

60 Second VBA – Newbie Edition

By Paul The Programmer

©2008 by Paul The Programmer

You have permission to post this, email this, print this and pass it along for free to anyone you like, as long as you make no changes or edits to its contents or digital format. Please feel free to make lots and lots of copies and send them to your friends, work colleagues, classmates etc. The right to bind this and sell it as a book, however, is strictly reserved by me. I'd also like to keep the theatre, TV series and movie rights too. Unless you can get Jim Carey to play me.

60 Second VBA is a trademark of Paul The Programmer

Designed by Paul The Programmer

You can find this eBook, along with lots of other good VBA tutorials, video and other stuff, at Paultheprogrammer.webs.com .

This Newbie version is current until June 2009. After that date, please go to paultheprogrammer.webs.com and get an updated version.

This book is dedicated to Lisa, Paul The Programmers Wife, without whom I'd go to bed much later.

Introduction

Lesson 1. A 60 Second Introduction to VBA

Lesson 2. Where Do I write VBA programs?

Lesson 3. My First Program

Lesson 4. Whats all this Sub business ?

Lesson 5. How to start doing something interesting.

Lesson 6. The Art of Repeating yourself

Lesson 7. Loops with strings attached

Lesson 8. Selecting the right thing to do

Lesson 9. More variables than a Unicycling Juggler

Lesson 10. Being Invited to Functions

Introduction

There's nothing to it.

Visual Basic for Applications (hereby known as VBA, because I'm too lazy to keep writing it longhand) is a simple language that simple people (just like me) can learn and use to speed up our Microsoft Office documents (e.g. spreadsheets, Access databases); add extra functionality; and remove repetitive tasks – in theory.

This is the first eBook in the series, written as a number of 60 second bite-sized lessons that will help the newbie (or novice if you prefer) to get started understanding a little VBA code, such as how to put messages on screen, how to store information and how to get a calculated result. I don't cover the science behind programming languages, how the computer understands the instructions you send it – there are many books on those subjects out there, which are as stale as a week old slice of bread.

I just want you to get quickly up to speed, get easy to understand results and learn by doing.

If you like what you read, please pass on this book to other people. If you don't like it, then please tell me – and I'll cry into my Cocoa as I lay awake at night worrying.

All the best, and good luck, Paul The Programmer

Paultheprogrammer.webs.com or follow me on twitter <http://twitter.com/paulthehacker>

Lesson 1. A 60 Second Introduction to VBA

VBA is short for Visual Basic for Applications or, as I like to call it, very basic.

VBA is extremely simple to understand and learn, unless your complex, in which case it isn't.

VBA can carry out complex and repetitive tasks, or 'bugs' as most programmers prefer to call them.

VBA can't carry your shopping home on the bus, or drive your car.

VBA is a computer language, easier to understand than Bulgarian (believe me I'm trying to).

VBA - think of it as an ordered shopping list of 'To Do' instructions for the computer, without the eggs, milk and bread.

VBA is built into Microsoft Office programs, like Excel, Word and Access, and it's how you get them to talk with one another. Unless they've fallen out.

VBA is written using an editor program. This will point out your mistakes. It would also be nice if it corrected them too, but that up to you, my friend.

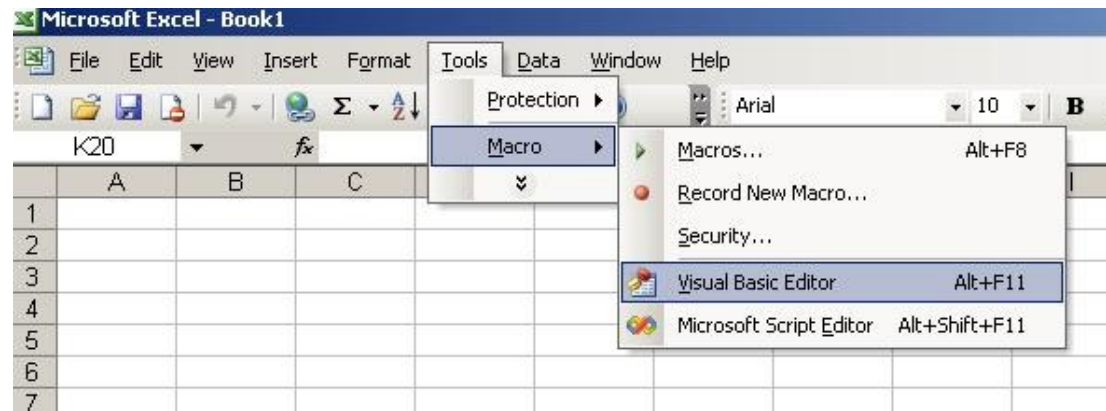
Lesson 2. Where Do I write VBA programs?

So you now know what VBA is, and roughly what it might be able to do. Great. But where do we actually create the list, or code.

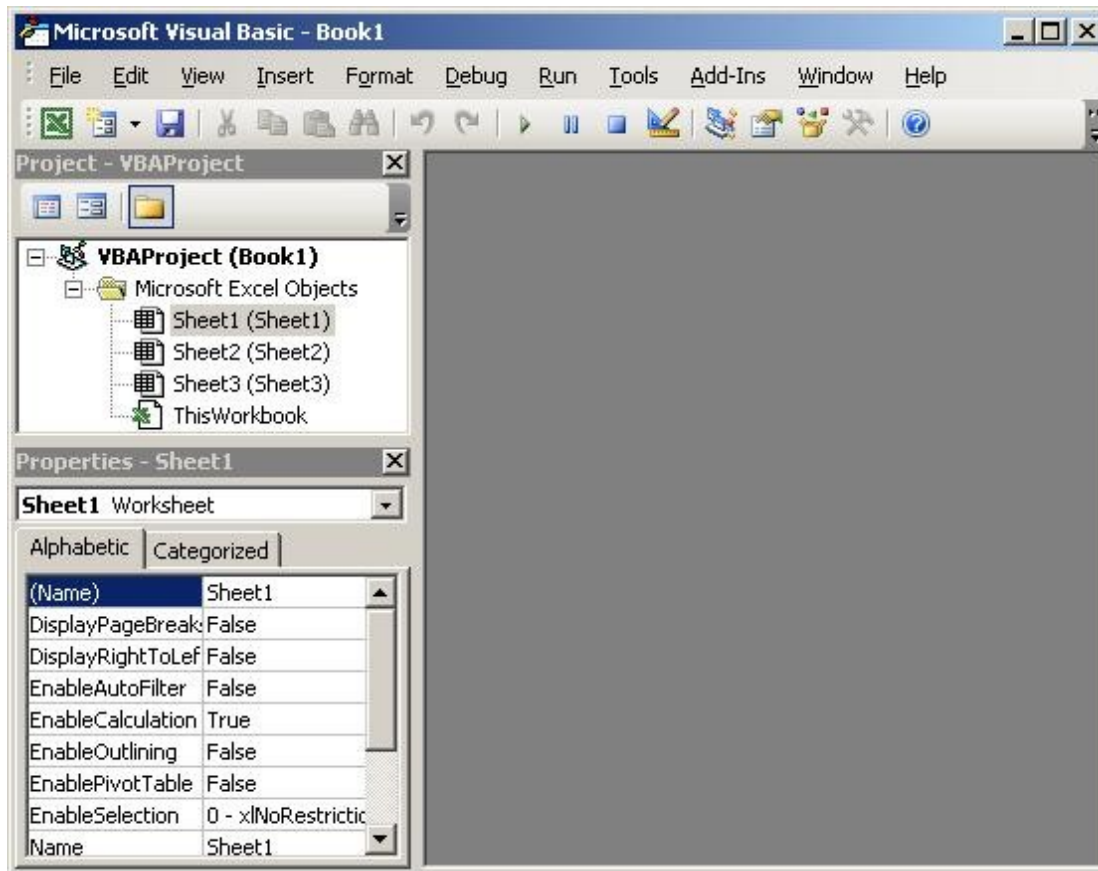
Well, the samples I'll show you on my tutorials are all in Microsoft Excel. Firstly because it's the first office application on my menu bar (and lazy is my second name), but more importantly screen grab works well so I can show you what I'm doing.

So open up excel.....Pause for effect....Now open up the VBA editor.

Gasp – shock horror, where is it? Easy, it can be found under Tools > Macro > Visual Basic Editor – see the picture below. Go on click it, I know you want to.



Now you have broken your computer, unless you can see the editor environment that looks like this

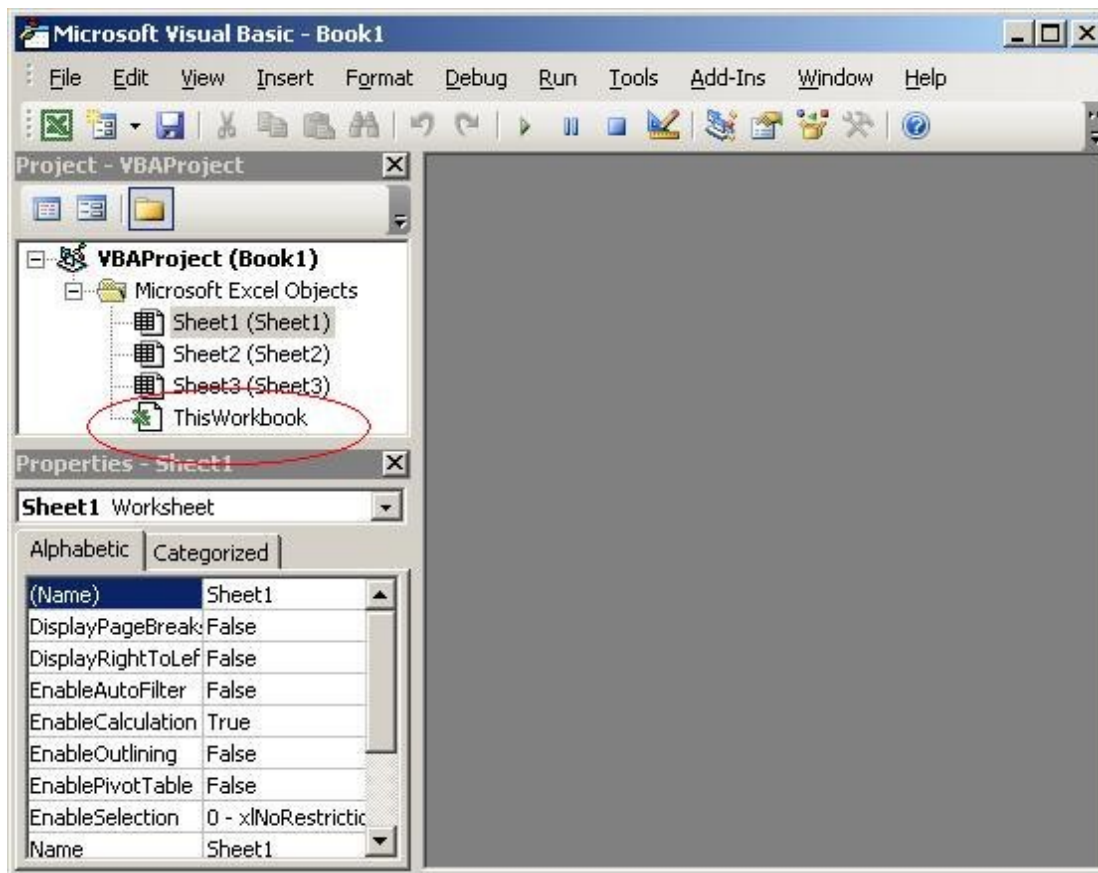


In which case congratulations; you now know how to get into the VBA editor. Well done.

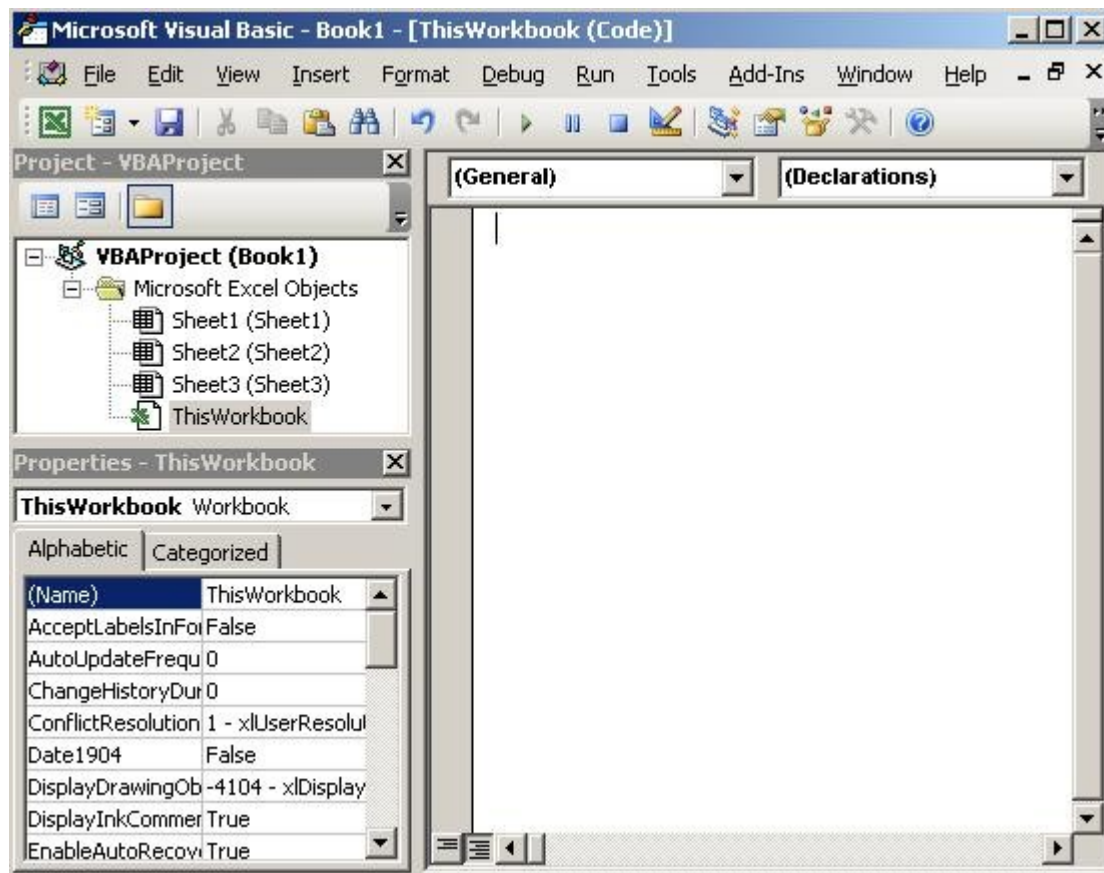
Lesson 3. My First Program

Next we will learn that programming involves typing. I bet that gave you a start. Yes, sorry but you have to do some real work now. Here is how we write some code.

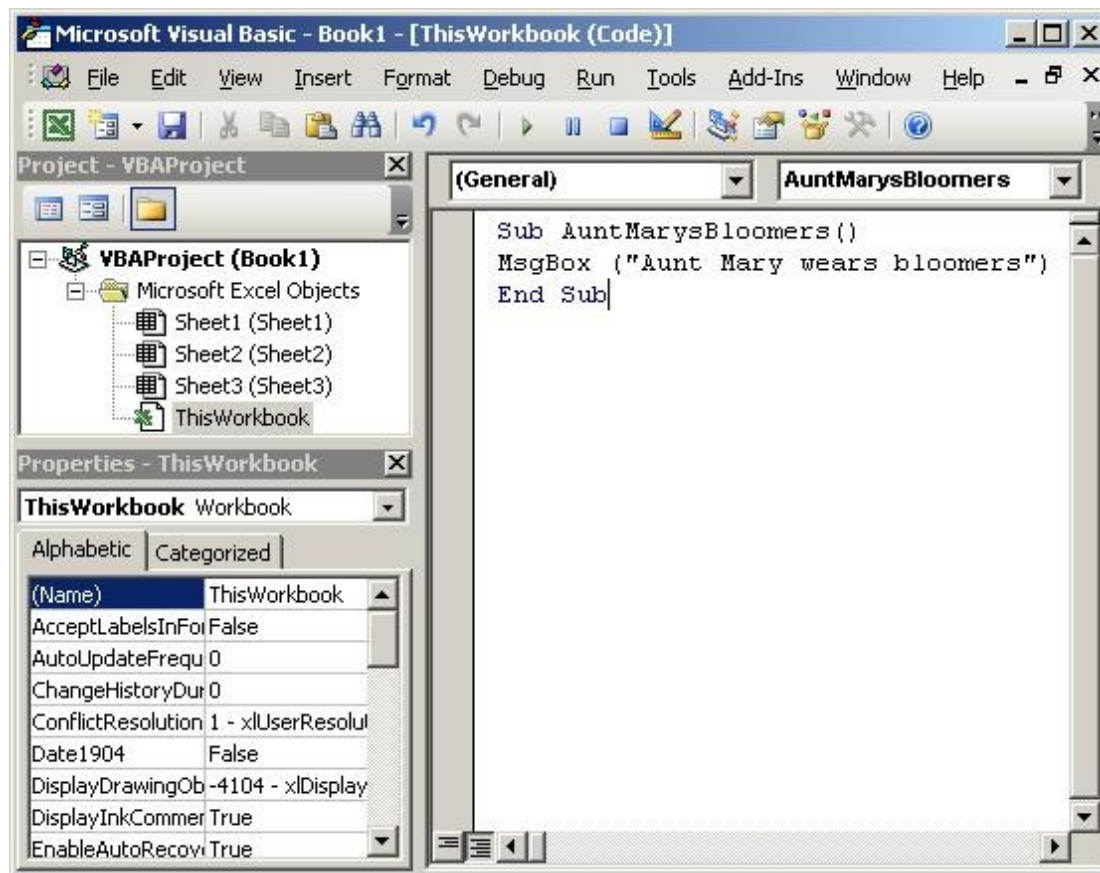
In the editor, you opened it in lesson 2, double click (that is two left hand mouse button clicks – click click) where it says ThisWorkbook



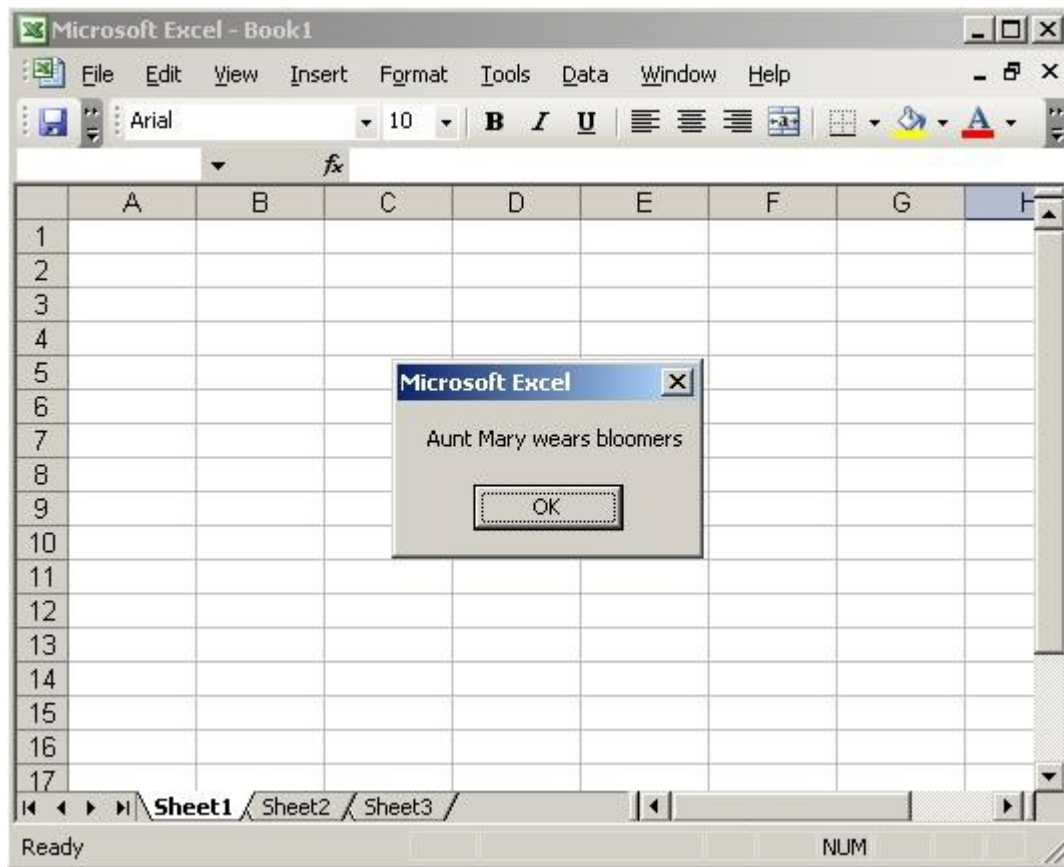
Once you've double clicked your left mouse button, you should see this screen



What you've done, is create an editable area to write your VBA code in (the bug nest). We're now going to type in some code and run it. Imagine you have an aunty Mary and she wears bloomers. We want our program to pop up a box and tell everyone about Aunt Mary and her bloomers. So we'd type the following



To get this program to run, we press the F5 key (function key) on our keyboard. This will run our first program and pop up a message like this



Oh what joy, didn't take you long to write your first program. You're flying along now.

Lesson 4. Whats all this Sub business ?

As you saw in Lesson 3, a program looks a lot like (actually just like) this...

```
Sub AuntMarysBloomers()  
MsgBox ("Aunt Mary wears bloomers")  
End Sub
```

The Sub part acts as a nice neat container for the “to do” list. We always have to contain the list; otherwise it might run off the screen and get married. Sub is a shortened version of Subroutine (code is nearly all about short things, my wife entirely understands short).

A Sub always ends in an End Sub, otherwise it wouldn't know when to stop. I would have just said “when the code runs out” but apparently Microsoft never thought of it.

There is another container called Function that we'll come to later. Strangely, they didn't shorten it to Fun, I think that might have been more accurate.

A Sub always needs a name, or it would never know when to do its job. In this case we called it [AuntMarysBloomers](#). The name can be almost anything you like except for words the computer already uses to do other things. Do not panic. If the computer doesn't like your word it will let you know by not running your code.

The brackets () are containers too. They can hold information (often known as gibberish) that we can get the code (we tried to write) to work on.

Confused? Me too. Time for a coffee, doughnut and a lie down by now. Just let it sink in slowly, we have a long way to go.

Lesson 5. How to start doing something interesting.

Perhaps the most interesting things in any program are the variables. Variables are like little bundles of information, tied to a name that humans can easily understand (we have difficulty remembering 1245642, but we can easily remember NumberOfBloomers)

Imagine you have a box of apples and people can put apples into the box or take them out. Any number they like, at any time. How many apples are currently in the box?

That's where a variable comes in. A variable will hold the current number of apples (or any other unit, money, time, number of characters for example), and we can ask the computer for this number at any time during our program. You'd think the computer would get fed up asking for the same number a thousand times, but it never does.

Whole numbers (numbers without a decimal point) are called integers (shortened sometimes to int, just to add to the confusion). We create an integer variable like this...

```
Dim intBoxOfApples as Integer
```

To add an apple to the box we would write

```
intBoxOfApples = intBoxOfApples + 1
```

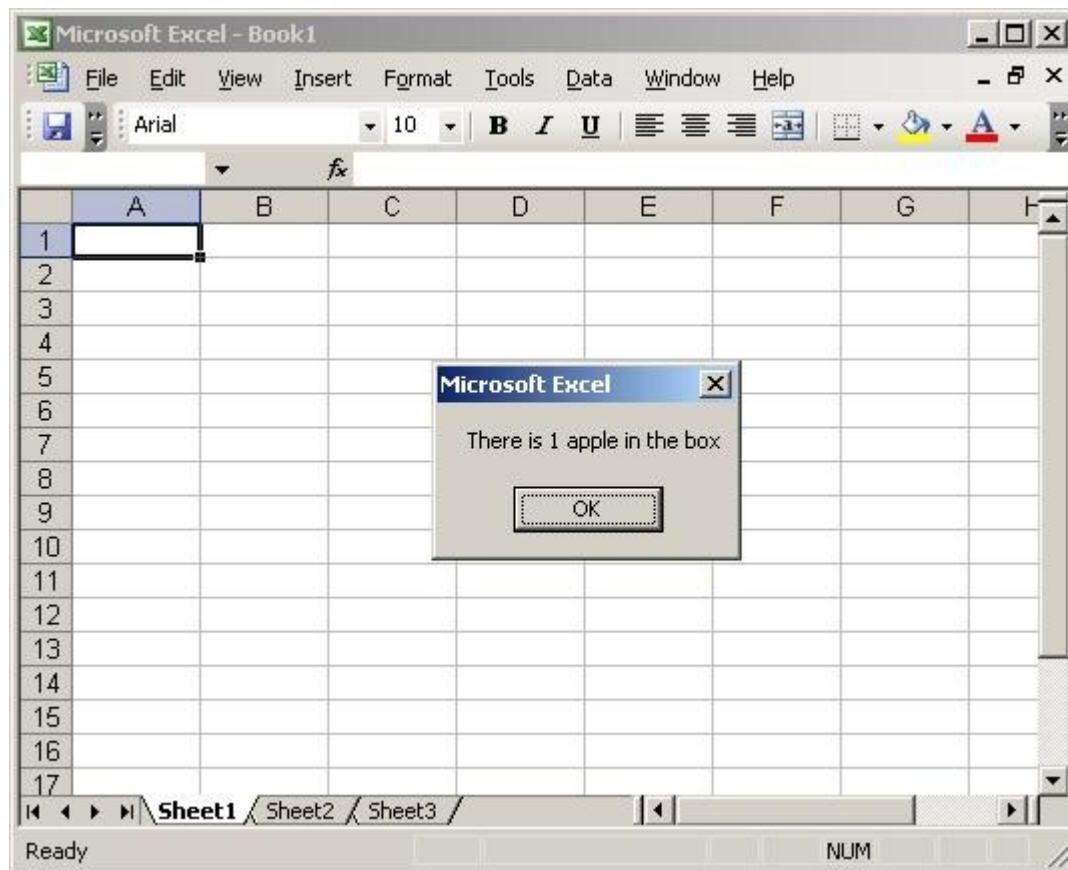
and guess what, to take an apple out of the box we would write

```
intBoxOfApples = intBoxOfApples - 1
```

So a program to put an apple into a box, and pop up a message to tell us about it, would look like this ...

```
Sub PutAppleInBox()  
Dim intBoxOfApples as Integer  
intBoxOfApples = intBoxOfApples + 1  
MsgBox ("There is " & intBoxOfApples & " apple in the box")  
End Sub
```

To try it out, swap the Aunt Marys Bloomers code from Lesson 3 and run the code once you have typed it in (and debugged it). The result will look like this ...



Lesson 6. The Art of Repeating yourself

Let us recap, quickly. Your writing VBA code, you know where to find the VBA editor, how to create a subroutine, what a variable is and how to get a message box on screen. Not bad for 5 minutes work, it sure took me a lot longer than that to learn it! What were you worried about? VBA is easy.

Now we're going to learn how to make instructions repeat themselves, we call this a loop. If you have OCD, then this is the Lesson for you.

There are a lot of ways to do loops in VBA, the simplest is to say "Do these instructions, repeat this many times". In code we'd write what's called a For ... Next loop, like this

```
For intLoop = 1 to 10  
....Some instruction or other  
Next intLoop
```

So, if we put a loop in our [PutAppleInBox](#) subroutine, we get this

```
Sub PutAppleInBox()  
  
Dim intRepeatThisLoop As Integer  
Dim intBoxOfApples As Integer  
  
For intRepeatThisLoop = 1 To 10  
    intBoxOfApples = intBoxOfApples + 1
```

Next

```
MsgBox ("There are " & intBoxOfApples & " apples in the box")
```

End Sub

Go try it now, modify your previous program, remember to press F5 key to run the code. I changed “There is” to “There are”, just to make the sentence look right. Now you’ve gone loopy too. You can change the 1 to 10 to any number you like; you can even go backwards like this

```
For intRepeatThisLoop = 10 to 1 step -1
```

By now you’re probably thinking either

- i. This guys a genius and teaches VBA really easily
- ii. This guys a nutcase and should be committed
- iii. Lets do another lesson (that’s what I was thinking)

Lesson 7. Loops with strings attached

Previously, we saw that you can drive yourself loopy, and repeat instructions over and over again, a time saver once you get it right. But what if you want to stop the loop early because something has happened?

These are called conditional loops. Easy real world definition: Going with your girl, stopping because you have an argument, getting back together again, another argumental split and so on.

Expanding a bit on the [PutAppleInBox](#) subroutine, we can add an IF....THEN.....ENDIF clause (not Santa). We do this to get the routine to “jump” over a few numbers, once a certain number or condition has been reached (my age went from 15 to 18 every Friday night so I could go to the pub):

```
Sub PutAppleInBox()  
  
Dim intRepeatThisLoop As Integer  
Dim intBoxOfApples As Integer  
  
For intRepeatThisLoop = 1 To 10  
  
    If intBoxOfApples = 8 Then  
        intBoxOfApples = 10  
    End If  
  
    intBoxOfApples = intBoxOfApples + 1
```

Next

MsgBox ("There are " & intBoxOfApples & " apples in the box")

End Sub

Using the code (above) Once `intBoxOfApples` reaches 8, the code makes `intBoxOfApples = 10` and completely misses out 9. This ends the `For` loop early. There are many uses of `If`.

If only she went out with me

If only I hadn't gone out with him

If only the tooth fairy hadn't pulled out all my teeth the night I slept with my head under the pillow.

Lesson 8. Selecting the right thing to do

When we write code that actually works (greatest feeling in the world) we most often want to make a decision or choice. Essentially, in human terms (or Martian, if there are any Martians reading this), we would say “when this happens then do this” or “if this had occurred, do that”. We wrote pretty much the same thing in our code from lesson 7, but like this:-

```
If intBoxOfApples = 8 Then
    intBoxOfApples = 10
End If
```

But if we have more than one choice (goodness knows we nearly always do) then a good method to decide what to do is **Select Case End Select**. This command allows us to have several options (Cases, no suitcases – more like a legal or medical case) to choose from and carry out some code on one or several of these options. It works like this:-

```
Select Case intBoxOfApples
Case 4:
    intBoxOfApples = 6
Case 8:
    intBoxOfApples = 10
End Select
```

Replace the `If...Then...End If` code from the previous lesson with the Select Case code (above) and run it. If you can follow the logic, or flow, of the code you'll see that when `intBoxOfApples` is equal to 4 (Case 4:) the code will let `intBoxOfApples` equal 6, and again when it reaches 8 (Case 8:) it will make `intBoxOfApples` equal to 10.

Try modifying with a few more results. You could even put in a few more `MsgBox`'s to test the results under each Case statement.

Programming is like lots of things in life, the more you do, the easier it is, until it becomes so easy it's almost a no-brainer. Well I've almost no brains so everything must be easy?

Lesson 9. More variables than a Unicycling Juggler

In an earlier lesson we met with Mr Integer, the variable for carrying around whole numbers. In this lesson, I thought it about time I showed you a number of other variable types which carry around values you'll more than likely want to store and certainly need in your future code.

Firstly, we have the string, which keeps a track of characters, such as names, addresses, zip codes, product descriptions, she is allocated like this

```
Dim strIsPaulSexy as String  
strIsPaulSexy = "NO, Paul has no sex appeal"
```



if we print out the value of strIsPaulSexy, the computer will tell us NO, Paul has no sex appeal – yeah, thanks a lot.

Then we have Mr Double. He can hold floating point numbers, which are numbers with a decimal place, like Pi or the money in your bank account :

```
Dim dbIPi as Double
dbIPi = 3.14159265
MsgBox (dbIPi * 10 & "cm")
```

Run this code and you will get a message box showing you the circumference of a 10cm diameter circle. Remember the difference between an integer and a double. Integers will round up or down to the nearest whole number (e.g. 3.14 = 3, or 4.501 = 5)

Then there is the Gemini like Boolean, the True/False variable of two halves. Useful as a switch within our code, making something happen in one position, or ignoring the instructions in another perhaps.

```
Dim boollsPaulSexy as Boolean
boollsPaulSexy = False

If boollsPaulSexy = True Then
    MsgBox ("Paul has sex appeal")
End If
```

The computer, like most ladies, will say nothing.

Lesson 10. Being Invited to Functions

In Lesson 4 I promised to talk about functions, OK here we go. A function acts in a similar way to a bicycle messenger. We send it off with a piece of information; the information is delivered; some processing occurs; and the messenger is sent back with a reply. Unless he's gone off for pizza or to meet a date. Our functions return results, as a variable.

Take a look at this:

```
Sub PutAppleInBox()  
MsgBox ("There are " & AppleCount(10) & " apples in the box")  
End Sub
```

```
Function AppleCount(intTotal As Integer) As Integer  
Dim intRepeatThisLoop As Integer  
Dim intBoxOfApples As Integer
```

```
For intRepeatThisLoop = 1 To intTotal  
    intBoxOfApples = intBoxOfApples + 1  
Next
```

```
AppleCount = intBoxOfApples  
End Function
```

There's our favourite subroutine [PutAppleInBox](#) from earlier, re-written so that there is only a [MsgBox](#) in the to do list. The [MsgBox](#) contains the messenger function "[AppleCount](#)", which is being asked to go off and calculate the number of apples then, bring back the result.

[AppleCount](#) is the function, we pass in a variable (in this case an integer called [intTotal](#)) and the messenger function uses this variable to carry out its own to do list of instructions. Once complete the function passes back the resulting variable, in this case As an Integer.

Functions and subroutines can be as complex or simple as you want to make them. By splitting down a longer program into a series of subs and functions you ensure you've left yourself lots of bug hunting and code-re-writing for years to come!