VBA Made Easy

Loops

09

www.accessallinone.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

Contents

Loops

After conditionals and arrays, loops form the next major component in VBA. A loop is a block of code that executes again and again until either an expression equates to false or is broken by way of an Exit statement.

What makes loops useful is that they can work with arrays and collections; they can perform tasks over and over until a condition is met and they can perform calculations over and over until you force them to stop.

There are several ways to express this need to loop and VBA isn't short on constructs for doing that. So we will get straight into a For loop, but first…

A Word of Warning - Infinite Loops

If you get stuck in an infinite loop or the loop is taking a lot longer than you expected, use CTRL + Break to stop VBA from executing.

For…Next

A For loop goes around and around incrementing some variable counter by a figure you determine (the default is 1). It executes a code block between the keywords For and Next until some condition with the variable is met.

The Standard Syntax For A For…Next Loop

Let's get straight into the code and see what a For loop does.

```
1   For counter = start To end
2      ...
3   next i
```

Figure 9.1

The code block contains a **Debug.Print** statement which prints the value of **i**. The **For** statement increments **i** by 1 on each iteration and stops when **i** gets to 10. Although i increments by 1, we can change the way it increments.

```
1   Sub forLoop2()
2       Dim i As Integer
3
4       For i = 1 To 10
5           Debug.Print i
6           i = i + 1
7           'Because of the above statement
8           'this loop will increment by 2
9       Next i
10  End Sub
```

```
    The output to the immediate window will be:

    1
    3
    5
    7
    9
```

Figure 9.2

Here ***forLoop2*** executes the code block but adds an extra **1** on each iteration.

What happens if we start the **For loop** at 10 instead of 1?

```
1   Sub forLoop3()
2       Dim i As Integer
3
4       For i = 10 To 1
5       'Starting i at 10 means that this
6       'loop will not print anything out
7       'as it (by default) increments and
8       'there is nothing after 10
9           Debug.Print i
10      Next i
11  End Sub
```

Figure 9.3

Well, nothing actually. The **for loop** moves forward by default and as 10 is the maximum number in the range, it has nowhere else to go!

Although we are incrementing **i**, we are also able to increment other variables inside the loop.

```
1   Sub forLoop4()
2       Dim i As Integer
3       Dim t As Integer
4
5       t=0
6
7       For i = 1 To 10
8           Debug.Print t
9           t = t + 3
10          'Although we are incrementing the
11          'i variable, we are printing out
12          'the value associated with the t
13          'variable
14      Next i
15  End Sub
```

```
    The output to the immediate window will be:

    0
    3
    6
    9
    12
    15
    18
    21
    24
    27
```

Figure 9.4

In the code below, we demonstrate that the end value of the **For loop** (5+5) can be an expression.

```
1   Sub forLoop5()
2       Dim i As Integer
3       Dim t As Integer
4
5       For i = 1 To 5 + 5
6       'Here we are using an expression (5+5)
7       'rather than simply using the number 10
8           Debug.Print i
9       Next i
10  End Sub
```

Figure 9.5

For…Step…Next

In **forLoop2** we adjusted the counter **i** to increment by an additional 1 for each loop. We can do the same by using the Step option in the For loop

Step tells the **For Loop** to increment its counter by a value other than the default value of 1.

```
1   Sub forLoop6()
2       Dim i As Integer
3
4       For i = 1 To 10 Step 2
5       'We are using the Step command
6       'to increment i by 2 on each
7       'iteration
8           Debug.Print i
9       Next i
10  End Sub
```

```
    The output to the immediate window will be:

    1
    3
    5
    7
    9
```

Figure 9.6

Using Step To Count Backwards

We can go backwards through a loop by using **Step – 1** in the **For Loop**.

```
1   Sub forLoop7()
2       Dim i As Integer
3
4       For i = 10 To 1 Step -1
5       'This is how you go backwards through
6       'a for loop : Step -1
7           Debug.Print i
8       Next i
9   End Sub
```

```
    The output to the immediate window will be:

    10
    9
    8
    7
    6
    5
    4
    3
    2
    1
```

Figure 9.7

Using Dynamic startValue, endValue and stepValues

In the below code, **startValue**, **endValue** and **stepValue** are all expressions, so as long as the expressions evaluate to a number, the **For Loop** will accept them. Here we start at 4, step by 3 and finish at 16.

```
1   Sub forLoop8()
2       Dim startValue As Integer
3       Dim endValue As Integer
4       Dim stepValue As Integer
5       Dim i As Integer
6
7       startValue = 4
8       endValue = 16
9       stepValue = 3
10
11      For i = startValue To endValue Step stepValue
12      'Each part of the for expression now contains
13      'a variable
14          Debug.Print i
15      Next i
16
17  End Sub
```

```
    The output to the immediate window will be:

    4
    7
    10
    13
    16
```

Figure 9.8

For…Each

The For…Each loop differs from a For…Next loop in that it iterates over arrays and collections and therefore knows how many iterations to perform.

Let's take a look at the For…Each loop over a standard array.

```
1   Sub forEachArray()
2
3       Dim element As Variant
4       Dim animals(0 To 5) As String
5       'We have created an array that can hold 6 elements
6
7       animals(0) = "Dog"
8       animals(1) = "Cat"
9       animals(2) = "Bird"
10      animals(3) = "Buffalo"
11      animals(4) = "Snake"
12      animals(5) = "Duck-billed Platypus"
13      'We fill each element of the array
14
15
16      For Each element In animals
17      'animals consists of 6 "elements"
18
19          Debug.Print element
20          'printing to the immediate window
21
22      Next
23
24  End Sub
```

```
    The output to the immediate window will be:

    Dog
    Cat
    Bird
    Buffalo
    Snake
    Duck-billed Platypus
```

Figure 9.9

Using Loops with Collections

Let's take a look at the For…Each loop over a collection.

```
1   Public Sub forEachCollection1()
2
3       Dim element As Variant
4       Dim animals  As Collection
5       Set animals = New Collection
6       'Collections are literally collections of objects
7       'and are a useful feature of MS Access
8       'They have 4 methods - add, count, item, remove
9
10      animals.Add "Dog"
11      animals.Add "Cat"
12      animals.Add "Bird"
13      animals.Add "Buffalo"
14      animals.Add "Snake"
15      animals.Add "Duck-billed Platypus"
16      'We utilise the add method to add the various
17      'animals to the collection
18
19      For Each element In animals
20          Debug.Print element
21      Next
22      'We print out all the elements in the
23      'animals collection
24
25      animals.Remove 3
26      'We remove an element from the animals collection
27      'we are removing the 3rd item in the collection (bird)
28
29      Debug.Print ""
30      'prints a blank line
31
32      For Each element In animals
33          Debug.Print element
34          'printing to the immediate window
35
36      Next
37      'Here we are printing out all the elements in the
38      'animals collection minus the bird
39  End Sub
```

```
    The output to the immediate window will be:

    Dog
    Cat
    Bird
    Buffalo
    Snake
    Duck-billed Platypus

    Dog
    Cat
    Buffalo
    Snake
    Duck-billed Platypus
```

Figure 9.10

Access contains some collections of its own! Knowing how to utilise these collections, can make life much simpler when coding.

```
1   Sub forEachCollection2()
2       Dim i As Integer
3       For i = 0 To CurrentProject.AllForms.Count - 1
4       'CurrentProject.AllForms is a collection and
5       'therefore has the add, count, item and remove
6       'methods available
7
8           Debug.Print CurrentProject.AllForms(i).Name
9           'Here we print the names of the forms to the
10          'immediate window
11      Next
12
13  End Sub
```

In a database with 3 forms (Form1, Form2, Form3), the output to the immediate window would be:

```
Form1
Form2
Form3
```

Figure 9.11

Let's take a look at how to exit a for each loop.

Exit For

To leave the For Each loop before its natural end, we can use the Exit For statement.

```
1   Sub forEachExit()
2       Dim element As Variant
3       Dim animals(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       animals(0) = "Dog"
7       animals(1) = "Cat"
8       animals(2) = "Bird"
9       animals(3) = "Buffalo"
10      animals(4) = "Snake"
11      animals(5) = "Duck-billed Platypus"
12      'Here we fill each element of the array
13
14      For Each element In animals
15      'iterates over the animals collection
16
17          Debug.Print element
18          'print each element to the immediate window
19
20          If element = "Buffalo" Then Exit For
21          'if, at any point, the element becomes equal
22
23      Next
24
25  End Sub
```

```
The output to the immediate window will be (we exited the loop before
all items could be printed):

Dog
Cat
Bird
Buffalo
```

Figure 9.12

While...Wend

A While loop executes its code blocks over and over until its expression is not True.

The basic syntax of a while loop is:

```
1   While(someExpression)
2   ...
3   Wend
```

Figure 9.13

The loop will continue to operate as long as someExpression is equal to true. When it becomes false, the while loop exits.

```
1   Sub whileLoopArray()
2       Dim i As Long
3       Dim kitchenItems(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       kitchenItems(0) = "Cooker"
7       kitchenItems(1) = "Fridge"
8       kitchenItems(2) = "Cutlery"
9       kitchenItems(3) = "Crockery"
10      kitchenItems(4) = "Dishwasher"
11      kitchenItems(5) = "Table and Chairs"
12      'Here we fill each element of the array
13
14      i = 0
15
16      While (i < UBound(kitchenItems) + 1)
17          'This line of code essentially says:
18          ' As long as the value of i is less
19          'than 6 execute the next line. Otherwise
20          'exit the loop
21
22          Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
23          'This line prints a string to the immediate window.
24          'An example would be:
25          'Item 4 is Dishwasher
26
27          i = i + 1
28          'We need to increment i or we will be stuck
29          'in a loop forever...
30      Wend
31
32  End Sub
```

```
    The output to the immediate window will be:

    Item 0 is Cooker
    Item 1 is Fridge
    Item 2 is Cutlery
    Item 3 is Crockery
    Item 4 is Dishwasher
    Item 5 is Table and Chairs
```

Figure 9.14

The While Loop is often used to cycle through Recordsets and Files.

```
1   Sub whileLoopRecordset()
2   On Error GoTo ErrorHandler
3
4   Dim strSQL As String
5   Dim rs As DAO.Recordset
6
7   strSQL = "tblTeachers"
8   'For the purposes of this post, we are simply going to make
9   'strSQL equal to tblTeachers.
10  'You could use a full SELECT statement such as:
11  'SELECT * FROM tblTeachers (this would produce the same result in
12  fact).
13  'You could also add a Where clause to filter which records are
14  returned:
15  'SELECT * FROM tblTeachers Where ZIPPostal = '98052'
16  ' (this would return 5 records)
17
18  Set rs = CurrentDb.OpenRecordset(strSQL)
19  'This line of code instantiates the recordset object!!!
20  'In English, this means that we have opened up a recordset
21  'and can access its values using the rs variable.
22
23  With rs
24
25
26      If Not .BOF And Not .EOF Then
27      'We don't know if the recordset has any records,
28      'so we use this line of code to check. If there are no records
29      'we won't execute any code in the if..end if statement.
30
31          .MoveLast
32          .MoveFirst
33          'It is not necessary to move to the last record and then back
34          'to the first one but it is good practice to do so.
35
36         While (Not .EOF)
37         'With this code, we are using a while loop to loop
38         'through the records. If we reach the end of the recordset, .EOF
39         'will return true and we will exit the while loop.
40
41        Debug.Print rs.Fields("teacherID") & " " & rs.Fields("FirstName")
42              'prints info from fields to the immediate window
43
44              .MoveNext
45              'We need to ensure that we use .MoveNext,
46              'otherwise we will be stuck in a loop forever…
47              '(or at least until you press CTRL+Break)
48          Wend
49
50      End If
51
52      .Close
53      'Make sure you close the recordset...
54  End With
55
56  ExitSub:
57      Set rs = Nothing
58      '..and set it to nothing
```

```
59      Exit Sub
60  ErrorHandler:
61      Resume ExitSub
62  End Sub
63
64
```

```
    The output to the immediate window will be:

    1 Anna
    2 Antonio
    3 Thomas
    4 Christina
    5 Martin
    6 Francisco
    7 Ming-Yang
    8 Elizabeth
    9 Sven
```

Figure 9.15

Exit While

To exit a while loop isn't as trivial a task as with other looping structures. To exit a While one must force the While expression to be false.

```
1   Sub whileLoopExit()
2       Dim i As Long
3       Dim kitchenItems(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       Dim stayInLoop As Boolean
7       'This is the variable we will use for the
8       'condition of the while loop
9
10      kitchenItems(0) = "Cooker"
11      kitchenItems(1) = "Fridge"
12      kitchenItems(2) = "Cutlery"
13      kitchenItems(3) = "Crockery"
14      kitchenItems(4) = "Dishwasher"
15      kitchenItems(5) = "Table and Chairs"
16      'Here we fill each element of the array
17
18      i = 0
19
20      stayInLoop = True
21      'sets stayInLoop as true
22
23      While (stayInLoop)
24
25          'As long as stayInLoop resolves to true,
26          'we will stay in the loop
27
28          Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
29          'This line prints a string to the immediate window.
30          'An example would be:
31          'Item 4 is Dishwasher
32
33          i = i + 1
34
35          If i = 3 Then
36          'If, at any point, i becomes equal to 3, we will change
```

```
37          'stayInLoop to false and exit the while loop
38              stayInLoop = False
39          End If
40
41      Wend
42
43
44  End Sub
```

```
The output to the immediate window will be:

Item 0 is Cooker
Item 1 is Fridge
Item 2 is Cutlery
```

Figure 9.16

Loop/Do…Until/While

The Do loops are another set of statements that perform like a While loop and permit exiting the loop at any point without changing the statement's expression.

Do…While

```
1    Sub doWhile1()
2
3        Dim i As Long
4        Dim kitchenItems(0 To 5) As String
5        'We have created an array that can hold 6 elements
6
7        kitchenItems(0) = "Cooker"
8        kitchenItems(1) = "Fridge"
9        kitchenItems(2) = "Cutlery"
10       kitchenItems(3) = "Crockery"
11       kitchenItems(4) = "Dishwasher"
12       kitchenItems(5) = "Table and Chairs"
13       'Here we fill each element of the array
14
15       i = 0
16
17       Do While (i < UBound(kitchenItems) + 1)
18           'This line of code essentially says:
19           ' As long as the value of i is less
20           'than 6 execute the next line. Otherwise
21           'exit the loop
22
23           Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
24           'This line prints a string to the immediate window.
25           'An example would be:
26           'Item 4 is Dishwasher
27
28           i = i + 1
29           'We need to increment i or we will be stuck
30           'in a loop forever...
31       Loop
32
33   End Sub
```

```
    The output to the immediate window will be:

    Item 0 is Cooker
    Item 1 is Fridge
    Item 2 is Cuttlery
    Item 3 is Crockery
    Item 4 is Dishwasher
    Item 5 is Table and Chairs
```

Figure 9.17

doWhile2 below performs the same operation as doWhile1 above except it uses Exit Do to exit the loop.

```
1   Sub doWhile2()
2       Dim i As Long
3       Dim kitchenItems(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       kitchenItems(0) = "Cooker"
7       kitchenItems(1) = "Fridge"
8       kitchenItems(2) = "Cutlery"
9       kitchenItems(3) = "Crockery"
10      kitchenItems(4) = "Dishwasher"
11      kitchenItems(5) = "Table and Chairs"
12      'Here we fill each element of the array
13
14      i = 0
15
16      Do While (True)
17          'Because True evaluates to true (obviously) we have
18          'created a never-ending loop. We will need to force
19          'an exit if we want to leave
20
21          Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
22          'This line prints a string to the immediate window.
23          'An example would be:
24          'Item 4 is Dishwasher
25
26          i = i + 1
27          'We need to increment i or we will be stuck
28          'in a loop forever...
29
30          If i = UBound(kitchenItems) + 1 Then Exit Do
31          'This line of code essentially says:
32          ' If, at any point, the value of i becomes
33          'greater than 6, exit the do loop
34      Loop
35
36  End Sub
```

```
    The output to the immediate window will be:

    Item 0 is Cooker
    Item 1 is Fridge
    Item 2 is Cuttlery
    Item 3 is Crockery
    Item 4 is Dishwasher
    Item 5 is Table and Chairs
```

Figure 9.18

Do Until executes its code block *until* a certain condition is met.

```
1   Sub doUntil()
2       Dim i As Long
3       Dim kitchenItems(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       kitchenItems(0) = "Cooker"
7       kitchenItems(1) = "Fridge"
8       kitchenItems(2) = "Cutlery"
9       kitchenItems(3) = "Crockery"
10      kitchenItems(4) = "Dishwasher"
11      kitchenItems(5) = "Table and Chairs"
12      'Here we fill each element of the array
13
14      i = 0
15
16      Do Until (False)
17          'The Do until Loop fires until a condition is met
18          'Because False can never evaluate to true (obviously)
19          'we have created a never-ending loop. We will need
20          'to force an exit if we want to leave
21
22          Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
23          'This line prints a string to the immediate window.
24          'An example would be:
25          'Item 4 is Dishwasher
26
27          i = i + 1
28          'We need to increment i or we will be stuck
29          'in a loop forever...
30
31          If i = UBound(kitchenItems) + 1 Then Exit Do
32          'This line of code essentially says:
33          'If, at any point, the value of i becomes
34          'equal to 6, exit the do loop
35      Loop
36
37  End Sub
```

```
    The output to the immediate window will be:

    Item 0 is Cooker
    Item 1 is Fridge
    Item 2 is Cutlery
    Item 3 is Crockery
    Item 4 is Dishwasher
    Item 5 is Table and Chairs
```

Figure 9.19

Finally, the Do…Loop executes until you force it to stop.

```
1   Sub doLoop()
2       Dim i As Long
3       Dim kitchenItems(0 To 5) As String
4       'We have created an array that can hold 6 elements
5
6       kitchenItems(0) = "Cooker"
7       kitchenItems(1) = "Fridge"
8       kitchenItems(2) = "Cutlery"
9       kitchenItems(3) = "Crockery"
10      kitchenItems(4) = "Dishwasher"
11      kitchenItems(5) = "Table and Chairs"
12      'Here we fill each element of the array
13
14      i = 0
15
16      Do
17      'The Do loop just does! There is no condition
18      'to evaluate to so we will need to force an exit.
19
20          Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
21          'This line prints a string to the immediate window.
22          'An example would be:
23          'Item 4 is Dishwasher
24
25          i = i + 1
26          'We need to increment i or we will be stuck
27          'in a loop forever...
28
29          If i = UBound(kitchenItems) + 1 Then Exit Do
30          'This line of code essentially says:
31          'If, at any point, the value of i becomes
32          'equal to 6, exit the do loop
33      Loop
34
35  End Sub
```

```
    The output to the immediate window will be:

    Item 0 is Cooker
    Item 1 is Fridge
    Item 2 is Cutlery
    Item 3 is Crockery
    Item 4 is Dishwasher
    Item 5 is Table and Chairs
```

Figure 9.20

Nested Loops

A loop inside a loop is termed a nested loop. We'll make a grid of numbers to illustrate.

```
1   Sub nestedLoop1()
2       Dim y As Integer
3       Dim x As Integer
4       Dim xString As String
5
6       For y = 0 To 9
7       'We start by looping through 0 - 9. This will provide
8       'us with 10 loops
9           For x = 0 To 9
10          'Adding a second loop will mean that we end up
11          'looping a hundred times (10 x 10)
12              xString = xString & x & " "
13              'On each loop we are concatenating the x
14              'variable with a space so we have a line that
15              'goes 0 1 2 3 4 5 etc.
16          Next x
17
18          Debug.Print xString
19          'Here we print out the full xString
20
21          xString = ""
22          'We reset the xString to nothing
23      Next y
24  End Sub
```

```
    The output in the immediate window will be:
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
    0 1 2 3 4 5 6 7 8 9
```

Figure 9.21

Nested Loops and Multidimensional Arrays

Nested loops work very well with multidimensional arrays.

```
1   Sub nestedLoop2()
2       Dim y As Integer
3       Dim x As Integer
4       Dim xString As String
5       Dim MyArray(10, 10) As String
6       'Here we have a multidimensional array of 10 x 10
7       'This array will be able to hold 100 items
8
9       For y = 0 To 9
10      '10 loops here...
11          For x = 0 To 9
12          '...and 10 more here give us 100 loops!
13              MyArray(y, x) = y * x
14              'We fill the array element with the
15              'multiple of x and y
16          Next x
17      Next y
18
19      For y = 0 To 9
20          For x = 0 To 9
21          'And now we loop again and print out the
22          'results of the code above
23              xString = xString & MyArray(y, x) & " "
24          Next x
25
26          Debug.Print xString
27          xString = ""
28      Next y
29
30  End Sub
```

```
    The output in the immediate window will be:

    0 0 0 0 0 0 0 0 0 0
    0 1 2 3 4 5 6 7 8 9
    0 2 4 6 8 10 12 14 16 18
    0 3 6 9 12 15 18 21 24 27
    0 4 8 12 16 20 24 28 32 36
    0 5 10 15 20 25 30 35 40 45
    0 6 12 18 24 30 36 42 48 54
    0 7 14 21 28 35 42 49 56 63
    0 8 16 24 32 40 48 56 64 72
    0 9 18 27 36 45 54 63 72 81
```

Figure 9.22

A Useful Implementation of Nested Loops

A more practical example is to iterate over a Collection within a Recordset.

```
1   Sub nestedLoop3()
2
3       Dim rs As DAO.Recordset, field As DAO.field
4       Dim rowText As String
5       Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblStudents")
6
7       While (Not rs.EOF)
8       'Loop no1
9           For Each field In rs.Fields
10    'Loop no2
11    'we will be looping through all of the field names in tblStudents
12
13     rowText = rowText & field.Name & "=" & rs.Fields(field.Name) & ", "
14    'we use the field name to get the value of that field and create
15    'a concatenated string to print out.
16    'e.g. StudentID=15, LastName=Kupova, etc.
17          Next
18
19          Debug.Print rowText
20          rowText = ""
21          rs.MoveNext
22      Wend
23
24  End Sub
```

```
    nestedLoop3
    StudentID=1, LastName=Bedecs, FirstName=Anna ' … more commented out
    StudentID=2, LastName=Gratacos Solsona, FirstName=Antonio '…
    StudentID=3, LastName=Axen, FirstName=Thomas, '…
```

Figure 9.23

Here the Fields collection is being iterated over and rowText populated with the field's name and value.

Note: *The* `On Error` *statement forces VBA to skip any error messages and* `Resume Execution`.

DoEvents

DoEvents is a simple command that pauses a loop and allows the operating system to carry out any tasks that have been queued.

If you have a loop that can take a significant time to fire, DoEvents enables the loop to pause at periodic intervals. In the code below, we have created a very long loop and added in a DoEvents command every 1 second or so.

```
1   Sub CPUTask()
2       Dim t As Double, zzz As Single
3       Debug.Print "CPUTask2 Start Now() = " & Now()
4
5       For t = 1 To 100000000
6       'We create a loop that will take 5-10 seconds to
7       'complete
8           zzz = zzz + (t / 2)
9           If (t Mod 10000000) = 0 Then
10              DoEvents
11              'DoEvents pauses the loop so the operating
12              'system can perform queued functions
13
14              Debug.Print t
15          End If
16      Next
17
18      Debug.Print "CPUTask End Now() = " & Now()
19  End Sub
```

```
    The output in the immediate window will be:

    CPUTask Start Now() = 25/12/2012 14:52:07
    10000000
    20000000
    30000000
    40000000
    50000000
    60000000
    70000000
    80000000
    90000000
    100000000
    CPUTask End Now() = 25/12/2012 14:52:18
```

Figure 9.24

DoEvents is a useful function so you can create long loops that don't hold up the operating system.

Questions

1) True or False
   a. A loop is a circular object instantiated by ReDim'ing an object reference
   b. For and Step are part of the For statement
   c. Next denotes the end of a For code block
   d. An infinite loop is magic
   e. While used with Step is valid

2) What is the output of the following code

```
1   Function forLoop2()
2     Dim i As Integer
3     For i = 1 To 50 Step 10
4       Debug.Print "i=" & cstr(i)
5     Next i
6
7   End Function
```

3) Change the following code to print hello world ten times using i as your counter

```
1   Function whileLoop1()
2     Dim a As Boolean, i As Integer
3     While (Not a)
4       Debug.Print "Hello World!"
5     Wend
    End Function
```

4) Which of the following pieces of code are infinite loops

(a)	(b)	(c)

`While(true)`
`  Debug.print 1`
`Wend`
	`Do while(true)`
	`  Exit Do`
	`Loop`
		`Do Until(false)`
		`  'Exit`
		`Loop`

(d)	**(e)**	**(f)**

`For I = 1 to 10`
`  I = I -1`
`Next`
	`A=1`
	`Loop While (A=1)`
	`Loop`
		`A = 3`
		`While(A=0)`
		`A=A-1 : Wend`

5) When iterating over a collection, which loop structures would you use?

6) Which of the following are multi-dimensional arrays
   a. A = Array(10,5)
   b. Dim myString(20) As String

c. B(50,50)
d. Dim (9,9)myVar as Integer

7) Which of the following are characteristic of DoEvent
   a. Allows non-multi-tasking OS to “multi-task”
   b. Schedules a future event
   c. Allows Access forms to repaint
   d. Used in loops to relinquish CPU resources
   e. Reserves memory for an array

8) Write a For loop that prints out the following array
   carParts = Array(“Wheel”,”Door”,”Clutch”,”Flywheel”,”Wishbone”,”Sump”)

9) Write a While loop that loops 100 times printing to the immediate window every second iteration.

10) Write a For Each loop that iterates over the CurrentProject.AllMacros collection and prints their names to the immediate window.

11) Using the following arrays, complete the questions that follow
   aa = array(10,6,20,99)
   bb = array(1,2,3,4)
   cc = array(aa,bb)

aa(0)=	bb(3)=	cc(0)(0)=	cc(1)(0)=	aa(bb(0))=
bb(4)=	cc(1)(3)=	bb(8-aa(1))=	aa(0)+bb(3)=	cc(0)(2)=

   a. Could the above array be iterated using loops?
   b. Which loops would be most suitable and why?

12) Using a Integer array called “IDs” with 10 elements, populate the array with numbers 1 to 10

13) How many “Running!” lines are printed to the immediate window?

```
1  Function runningLoop()
2    While (false)
3      Debug.print “Running!”
4    Wend
5  End Function
```

14) When does the following loop exit?

```
1  Function exitAtFive()
2    Dim a as Integer : a = 100
3    While (a>=5)
4      a = a - 1
5    Wend
6  End Function
```

15) What is the result of the following:
    a. Dim a1(20) : UBound(a1) = ?
    b. Dim b(10) : LBound(b10) = ?
    c. Dim c As New Collection: c.Add "Hi": c.Add "#12/12/2010#": c.Count = ?

16) Examine the following function newChessboard()

```
   Function newChessboard()
1    Dim chessboard(8), pieces1, pieces2, places, none As String
2    pieces1 = Array("rook", "knight", "bishop", "king", _
3                    "queen", "bishop", "knight", "rook")
4    pieces2 = Array("pawn", "pawn", "pawn", "pawn", "pawn", _
5                    "pawn", "pawn", "pawn")
     none = "empty"
     places = Array(none, none, none, none, none, none, none, none)
     chessboard(0) = pieces1
     chessboard(1) = pieces2
     chessboard(2) = places
     chessboard(3) = places
     chessboard(4) = places
     chessboard(5) = places
     chessboard(6) = pieces2
     chessboard(7) = pieces1
     newChessboard = chessboard
   End Function
```

    a. Describe the output of the function

17) What is the difference between chessboard(8,8) and newChessboard in the above function?
    a. What I the purpose of the array pieces1

18) Write a loop that prints out chessboard(7)
    a. And, write a loop that prints out column 1 of the chessboard

19) Write a loop that prints only the positions "(x)(y)={content}" of squares that are not "empty"
    hint: you will need to use If, Loops and arrays

20) What happens if we ask what is in element chessboard(9)(2)?

Answers

1) True or false
    a. False
    b. True
    c. True
    d. False
    e. False
2) i=1
   i=11
   i=21
   i=31
   i=41
3)

```
1   Function whileLoop1()
2     Dim a As Boolean, i As Integer
3     While (Not a)
4       Debug.Print "Hello World!"
5       i = i + 1: If i = 10 Then a = True
6     Wend
7   End Function
```

4) True and false
    a. True
    b. False
    c. True
    d. True
    e. True
    f. False
5) For Each
6) True or false
    a. True
    b. False
    c. Could be true if option explicit is not set
    d. False
7) True or false
    a. True
    b. False
    c. True
    d. True
    e. False
8) One of the following

```
1   For each p in carParts
2     Debug.print p
3   next
4   --or--
5   For p = 0 to ubound(carParts)-1
6     Debug.print carParts(p)
7   Next
```

9) As follows

```
1    While (t < 100)
2    t = t + 1
3    If t Mod 2 Then Debug.Print t
4    Wend
```

10)

```
1    While (t < 100)
2    t = t + 1
3    If t Mod 2 Then Debug.Print t
4    Wend
```

11) aa = array(10,6,20,99)
bb = array(1,2,3,4)
cc = array(aa,bb)

aa(0)=10	bb(3)=4	cc(0)(0)=10	cc(1)(0)=6	aa(bb(0))=10
bb(4)=error	cc(1)(3)=4	bb(7-aa(1))=2	aa(0)+bb(3)=14	cc(0)(2)=20

a) Yes
b) For loop or for each. For loops clearly show and restrict how many elements will be iterated in each loop. While and other loops are not restricted and could execute infinitely.

12) any loop structure that increments a variable and assigns that value to IDs(variable)=variable

13) none

14) when a is less than 4

15) values
a. 20
b. 0
c. 2

16) An array chessboard (8) with each element containing another array.
chessboard(0) and chessboard(7) are the main pieces
chessboard(1) and chessboard(6) are the pawns
chessboard(2-5) are empty

17) Chessboard(8,8) creates a two dimensional array
newChessboard() returns a one-dimensional array, each dimension having another one-dimensional array.

18)

```
1   For each sq in chessboard(7)
2     Debug.print sq
3   Next
4
```

a)

```
1   For t=0 to 7
2     Debug.print chessboard(t)(1)
3   Next
4
```

19)

```
1   chessboard = newChessboard()
2   For y = 0 To 7
3     For x = 0 To 7
4      If chessboard(y)(x) <> "empty" Then
5       Debug.Print "position(" + CStr(y) + "," + CStr(x) + ")=" + chessboard(y)(x)
6      End If
7     Next
8   Next
9
```

20) out of bounds error