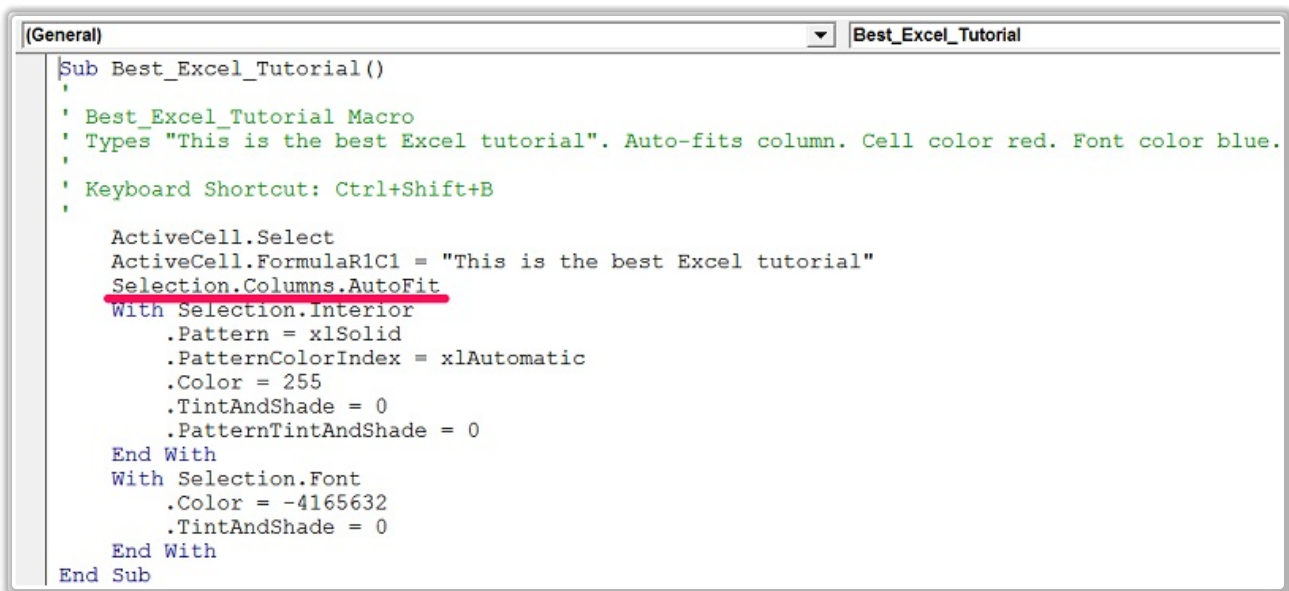
```

ActiveCell returns an object. More precisely, it returns the current active cell.

So what is FormulaR1C1? A property or a method?

Property. More precisely, **Formula R1C1 sets the formula** for ActiveCell.

- "Selection.Columns.AutoFit" instructs Excel to auto-fit the column of the active cell.



```

Sub Best_Excel_Tutorial()
'
' Best_Excel_Tutorial Macro
' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

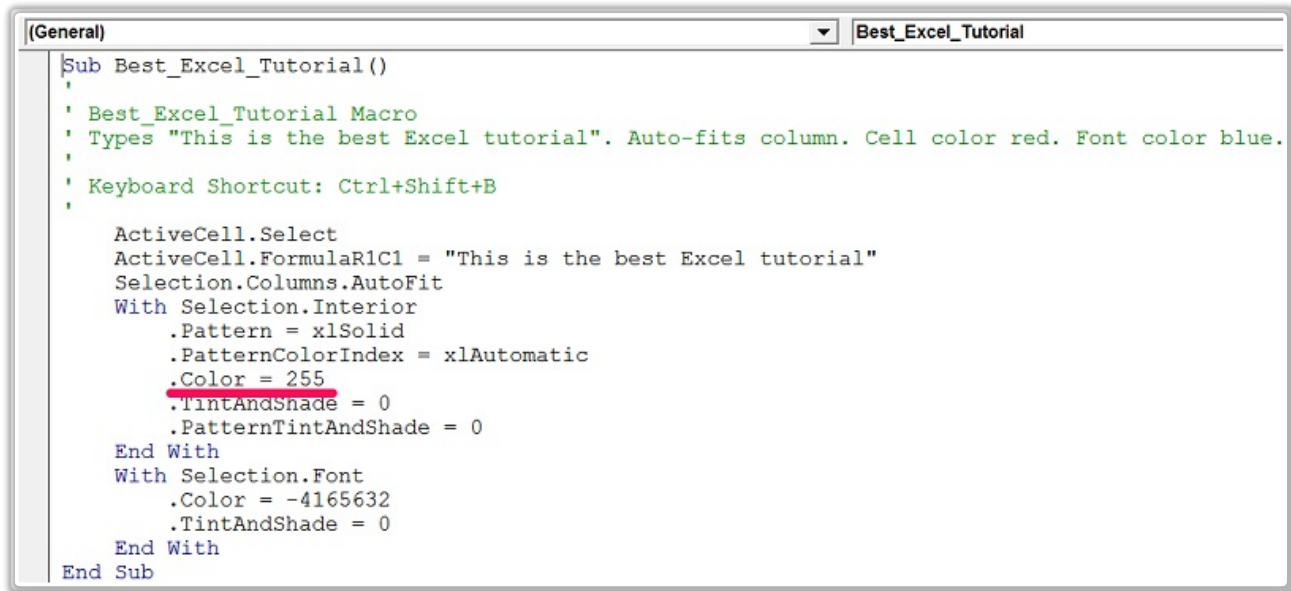
```

In this case, the object is represented by Selection.Columns which represents **the column of the current active cell**.

So what is the remaining part of the statement? Is AutoFit a property or a method?

If you answered method, you are correct. **AutoFit is changing the width** of the relevant column to

- The part of the first **With... End With Statement** that actually sets the fill color of the active cell is `".Color = 255"`.



```
Sub Best_Excel_Tutorial()  
    ' Best_Excel_Tutorial Macro  
    ' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.  
    ' Keyboard Shortcut: Ctrl+Shift+B  
    '  
    ActiveCell.Select  
    ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"  
    Selection.Columns.AutoFit  
    With Selection.Interior  
        .Pattern = xlSolid  
        .PatternColorIndex = xlAutomatic  
        .Color = 255  
        .TintAndShade = 0  
        .PatternTintAndShade = 0  
    End With  
    With Selection.Font  
        .Color = -4165632  
        .TintAndShade = 0  
    End With  
End Sub
```

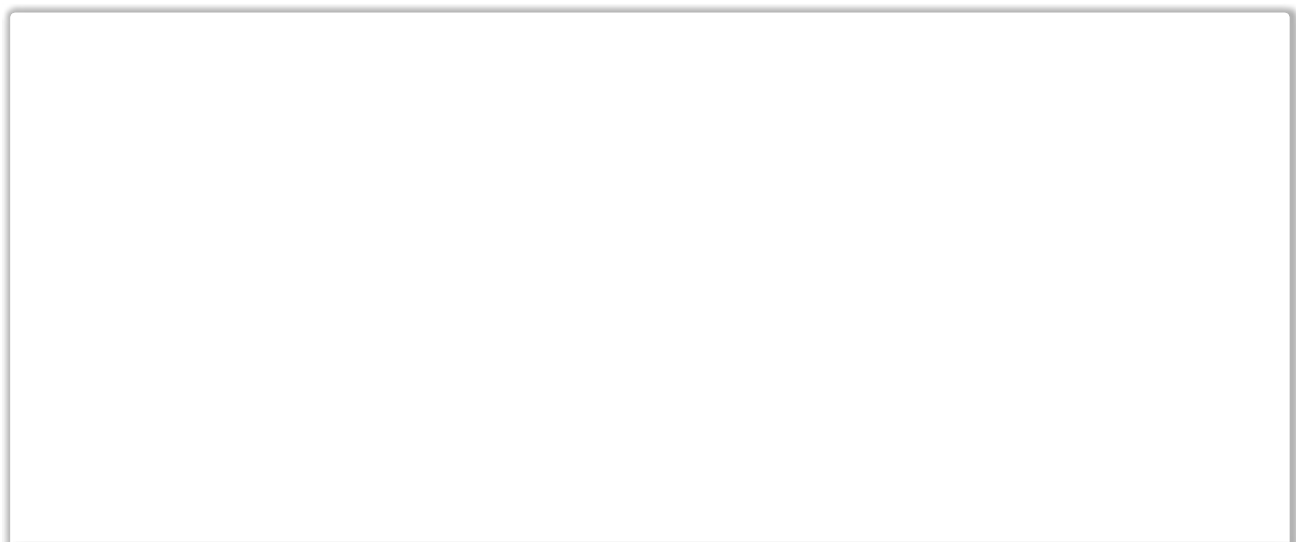
All the individual statements within this With... End With statement (including `".Color = 255"`) refer to `Selection.Interior`. This is the interior of the current selection (in this case, the active cell).

You know which question is coming:

Is Color a property or a method?

Since **Color is setting the main color** of the interior of the active cell, the answer is property.

- Finally, the part of the second With... End With Statement that determines the font color is `".Color = -4165632"`.



```

' Types "This is the best Excel tutorial". Auto-fits column. Cell color red. Font color blue.
' Keyboard Shortcut: Ctrl+Shift+B
'
ActiveCell.Select
ActiveCell.FormulaR1C1 = "This is the best Excel tutorial"
Selection.Columns.AutoFit
With Selection.Interior
    .Pattern = xlSolid
    .PatternColorIndex = xlAutomatic
    .Color = 255
    .TintAndShade = 0
    .PatternTintAndShade = 0
End With
With Selection.Font
    .Color = -4165632
    .TintAndShade = 0
End With
End Sub

```

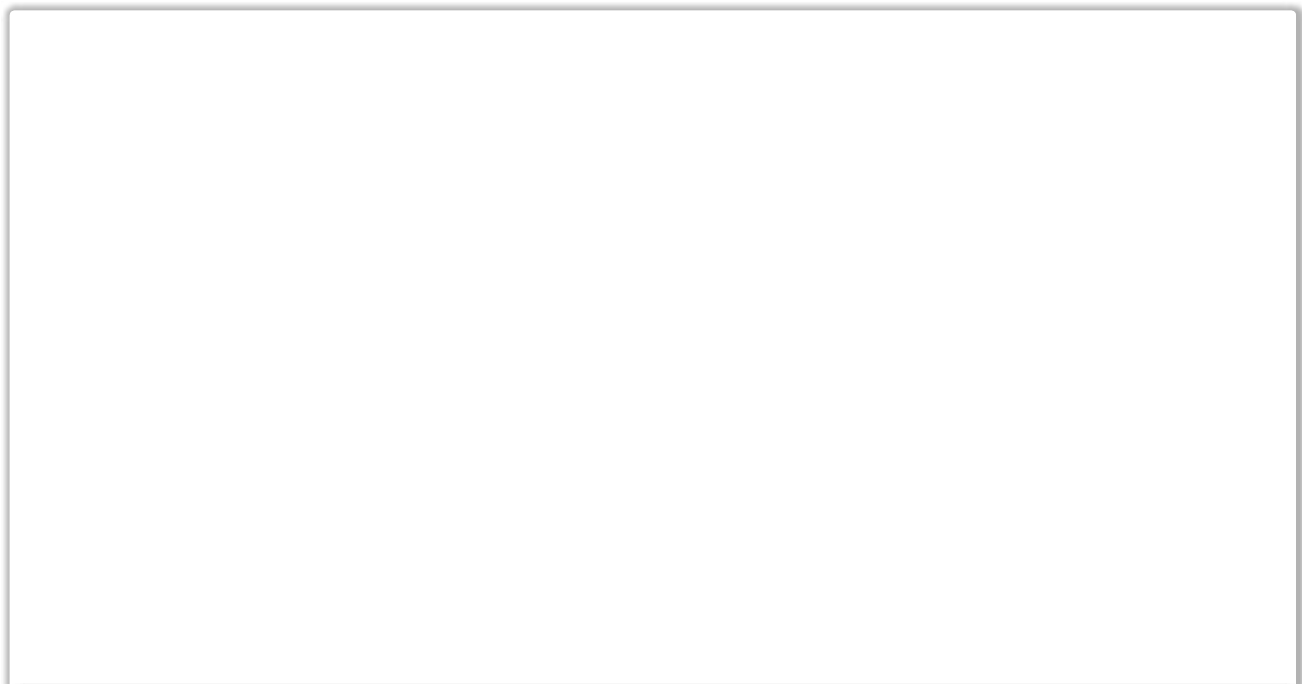
This is, for purposes of this Excel VBA tutorial for beginners, substantially the same case as above. In this case reference is made to Selection.Font which, in this example, is the font of the text in the active cell.

And, as you already know that Color is a property, I won't ask you again 😊 .

What Are Variables And Arrays?

In computer science, **a variable is a storage location** that you pair with a name and use to represent a particular value. That value is stored in the computer's memory.

In other words, you use **variables as placeholders** for particular values. You can think about variables as **envelopes**.





How can you use envelopes to store information?

You can, for example:

- Put some information inside an envelope. This is the content of the envelope or, for programming purposes, the value of the variable.
- Put a name to the envelope. This is the name of the envelope or variable.

Now, let's imagine that you need to tell somebody to get the information that is inside a particular envelope. You can describe the information that you need in either of the following ways:

- By describing the information itself. In this case, the person that is helping you has to open each and every envelope to check out their contents.
- By mentioning the name of the relevant envelope. In this case, the person helping you doesn't need to open each envelope to know where the piece of information they need to get is.

Can you picture why referring to the name of the envelope, instead of the information itself, can be a more efficient option?

But let's get back to the topic that really matters:

I bet you were not expecting that 🍬 ... but let's imagine that you own one of the horses in the images above and a very important part of your horse's diet is sugar cubes.



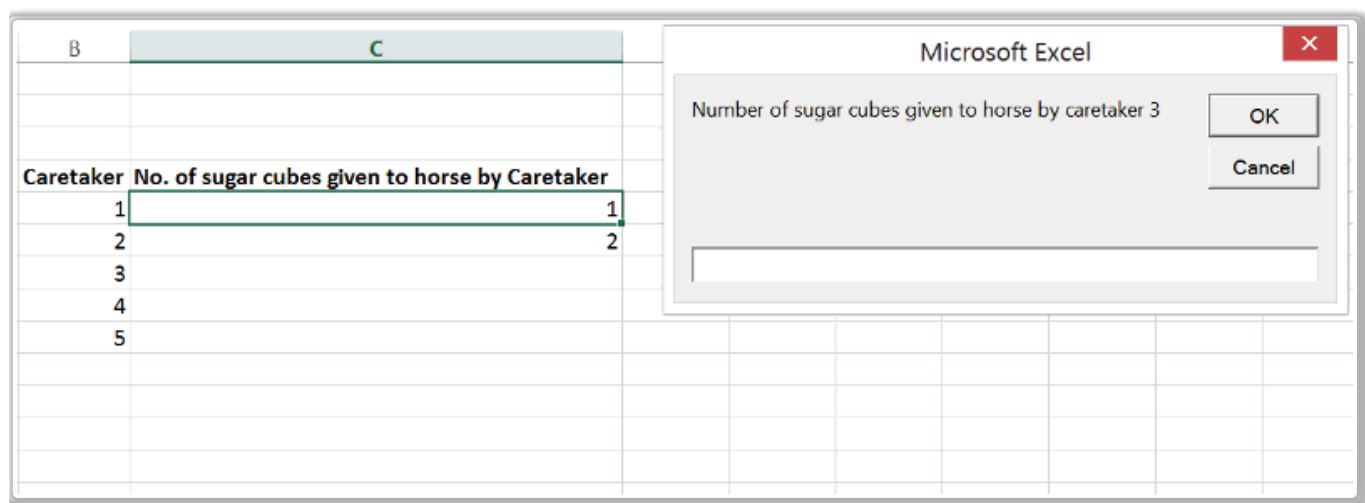
The horse has 5 different caretakers and, each day, he should eat between 5 and 10 sugar cubes. In order to guarantee that this is the case, you set the following rule: each caretaker should give 1 or 2 sugar cubes per day to the horse.

At the beginning of each day, each caretaker needs to report how many sugar cubes he has given to the horse the day before. This report is done by filling the following Excel table.

Caretaker	No. of sugar cubes given to horse by Caretaker
1	
2	
3	
4	
5	

- Asks each caretaker how many sugar cubes they have given to the horse.
- If any of the caretakers has not followed the rule that requires him to give 1 or 2 sugar cubes to the horse, issues a reminder.

Before we take a look at the actual VBA code, check out how the Horse_Sugar_Cubes macro works in practice:



You can get the Excel workbook that contains the Horse_Sugar_Cubes macro for free by [clicking here](#).

Let's start the set up of the Horse_Sugar_Cubes application. For these purposes, you may want to use the following two variables:

- A variable that stores the number of sugar cubes given by a particular caretaker to the horse. You can name this variable `sugarCubes`.
- A variable that stores the identification number of the caretaker. You can name this variable `caretakerNumber`.

How do you create these variables?

To create a variable in Visual Basic for Applications, you must declare it. When you declare a variable, you determine what is the name and what are the characteristics of the particular variable, and tell the computer to allocate some storage space.

You declare a variable in VBA by using the **Dim statement**. I explain the topic of variable declaration in [this Excel tutorial](#). For the moment, bear in mind the following:

For example, you can declare a variable at the top of a module. This variable is known as a module-level variable and exists as long as the module is loaded. Additionally, they're available for use in any procedure within the relevant module.

You can also create a variable with a more limited reach, by declaring the variable within a procedure. In this case, this variable is known as a procedure-level variable. Procedure-level variables can only be used within the relevant procedure in which they have been declared.

- Since you use variables to store different types of data, you can define **different types for a variable**. You do this by using the As keyword.

Some examples of types that you can specify for a variable are Integer, Boolean, String and Range.

Let's take a look at how you can declare the variables caretakerNumber and sugarCubes in practice.

```
Dim caretakerNumber As Integer
Dim sugarCubes As Range
```

I come back to the topic of variables further down this Excel VBA tutorial for purposes of explaining how you can use variables in the VBA macro to help you keep track of how many sugar cubes are given to the horse each day.

As a final note, allow me to make **a brief introduction of arrays**. I provide **a more detailed introduction to arrays in this VBA tutorial**.

Variables that contain a single value are known as scalar variables. You use scalar variables **when working with a single item**.

What do you do if you are working with a group of items that are related to each other?

In these cases, you use arrays. **Arrays are sets of indexed elements that share the same data type and have a logical relationship between them**. The function is substantially the same as that of a variable: holding values. The main difference is that **arrays can store several values** while scalar variables can hold only one value.

When you use an array, you refer to the different elements of the array using the common name and distinguish among them with a number (called a subscript or index). For example, if you had a group of 10



Let's continue with the study of the main VBA components that appear in the Horse_Sugar_Cubes application by understanding...

What Is A Condition?

A **condition** is a statement or expression that **evaluates to either true or false**. Then, **depending on whether the statement has evaluated to true or false, Excel executes (or doesn't execute) a group of statements**.

As explained by **David Malan** (Professor of Computer Science at Harvard University), a condition can be seen as:

“ Something that must be true in order for something to happen.

Can you think of a way you can apply a conditional statement in the VBA application you are developing to keep track of how many sugar cubes are given to your horse by the caretakers?

Hint: conditional statements often use the if-then structure.

- If any of the caretakers doesn't follow the rule that requires him to give 1 or 2 sugar cubes to the horse...
- Then the Horse_Sugar_Cubes macro issues a reminder.

If you're an Excel user, you may have noticed that conditions are not exclusive to VBA programming. For example, several **Excel functions such as the IF function** allow you to check whether a condition is true or not and, based on the result, do one thing or another. Additionally, you can use **other functions such as ISNUMBER** to perform logical tests.

There are several ways to structure conditional statements in Visual Basic Applications and I will cover them in future tutorials. However, for purposes of the Horse_Sugar_Cubes application, you can use an **If...Then...Else statement**.

How does this look inside the Visual Basic Editor?

You can use set up the following If...Then statement for the Horse_Sugar_Cubes macro:

```
If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then  
    MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")  
End If
```

Let's take a look at each of the lines in this code snippet:

- The first line states 2 conditions that can evaluate to either true or false.

This particular line asks Excel to check whether the value of sugarCubes (the amount of sugar cubes that a particular caretaker has given to the horse) is less than 1 **or** more than 2. In other words, this is where Excel confirms whether a particular caretaker has complied with the rule that requires him to give 1 or 2 sugar cubes per day to the horse.

If either of these 2 conditions is true, Excel executes the statement that appears in the second row. If neither of the 2 conditions is true (both are false), Excel doesn't execute the statement in the second line.

- The second line tells Excel what it should do if either of the 2 conditions set in the first line is true.

In this case, if a caretaker has not given any sugar cubes (less than 1) or has given more than 2 sugar cubes to the horse in a particular day, Excel displays a **message in a dialog box** reminding that

- The third line terminates the If...Then...Else block.

You now know how to create the variables that store the number of sugar cubes given to the horse by each caretaker and the identification number of each caretaker. You also know how to get Excel to remind a caretaker of the rule that states they should give 1 or 2 sugar cubes per day to the horse in case they haven't done so.

You only need one extra VBA component to complete the basic structure of your Horse_Sugar_Cubes macro:

How do you get Excel to ask each of the 5 caretakers how many sugar cubes he has given to the horse?

I help you answer this question in the following section.

What Is A Loop?

Loops are statements that:

- Are specified once, but...
- Are carried out several times.

In other words, **a loop is a particular statement that makes a group of instructions be followed multiple times**. Just as with conditional statements, there are several ways in which you can structure loops and I will cover them in future tutorials.

However, for purposes of the Horse_Sugar_Cubes application, you can use a **For Each...Next statement**. This statement asks Excel to execute a group of statements repeatedly for each of the components of a particular group.

In practice, this looks roughly as follows. This is the For Each...Next statement of the Horse_Sugar_Cubes macro that I am using as example in this Excel VBA tutorial for beginners.

```
For Each sugarCubes In Range("C5:C9")
    sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
    If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
        MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
    End If
    caretakerNumber = caretakerNumber + 1
Next sugarCubes
```

Notice that the conditional statement that I explain in the section above is there.


```
    caretakerNumber = caretakerNumber + 1  
Next sugarCubes
```

However, what is more relevant for purposes of this section is the general structure of a For Each...Next statement, so let's take a look at it.

- At the beginning, a For Each...Next statement must say "For Each element of a certain type in the particular group". In the example above, this line is as follows:

For Each sugarCubes In Range ("C5:C9")

The word element refers to the particular items of a collection through which the loop should run. In the case I am using as an example, the elements are sugarCubes.

In this particular case, sugarCubes is defined as a Range object variable. If you're in a situation where the element has not been declared previously, you declare its data type.

The last portion of the statement refers to the group in which the elements are. In this case, this is the range of cells C5 through C9. This is the collection over which the statements inside the loop is repeated.

In other words, Excel applies the set of instructions inside the loop to each of the cells highlighted in the following screenshot.

	A	B	C
1			
2			
3			
4		Caretaker	No. of sugar cubes given to horse by Caretaker
5		1	
6		2	
7		3	
8		4	
9		5	

- The body of a simple For Each...Next statement such as the one in the Horse_Sugar_Cubes macro includes a certain number of instructions that Excel applies to each of the elements in the group (according to what has been stated in the first line).

In the example being used in this Excel VBA tutorial for beginners, the body of statements looks as follows:

```
sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
    MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
End If
caretakerNumber = caretakerNumber + 1
```

The above group of statements is a very simple example. There are structures that are more complicated than this, involving statements (such as Continue For or Exit For) which can be used to transfer the control to different parts of the VBA code.

- The last statement of the For Each...Next statement is of the form "Next element". In the example above, this looks as follows:

Next sugarCubes

This statement simply **terminates the definition** of the loop and tells Excel that, after carrying out the instructions inside the loop, it should move to the next element (in this case, the next sugarCubes).

To summarize, the opening and closing lines of the For Each...Next statement tell Excel that it should run the instructions that are inside the loop for each of the 5 cells where the number of sugar cubes given to the horse by each caretaker are to be recorded.

Macro Example: Horse_Sugar_Cubes

Before I end this Excel VBA tutorial for beginners, let's take a final line-by-line look at the complete VBA code behind the Horse_Sugar_Cubes macro to review some of the essential terms that have been covered in this guide and understand each of the instructions behind the application.

```

' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes
'
' Keyboard Shortcut: Ctrl+Shift+H
'

Dim caretakerNumber As Integer
Dim sugarCubes As Range

caretakerNumber = 1
For Each sugarCubes In Range("C5:C9")
    sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
    If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
        MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
    End If
    caretakerNumber = caretakerNumber + 1
Next sugarCubes

End Sub

```

For ease of reference, here is once more an illustration of how the Horse_Sugar_Cubes macro works in practice:

[illegible]

Let's go through the main items of this macro while making some general comments to further illustrate each of the terms covered in this tutorial:

#1. General aspects.

The Horse_Sugar_Cubes application is written in **Visual Basic for Applications or VBA**, the programming language you can use to communicate instructions to Excel.

Horse_Sugar_Cubes itself is a **macro, a sequence of instructions for Excel to follow**. The code that appears in the screenshot above is an example of VBA code. The terms macro, VBA code, Sub procedure, routine and procedure are sometimes used interchangeably.

The VBA code behind the Horse_Sugar_Cubes application is stored by Excel in **a module, the container where Excel stores the VBA code.**

The Horse_Sugar_Cubes macro has several **statements, or instructions**.

```

Sub Horse_Sugar_Cubes()
    '
    ' Horse_Sugar_Cubes Macro
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes
    ' Keyboard Shortcut: Ctrl+Shift+H
    '

    Dim caretakerNumber As Integer
    Dim sugarCubes As Range

    caretakerNumber = 1
    For Each sugarCubes In Range("C5:C9")
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
        End If
        caretakerNumber = caretakerNumber + 1
    Next sugarCubes
End Sub

```

The first line of code in the screenshot above declares the sub procedure Horse_Sugar_Cubes. **A Sub procedure is** the series of statements that are between the Sub and the End Sub statements and, more precisely, is a **part of a computer program that performs an action**.

The other main type of procedure in VBA are **Function procedures, which carry out calculations and return a particular value**.

The last line of code in the screenshot above terminates the execution of the sub procedure Horse_Sugar_Cubes. Once Excel executes this line, the macro stops running.

#3. Dim caretakerNumber As Integer and Dim sugarCubes as Range.

```

Sub Horse_Sugar_Cubes()
    '
    ' Horse_Sugar_Cubes Macro
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes
    ' Keyboard Shortcut: Ctrl+Shift+H
    '

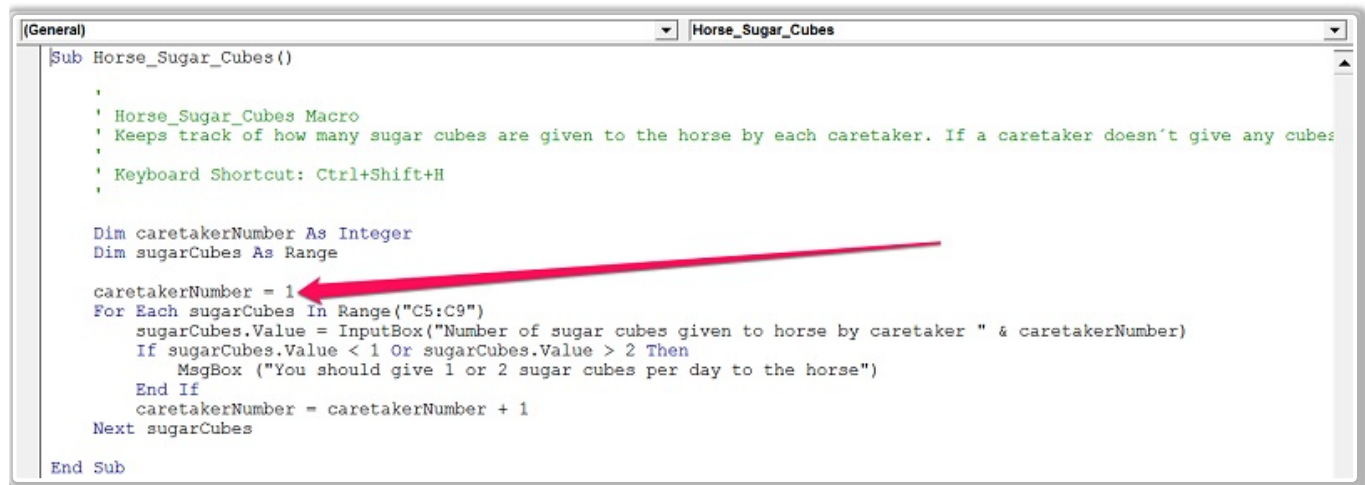
    Dim caretakerNumber As Integer
    Dim sugarCubes As Range

    caretakerNumber = 1
    For Each sugarCubes In Range("C5:C9")
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
        End If
        caretakerNumber = caretakerNumber + 1
    Next sugarCubes
End Sub

```

In Visual Basic for Applications, variables are usually declared using the Dim statement. After this, you can determine the name of the variable and its characteristics. The computer allocates a storage location to the variable and, then, **you can use the declared variable as a placeholder to represent a particular value**.

#4. caretakerNumber = 1.



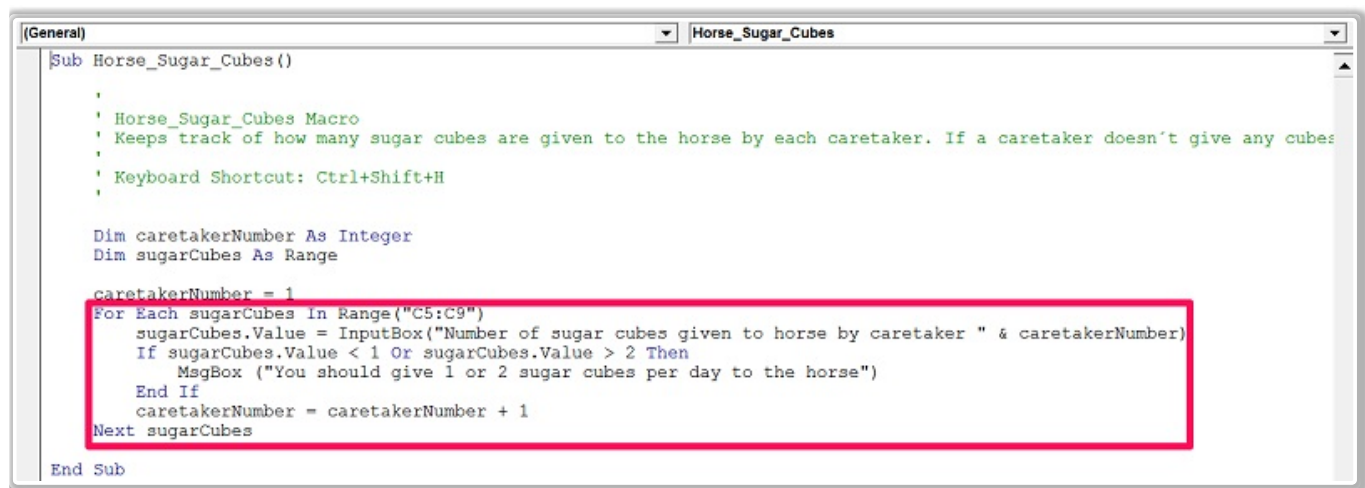
```
(General) Horse_Sugar_Cubes
Sub Horse_Sugar_Cubes()
    '
    ' Horse_Sugar_Cubes Macro
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes
    ' Keyboard Shortcut: Ctrl+Shift+H
    '

    Dim caretakerNumber As Integer
    Dim sugarCubes As Range

    caretakerNumber = 1
    For Each sugarCubes In Range("C5:C9")
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
        End If
        caretakerNumber = caretakerNumber + 1
    Next sugarCubes
End Sub
```

This line is an assignment statement which assigns the value 1 to the variable caretakerNumber. As a consequence of this assignment, every time that the Horse_Sugar_Cubes macro is executed, caretakerNumber is set to the initial value of 1.

#5. For Each...Next statement.



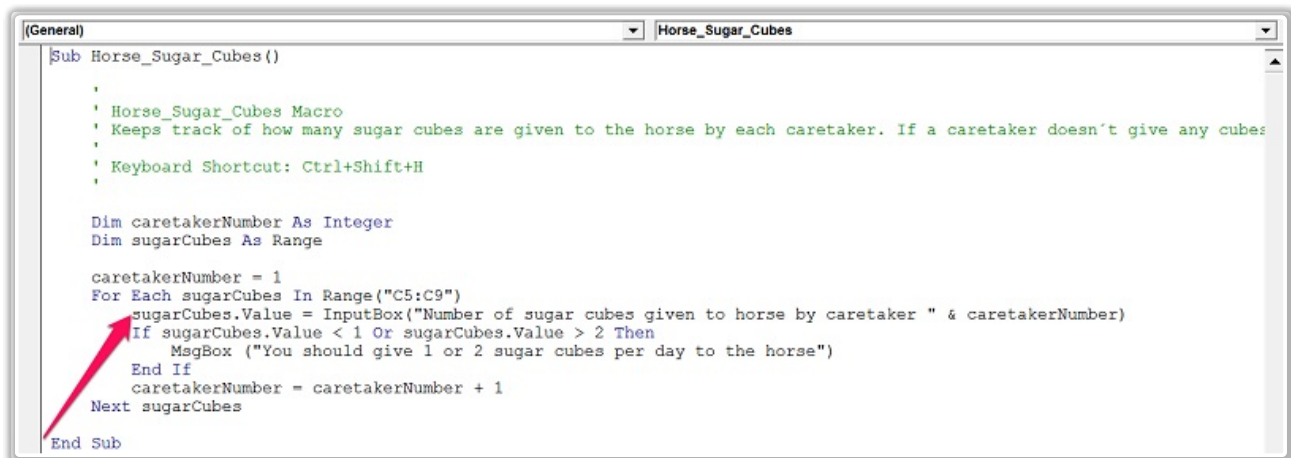
```
(General) Horse_Sugar_Cubes
Sub Horse_Sugar_Cubes()
    '
    ' Horse_Sugar_Cubes Macro
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes
    ' Keyboard Shortcut: Ctrl+Shift+H
    '

    Dim caretakerNumber As Integer
    Dim sugarCubes As Range

    caretakerNumber = 1
    For Each sugarCubes In Range("C5:C9")
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")
        End If
        caretakerNumber = caretakerNumber + 1
    Next sugarCubes
End Sub
```

A For Each...Next statement asks Excel to execute a group of statements repeatedly for each member of a group. This type of statement is one of the simplest ways to implement **a loop, a statement that makes a particular group of instructions be repeated several times.**

In the case of the Horse_Sugar_Cubes macro, the loop asks Excel to repeat the relevant set of instructions for each of the 5 caretakers of the horse. Let's take a look at the body of the For Each...Next statement to understand the set of instructions that is repeated:



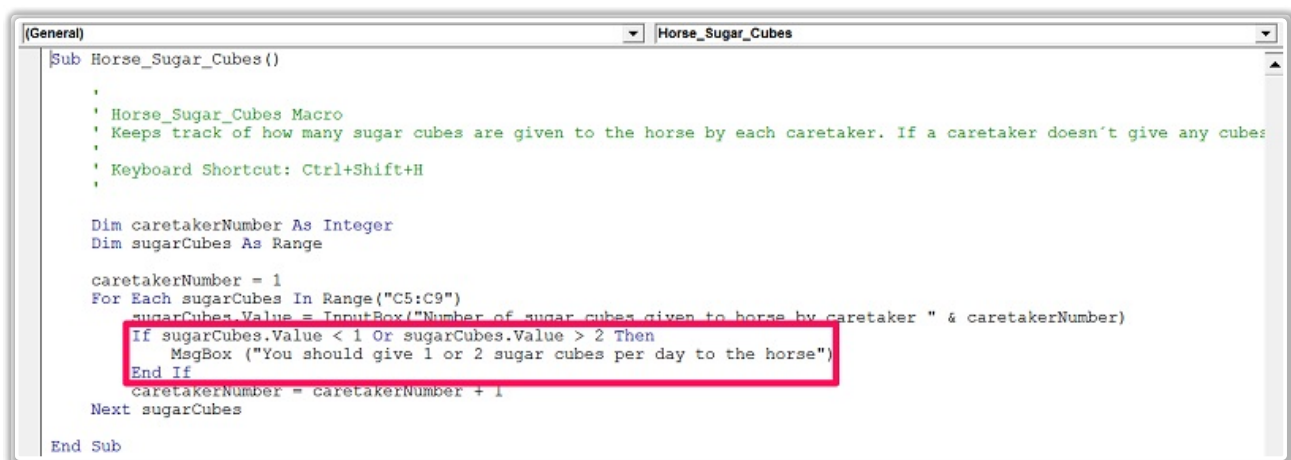
```
Sub Horse_Sugar_Cubes()  
    ' Horse_Sugar_Cubes Macro  
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes  
    ' Keyboard Shortcut: Ctrl+Shift+H  
    '  
    Dim caretakerNumber As Integer  
    Dim sugarCubes As Range  
  
    caretakerNumber = 1  
    For Each sugarCubes In Range("C5:C9")  
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)  
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then  
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")  
        End If  
        caretakerNumber = caretakerNumber + 1  
    Next sugarCubes  
End Sub
```

The first part of the line (sugarCubes.Value =) assigns a value to the variable sugarCubes.

The second part of the statement (InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)) instructs Excel to show a pop-up input box that asks what is the number of sugar cubes given to the horse by each caretaker. The number that is inputted in the box is recorded in the relevant cell of the Excel worksheet and is the value assigned to the variable sugarCubes.

The input box refers to each caretaker by its identification number (1 through 5) by calling the value of the variable caretakerNumber (for example, the first caretaker is referred to as caretaker 1, and so on). As a consequence of the statement immediately above the For Each...Next statement (caretakerNumber = 1), the value of the variable caretakerNumber at the beginning of the process is always 1. I explain further below which statement asks Excel to update the caretaker number for the relevant caretaker.

- **Conditional statement.**



```
Sub Horse_Sugar_Cubes()  
    ' Horse_Sugar_Cubes Macro  
    ' Keeps track of how many sugar cubes are given to the horse by each caretaker. If a caretaker doesn't give any cubes  
    ' Keyboard Shortcut: Ctrl+Shift+H  
    '  
    Dim caretakerNumber As Integer  
    Dim sugarCubes As Range  
  
    caretakerNumber = 1  
    For Each sugarCubes In Range("C5:C9")  
        sugarCubes.Value = InputBox("Number of sugar cubes given to horse by caretaker " & caretakerNumber)  
        If sugarCubes.Value < 1 Or sugarCubes.Value > 2 Then  
            MsgBox ("You should give 1 or 2 sugar cubes per day to the horse")  
        End If  
        caretakerNumber = caretakerNumber + 1  
    Next sugarCubes  
End Sub
```

Conditional statements evaluate a particular condition and, depending on the result (true or false),

The conditional statement in the `Horse_Sugar_Cubes` macro evaluates whether a caretaker has given less than 1 or more than 2 sugar cubes to the horse (therefore not complying with the rule that requires them to give either 1 or 2 sugar cubes per day to the horse). If any of the 2 conditions is met (the number of sugar cubes given to the horse is less than 1 or more than 2), Excel displays a message box with a reminder that states "You should give 1 or 2 sugar cubes per day to the horse".

- **`caretakerNumber = caretakerNumber + 1`.**

This statement increases the value of the variable `caretakerNumber` by 1 for each successive repetition of the `For Each...Next` statement.

Therefore, the second time the set of instructions is repeated by the macro, `caretakerNumber` is equal to 2. The third time, the variable has the value of 3. As you may expect, the values for the fourth and fifth times are 4 and 5 respectively.

Conclusion

You have made it to the end of this Excel VBA tutorial for beginners.

By now, based on what you have learned on this beginners guide, you know the meanings of at least 16 essential terms you need to know to learn VBA programming. You're also able to understand how some of these terms are sometimes used interchangeably and some of the discussions regarding their use.

Additionally, you've seen how these concepts come together to form a macro, and you know how some of the VBA components that have been explained in this Excel VBA tutorial for beginners can be implemented in practice.

I hope that this Excel tutorial for beginners has proved that the most essential terms you need to know to learn VBA programming are not that complicated to understand.

You're likely to find the terms that have been covered in this particular guide for beginners many times during your journey to becoming a VBA expert so you may want to bookmark this post and come back as required during your future Visual Basic for Applications studies. **If you have any doubts or questions regarding any of these terms, please leave a comment below.**

By the way...

Which other terms do you consider essential for purposes of learning VBA programming?

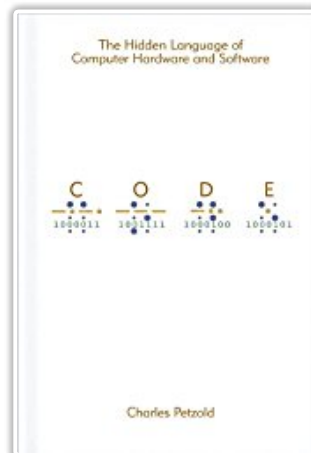
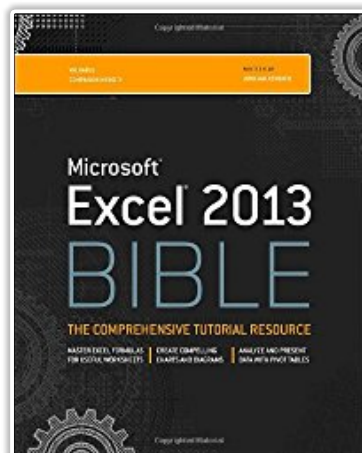
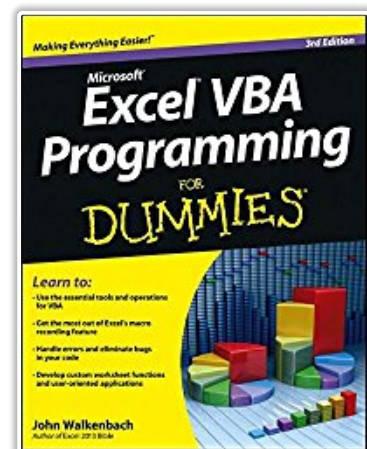
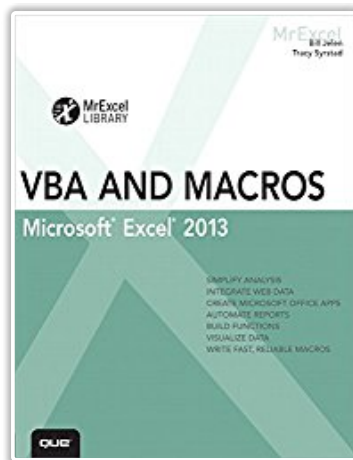
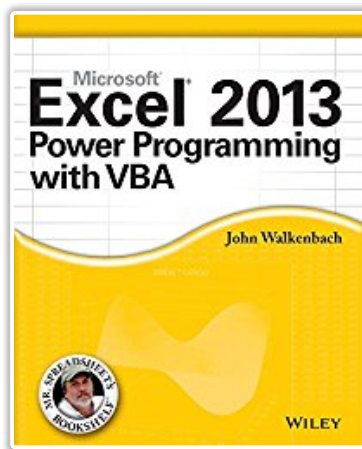
programming? Regardless of whether you're a beginner or an experienced Excel user, **I would love to hear which other terms do you consider essential for those purposes.**

Conversely, *do you think I should've left out one (or more) of the terms that have been explained in this guide?*

Please leave a comment below and let me know your opinion.

Books Referenced In This Excel Tutorial

Click on any of the images below to purchase the book at Amazon.



2 tanners • 6 months ago

Thanks. I'm just starting up again after a break of decades and this was very helpful. Possibly the only thing I would have preferred done different is LOOP before FOR EACH. I'm sure you had your reasons, this is just the way I think. BTW, do you really like horses, or is this your way of getting over a fear of them? :)

^ | ▾ • Share ›

Jorge A. Gomez Mod ➔ 2 tanners • 6 months ago

Many thanks for your comments 2 tanners!

I'm happy to read you've found the post helpful. I appreciate your comment regarding the loop/for each explanations. I'm hoping to be able to come back to some of these posts and update/improve them. I will consider this comment when doing that.

Regarding the horses: I do like them :-)

In reality, the main reason they ended up within the examples in this post was because I found several nice pictures with the Creative Commons License in Flickr while preparing the initial drafts of the post. I then adjusted the post/examples accordingly.

Thanks again for stopping by.

^ | ▾ • Share ›

PL • 7 months ago

thank you so much.

^ | ▾ • Share ›

Jorge A. Gomez Mod ➔ PL • 7 months ago

You're welcome PL. Thanks for visiting and for your nice comment, which I appreciate very much

^ | ▾ • Share ›

Trixi Rosales • a year ago

Wow. You write so clear and well! I had fun. Replace our prof, please? Haha

^ | ▾ • Share ›

Jorge A. Gomez Mod ➔ Trixi Rosales • a year ago

Thanks a lot for your very nice comments Trixi!

I'm very happy to read that you liked the blog post. I look forward to seeing you again in Power Spreadsheets

^ | ▾ • Share ›



Join thousands of Excel Power Users

Receive FREE updates about new
Tutorials and FREE resources that will
help you become an Excel Power User.

Enter your email here...

BECOME AN EXCEL POWER USER

POWER SPREADSHEETS IN SOCIAL MEDIA



Excel Resources | Excel Shortcuts



Contact

commission. This commission comes at no additional cost to you,