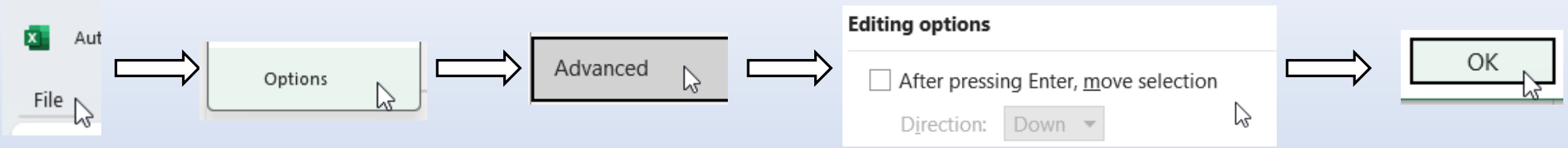
Excel Bootcamps 1, 2, 3 and 4

- ✓ 1: Getting up to speed with Excel
- 2: Introducing VBA
- 3: Learning to use Excel to solve typical problem scenarios
- 4: Detailed modeling of packed-bed and plug-flow reactors

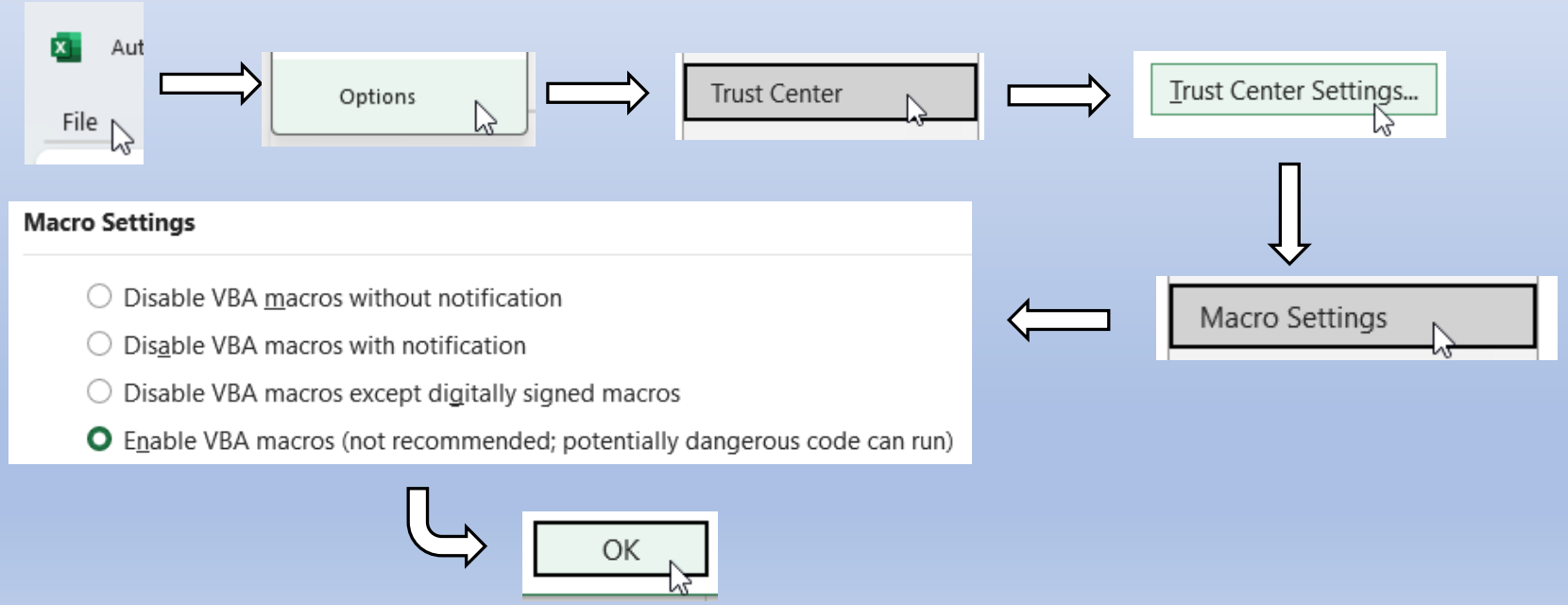
Bootcamp 2 Outline

Excel Settings for Use of VBA

Remove option to move selection down after Enter key is pressed.

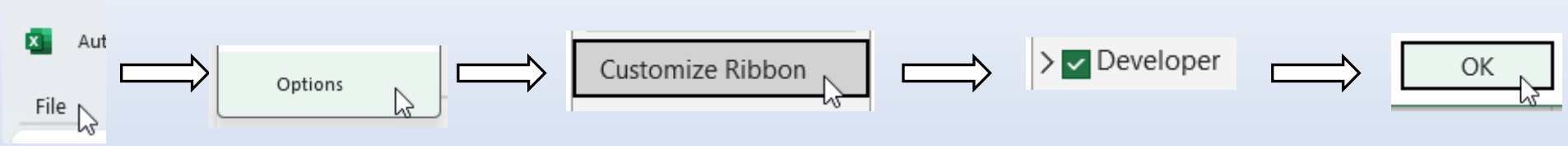


Enable VBA macros

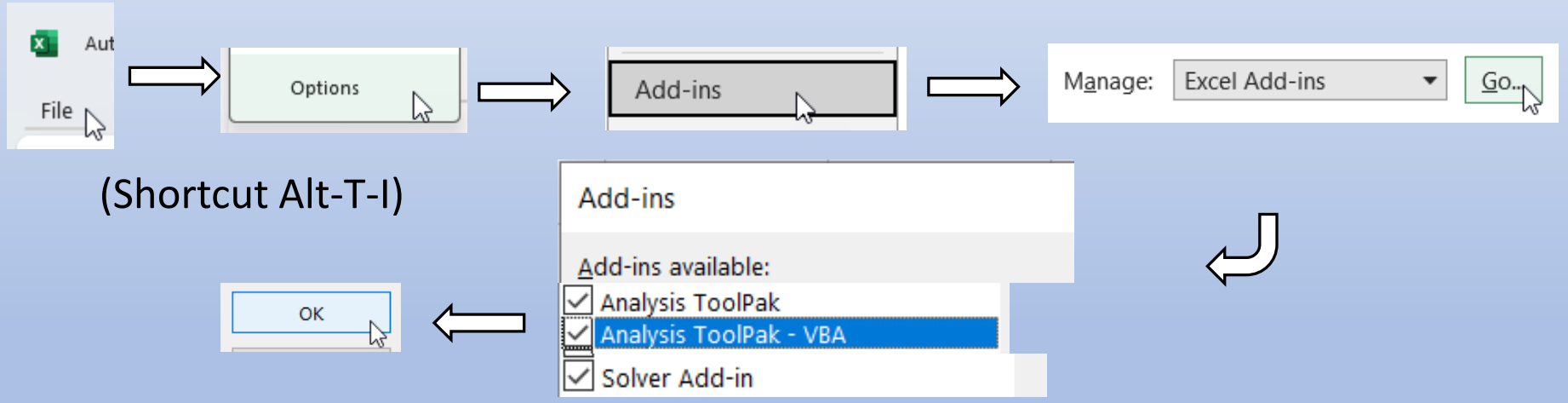


Excel Settings for Use of VBA

Enable the Developer tab on the Ribbon



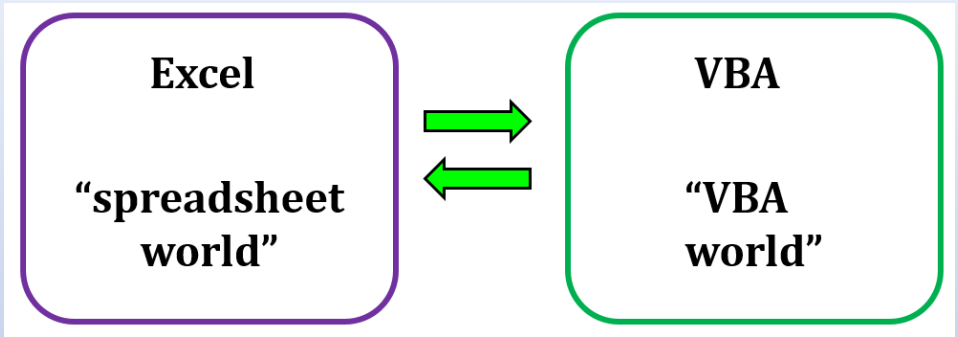
Enable the Analysis Toolpak – VBA
(in addition to the Analysis Toolpak and Solver)

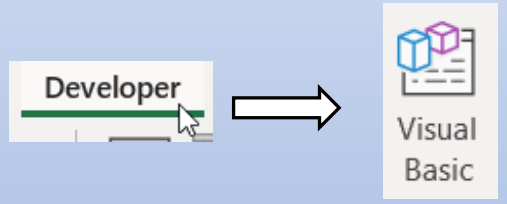


(Shortcut Alt-T-I)

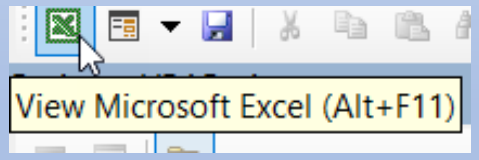
Excel and the Visual Basic Editor

Two separate environments – switching back and forth





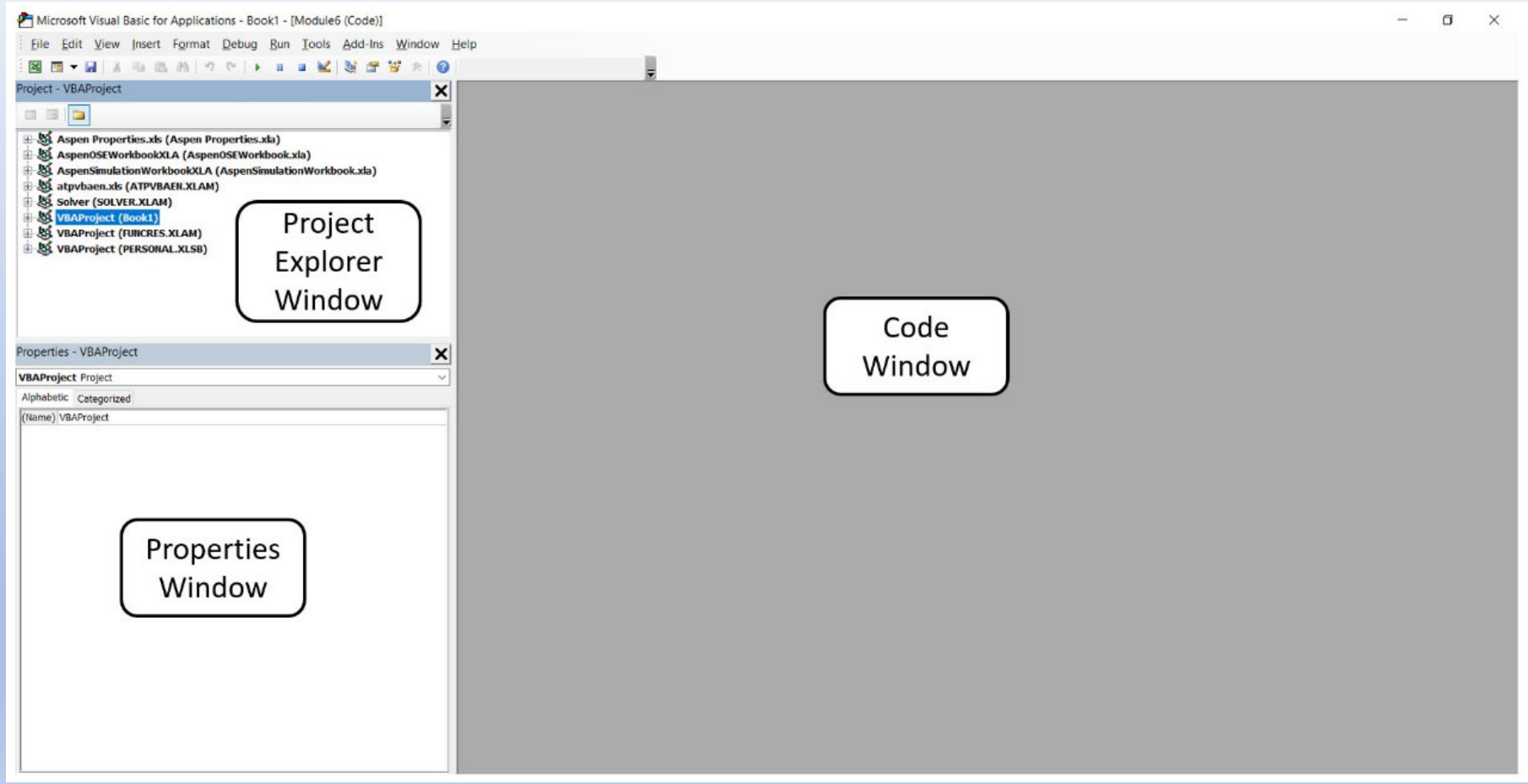
or Alt-F11



or Alt-F11

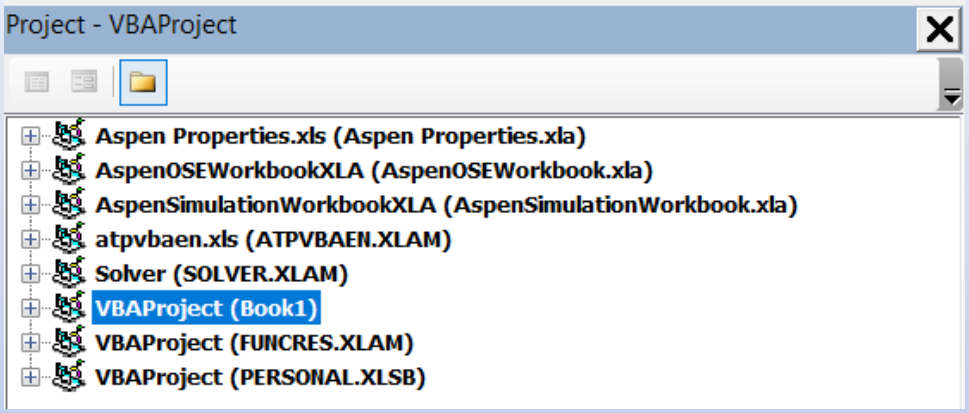
The Visual Basic Editor (VBE)

Layout of the VBE

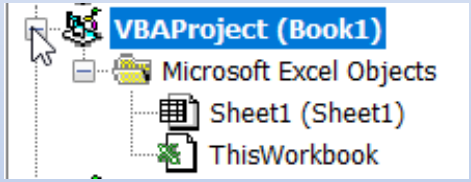


The Visual Basic Editor (VBE)

Project Explorer



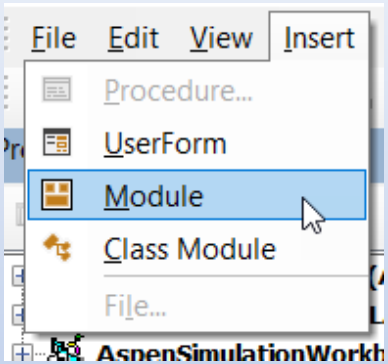
Open branch for VBAProject associated with current Excel workbook (Book1 here)

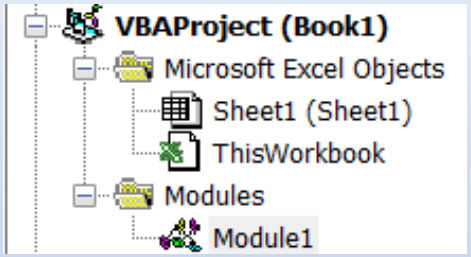


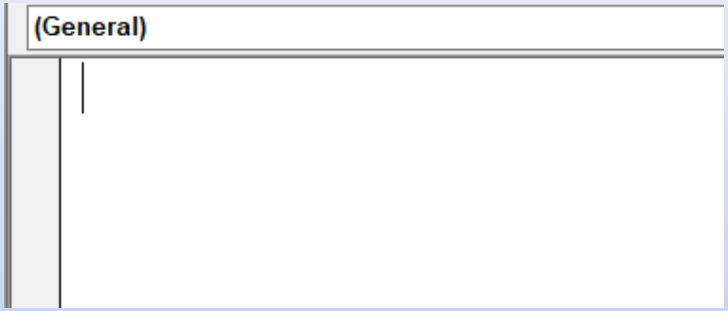
Always make certain that your VBA work is with respect to the appropriate project.

The Visual Basic Editor (VBE)

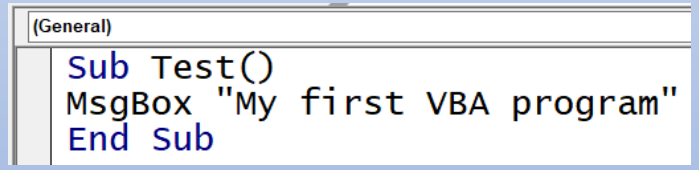
Adding a code module to the project



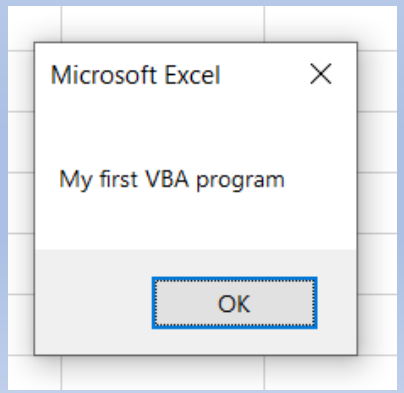




Type in a simple program



With the cursor (anywhere) in the code, run the Sub program with the F5 key

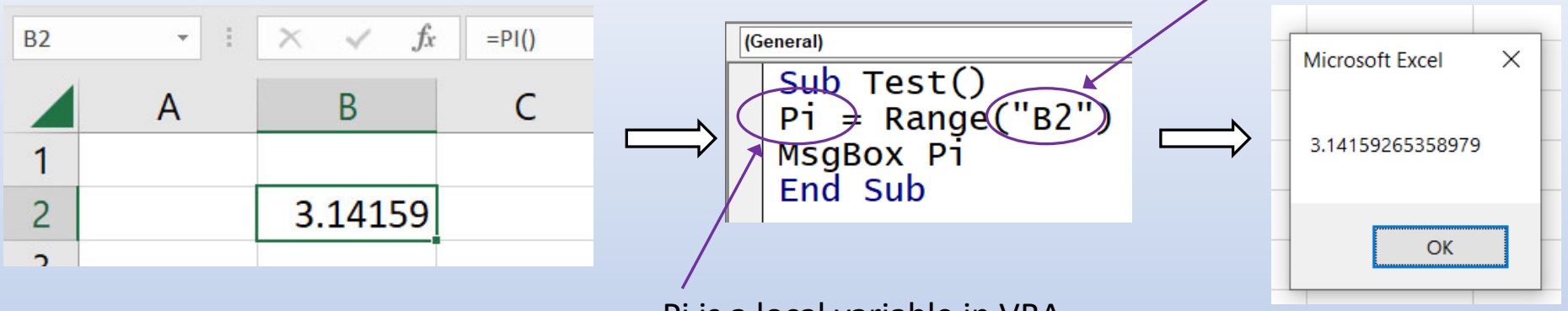


Appears on the Excel worksheet

Click OK to finish

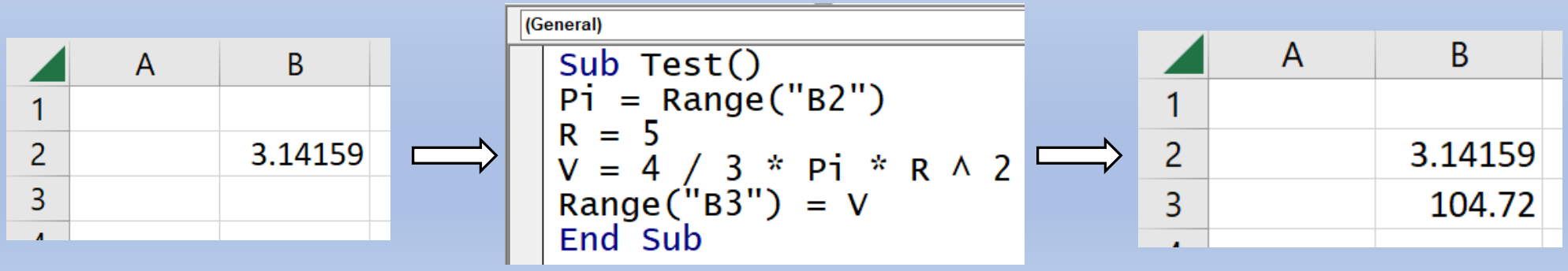
VBA – Exchanging Information with “Spreadsheet World”

Acquiring the value from a spreadsheet cell using the Range type



Sending a value from VBA to a spreadsheet cell using the Range type

Run Subs with F5

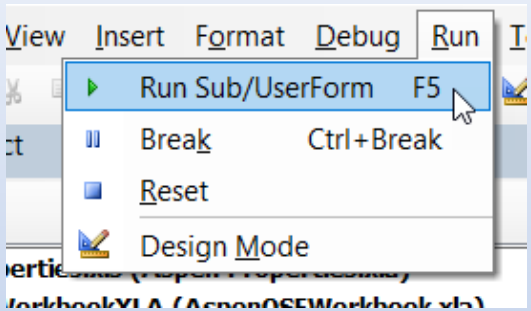


VBE provides spacing once Enter key is pressed on a line. (Must enter spaces around ^.)

Ways to Run a Sub

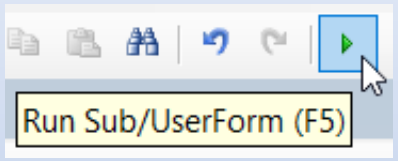
"Sub" is short for *subroutine procedure*. It is a subroutine because it is subservient to the Excel application.

From the VBE



Menu

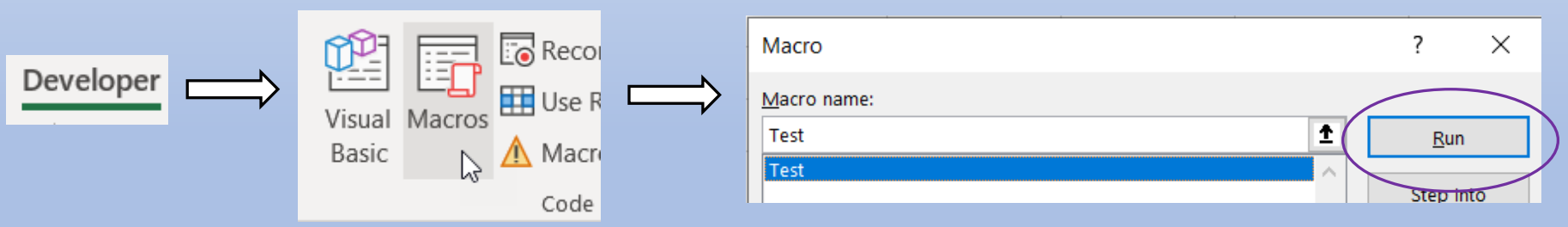
or



Toolbar

or the F5 key

From the spreadsheet



or Alt-F8

Recording a Sub as a Macro

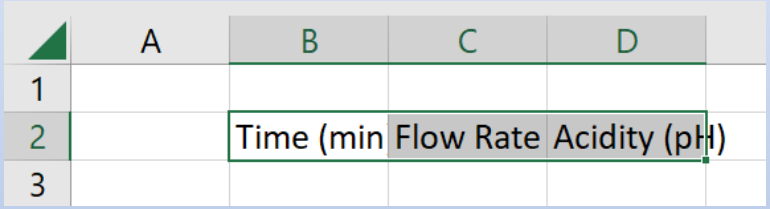
Suitable for spreadsheet procedures that are repeated often.

Example: Center and wrap table headings

Enter the headings:

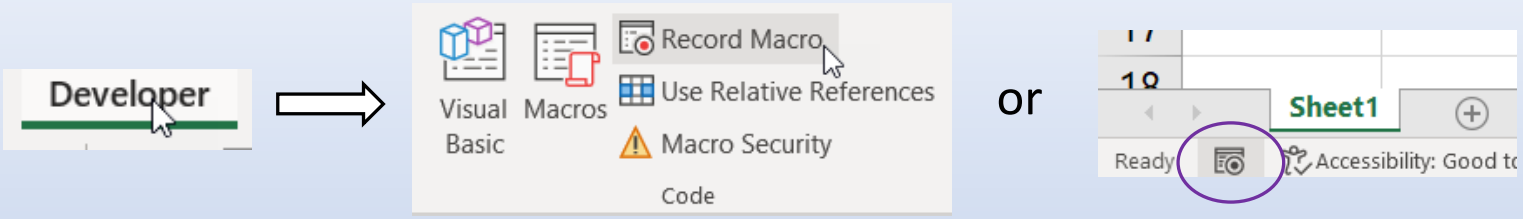
B2: Time (min) C2: Flow Rate (Lpm) D2: Acidity (pH)

and select those cells.



Recording a Sub as a Macro

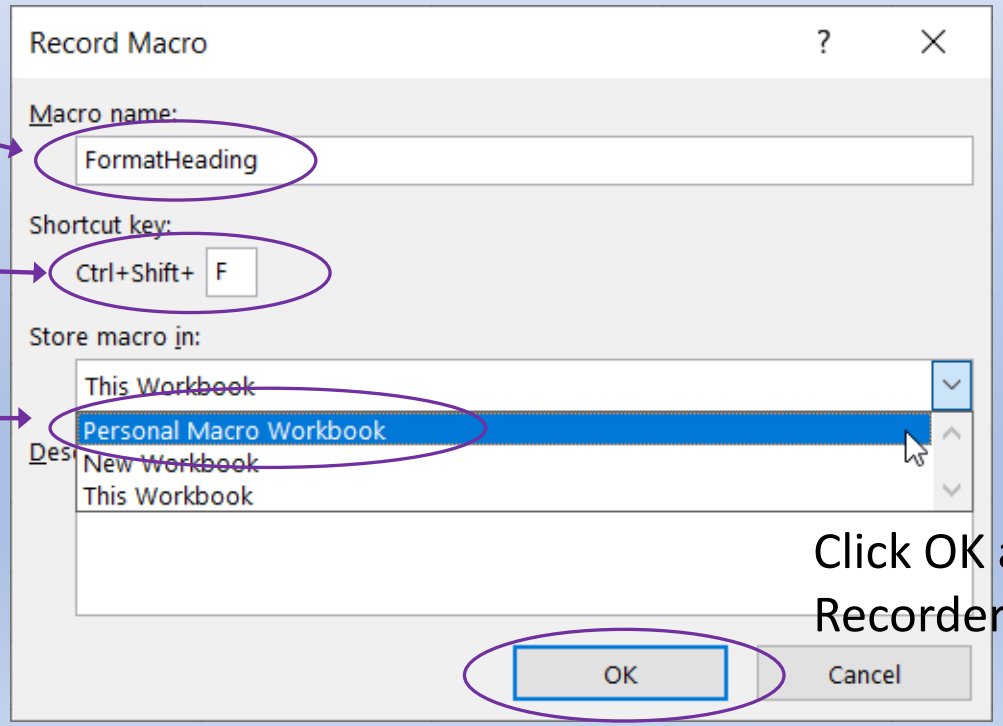
Start the Macro Recorder



Name the macro (no spaces)

Specify a shortcut letter (cap F here)

Store a generic macro here

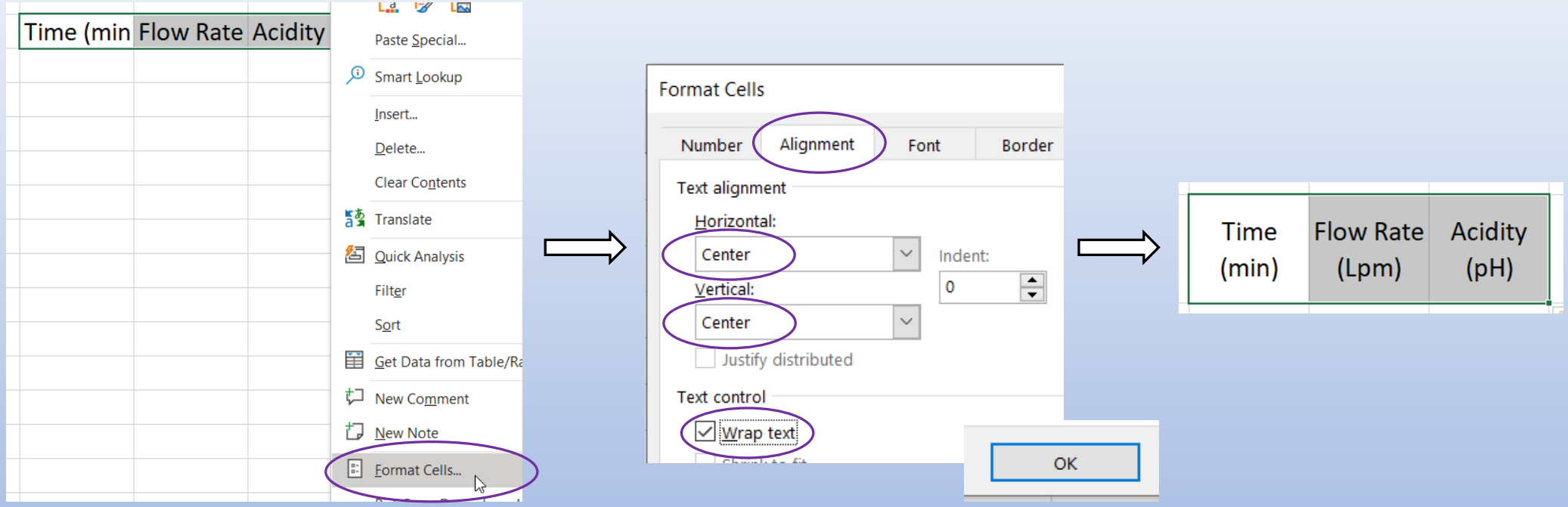


Click OK and the Recorder is ON.

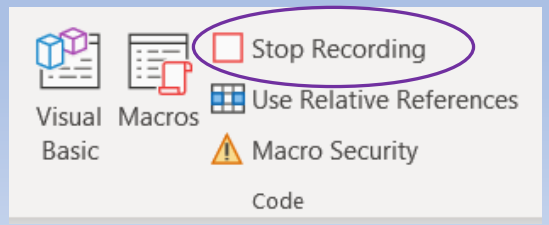
Avoid lower-case letters except e, j, l, m, q and t.
Avoid only upper-case F, O and P.

Recording a Sub as a Macro

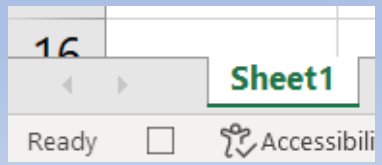
Carry out the procedure



Turn off the Recorder

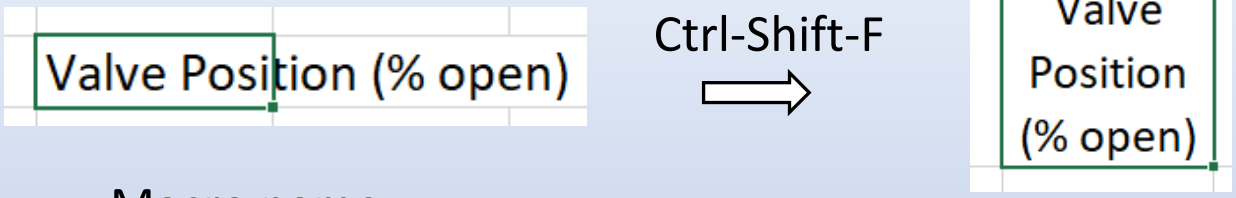


or



Recording a Sub as a Macro

Test the macro



Macro name

Look at the recorded VBA code in a module under the PERSONAL.XSLB project

Changes that were made

These are default values -- we can remove these lines.

```
Sub FormatHeading()
'
' FormatHeading Macro
'
' Keyboard Shortcut: Ctrl+Shift+F
'
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .WrapText = True
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = False
    End With
End Sub
```

Comments are in green, preceded by '

Selection is the object and the .properties are between **With** and **End With**.

Recording a Sub as a Macro

Edited macro

```
Sub FormatHeading()
'
' FormatHeading Macro
'
' Keyboard Shortcut: Ctrl+Shift+F
'
    With Selection
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .WrapText = True
    End With
End Sub
```

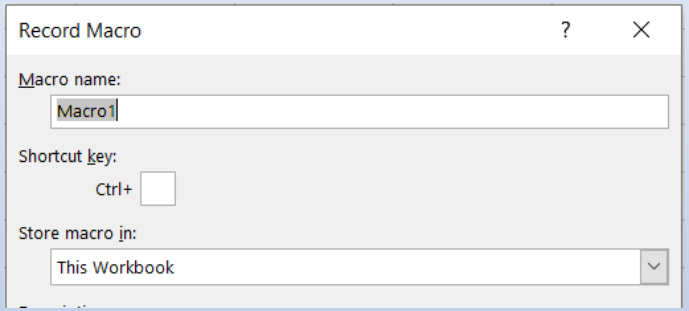
- Only create generic macros for frequently used operations.
- Macros particular only to the current project should be stored in This Workbook.
- The Macro Recorder will sometimes get things wrong, and you have to fix the code.
- VBA code may be difficult to write from scratch, but it is relatively easy to read.
- The first time you create a generic macro, Excel will ask whether you want to save the Personal Macro Workbook when you exit. Click YES.

The Macro Recorder as a Code Detective

Find obscure VBA commands for spreadsheet manipulations

Example: how to select a column of filled cells

	A	B
1		
2		0.26
3		0.08
4		0.79
5		0.88
6		0.31
7		0.30
8		0.81
9		0.43
10		0.98
11		1.00



Ctrl-Shift-↓

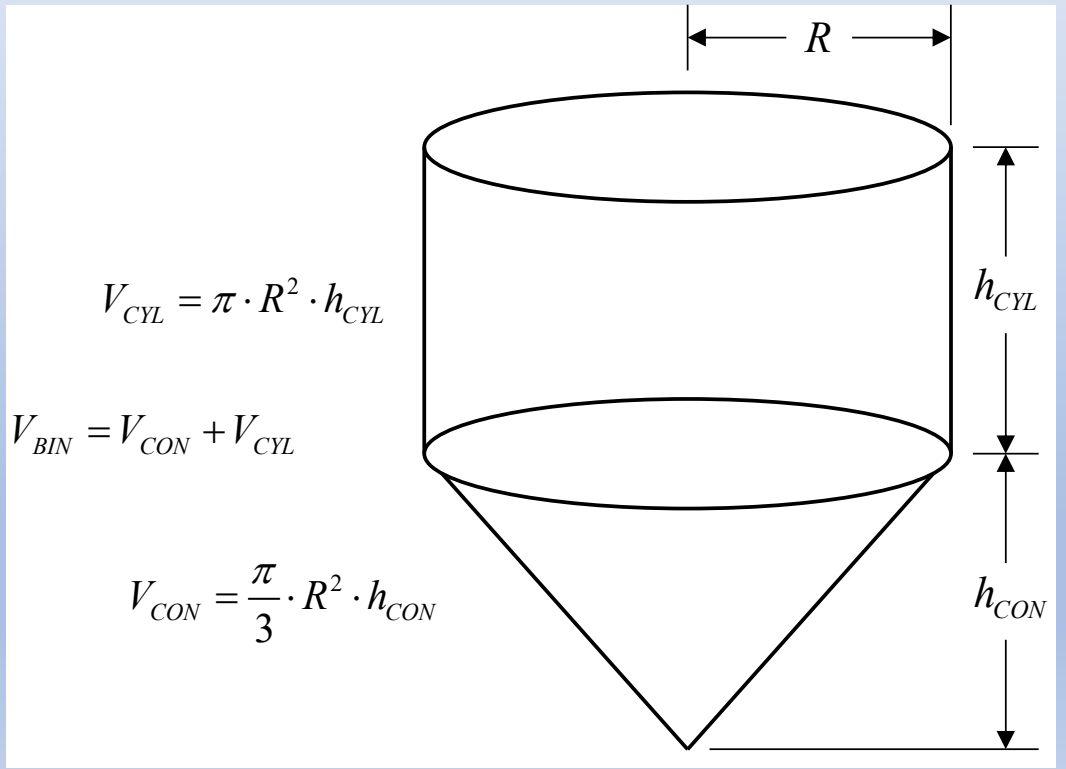
```
Sub Macro1()
'
' Macro1 Macro
'

'
    Range(Selection, Selection.End(xlDown)).Select
End Sub
```

Note or copy to other VBA code

User-defined Functions

Create your own functions when Excel or VBA doesn't have what you need.
On the spreadsheet, a common example is to create a function for an engineering formula.
In VBA, an example is to provide a mathematical function that VBA doesn't provide.

Example: Create a function to compute the volume of a cylindrical bin with a conical base.



User-defined Functions

Create a prototype calculation on the spreadsheet

	B	C	
2	Radius	1.5	m
3	h_{CYL}	3	m
4	h_{CON}	2.4	m
5	Volume		m^3

	B	C	D	E	F
2	Radius	1.5	m		
3	h_{CYL}	3	m		
4	h_{CON}	2.4	m		
5	Volume	=pi()*Radius^2*hCYL+pi()*Radius^2*hCON/3			

	B	C	
2	Radius	1.5	m
3	h_{CYL}	3	m
4	h_{CON}	2.4	m
5	Volume	26.86	m^3

User-defined Functions

Design what the function will be named and how it will work on the spreadsheet.

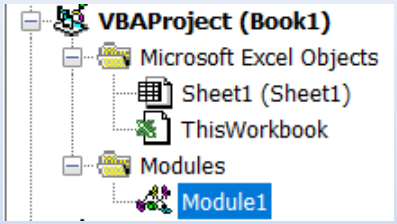
	B	C	D	E
2	Radius	1.5	m	
3	h_{CYL}	3	m	
4	h_{CON}	2.4	m	
5	Volume			
6		=BinVolume(Radius,hcyl,hcon)		

	B	C	
2	Radius	1.5	m
3	h_{CYL}	3	m
4	h_{CON}	2.4	m
5	Volume	26.86	m^3
6		#NAME?	

Of course, this generates a #NAME? error because we haven't programmed the function yet.

User-defined Functions

Insert a module in the project and enter the code as shown.




```
Function BinVolume(r, hcyl, hcon)
Pi = 4 * Atn(1)
Vcyl = Pi * r ^ 2 * hcyl
Vcon = Pi * r ^ 2 * hcon / 3
BinVolume = Vcyl + Vcon
End Function
```

Arguments **(r, hcyl, hcon)** don't have to match what we used on the spreadsheet, but the order has to be consistent.

The Excel **PI()** function isn't available in VBA, so we compute **Pi** as 4 times $\pi/4$.

Remember to enter spaces around the **^** operator.

The answer has to be assigned to the name of the function, **BinVolume**.

Edit the function formula on the spreadsheet (F2, Enter) and this is the result:

	B	C	
2	Radius	1.5	m
3	h_{CYL}	3	m
4	h_{CON}	2.4	m
5	Volume	26.86	m^3
6		26.8606	

The function can now be used elsewhere in the workbook with cell address, name or numerical arguments.

VBA Program Structure Elements

Selection – decisions, if structures
- If-Then
- If-Then-Else
- If-Then-Elseif
- Select Case

Repetition – iteration, looping structures
- Do-Loop
- For-Next

VBA Program Structure Elements

If-Then

If *condition* Then
statements
End If

One-line Verstion

If *condition* Then *statement(s)*

Example: Return the square of the argument but preserve its sign.

```
Function SgnSqr(x)
SgnSqr = x ^ 2
If x < 0 Then
    SgnSqr = -SgnSqr
End If
End Function
```

```
Function SgnSqr(x)
SgnSqr = x ^ 2
If x < 0 Then SgnSqr = -SgnSqr
End Function
```

VBA Program Structure Elements

If-Then-Else

If *condition* Then

true statements

Else

false statements

End If

Example: Calculate the Fanning friction factor based on the Reynolds number

```
Function Fanning(Re)
If Re < 2100 Then
    Fanning = 16 / Re
Else
    Fanning = 0.079 / Re ^ (1 / 4)
End If
End Function
```

VBA Program Structure Elements

If-Then-ElseIf

If *condition_1* Then
statements_1
ElseIf *condition_2* Then
statements_2
-
-
ElseIf *condition_n* Then
statements_n
Else (optional)
else statements
End If

Example: Calculate the temperature (°C) of the atmosphere at different elevations (m)

```
Function AtmTemp(h)
If h < 11000 Then
    T = 15.04 - 0.00649 * h
ElseIf h < 25000 Then
    T = -56.46
Else
    T = -131.21 + 0.00299 * h
End If
AtmTemp = T
End Function
```

TemperatureStandardAtmosphere.xlsm

VBA Program Structure Elements

Select Case

Select Case *test expression*
  Case *list_1*
    statements_1
  Case *list_2*
    statements_2
      •
      •
      •
  Case Else
    else statements
End Select

Example: Provide a menu-like choice for conversion from °C to other units.

```
Sub TempConvert()
TempIn = InputBox("Enter temperature in degC: ")
Choice = InputBox("Enter your letter of choice" _
                  & vbCrLf & _
                  "A: degF" & vbCrLf & _
                  "B: K" & vbCrLf & _
                  "C: degR" & vbCrLf)
Select Case Choice
    Case "A"
        TempOut = 1.8 * TempIn + 32
    Case "B"
        TempOut = TempIn + 273.15
    Case "C"
        TempOut = 1.8 * TempIn + 491.67
End Select
MsgBox TempOut
End Sub
```

TempOut.xlsm

VBA Program Structure Elements

Repetition – the Do-Loop

Do

pre-test statements

If *condition* Then Exit Do

post-test statements

Loop

Either pre-test or post-test code can be missing.
There can be more than one **If ... Exit Do** statements.

General example: Convergence of a numerical method

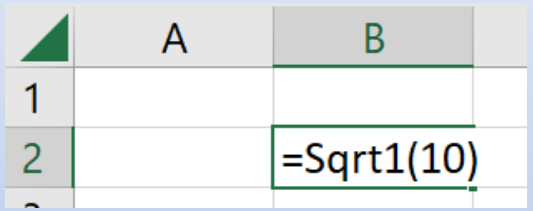
```
tolerance = small number, e.g., 1.e-7
x = initial estimate
Do
   xnew = get next estimate using x
   error = |(xnew-x)/x|
   If error < tolerance Then Exit Do
   x =xnew
Loop
```

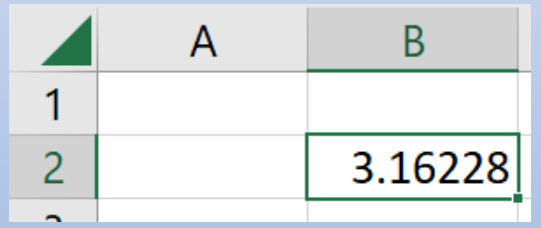
VBA Program Structure Elements

Repetition – the Do-Loop

Example: Finding the square root

```
Function Sqrt1(x)
If x <= 0 Then
    Sqrt1 = "error, x must be > 0"
Else
    tol = 0.0000001
    s = x / 2
    Do
        snew = (s + x / s) / 2
        er = Abs((snew - s) / snew)
        If er < tol Then Exit Do
        s = snew
    Loop
    Sqrt1 = snew
End If
End Function
```





Sqrt1Function.xlsm

VBA Program Structure Elements

Repetition – the For-Next loop

For *index* = *start* To *limit* Step *increment*
loop statements
Next *index*

Step *increment* is optional
If omitted, ***increment*** = 1

index variable often used as array subscript

Example: Case study of a function

```
Sub CaseStudy()
Range("B2").Select
For i = 0 To 100
    x = i / 100
    f = Cos(x) * Atn(x)
    ActiveCell.Offset(i, 0) = x
    ActiveCell.Offset(i, 1) = f
Next i
End Sub
```

0	0
0.01	0.01
0.02	0.01999
0.03	0.02998
0.04	0.03995
0.05	0.0499

• • •

0.97	0.43538
0.98	0.43186
0.99	0.42818
1	0.42435

VBA Program Structure Elements

Repetition – the For-Next loop

Premature exit from the For-Next loop with the Exit For command

```
Sub CaseStudy()
Range("B2").Select
For i = 0 To 100
    x = i / 100
    f = Cos(x) * Atn(x)
    ActiveCell.Offset(i, 0) = x
    ActiveCell.Offset(i, 1) = f
    If f >= 0.4 Then Exit For
Next i
End Sub
```

0	0
0.01	0.01
0.02	0.01999
0.03	0.02998
0.04	0.03995
0.05	0.0499
•	• •
0.46	0.38632
0.47	0.39172
0.48	0.39695
0.49	0.402

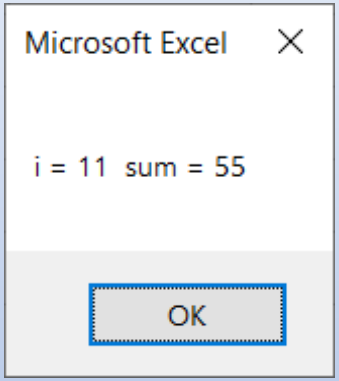
Case study stops when f(x) > 0.4

VBA Program Structure Elements

Repetition – the For-Next loop

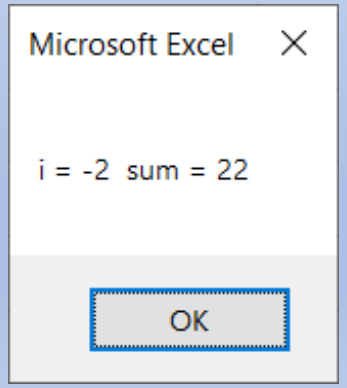
index variable is one step beyond the ***limit*** upon exit

```
Sub ForIndexExample()
For i = 1 To 10
    Sum = Sum + i
Next i
MsgBox "i = " & i & "  sum = " & Sum
End Sub
```



ForNextCaseStudy1.xlsm

```
Sub ForIndexExample()
For i = 10 To 1 Step -3
    Sum = Sum + i
Next i
MsgBox "i = " & i & "  sum = " & Sum
End Sub
```



ForNextCaseStudy2.xlsm

Working with Arrays in VBA

Subscript Base

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

ArrayExamples.xlsm

Base zero

```
Sub ArrayTest()
Dim a(1), B(1, 1)
a(0) = 1: a(1) = 2
B(0, 0) = 1: B(0, 1) = 2
B(1, 0) = 3: B(1, 1) = 4
End Sub
```

Base one

```
Option Base 1
Sub ArrayTest()
Dim a(2), B(2, 2)
a(1) = 1: a(2) = 2
B(1, 1) = 1: B(1, 2) = 2
B(2, 1) = 3: B(2, 2) = 4
End Sub
```

Working with Arrays in VBA

Arrays as Macro Arguments

FindMaxLocation.xlsm

```
Option Explicit
Option Base 1
Function MaxLoc(DataArray)
Dim m As Integer, n As Integer
Dim maxval, i As Integer, j As Integer
Dim imax As Integer, jmax As Integer
Dim MaxFind(2)
m = DataArray.Rows.Count
n = DataArray.Columns.Count
maxval = DataArray(1, 1)
For i = 1 To m
    For j = 1 To n
        If DataArray(i, j) > maxval Then
            maxval = DataArray(i, j)
            imax = i
            jmax = j
        End If
    Next j
Next i
MaxFind(1) = imax
MaxFind(2) = jmax
MaxLoc = Application.WorksheetFunction.Transpose(MaxFind)
End Function
```

Use the Rows.Count and Columns.Count properties to determine the size of the array.

Find the maximum value in DataArray and save the row and column location.

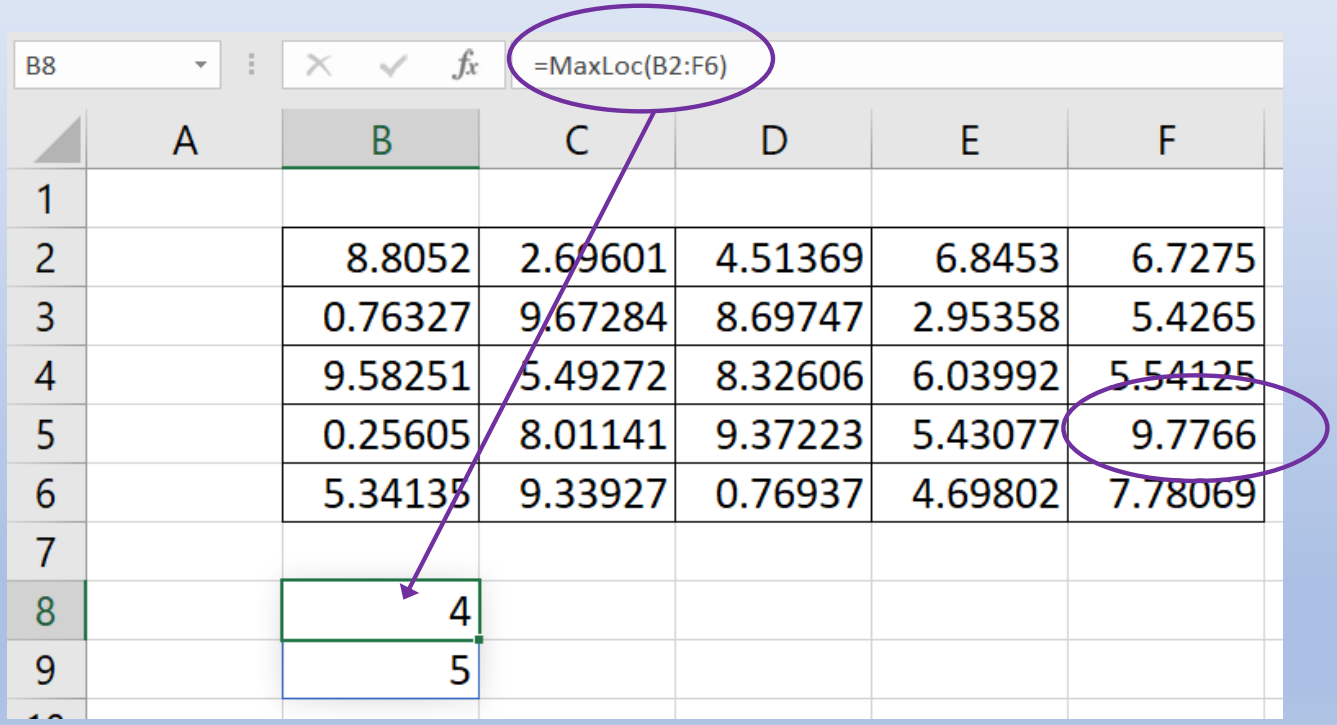
Return the row and column locations to the spreadsheet in two vertical cells.

Borrow the Transpose function from “spreadsheet world.”

Working with Arrays in VBA

Arrays as Macro Arguments

Example



VBA Details

Borrowing Excel's built-in functions

Application.WorksheetFunction.*function_name*(*arguments*)

not required

Examples

Application.WorksheetFunction.Atan2(x,y)

Application.WorksheetFunction.Average(DataArray)

Application.WorksheetFunction.Transpose(DataArray)

VBA Details

Using the Offset property

CellName.Offset(row count, column count).Value

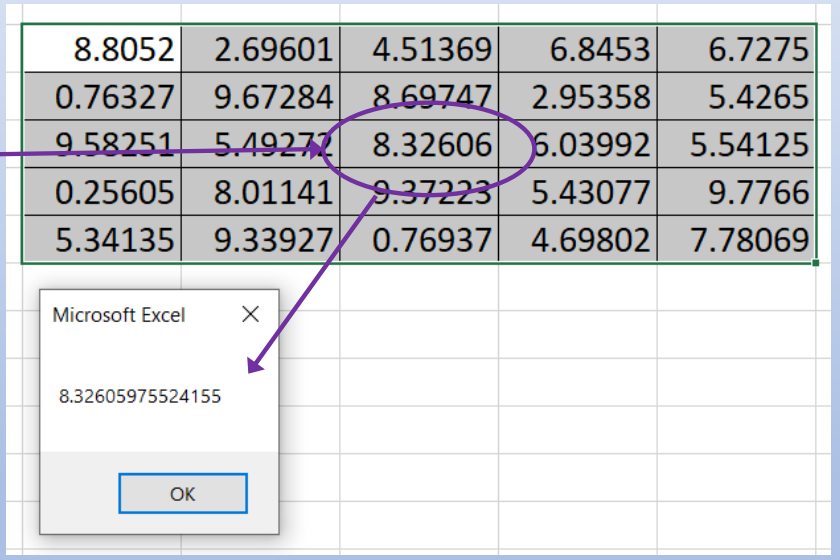
or .Select

Example

```
Option Explicit
Sub OffsetTest()
Dim DataValue
DataValue = ActiveCell.Offset(2, 2).Value
MsgBox DataValue
ActiveCell.Offset(2, 2).Select
End Sub
```

2 down, 2 to the right

8.8052	2.69601	4.51369	6.8453	6.7275
0.76327	9.67284	8.69747	2.95358	5.4265
9.58251	5.49272	8.32606	6.03992	5.54125
0.25605	8.01141	9.37223	5.43077	9.7766
5.34135	9.33927	0.76937	4.69802	7.78069



References:

Spreadsheet Problem Solving and Programming for Engineers and Scientists,
David E. Clough and Steven C. Chapra,
CRC Press - Taylor & Francis Group, 2024.

Excel 2019 Power Programming with VBA,
Michael Alexander and Dick Kusleika,
Wiley, 2019/

"Prof. Clough, may I be excused? My brain is full."

What's next?

Excel Bootcamps 1, 2, 3 and 4

- ✓ 1: Getting up to speed with Excel
- ✓ 2: Introducing VBA
- • 3: Learning to use Excel to solve typical problem scenarios
- • 4: Detailed modeling of packed-bed and plug-flow reactors