COMP1022Q
Introduction to Computing with Excel VBA

Using For Loops

David Rossiter and Gibson Lam

Outcomes

- After completing this presentation, you are expected to be able to:
  1. Write for loops to run code repeatedly in VBA
  2. Write Exit For code to stop the loops prematurely

For…Next

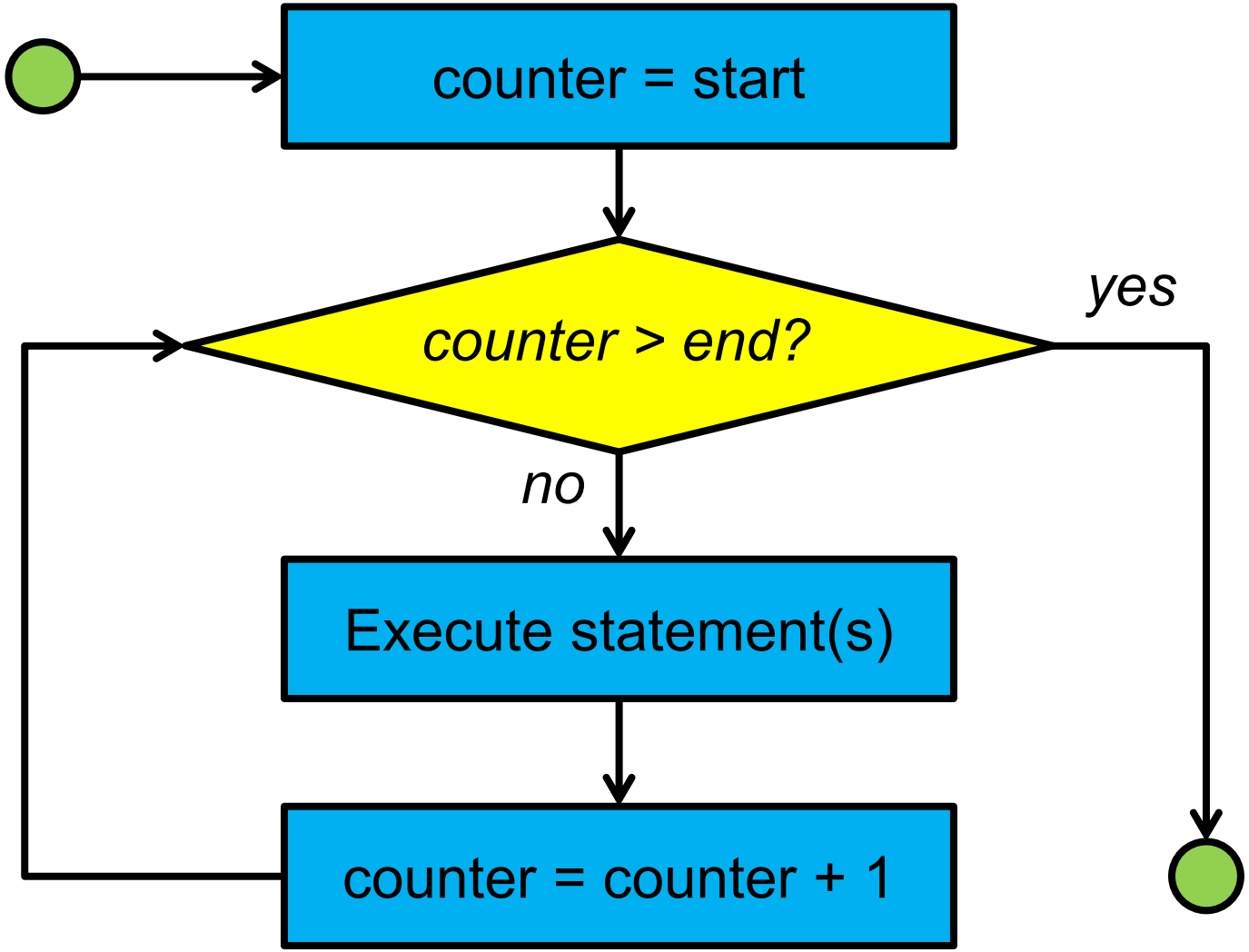
`For` *counter* `=` *start* `To` *end*

. . .statement(s). . .

`Next` *counter*

- *For…Next* uses a *counter* (a variable) that is equal to *start* at the start of the loop
- The *counter* increases after each iteration of the loop
- The loop executes up to and including the iteration when the value of *counter* is equal to *end*
- That means the number of times the loop repeats itself is (*end* - *start* + 1)

The Flow of For…Next



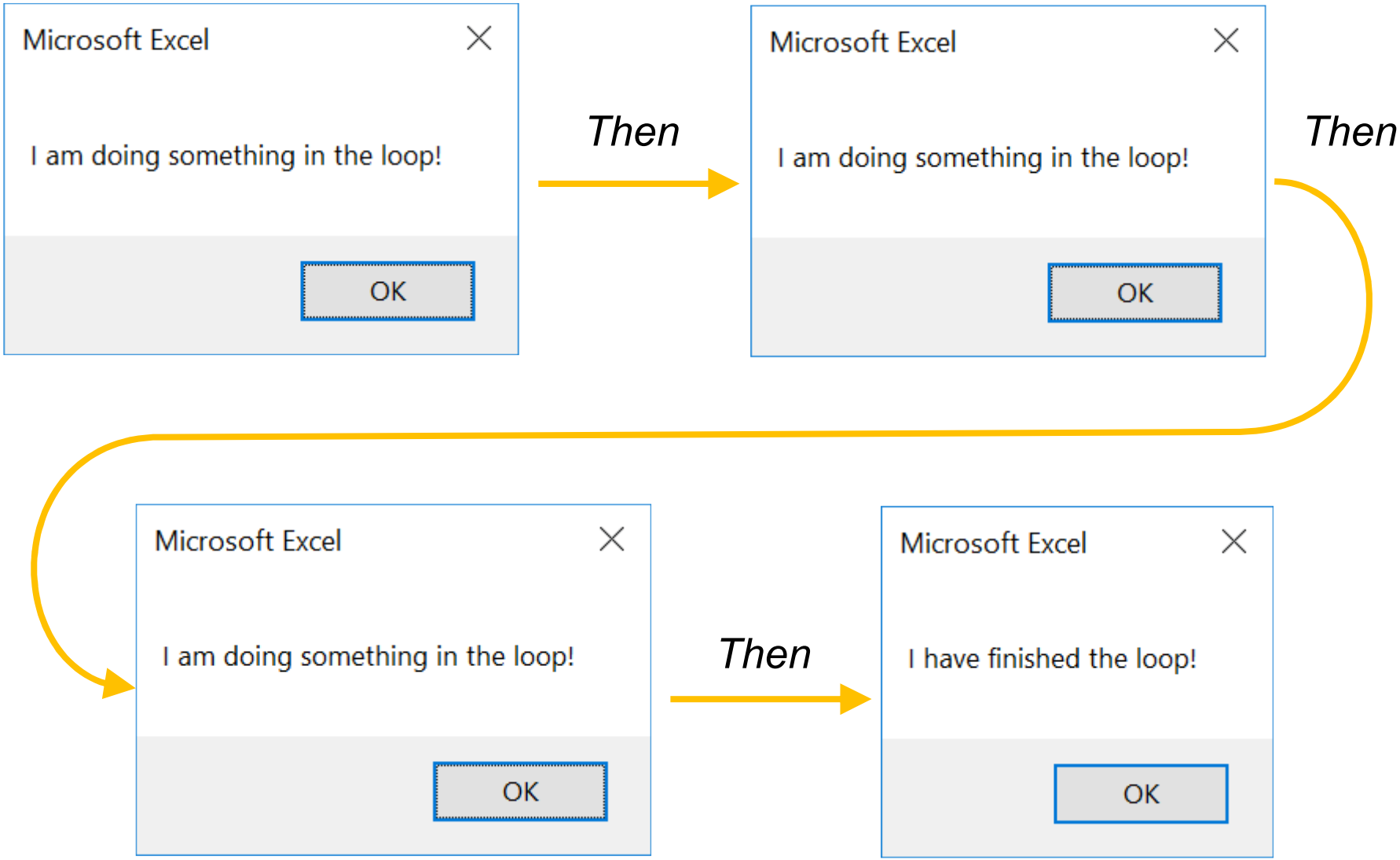
A Simple Example of For…Next

- Here is a simple example that runs the loop content three times

```
Dim Count As Integer

For Count = 1 To 3
    MsgBox "I am doing something in the loop!"
Next Count

MsgBox "I have finished the loop!"
```

This is the loop content

Running The Example



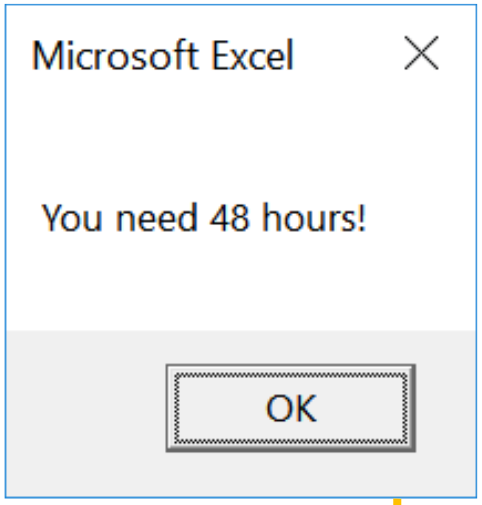
Another For Loop Example

- In this example, a for loop is used to count the number of hours you need to spend in the course
- The example assumes that:
  - There are 13 weeks in a semester
  - You attend 2 lectures per week (2 hours) for the course
  - You attend 1 lab per week (2 hours) starting from week 3 of the semester

```
Dim Week As Integer
Dim TotalHours As Integer

TotalHours = 0

For Week = 1 To 13
    ' Assume you go to two lectures a week
    TotalHours = TotalHours + 2

    ' Assume you go to one lab a week starting
    ' from week 3
    If Week >= 3 Then
        TotalHours = TotalHours + 2
    End If
Next Week

MsgBox "You need " & TotalHours & " hours!"
```

The variable to go through the 13 weeks of the course

You can choose to only write: `Next`
but it is clearer to write: `Next Week`



Using a While Loop

- The previous loop can be written using a while loop:

```
Dim Week As Integer
Dim TotalHours As Integer

TotalHours = 0
Week = 1
Do While Week <= 13
    TotalHours = TotalHours + 2
    If Week >= 3 Then
        TotalHours = TotalHours + 2
    End If
    Week = Week + 1
Loop
MsgBox "You need " & TotalHours & " hours!"
```

Week starts from 1 and ends at 13

Week increases by 1 inside the loop body

While Loops and For Loops

- Both while loops and for loops are used for repeating code
- For loops are good at going through a given range of numbers incrementally,
  whereas while loops are good at repeating things a number of times based on a flexible criteria
- As you can see, it is easy to write a while loop to do what a for loop does
  (however, it may not be so easy to write a for loop to do what a while loop does)

Using Exit For to Stop a For Loop

- A for loop normally repeats the loop body when the counter is from *start* to *end*

```
For counter = start To end
        . . .loop body. . .
Next counter
```

- If you want to, you can use `Exit For` inside the loop body to immediately stop the loop

```
For counter = start To end
        . . .
        Exit For
        . . .
Next counter
. . .
```

This means stop the loop immediately no matter what the value of the loop counter is

A Simple Example of Exit For

- Here is an example which puts a message in each of the first five rows of a worksheet:

```
Dim Row As Integer
For Row = 1 To 10
    Cells(Row, 1).Value = _
        "Hello, row " & Row
    If Row >= 5 Then
        Exit For
    End If
Next Row
```

The loop normally runs 10 times

Finish the loop immediately when the current row is the fifth row

	A
1	Hello, row 1
2	Hello, row 2
3	Hello, row 3
4	Hello, row 4
5	Hello, row 5
6	
7	

Storing A Big Integer

- So far we know about 2 types of variable:

```
Dim MyFavouriteText As String 'stores text
Dim MyFavouriteNumber As Integer 'stores an integer
```

- One problem with an *Integer* variable is that it cannot store a big number such as 40000

```
MyFavouriteNumber=40000 'This makes an error!
```

- If we want to do that we can use a *Long* variable:

```
Dim MyFavouriteBigNumber As Long
MyFavouriteBigNumber = 40000  'Doing this is OK
```

Another Example of Exit For

- In this example, a table shows your projected savings per year starting from the age of 21 to 70
- When you open the Excel file you are asked to enter the amount of money you need when you retire
- The VBA code then finds the age you can retire by accumulating the savings until the total is more than or equal to what you need

	A	B
4	**Age**	**Savings Per Year**
5	21	HK$ 20,000.00
6	22	HK$ 22,000.00
7	23	HK$ 23,100.00
8	24	HK$ 24,255.00
9	25	HK$ 25,467.75
10	26	HK$ 26,741.14
11	27	HK$ 28,078.19
12	28	HK$ 29,482.10
13	29	HK$ 30,956.21
14	30	HK$ 32,504.02
	⋮	
52	68	HK$ 207,553.68
53	69	HK$ 217,931.36
54	70	HK$ 228,827.93

The Code of the Example 1/3

- In this first part of the code, some variables are created and the code asks the user for the target savings:

```
Dim Target As Long, Total As Long
Dim Age As Integer, Row As Integer

Target = InputBox( _
   "How much do you " & _
   "need to stop working?")
```

- Here are two examples of how to create two variables in one line of VBA code
- `Target` and `Total` are both Long
- `Age` and `Row` are both Integer

The Code of the Example 2/3

- The second part of the code accumulates the savings using a for loop:

```
Total = 0
For Age = 21 To 70
    ' The values of yearly
    ' savings start from B5
    Row = Age - 16
    Total = Total + Cells(Row, 2).Value

    If Total >= Target Then
        Exit For
    End If
Next Age
```

The loop accumulates the savings from the age of 21 to 70

When the total is bigger than or equal to the target there is no need to accumulate the savings so the loop is stopped

The Code of the Example 3/3

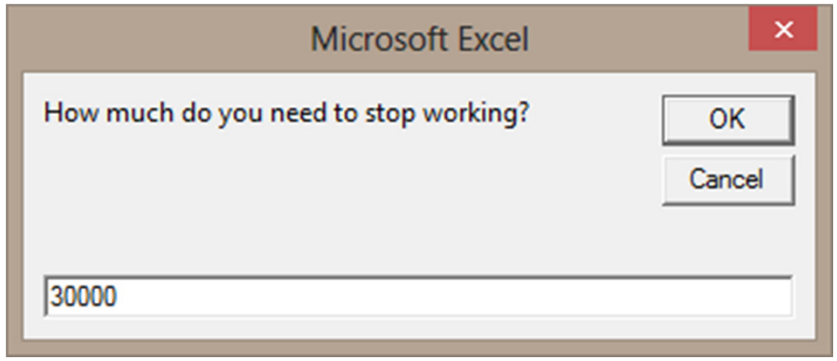
- The last part of the code shows the result by comparing the total savings (`Total`) and the target (`Target`):

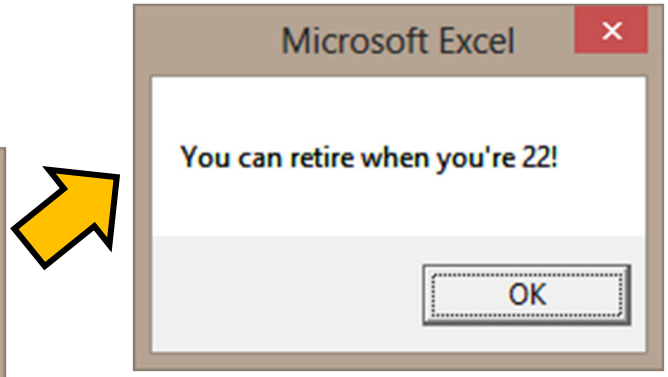
```
If Total >= Target Then
    MsgBox "You can retire when you're " & _
           Age & "!"
Else
    MsgBox "You cannot retire even at 70!"
End If
```

Running the Example

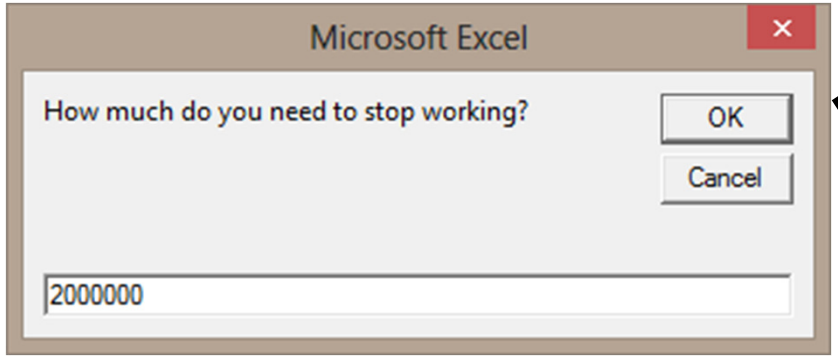
- Let's try the example:

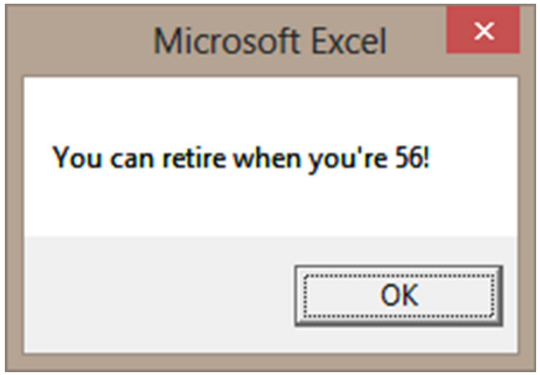
If you need HK$30,000:





If you need HK$2,000,000:





If you need HK$5,000,000: