(U4284) Python程式設計
Excel - VBA

Speaker: 吳淳硯

Fundamentals

- What is VBA?

Visual Basic for Applications.

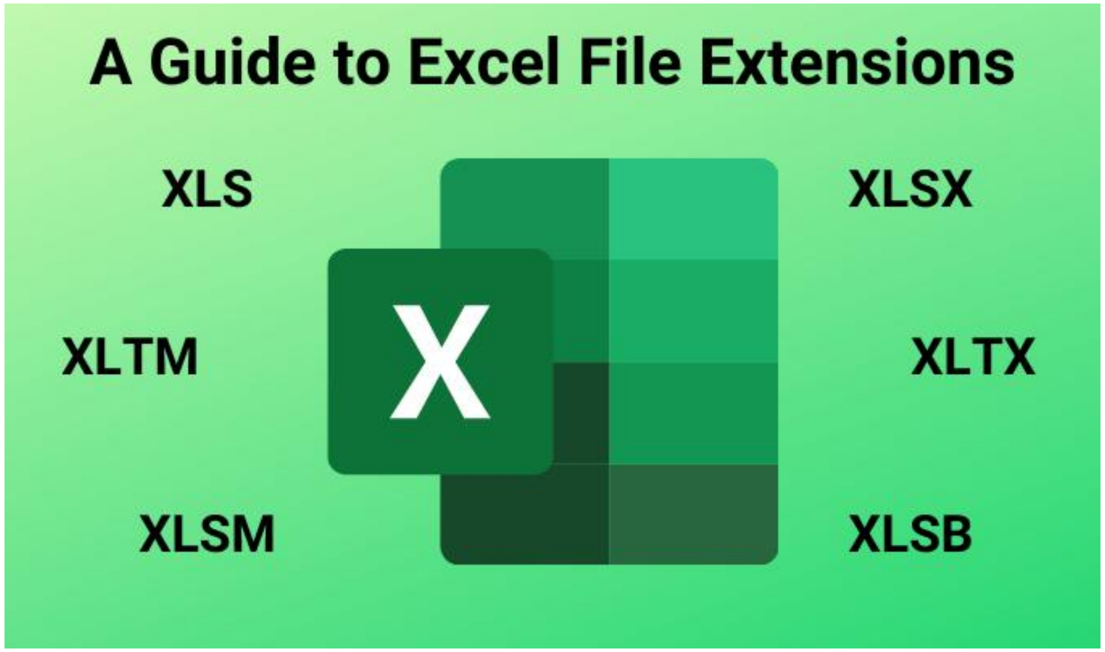
- Why learn VBA?
  - Save Time
  - Make Excel Smarter
  - Be More Valuable at Work

- Why VBA is easy to learn?
  - Simple syntax
  - Built-In Recorder

VBA makes Excel smarter, faster, and more powerful!

國立臺北大學 National Taipei University

File Type



- .xls(.xlt) file extension is the default Excel format for all versions of excel prior to Excel 2007. From Excel 2007 onwards, the default file extension for any Excel file was (and remains) .xlsx(.xltx).

- Other excel file extensions
  - .xlsm(.xltm): excel macro-enabled workbook
  - .xlsb: excel binary workbook

Comparison of file types

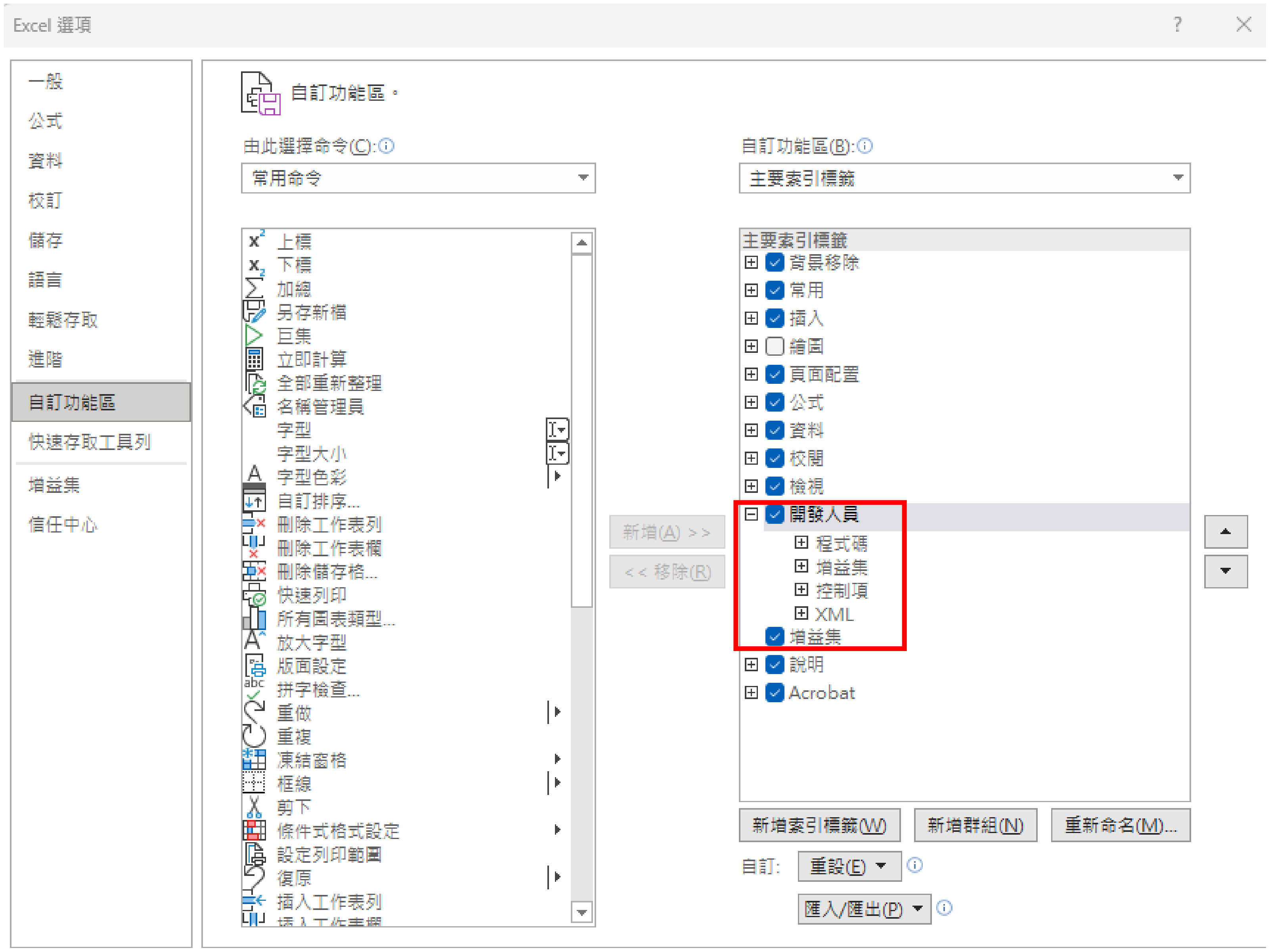
- Come to the following comparison:

Type	Formulas	Macros	Rows	Columns
XLS	Y	Y	65536	4096
XLSX	Y	N	1048576	16384
XLSM	Y	Y	1048576	16384
XLSB	Y	Y	1048576	16384
CSV	N	N	1048576	16384

		File Size	
		<10 MB	>10 MB
VBA macros	No	XLSX	XLSB
	Yes	XLSM	XLSB

國立臺北大學
National Taipei University

Setting


Excel 選項

一般
公式
資料
校訂
儲存
語言
輕鬆存取
進階
自訂功能區
快速存取工具列
增益集
信任中心

自訂功能區。

由此選擇命令(C):
常用命令

上標
下標
加總
另存新檔
巨集
立即計算
全部重新整理
名稱管理員
字型
字型大小
字型色彩
自訂排序...
刪除工作表列
刪除工作表欄
刪除儲存格...
快速列印
所有圖表類型...
放大字型
版面設定
拼字檢查...
重做
重複
凍結窗格
框線
剪下
條件式格式設定
設定列印範圍
復原
插入工作表列

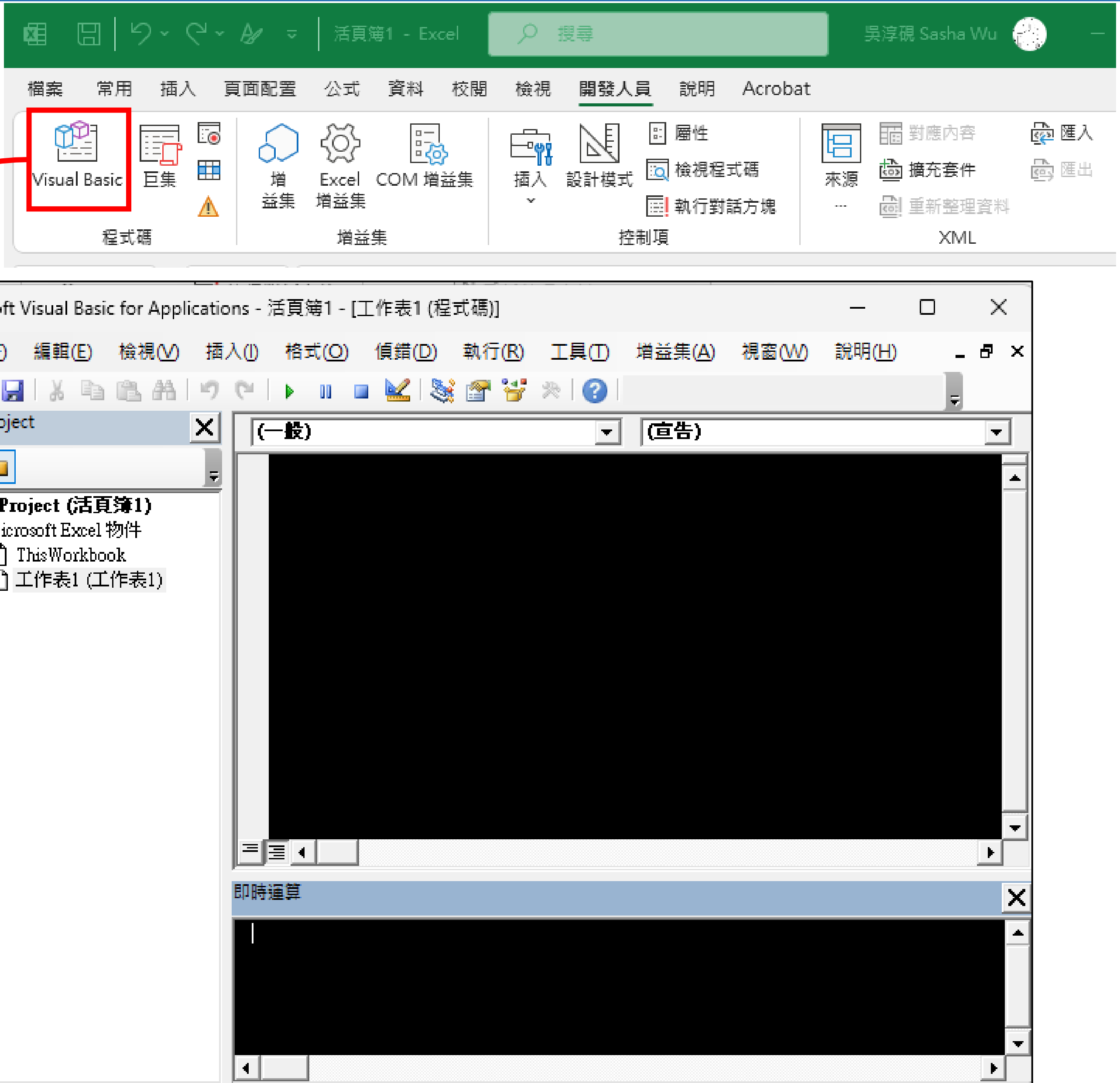
新增(A) >>
<< 移除(R)

自訂功能區(B):
主要索引標籤

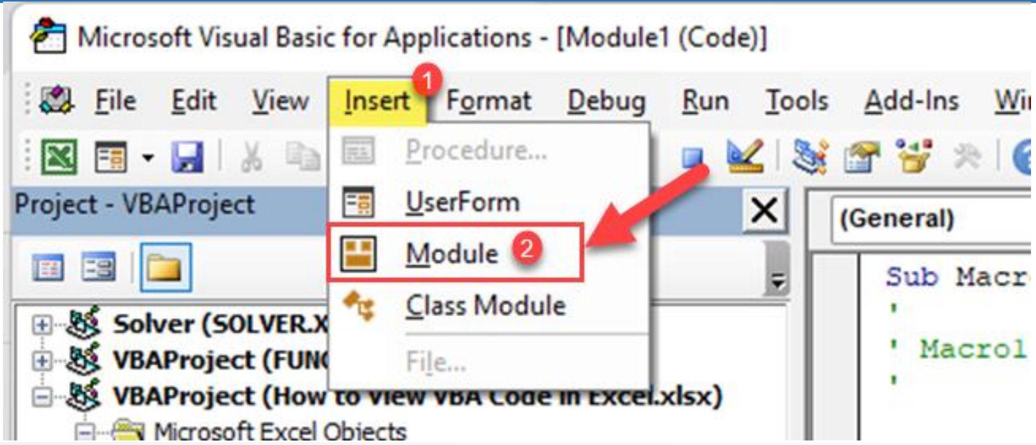
主要索引標籤
- ✓ 背景移除
- ✓ 常用
- ✓ 插入
- ☐ 繪圖
- ✓ 頁面配置
- ✓ 公式
- ✓ 資料
- ✓ 校閱
- ✓ 檢視
- ✓ 開發人員
  - 程式碼
  - 增益集
  - 控制項
  - XML
- ✓ 增益集
- ✓ 說明
- ✓ Acrobat

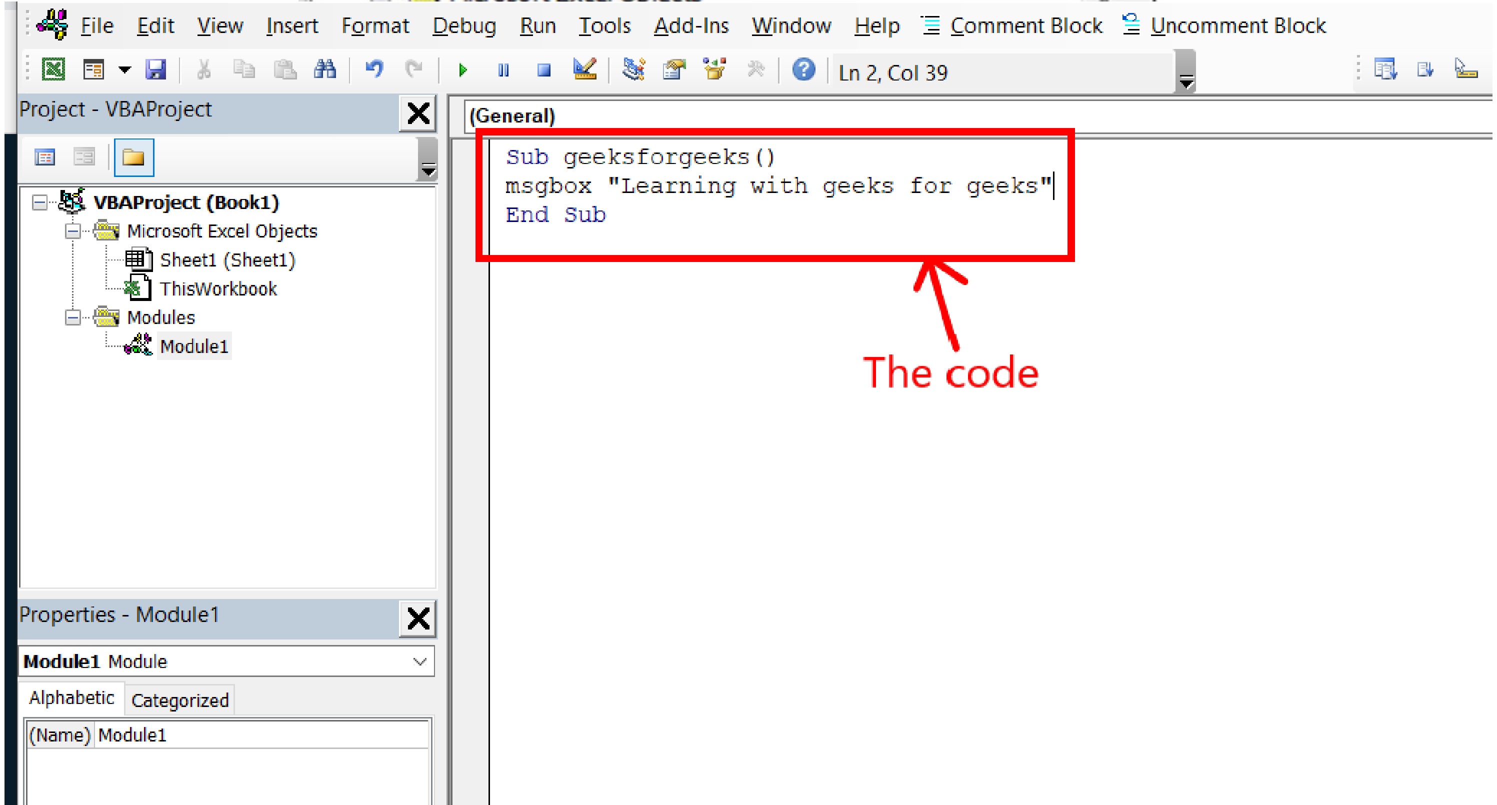
新增索引標籤(W)　新增群組(N)　重新命名(M)...
自訂: 重設(E)
匯入/匯出(P)

Visual Basic Editor



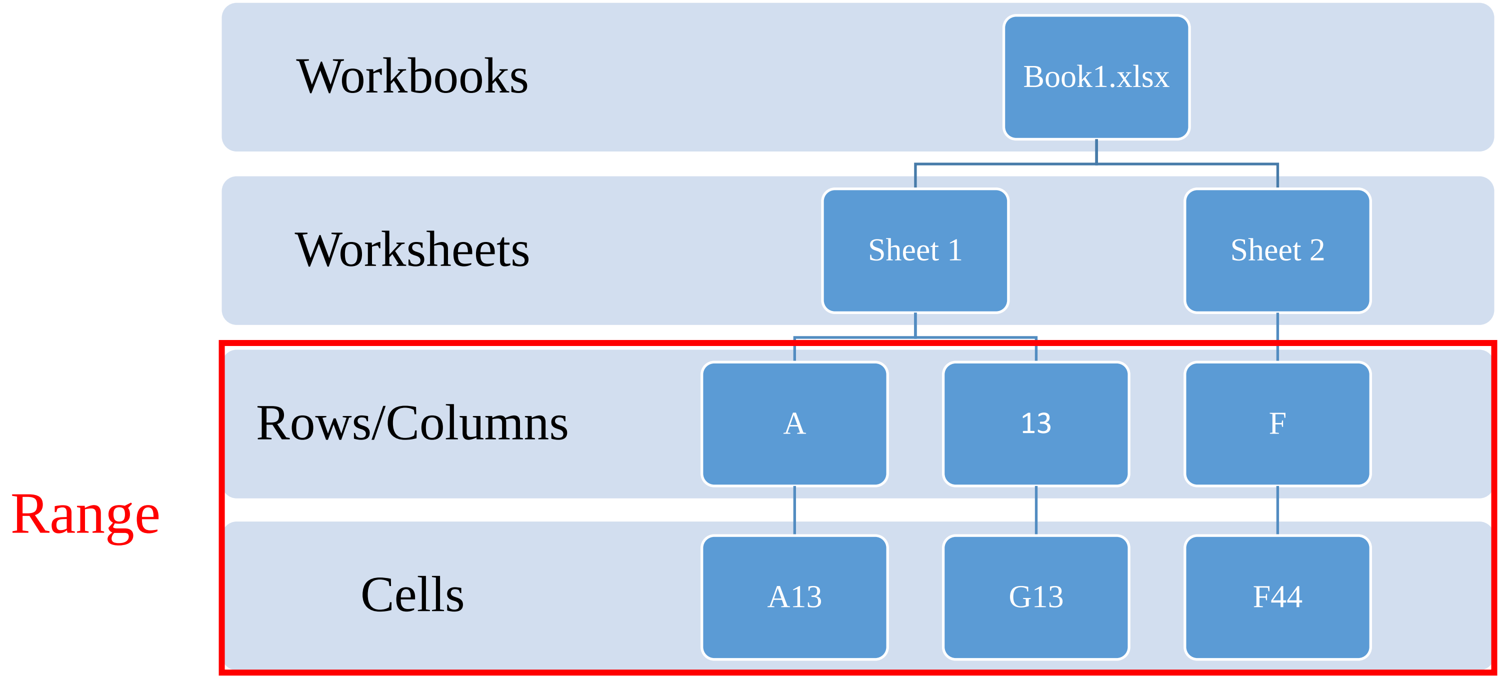
Modules

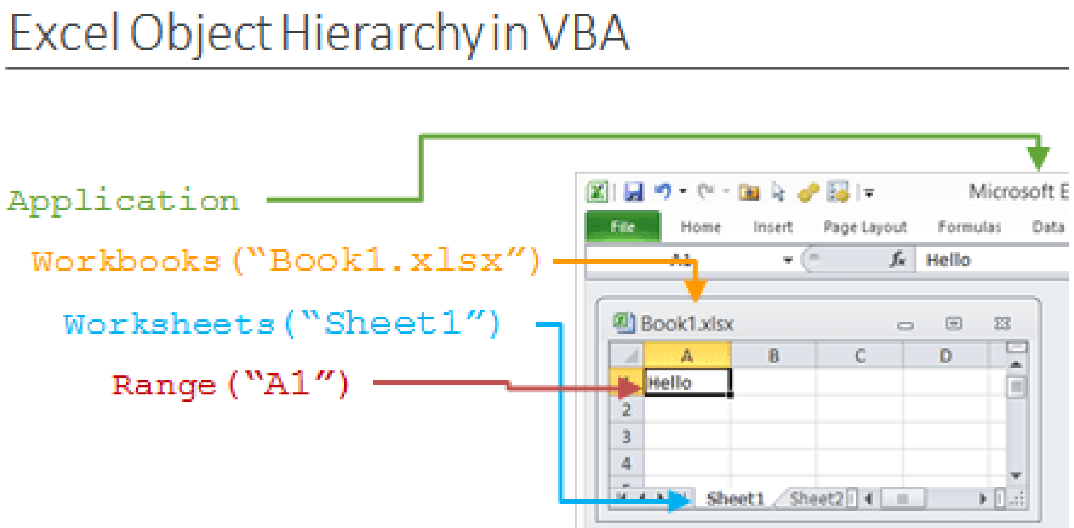

Microsoft Visual Basic for Applications - [Module1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins
1
Procedure...
UserForm
Module
2
Class Module
File...
Project - VBAProject
Solver (SOLVER.X
VBAProject (FUN
VBAProject (How to view VBA Code in Excel.xlsx)
Microsoft Excel Objects
(General)
Sub Macr
' Macro1



```
Sub geeksforgeeks()
msgbox "Learning with geeks for geeks"
End Sub
```

Excel Hierarchy





- Equivalence
  Cells(3,″A″)
  = Cells(3,1)
  = Range(″A3″)

Working with Ranges

- What is Range?
  - When we refer to a range in Excel we mean either a singular cell, a rectangular block of cells, or a union of many rectangular blocks.

- In VBA Range is an object with its own properties and methods.
  - Non-contiguous range: Range(″A1:B3,E1:O9″)
  - Range(″A1:E5″) = Range(Cells(1,1),Cells(5,5))
  - Range(″F9″) = Range(″D10:G20″).Cells(0,3)

- Fixing Reference

Fix	Column	Row
A27	N	N
\$A\$27	Y	Y
SA27	Y	N
A\$27	N	Y

Some Excel Function

Let [number] denote optional choice

- SUM(number,[number])
  - Add the values in cells.
- MIN (number,[number]), MAX (number,[number])
  - Return the smallest (largest) value in a set of values.
- COUNT(value,[value])
  - Counts how many numbers are in the list of arguments.
- AVERAGE(number,[number])
  - Return the average of its the arguments.
- UNIQUE(array,[by col],[exactly once])
  - Returns a list of unique values in a list or range.
- RANK(number, ref,[order])
  - Returns the rank of a number in a list of numbers.

Logical functions

- IF(logical test, value if true,[value if false])
  - Specifies a logical test to perform
- IFNA(value,[value if na])
  - Returns the value you specify if the expression resolves to #N/A. Otherwise returns the result of the expression
- IFERROR(value,[value if error])
  - Returns a value you specify if a formula evaluates to an error. Otherwise, it returns the result of the formula.
- AND(logical,[logical])
  - Return TRUE if all if its arguments are TRUE.
- OR(logical,[logical])
  - Return TRUE if any arguments is TRUE.
- NOT(logical)
  - Reverses the logic of its argument.

國立臺北大學
National Taipei University

Declare Variable - 1

- Syntax of Variable declaration

  Dim ⟨variable name⟩ As ⟨variable type⟩

  - Numerical data type:

    Byte, Integer, Long, Single, Double, Decimal, Currency.

  - Non-numerical data type:

    String, Date, Boolean, Object, Variant.

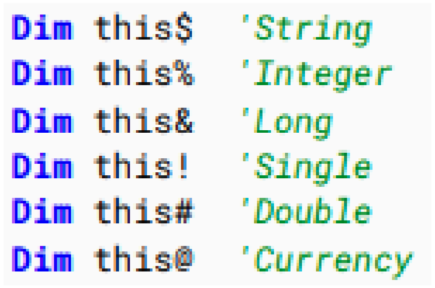
- Variable Names
  - You can't start a variable name with a number.
  - You can't have spaces in your variable names, or full stops (periods).
  - You can't use any of the following characters: !, %, ?, #, $.

MyVariable
My_Variable
myvariable2
●

2MyVariable
My Variable
$myvariable
✖

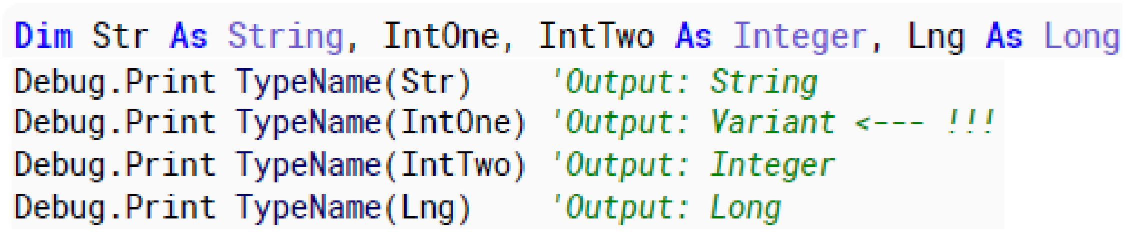
Declare Variables - 2

- Variables can also be declared using Data Type Character suffixes.



```
Dim this$   'String
Dim this%   'Integer
Dim this&   'Long
Dim this!   'Single
Dim this#   'Double
Dim this@   'Currency
```

- Multiple variables can be declared on a single line using commas as delimiters, but **each type must be declared individually**, or they will default to the Variant type.

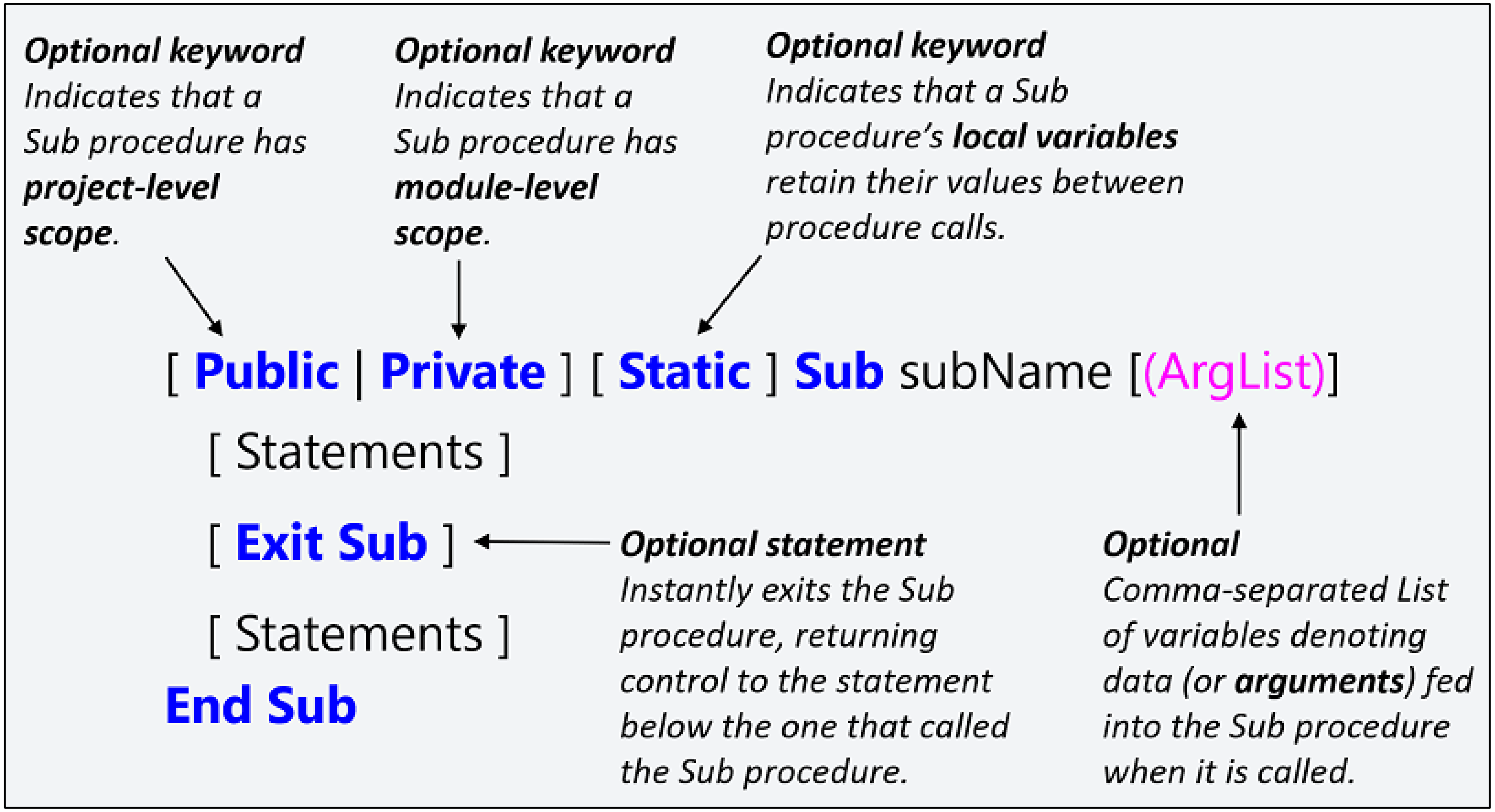


```
Dim Str As String, IntOne, IntTwo As Integer, Lng As Long
Debug.Print TypeName(Str)    'Output: String
Debug.Print TypeName(IntOne) 'Output: Variant <--- !!!
Debug.Print TypeName(IntTwo) 'Output: Integer
Debug.Print TypeName(Lng)    'Output: Long
```

- Const is a named memory location used to hold a value that can not be changed during the script execution.
- Use Option Explicit statement on 1st line of a module to force all variables to be declared before usage.

Sub Procedures

- A Sub can be described as a small program within the VBA Editor that performs a specific action in Excel.



Optional keyword
Indicates that a Sub procedure has ***project-level scope.***

Optional keyword
Indicates that a Sub procedure has ***module-level scope.***

Optional keyword
Indicates that a Sub procedure's ***local variables*** *retain their values between procedure calls.*

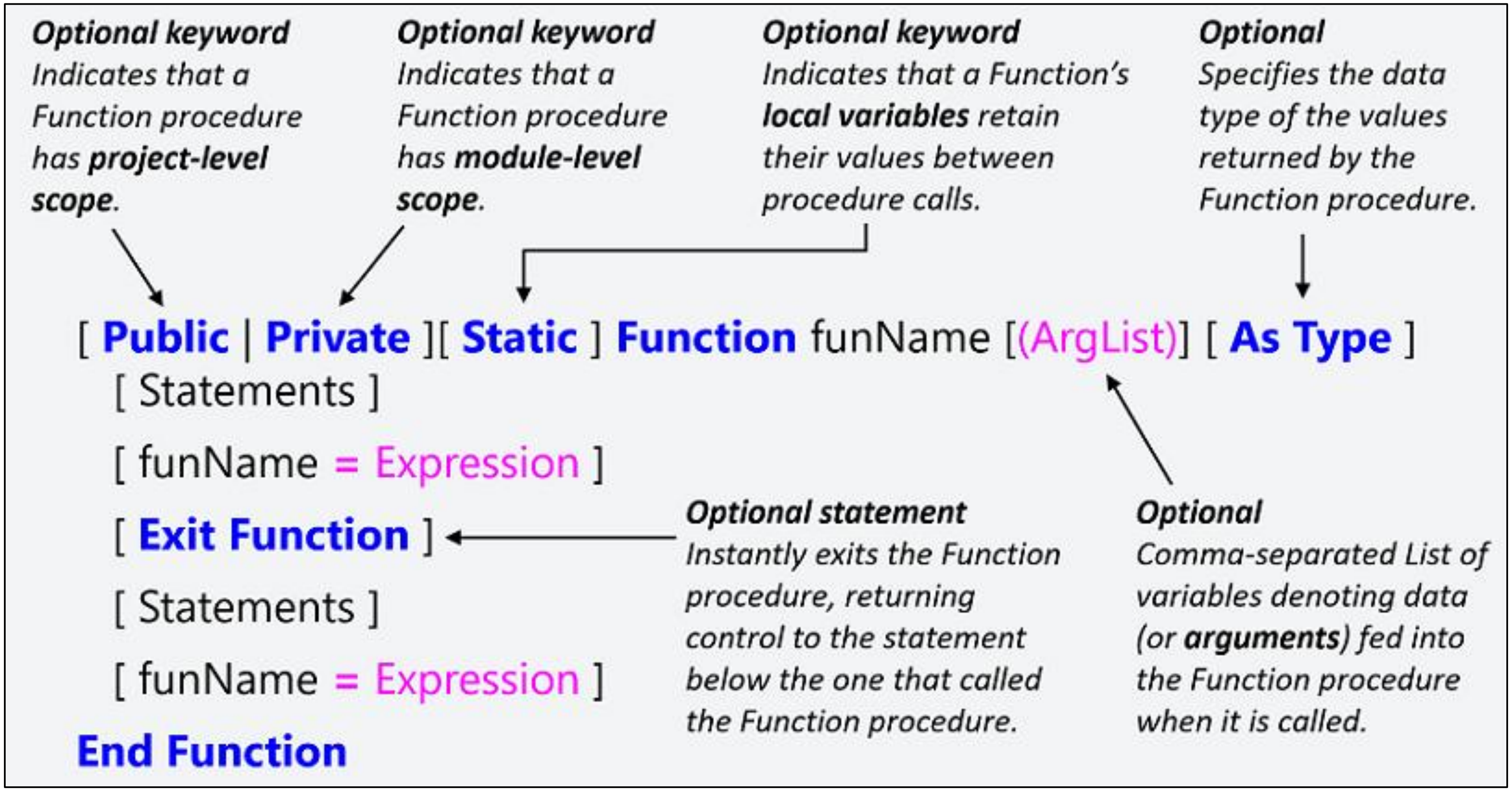
```
[ Public | Private ] [ Static ] Sub subName [(ArgList)]
    [ Statements ]
    [ Exit Sub ]
    [ Statements ]
End Sub
```

Optional statement
Instantly exits the Sub procedure, returning control to the statement below the one that called the Sub procedure.

Optional
Comma-separated List of variables denoting data (or ***arguments****) fed into the Sub procedure when it is called.*

Function

- A Function is similar to a Sub procedure, only that the former can return a value whereas the latter cannot.



Optional keyword *Indicates that a Function procedure has* ***project-level scope****.*

Optional keyword *Indicates that a Function procedure has* ***module-level scope****.*

Optional keyword *Indicates that a Function's* ***local variables*** *retain their values between procedure calls.*

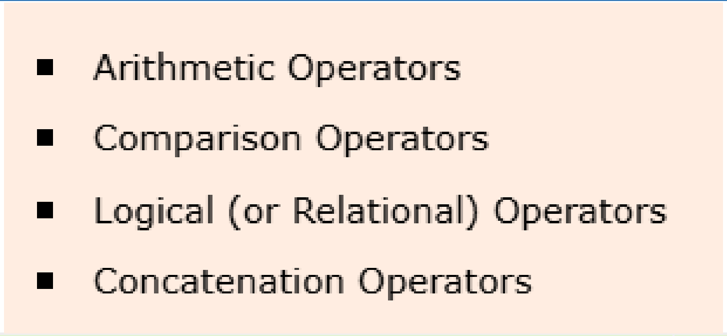
Optional *Specifies the data type of the values returned by the Function procedure.*

```
[ Public | Private ][ Static ] Function funName [(ArgList)] [ As Type ]
  [ Statements ]
  [ funName = Expression ]
  [ Exit Function ]
  [ Statements ]
  [ funName = Expression ]
End Function
```

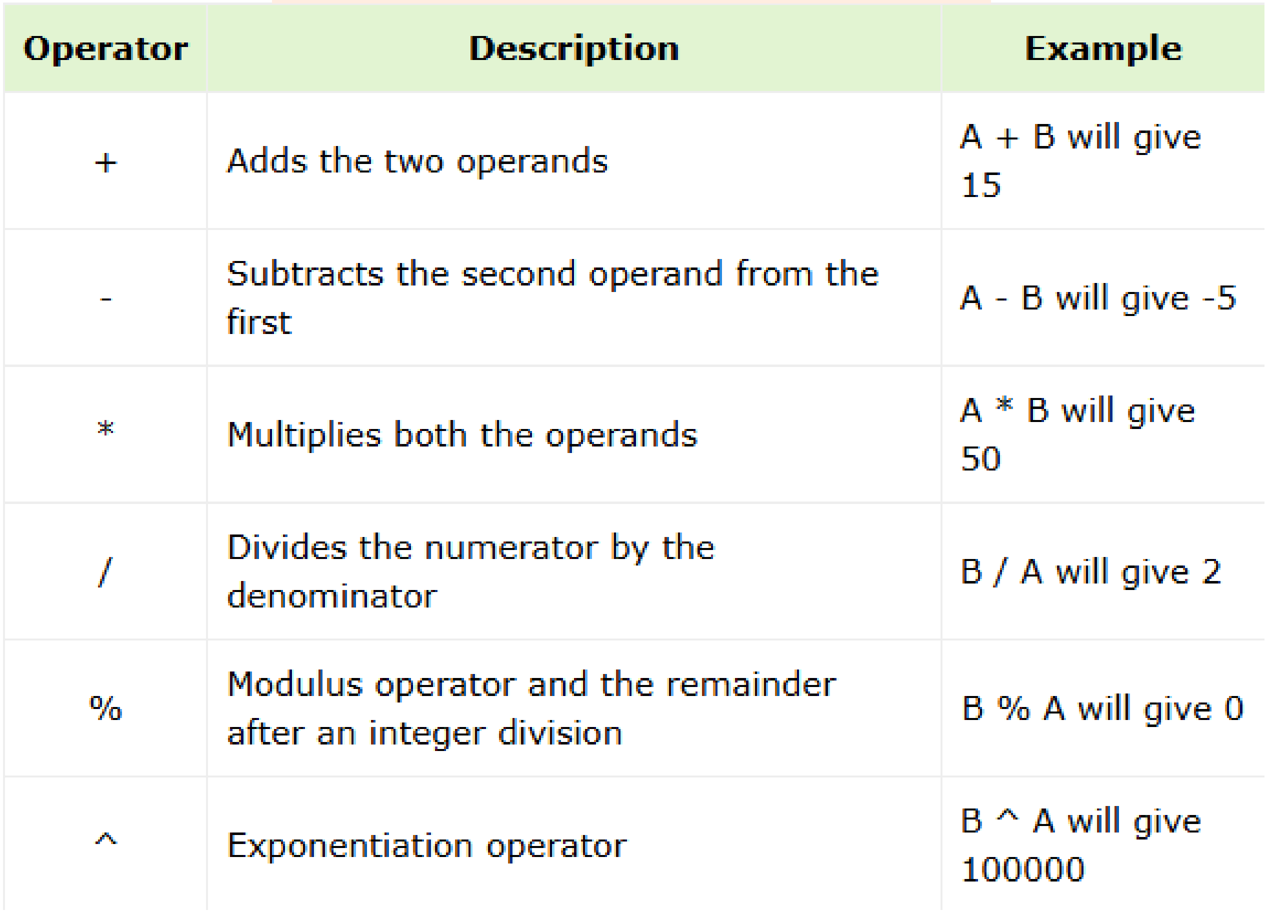
Optional statement *Instantly exits the Function procedure, returning control to the statement below the one that called the Function procedure.*

Optional *Comma-separated List of variables denoting data (or* ***arguments****) fed into the Function procedure when it is called.*

Arithmetic operators

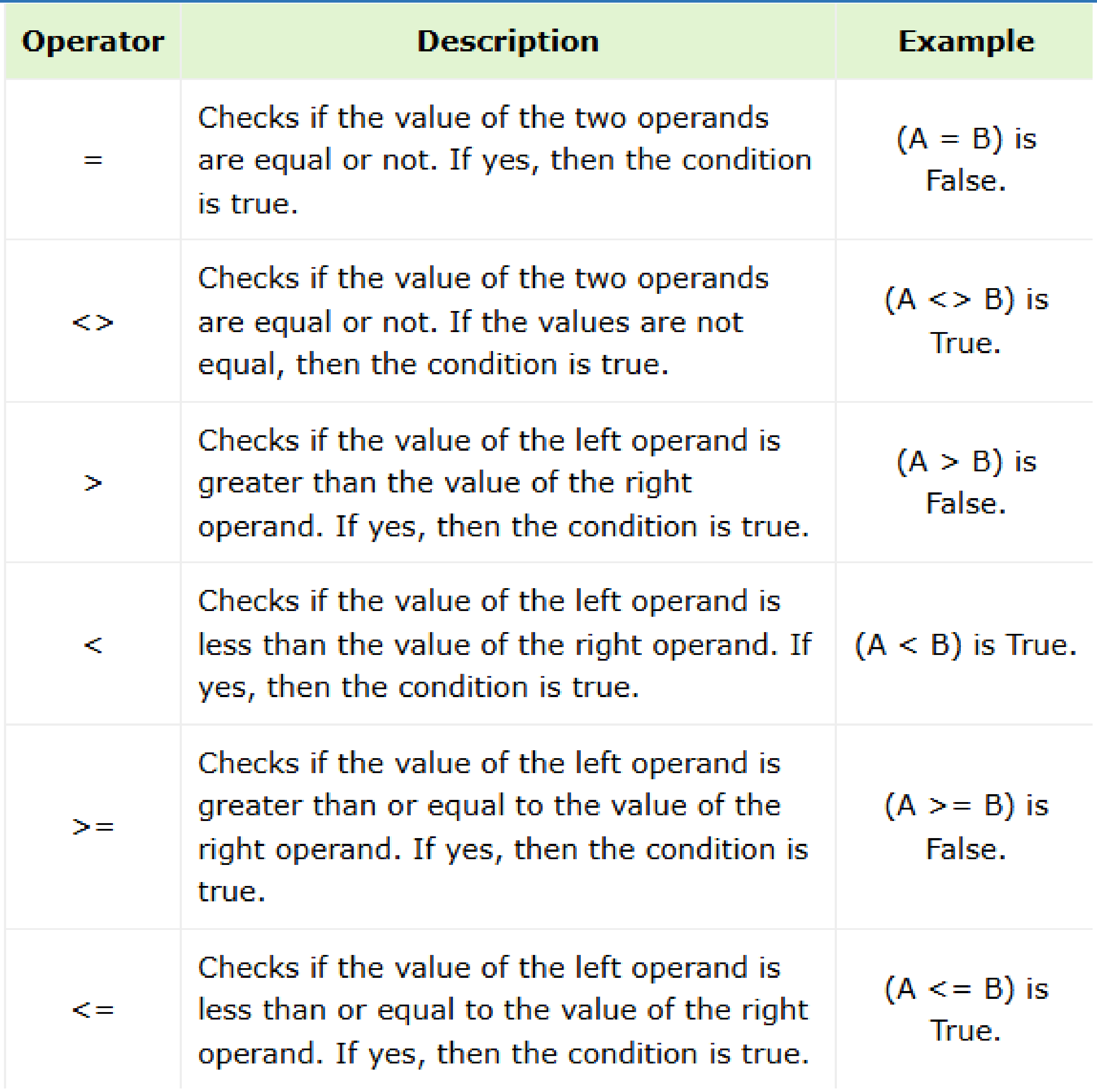


- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Concatenation Operators



Operator	Description	Example
+	Adds the two operands	A + B will give 15
-	Subtracts the second operand from the first	A - B will give -5
*	Multiplies both the operands	A * B will give 50
/	Divides the numerator by the denominator	B / A will give 2
%	Modulus operator and the remainder after an integer division	B % A will give 0
^	Exponentiation operator	B ^ A will give 100000

Comparison operator



Operator	Description	Example
=	Checks if the value of the two operands are equal or not. If yes, then the condition is true.	(A = B) is False.
<>	Checks if the value of the two operands are equal or not. If the values are not equal, then the condition is true.	(A <> B) is True.
>	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition is true.	(A > B) is False.
<	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition is true.	(A < B) is True.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition is true.	(A >= B) is False.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition is true.	(A <= B) is True.

國立臺北大學 National Taipei University

Logical and Concatenation operator

Operator	Description	Example
AND	Called Logical AND operator. If both the conditions are True, then the Expression is true.	a<>0 AND b<>0 is False.
OR	Called Logical OR Operator. If any of the two conditions are True, then the condition is true.	a<>0 OR b<>0 is true.
NOT	Called Logical NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	NOT(a<>0 OR b<>0) is false.
XOR	Called Logical Exclusion. It is the combination of NOT and OR Operator. If one, and only one, of the expressions evaluates to be True, the result is True.	(a<>0 XOR b<>0) is true.

Assume A holds 5 and B holds 10

Assume A= ″Microsoft″ and B = ″VBScript″

Operator	Description	Example	Example
+	Adds two Values as Variable. Values are Numeric	A + B will give 15	A + B will give MicrosoftVBScript
&	Concatenates two Values	A & B will give 510	A & B will give MicrosoftVBScript

國立臺北大學
National Taipei University

String - 1

- Following are String methods that are supported in VBA.
  - InStr([start],string1,string2,[compare])
    InstrRev(string1,string2,[start],[compare])

    Returns the first occurrence of one string within another string.
    The Search happens from the right to the left|right to the left.
  - Lcase(string)
    Ucase(string)

    Converting the entered string into lower|upper case letters.
  - Left(string, length)
    Right(string, length)
    Mid(string, start,[length])

    Returns a specified number of characters

    from the {
    left side of the string
    right side of the string
    a string based on specified parameters

String - 2

- Ltrim(string)
  Rtrim(string)
  Trim(string)

  Returns a string after

  $$\text{removing}\begin{cases}\text{the spaces on the left side of the specified string}\\ \text{the spaces on the right side of the specified string}\\ \text{both the leading and the trailing blank spaces}\end{cases}$$

- Replace(string, find, replace with,[start],[count],[compare])

  Replaces a specified part of a string with a specific string, a specified number of times.

- Space(number)

  Fills a string with a specific number of spaces.

- StrReverse(string)

  Reverses the specified string

Array - 1

- Syntax

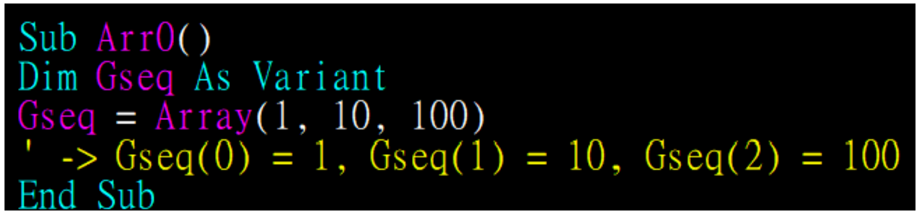
  Dim MyArray(n) As ⟨variable type⟩

  where n means array with index 0 to n, if the 1st position is not consider to be 0 then

  Dim MyArray(k to n + k) As ⟨variable type⟩

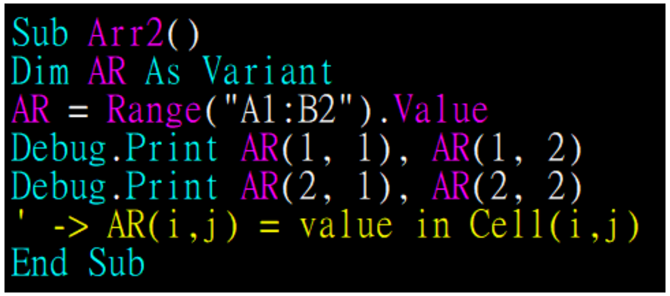
  Or use Option Base 1 at the top of the module to make default lower bound equal to 1

- Using Array() function



```
Sub Arr0()
Dim Gseq As Variant
Gseq = Array(1, 10, 100)
' -> Gseq(0) = 1, Gseq(1) = 10, Gseq(2) = 100
End Sub
```

  Or from Range

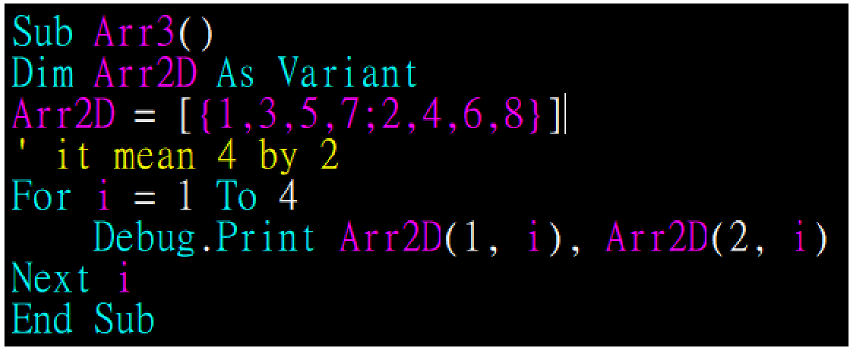


```
Sub Arr2()
Dim AR As Variant
AR = Range("A1:B2").Value
Debug.Print AR(1, 1), AR(1, 2)
Debug.Print AR(2, 1), AR(2, 2)
' -> AR(i,j) = value in Cell(i,j)
End Sub
```

National Taipei University

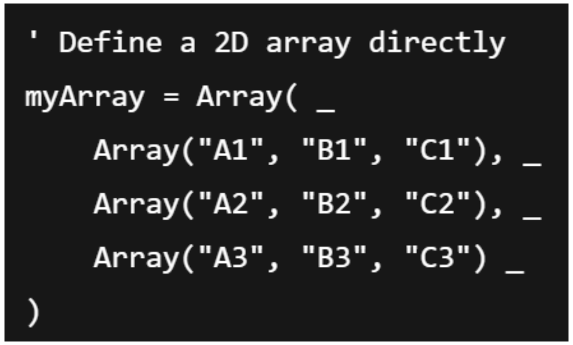
Array - 2

- Syntax:
  Ex. 3 by 4 array
  Dim Arr2D(1 to 3,1 to 4) as Variant
- 2D array (Only valid for all constants)



```
Sub Arr3()
Dim Arr2D As Variant
Arr2D = [{1,3,5,7;2,4,6,8}]
' it mean 4 by 2
For i = 1 To 4
    Debug.Print Arr2D(1, i), Arr2D(2, i)
Next i
End Sub
```

- Jagged array : Array of Arrays.



```
' Define a 2D array directly
myArray = Array( _
    Array("A1", "B1", "C1"), _
    Array("A2", "B2", "C2"), _
    Array("A3", "B3", "C3") _
)
```

國立臺北大學 National Taipei University

Array - 3

- You can not assign jagged array without Array() or Transpose.

```
' Assign to range using double transpose to reshape jagged array
Sheet1.Range("A1:C3").Value = _
    Application.WorksheetFunction.Transpose( _
    Application.WorksheetFunction.Transpose(myArray))
```

- Array functions

Function	Purpose
`LBound(arr)`	Lowest index of array
`UBound(arr)`	Highest index of array
`IsArray(var)`	Returns True if variable is an array
`Join(arr, ",")`	Combines array elements into string
`Split(str, ",")`	Converts string into array
`Filter()`	Filters array elements (Variant only)

Cells - 1

- Syntax

  $$\text{Cells(row, column)}$$

- Ways to refer to a single cell
  - Enclose the A1 form of its reference in square brackets

    $$\text{[A1] = "Hello"}$$

  - Evaluate method

    $$\text{Application. Evaluate("A2") = "Hello“}$$

  - Call Cells method

    $$\text{Cells(3,1). Formula = " = A1 \& A2 “}$$

  - In previous case, we did not specify a worksheet, so Excel will use the active sheet. We can provide the name of a particular sheet.

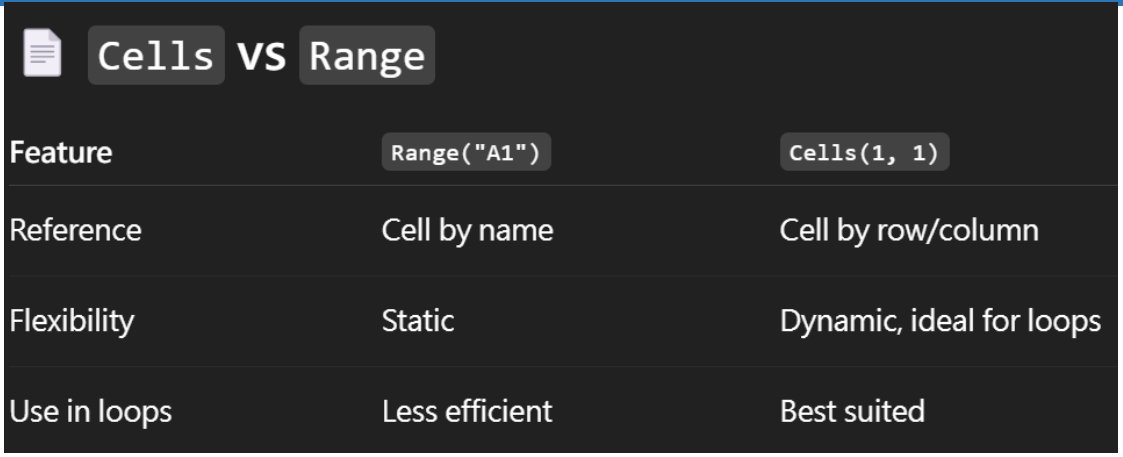
    $$\text{Sheets("name"). Cells(3,1). Formula = " = A1 \& A2“}$$

  - Rows method + Cells method
    Columns method + Cells method

    $$\text{Rows(4). Cells(1). Value = [A3] \& "!"}$$
    $$\text{Columns(1). Cells(5). Value = "Hey"}$$

國立臺北大學 National Taipei University

Cells - 2



Cells vs Range

Feature	`Range("A1")`	`Cells(1, 1)`
Reference	Cell by name	Cell by row/column
Flexibility	Static	Dynamic, ideal for loops
Use in loops	Less efficient	Best suited

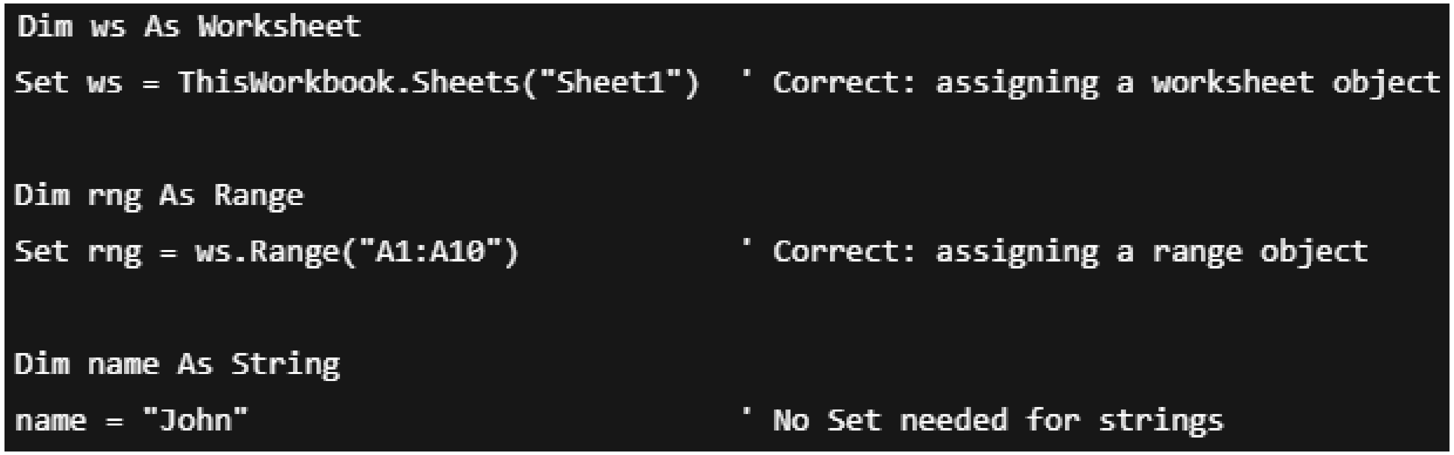
- Formatting a Cell
  - Interior (Fill color)
  - Borders
  - Number Formats
  - Alignment
  - Row Height & Column Width
  - Cleaning Cells
  - Validation and Comments
  - Merging & Unmerging
  - Selection / Activating / Copy-Paste

國立臺北大學
National Taipei University

Set Keyword

- Why is it needed?
  Because objects (like worksheets, ranges, or workbooks) are more complex than basic data types, VBA needs to differentiate between:
  - assigning a value → no Set needed
  - assigning an object reference → Set is required

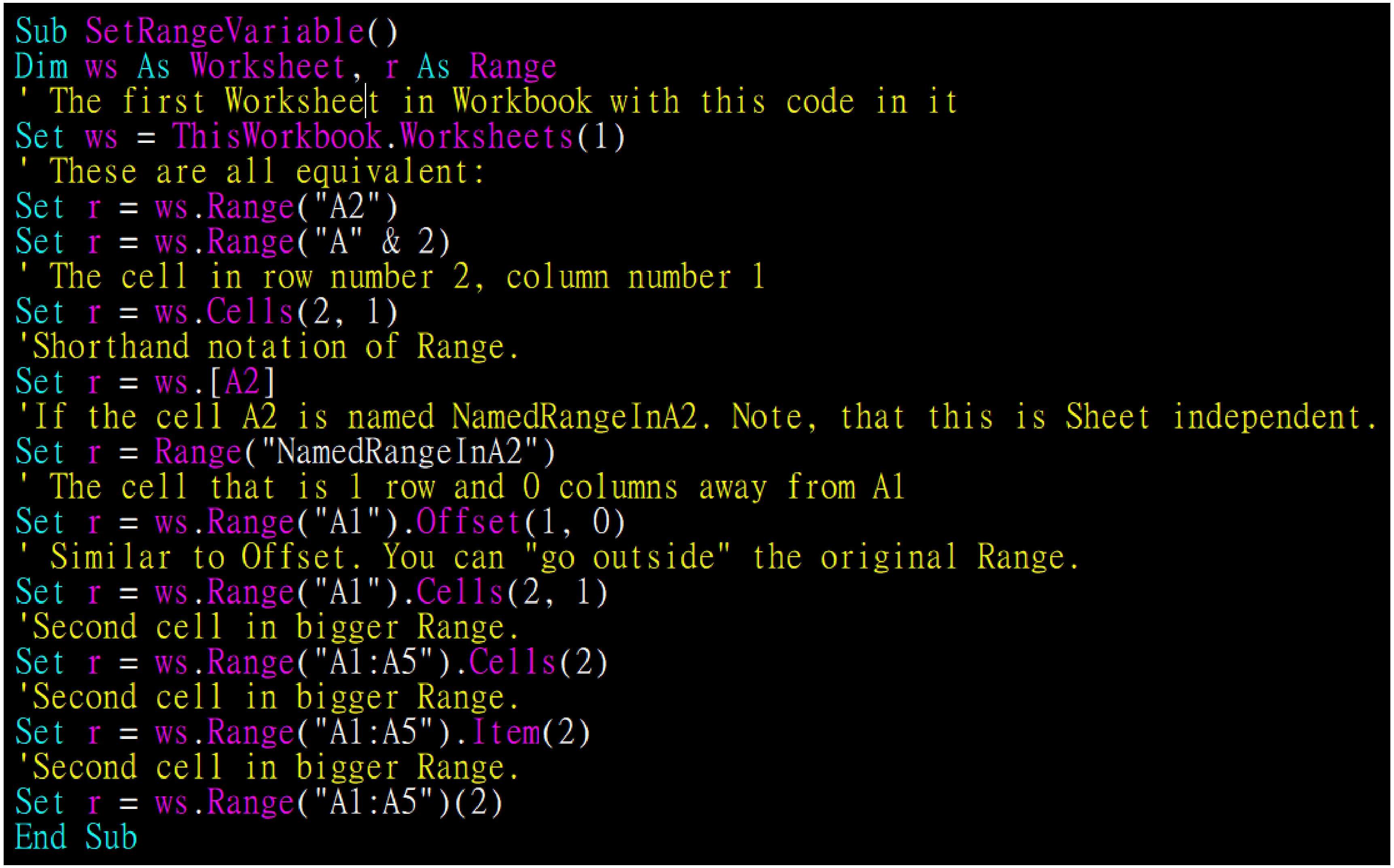
- Syntax
  Set objectvar = {[New]objectexpression|Nothing}



```
Dim ws As Worksheet
Set ws = ThisWorkbook.Sheets("Sheet1")  ' Correct: assigning a worksheet object

Dim rng As Range
Set rng = ws.Range("A1:A10")            ' Correct: assigning a range object

Dim name As String
name = "John"                           ' No Set needed for strings
```

Range

- There are different ways to create the same Range

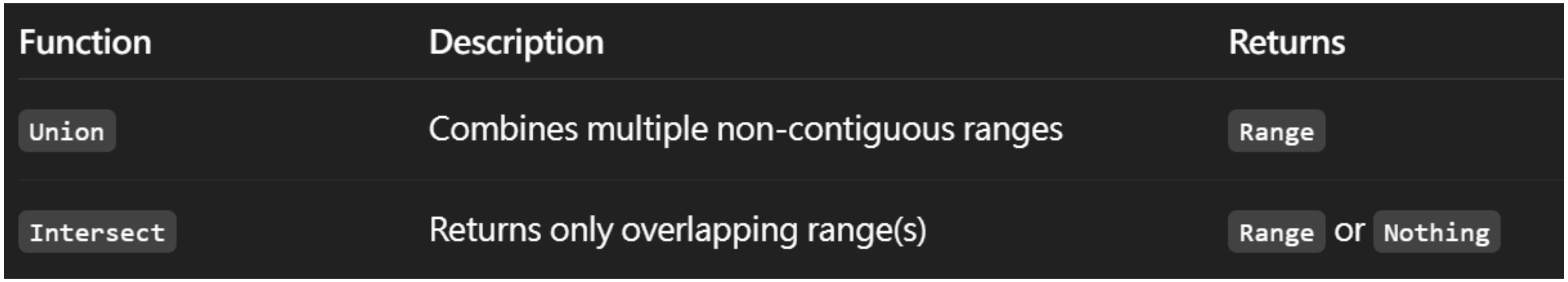


```
Sub SetRangeVariable()
Dim ws As Worksheet, r As Range
' The first Worksheet in Workbook with this code in it
Set ws = ThisWorkbook.Worksheets(1)
' These are all equivalent:
Set r = ws.Range("A2")
Set r = ws.Range("A" & 2)
' The cell in row number 2, column number 1
Set r = ws.Cells(2, 1)
'Shorthand notation of Range.
Set r = ws.[A2]
'If the cell A2 is named NamedRangeInA2. Note, that this is Sheet independent.
Set r = Range("NamedRangeInA2")
' The cell that is 1 row and 0 columns away from A1
Set r = ws.Range("A1").Offset(1, 0)
' Similar to Offset. You can "go outside" the original Range.
Set r = ws.Range("A1").Cells(2, 1)
'Second cell in bigger Range.
Set r = ws.Range("A1:A5").Cells(2)
'Second cell in bigger Range.
Set r = ws.Range("A1:A5").Item(2)
'Second cell in bigger Range.
Set r = ws.Range("A1:A5")(2)
End Sub
```

Combine Multiple Ranges

- The Union function lets you combine two or more ranges into one Range object even if they're non-contiguous
  - Set result = Union(Range1, Range2, ...)

- The Intersect function returns only the overlapping part of two (or more) ranges.
  - Set result = Intersect(Range1, Range2, ...)



Function	Description	Returns
`Union`	Combines multiple non-contiguous ranges	`Range`
`Intersect`	Returns only overlapping range(s)	`Range` or `Nothing`

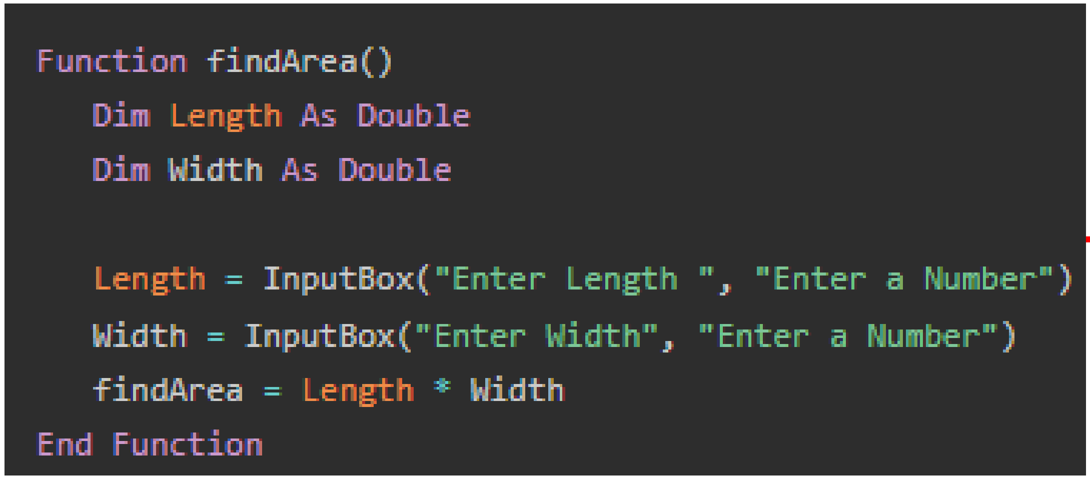
- Union of rows or columns

Set uR = Union(Rows(2), Rows(4))
Set uC = Union(Columns(2), Columns(4))

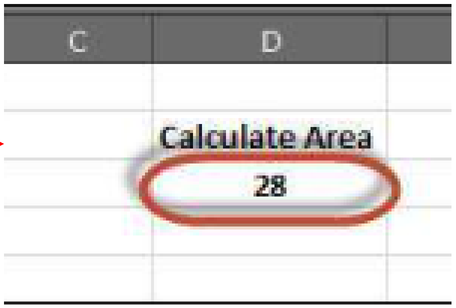
InputBox

- The InputBox function prompts the users to enter values. After entering the values, if the user clicks the OK button or presses ENTER on the keyboard, the InputBox function will return the text in the text box. If the user clicks the Cancel button, the function will return an empty string ("").

- Syntax
  InputBox(prompt, [title], [default], [xpos], [ypos], [helpfile], [context])



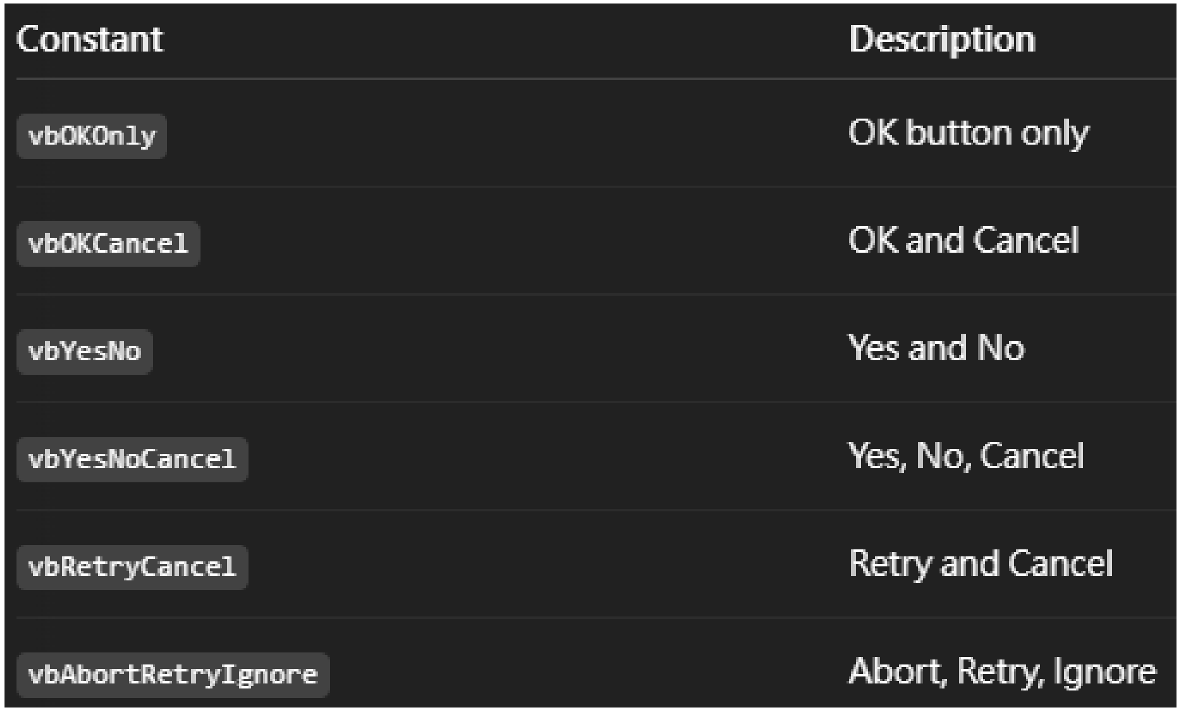
```
Function findArea()
   Dim Length As Double
   Dim Width As Double

   Length = InputBox("Enter Length ", "Enter a Number")
   Width = InputBox("Enter Width", "Enter a Number")
   findArea = Length * Width
End Function
```



Message Box - 1

- Displays a message box and waits for the user to click a button and then an action is performed based on the button clicked by the user.
- Syntax

  MsgBox(prompt, [button], [title], [helpfile], [context])
- Buttons Types

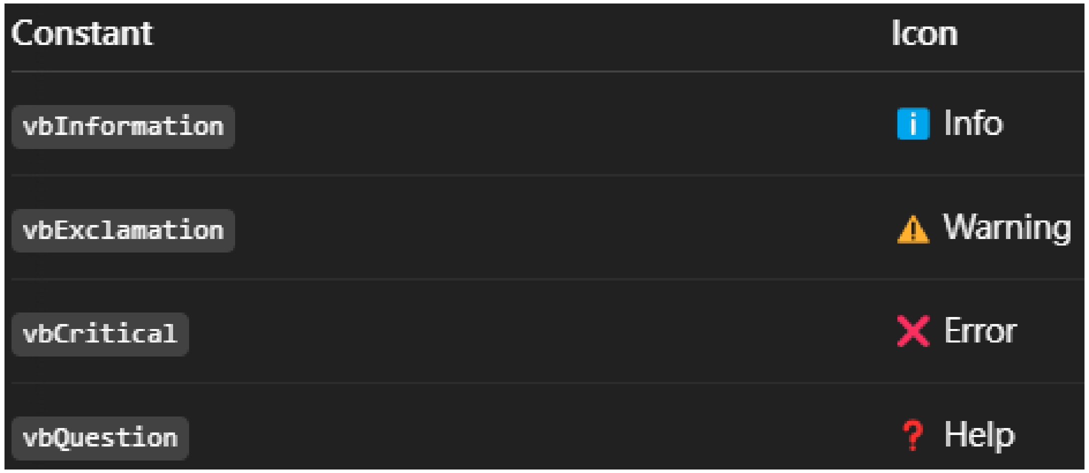


Constant	Description
`vbOKOnly`	OK button only
`vbOKCancel`	OK and Cancel
`vbYesNo`	Yes and No
`vbYesNoCancel`	Yes, No, Cancel
`vbRetryCancel`	Retry and Cancel
`vbAbortRetryIgnore`	Abort, Retry, Ignore

國立臺北大學 National Taipei University

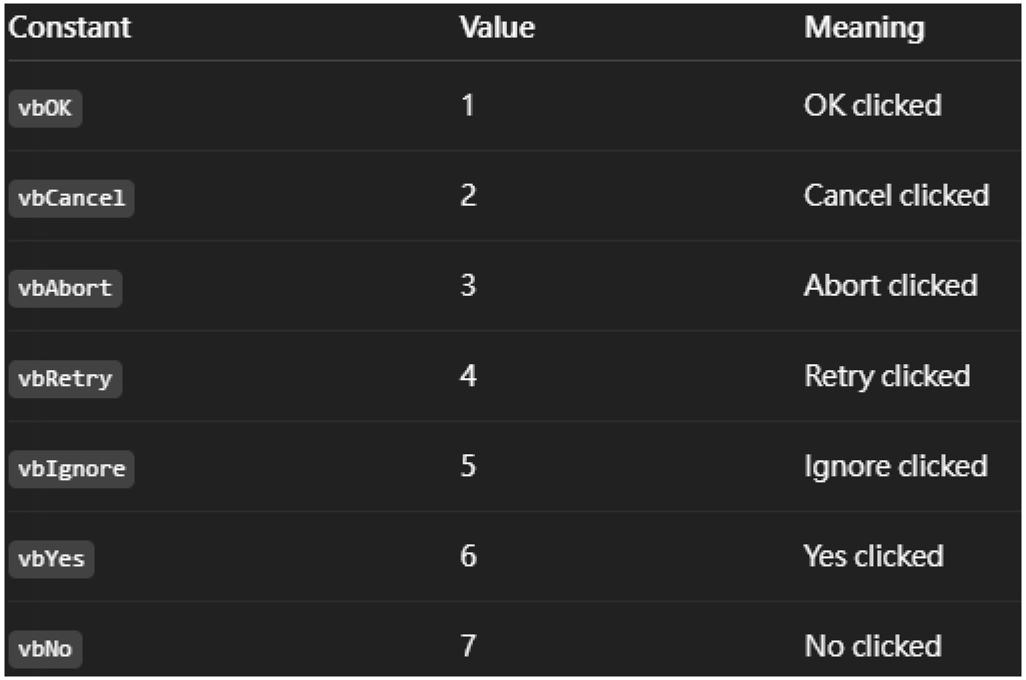
Message Box - 2

- Icon Types



Constant	Icon
`vbInformation`	ℹ Info
`vbExclamation`	⚠ Warning
`vbCritical`	✕ Error
`vbQuestion`	? Help

- Return Constants



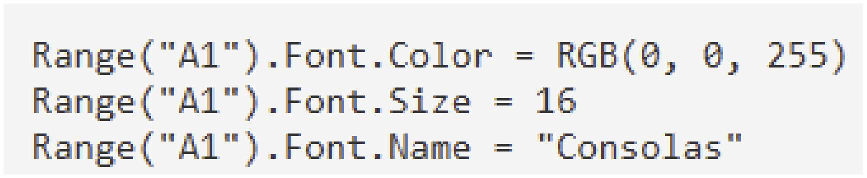
Constant	Value	Meaning
`vbOK`	1	OK clicked
`vbCancel`	2	Cancel clicked
`vbAbort`	3	Abort clicked
`vbRetry`	4	Retry clicked
`vbIgnore`	5	Ignore clicked
`vbYes`	6	Yes clicked
`vbNo`	7	No clicked

With statement

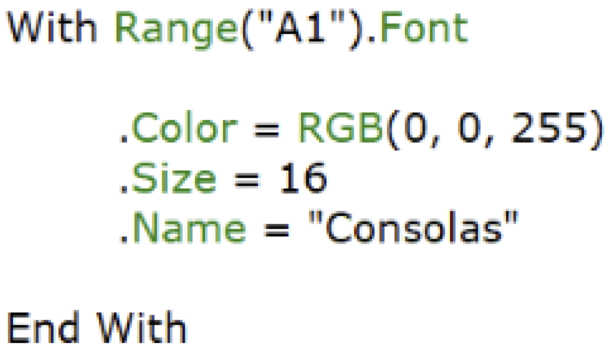
- Executes a series of statements on a single object.
- Syntax

```
With object
        $.\text{property}_1$
            ⋮
        $.\text{property}_k$
End With
```

- Equivalence



```
Range("A1").Font.Color = RGB(0, 0, 255)
Range("A1").Font.Size = 16
Range("A1").Font.Name = "Consolas"
```

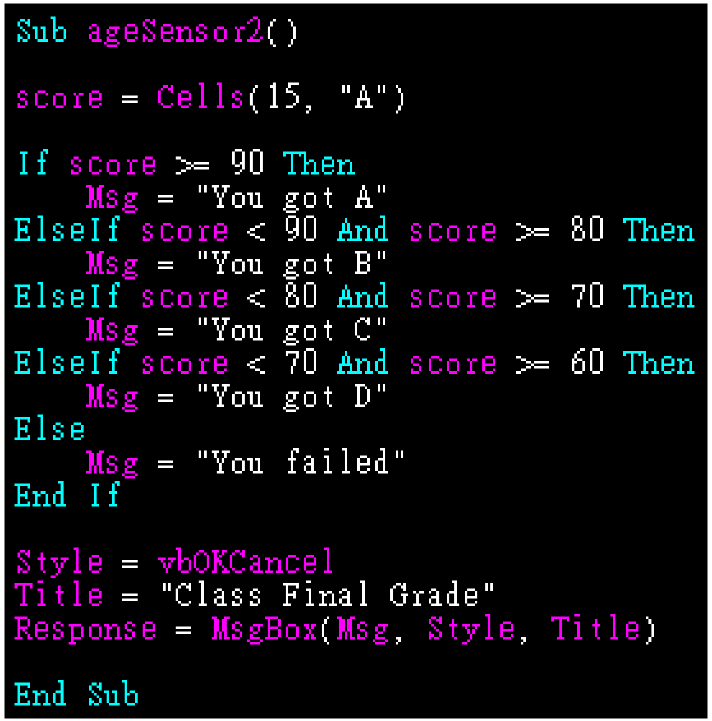


```
With Range("A1").Font

    .Color = RGB(0, 0, 255)
    .Size = 16
    .Name = "Consolas"

End With
```

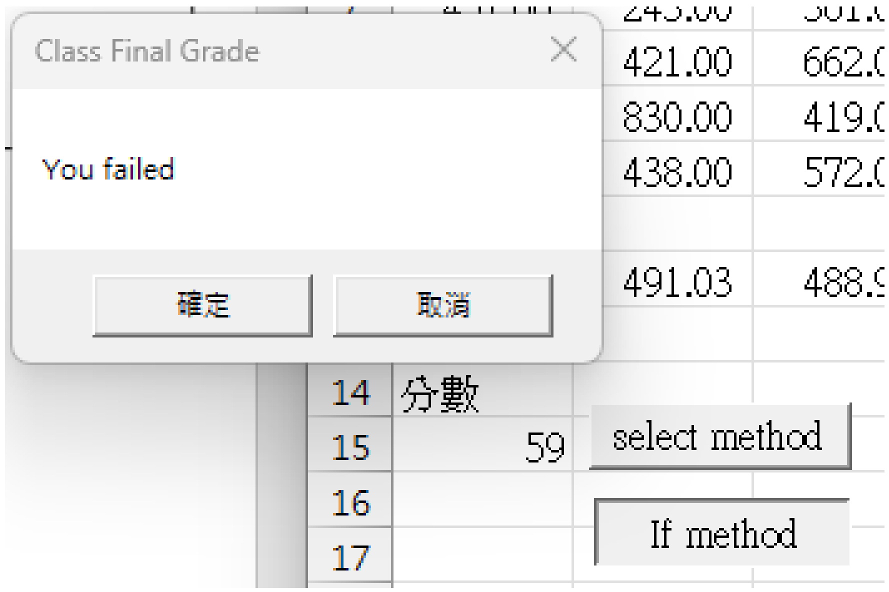
If Statement

- If, Elseif, …, Else block

```
If [Some condition is True] Then
   [Do some thing(s)]
ElseIf [Some other condition is True] Then
   [Do some different thing(s)]
Else    'Everything above has evaluated to False
   [Do some other thing(s)]
End If
```



```
Sub ageSensor2()

score = Cells(15, "A")

If score >= 90 Then
    Msg = "You got A"
ElseIf score < 90 And score >= 80 Then
    Msg = "You got B"
ElseIf score < 80 And score >= 70 Then
    Msg = "You got C"
ElseIf score < 70 And score >= 60 Then
    Msg = "You got D"
Else
    Msg = "You failed"
End If

Style = vbOKCancel
Title = "Class Final Grade"
Response = MsgBox(Msg, Style, Title)

End Sub
```



Class Final Grade

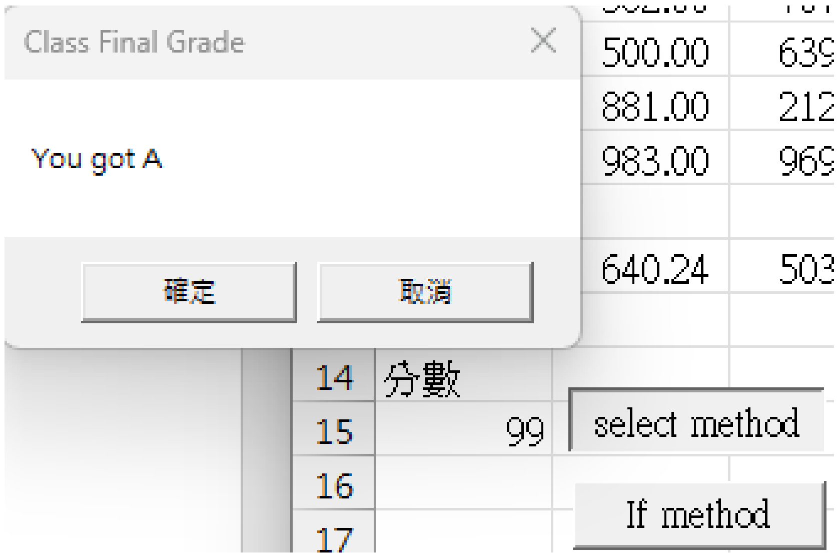
You failed

Select Case

- Syntax

```
Select Case expression
    Case val1:    'Code for val1
    Case val2:    'Code for val2
    Case val3, val4: 'Code for val3 or val4
    Case Else:    'Default code
End Select
```

```
Sub ageSensor1()

Select Case Cells(15, "A")
    Case Is >= 90
        Msg = "You got A"
    Case 80 To 89
        Msg = "You got B"
    Case 70 To 79
        Msg = "You got C"
    Case 60 To 69
        Msg = "You got D"
    Case Is < 60
        Msg = "You failed"
End Select

Style = vbOKCancel
Title = "Class Final Grade"
Response = MsgBox(Msg, Style, Title)


End Sub
```


Class Final Grade

You got A

For loop vs For each Group

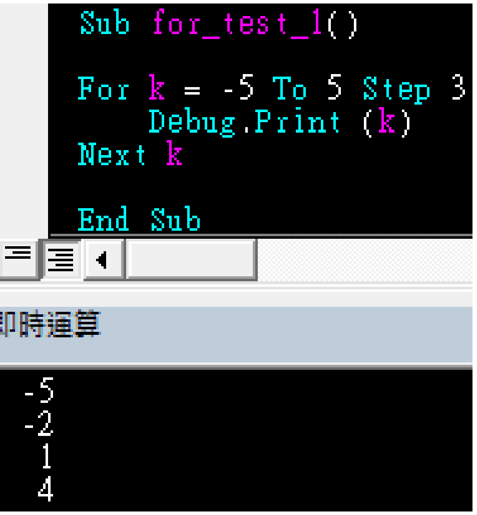
```
For Counter = Start To End [ Step StepIncrement ]
    '...Code here...
Next [ Counter ]
```

Items	Description
Counter	Is a numeric variable, the counter index for the loop. This can be only of VBA Native data types (e.g. Long, Integer, Double etc.)
Start and End	The starting value of the **Counter** and the ending value of the **Counter**
Step	Statement indicating that the **StepIncrement** between increments will be defined
StepIncrement	Optional, defaults to 1. A defined step between **Counter** values. E.g. for **Step 2** the **Counter** value will be incremented by 2 instead of 2

```
For Each Iterator in Items
    '...Code here...
Next [ iterator ]
```

Items	Description
Iterator	The iterating variable. Used to iterate through the elements of the collection or array
Items	A collection or array of items
Next	Closing statement for the loop. Optionally you can specify the **Iterator** variable

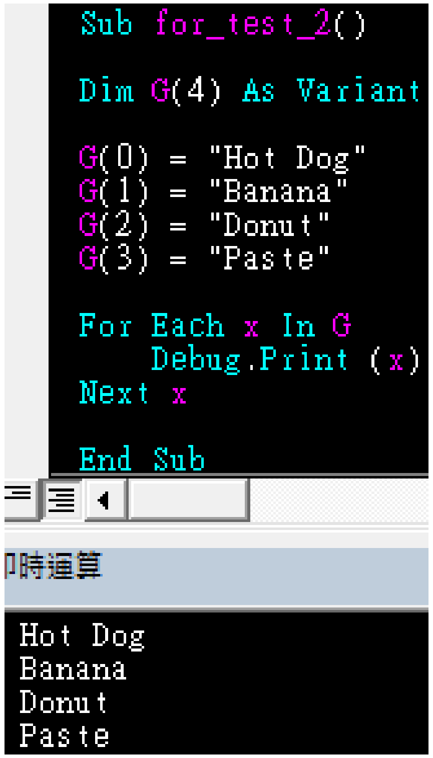
Example



```
Sub for_test_1()

For k = -5 To 5 Step 3
    Debug.Print (k)
Next k

End Sub
```

即時運算

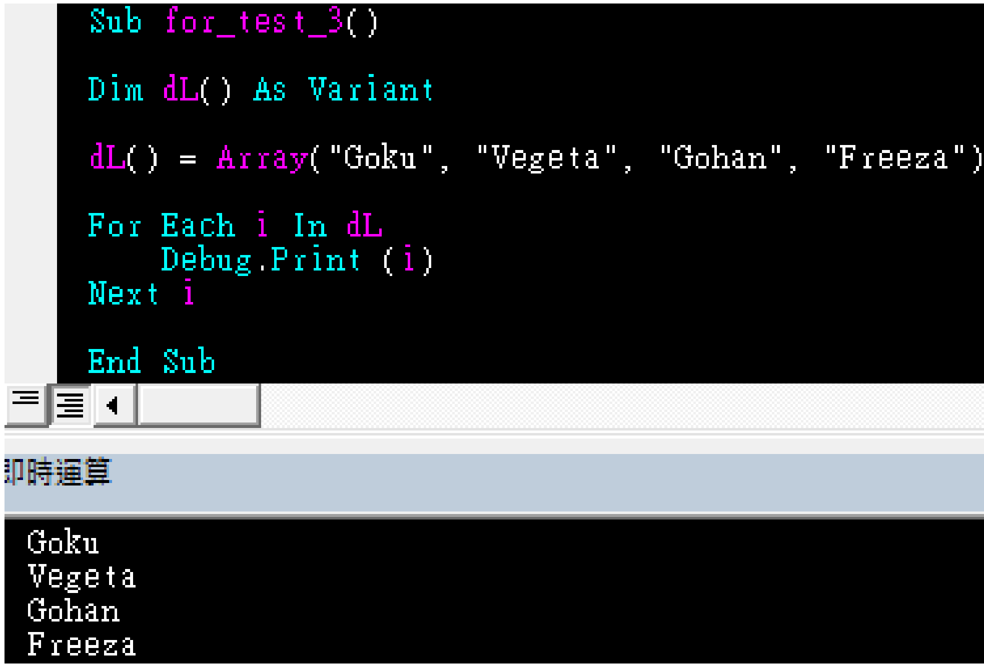
```
-5
-2
 1
 4
```



```
Sub for_test_2()

Dim G(4) As Variant

G(0) = "Hot Dog"
G(1) = "Banana"
G(2) = "Donut"
G(3) = "Paste"

For Each x In G
    Debug.Print (x)
Next x

End Sub
```

即時運算

```
Hot Dog
Banana
Donut
Paste
```

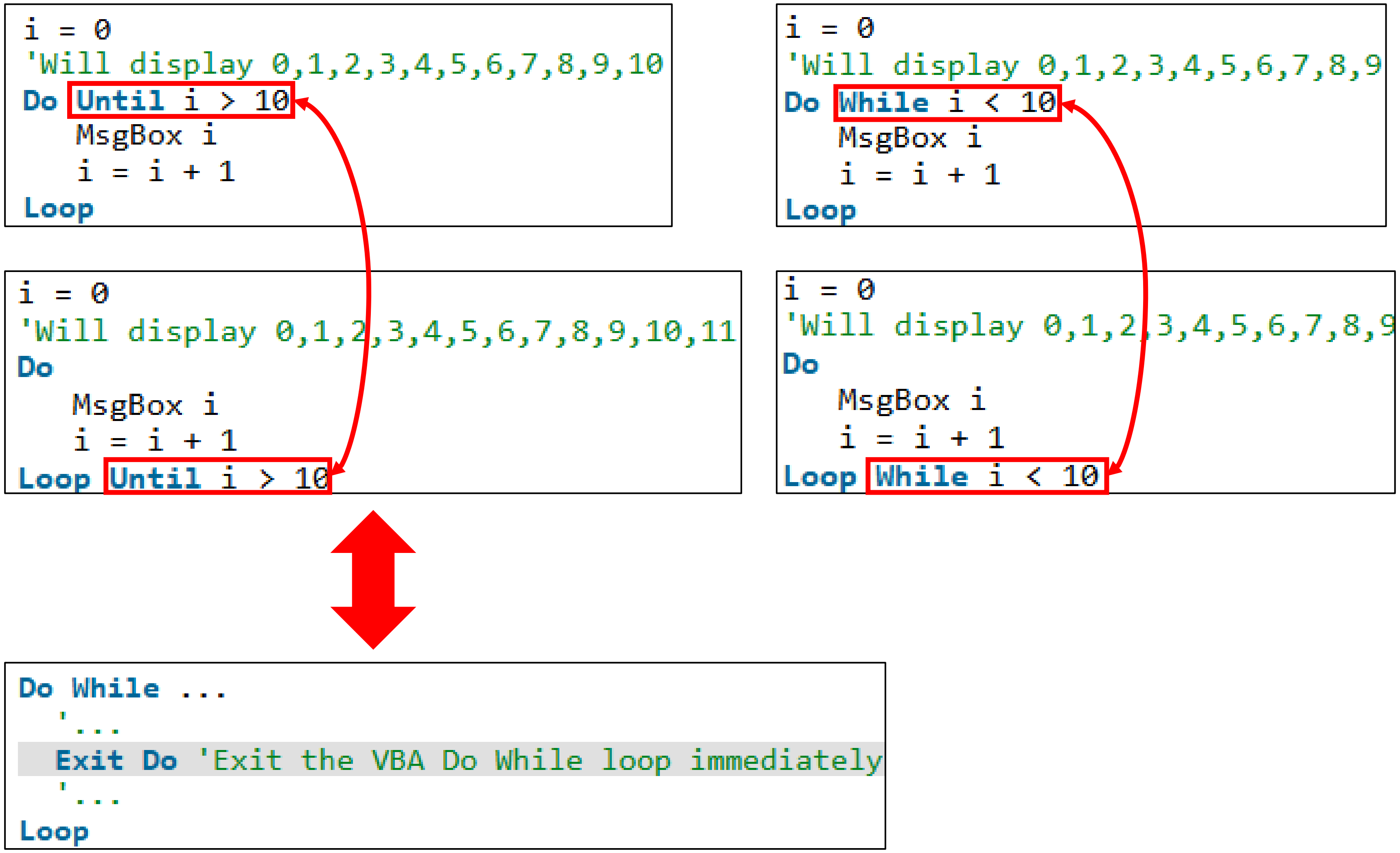


```
Sub for_test_3()

Dim dL() As Variant

dL() = Array("Goku", "Vegeta", "Gohan", "Freeza")

For Each i In dL
    Debug.Print (i)
Next i

End Sub
```

即時運算

```
Goku
Vegeta
Gohan
Freeza
```

Do until loop & Do while loop



```
i = 0
'Will display 0,1,2,3,4,5,6,7,8,9,10
Do Until i > 10
    MsgBox i
    i = i + 1
Loop
```

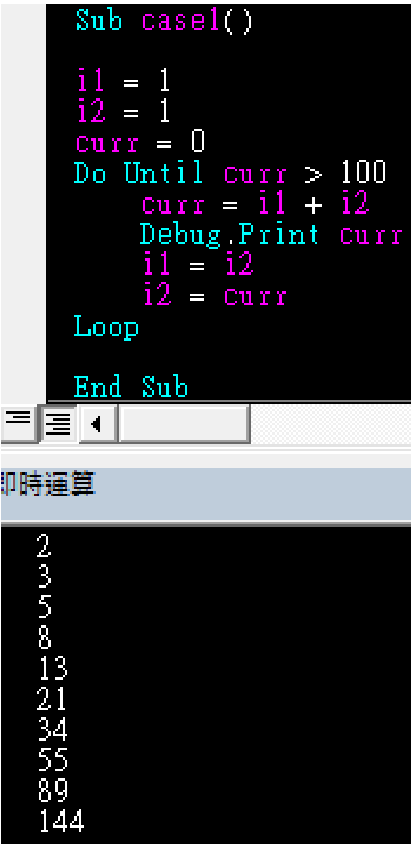
```
i = 0
'Will display 0,1,2,3,4,5,6,7,8,9,10,11
Do
    MsgBox i
    i = i + 1
Loop Until i > 10
```

```
i = 0
'Will display 0,1,2,3,4,5,6,7,8,9
Do While i < 10
    MsgBox i
    i = i + 1
Loop
```

```
i = 0
'Will display 0,1,2,3,4,5,6,7,8,9
Do
    MsgBox i
    i = i + 1
Loop While i < 10
```

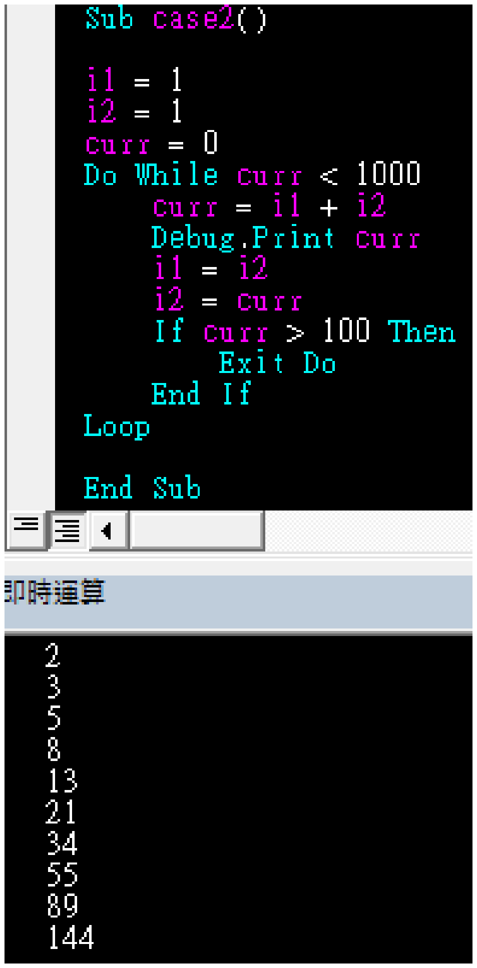
```
Do While ...
  '...
  Exit Do 'Exit the VBA Do While loop immediately
  '...
Loop
```

Example



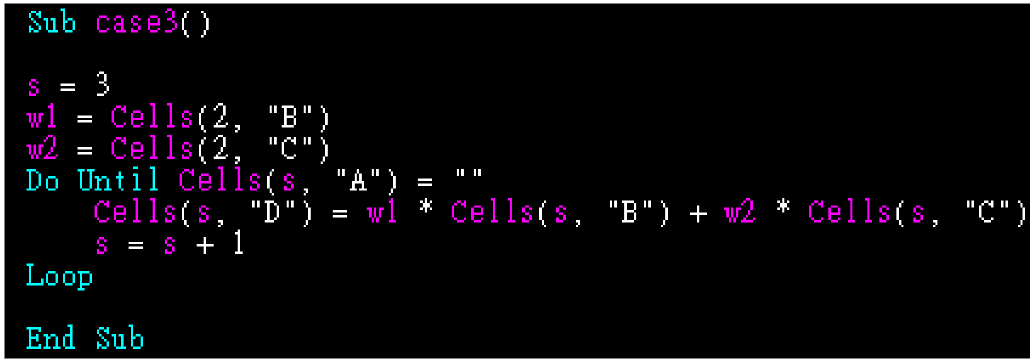
```
Sub case1()

i1 = 1
i2 = 1
curr = 0
Do Until curr > 100
    curr = i1 + i2
    Debug.Print curr
    i1 = i2
    i2 = curr
Loop

End Sub
```

```
 2
 3
 5
 8
 13
 21
 34
 55
 89
 144
```

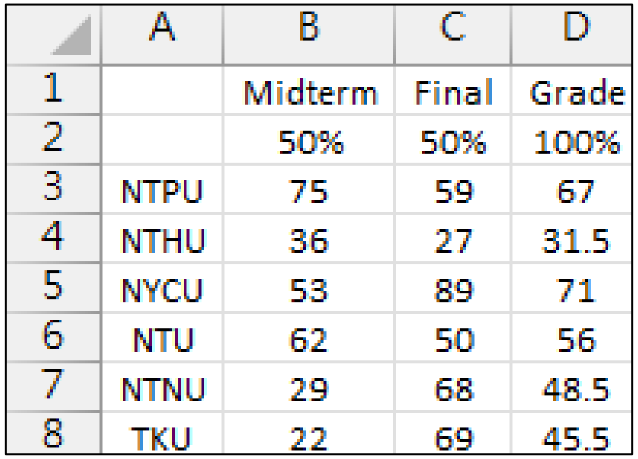


```
Sub case2()

i1 = 1
i2 = 1
curr = 0
Do While curr < 1000
    curr = i1 + i2
    Debug.Print curr
    i1 = i2
    i2 = curr
    If curr > 100 Then
        Exit Do
    End If
Loop

End Sub
```

```
 2
 3
 5
 8
 13
 21
 34
 55
 89
 144
```



```
Sub case3()

s = 3
w1 = Cells(2, "B")
w2 = Cells(2, "C")
Do Until Cells(s, "A") = ""
    Cells(s, "D") = w1 * Cells(s, "B") + w2 * Cells(s, "C")
    s = s + 1
Loop

End Sub
```



	A	B	C	D
1		Midterm	Final	Grade
2		50%	50%	100%
3	NTPU	75	59	67
4	NTHU	36	27	31.5
5	NYCU	53	89	71
6	NTU	62	50	56
7	NTNU	29	68	48.5
8	TKU	22	69	45.5

國立臺北大學 National Taipei University

Error Handling

On Error Resume Next

Skip any raised errors

```
Dim x, y
x = y /0 'Divide by 0 error!
On Error Resume Next
x = y /0 'No error raised
```

On Error Goto 0

Disable any previous VBA error handling

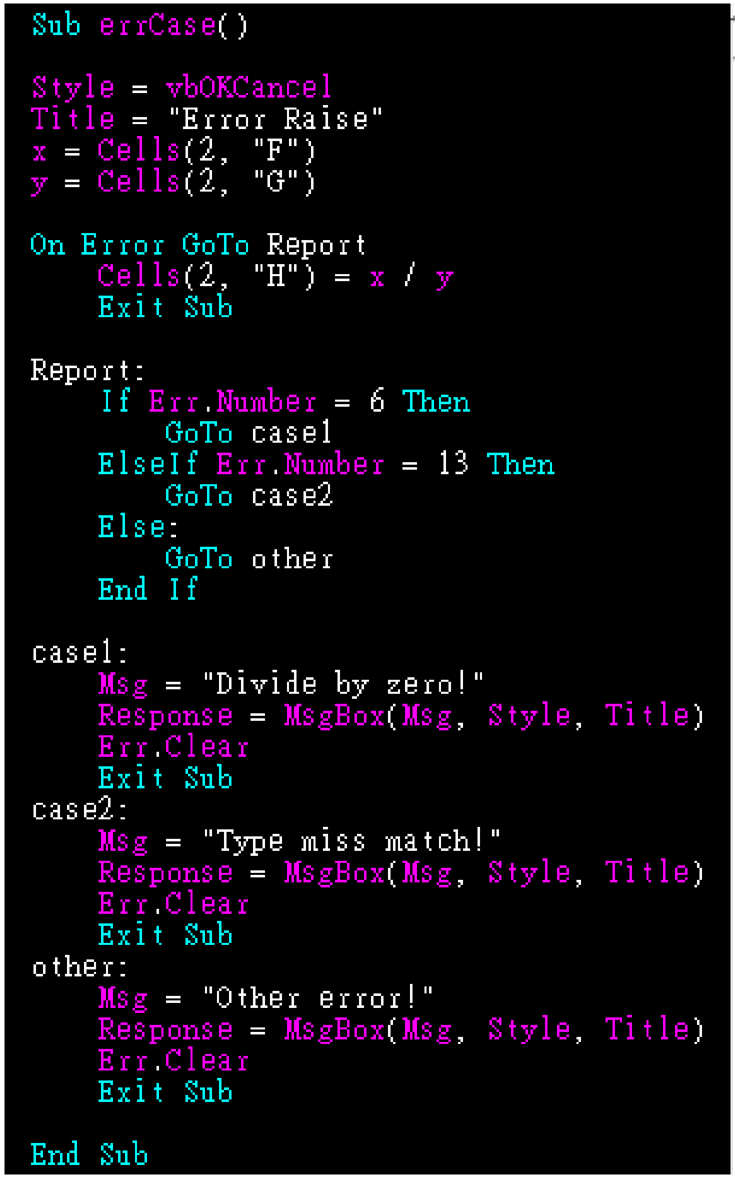
```
Dim x, y
On Error Resume Next 'Skip errors
x = y /0 'No error raised
On Error Goto 0 'Disable error handling
x = y /0 'Divide by 0 error!
```

On Error Goto Label

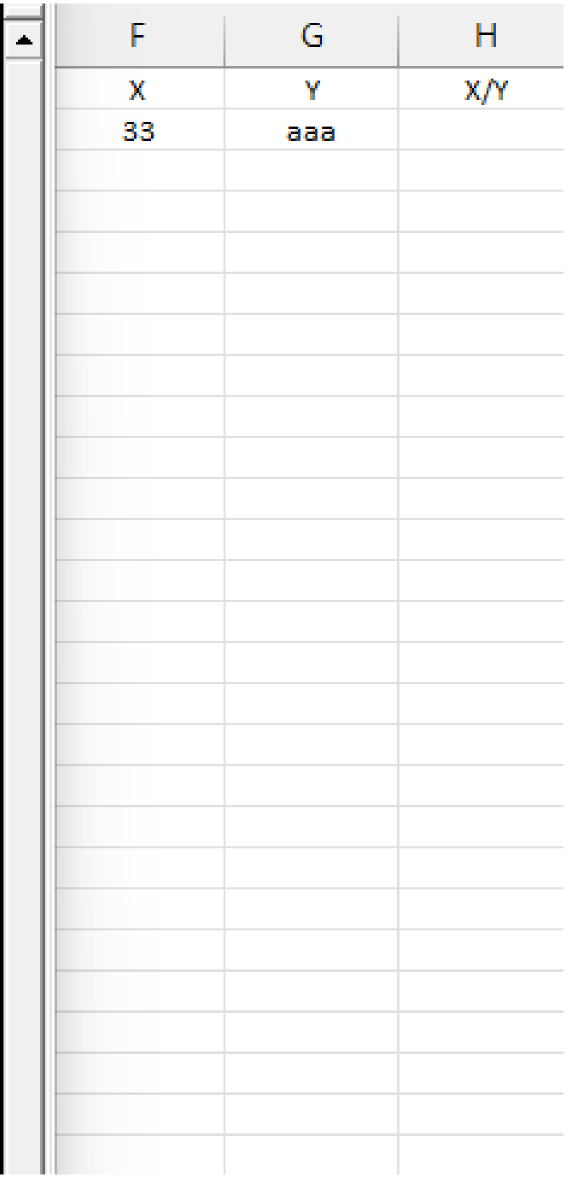
On error raised jump to a specific line label

```
Dim x, y
On Error Goto ErrorHandl
x = y /0 'No error raised
On Error Goto 0 'Disable error handling
x = y /0 'Divide by 0 error!
```

Example



```
Sub errCase()

Style = vbOKCancel
Title = "Error Raise"
x = Cells(2, "F")
y = Cells(2, "G")

On Error GoTo Report
    Cells(2, "H") = x / y
    Exit Sub

Report:
    If Err.Number = 6 Then
        GoTo case1
    ElseIf Err.Number = 13 Then
        GoTo case2
    Else:
        GoTo other
    End If

case1:
    Msg = "Divide by zero!"
    Response = MsgBox(Msg, Style, Title)
    Err.Clear
    Exit Sub
case2:
    Msg = "Type miss match!"
    Response = MsgBox(Msg, Style, Title)
    Err.Clear
    Exit Sub
other:
    Msg = "Other error!"
    Response = MsgBox(Msg, Style, Title)
    Err.Clear
    Exit Sub

End Sub
```



F	G	H
X	Y	X/Y
33	aaa	

LinEst

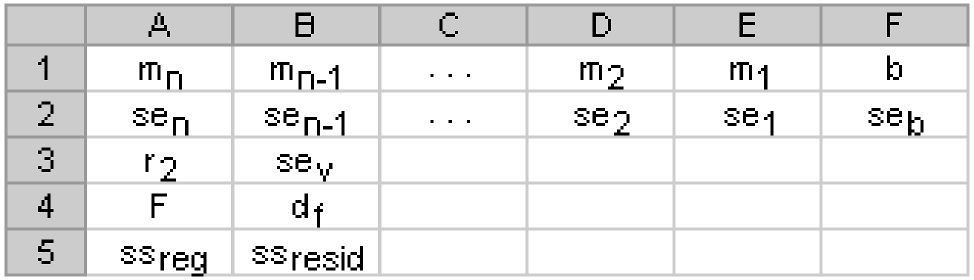
- Syntax

  WorksheetFunction. LinEst(data Y, data X, const, stats)

- If const $= \begin{cases} \text{True, intercept is calculated} \\ \text{False, no intercept} \end{cases}$

- If stats $= \begin{cases} \text{True, return regression statistics} \\ \text{False, return only coefficients} \end{cases}$

- The following illustration shows the order in which the additional regression statistics are returned

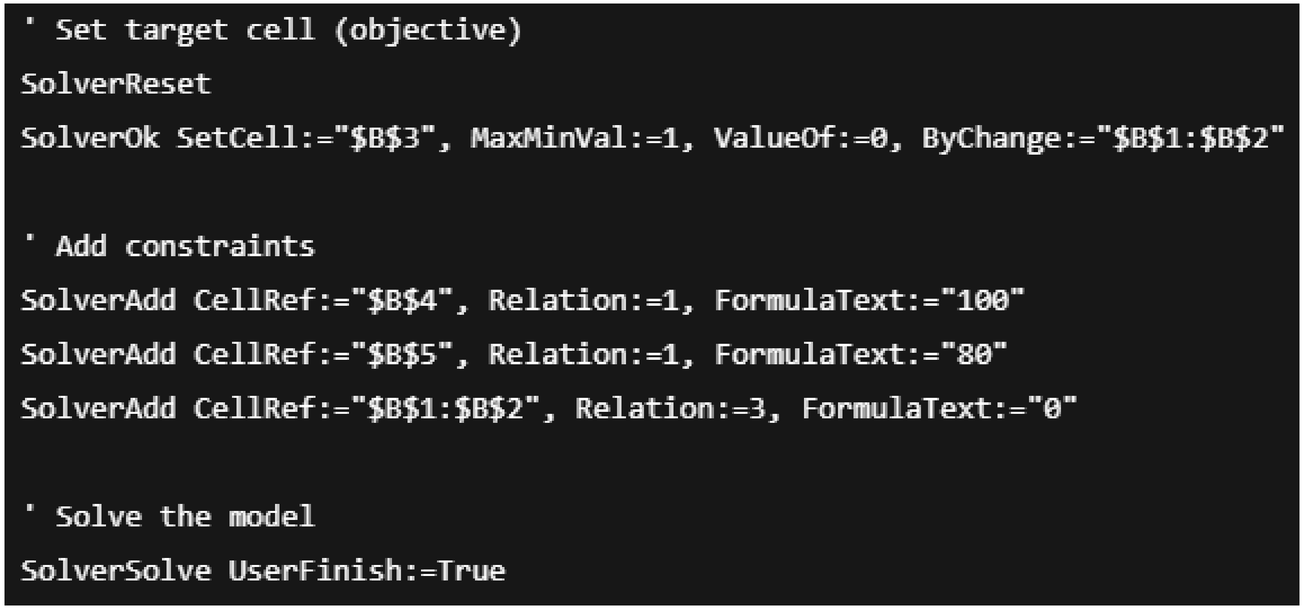


	A	B	C	D	E	F
1	m_n	m_{n-1}	. . .	m_2	m_1	b
2	se_n	se_{n-1}	. . .	se_2	se_1	se_b
3	r_2	se_y				
4	F	d_f				
5	ss_{reg}	ss_{resid}				

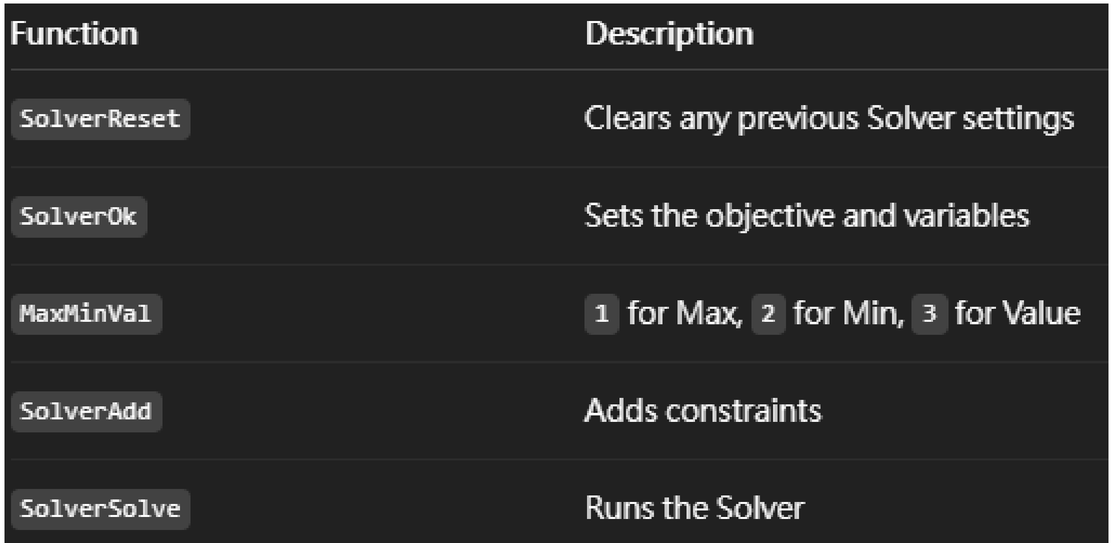
國立臺北大學
National Taipei University

LP Solver

- Syntax



```
' Set target cell (objective)
SolverReset
SolverOk SetCell:="$B$3", MaxMinVal:=1, ValueOf:=0, ByChange:="$B$1:$B$2"

' Add constraints
SolverAdd CellRef:="$B$4", Relation:=1, FormulaText:="100"
SolverAdd CellRef:="$B$5", Relation:=1, FormulaText:="80"
SolverAdd CellRef:="$B$1:$B$2", Relation:=3, FormulaText:="0"

' Solve the model
SolverSolve UserFinish:=True
```



Function	Description
`SolverReset`	Clears any previous Solver settings
`SolverOk`	Sets the objective and variables
`MaxMinVal`	`1` for Max, `2` for Min, `3` for Value
`SolverAdd`	Adds constraints
`SolverSolve`	Runs the Solver

Exercise – Build a Regression Tool

- Objective:

  Write a VBA Marco to perform simple linear regression (Include no intercept case) without using Excel function.

- Dataset

  X in column A

  Y in column B

- Predict set

  Predict Y in column C
  Residual in column D

- Output

  T test (include coefficient, p-value and test statistic)

  F test (include p-value and test statistic)
  R-squared value

- Submit:

  學號.xlsm file