**Week 8**

**Objects and Collections; Using the Macro Recorder; Shapes**

---

**Parameters revisited**

- Values passed to procedures are called **arguments**.
- Variables defined to hold these values are called **parameters**.
- Parameters are local variables, like ones declared using **Dim**.
- Data can be passed in two different ways:
  - **by value** (argument is expression, only value transferred)
  - **by reference** (argument is variable, parameter acts as a temporary alias)
- Keywords **ByVal** and **ByRef** specify which (**ByRef** is assumed if omitted)

---

**Example: Swapping values**

```vba
Sub Swap(ByVal x1 As Variant, ByVal x2 As Variant)
    Dim dblTemp As Variant
    dblTemp = x1
    x1 = x2
    x2 = dblTemp
End Sub
```

\[
y = -23.5; z = 6
\]

We used the **Variant** type so the procedure can be used to exchange the contents of variables any type. This is the only recommended use for Variants. Unfortunately, the following does **NOT** work:

```vba
Swap ActiveSheet.Cells(1,1), ActiveCell
```

---

**Reference Parameters**

```vba
Sub halve2(ByVal x As Double)
    x = x / 2
End Sub
Sub halve2(ByRef x As Double)
    x = x / 2
End Sub
y = 6.4: z = 6.4
halve2 y
' what is the value of y?
halve2 z
' what is the value of z?
```

The keyword **ByRef** (or none) passes a reference to the argument rather than its value. \( x \) becomes an alias for \( z \) and changes to \( x \) also affect \( z \) immediately.

**Named Arguments**

For procedures with long argument lists (especially built-in procedures), arguments can be specified in any order using the notation **paramsName=argument**.

**Example** (see **MsgBox Function** in VBA Help)

```vba
reply = MsgBox(title:="Bummer", prompt:="No solution to your equation", buttons:=vbCritical + vbCancelRetry)
```

---

**References**

Chapra (Part 2 of ENGG1811 Text)
- Topic 10 (chapter 3) Macro Recorder
Objects

VBA is a mixture of conventional procedural notation and object oriented programming. Excel data is represented by objects that have properties and methods are applied:

- `ActiveSheet.Cells(row,col).Formula = "=B3/4"
- `ActiveSheet.Rows(2).Clear`
- `row = ActiveCell.Row`

Methods are procedures that are associated with a type of object and applied to objects of that kind.

Collections

Array concept relies on strict ordering of elements. If elements can be named, get more general way of describing the group. A **Collection** is an array-type object indexed either by number or by name.

- **Properties** include: `Count` (number of elements)
- **Methods** include: `Add` (a named element, relatively positioned), `Item` (retrieve "current" element)

Manipulating Collections

Declare variable to reference elements:

```vba
Dim wks As Worksheet
```

Methods include:

- `wks.Visible` (property, `True/False`)
- `wks.Unprotect` (method)
- `wks.Cells(row,col).Formula = "=A1+1"`

Object assignment uses the **Set** keyword:

- `Set wks = Sheets(1)` this is easy to forget.

Collections, continued

Some Excel objects are collections:

- `Application.Workbooks (base type: Workbook)`
- `Application.ActiveWorkbook.WorkSheets (Worksheet)`
- `ActiveWorkbook.Sheets (Chart or Worksheet)`
- `ActiveSheet.Charts (Displayed charts)`
- `ActiveSheet.Shapes (Drawing objects: later slide)`

Examples:

```vba
Sheets.Add Name:="New sheet", After:=Sheets(3)
Sheets.Add Name:="New sheet", Before:=Sheets(3)
```

Iterating over a Collection

```vba
For Each wks In ActiveWorkbook.Sheets
  Debug.Print wks.Name
Next wks
```
### With Statement

Sometimes the same object needs to be referenced several times in succession:

```vba
ActiveSheet.Cells(row, col) = intVal1
ActiveSheet.Cells(row+1, col) = intVal2
ActiveSheet.Cells(row+2, col) = intVal3
```

Use a `With` statement for the common prefix:

```vba
With ActiveSheet
    .Cells(row, col) = intVal1
    .Cells(row+1, col) = intVal2
    .Cells(row+2, col) = intVal3
End With
```

*note dot is retained in the shortened reference*

### Note

The following topics contain a lot of detail:

- Most of the info can be found in Help screens or reference books
- **You don't have to learn the detail**, but you do need to understand the way that these structures are put together
- Lab work will guide you through the essential skills needed to manipulate objects, including:
  - referencing ranges
  - using the macro recorder
  - creating and manipulating drawings
- ...but you need to follow what's presented here!

### Ranges

- The `Range` type represents a rectangular group of (one or more) cells on a worksheet
- Can be referred to in many ways:
  - Single cell, separate indexes: `.Cells(row, col)`
  - Single cell, RC format: `.Range("B5")`
  - Single row, index: `.Rows(row)`
  - Single column, index: `.Columns(col)`
  - Row/column block: `.Rows("4:6")` or `.Columns("A:C")`
  - Any block, RC format: `.Range("B4:H7")`
  - Named range: `.Range("polynomial")`
  - Derived from another range: `.ActiveSheet.EntireRow`
  - Whole sheet: `.Cells`

### Some Range Properties

- **Borders** Object representing outline
- **Comment** Comment text
- **Font** Text font
- **Formula** Formula as a string (such as `"=A$1/24"`)
- **Height, Width** Height or width in screen units
- **HorizontalAlignment** `xlLeft`/`xlCenter`/`xlRight`
- **Interior** Object representing cell background
- **Locked** `True` ⇒ Cells can't be changed by the user
- **Name** Name if defined
- **NumberFormat** Name or specification of display format
- **Value** Stored value

*Note: Properties of a multi-cell range are undefined if the cell properties are not consistent*

### Some Range Methods

- **AutoFilter** Apply column filters
- **Calculate** Re-evaluate formulae
- **Clear, ClearComments** Remove cell contents or comments
- **Copy, Cut** Copy to clipboard, Cut=remove after pasting
- **Delete** Remove cells and shift in
- **FillDown, FillRight** Apply fill operations
- **Find** Search for cell contents matching value
- **Offset** Generate reference to nearby cell (used in week 5)
- **GoalSeek** Apply Excel's Goal Seek (range must be one cell)
- **Merge** Merge cells, retaining value from top left cell
- **PasteSpecial** Paste from Clipboard with optional transforms
- **Replace** Replace cell contents
- **Select** Make range the current selection (available as the Selection object)
- **Sort** Reorder rows (has many parameters)

### Range as a Collection

*From an example at [http://www.fontstuff.com/casebook/casebook01.htm](http://www.fontstuff.com/casebook/casebook01.htm)*

```vba
Function SumIfColoured(rngCells As Range) As Double
    Dim cell As Object
    Dim sum As Double

    sum = 0
    For Each cell In rngCells
        If cell.Interior.ColorIndex <> xlNone Then
            sum = sum + Cell.Value
        End If
    Next cell
    SumIfColoured = sum
End Function
```
Macro Recorder

- Almost everything the user can do with Excel can be tracked as a Macro and replayed*
- Macros are stored as VBA procedures
- Macros can be edited (or cannibalised!) to adapt or generalise their purpose
- Effective way to find out how to achieve a particular effect (especially as VBA Help is poor)


Using the Macro Recorder

- Menu path
  Developer – Record Macro…
  (Tools – Macro – Record New Macro… for Excel 2003)
- Name, location, description and optional shortcut key can be set.
- Creates new module if necessary
- Tracks all menu functions and relevant mouse activity (cell selections, dialogue responses, etc)
- When finished,
  Developer – Stop Recording
- View/edit macro with VBE

Macro Recording conventions

- Use of `Select` method and `Selection` object – since the user has to identify an element before doing anything to it
- Use of `With Selection`...
- Sets all properties of an object, even if only one was changed
- Uses `Range()` and related objects extensively
- Uses absolute references, where a programmer would use a variable or constant (hence readability is poor)

Example: Cell operations

Apply the following to a two-column table such as the solution to Lab 7 Part B:
1. Adjust the alignment and indent of column B;
2. Change the background of the header cells
3. Change the font weight and colour of the header cells

VBA Created by Recorder

```
Columns("B:B").Select
With Selection
  .HorizontalAlignment = xlRight
  .VerticalAlignment = xlBottom
  .WrapText = False
  .Orientation = 0
  .AddIndent = False
  .ShrinkToFit = False
  .ReadingOrder = xlContext
  .MergeCells = False
End With
Selection.InsertIndent 1
Range("A1:B1").Select
With Selection.Interior
  .Pattern = xlSolid
  .Color = 192
End With
Selection.Font.Bold = True
With Selection.Font
  .Color = -16711681
  ' other font changes...
End With
```

Colour Representation

- Prior to 2007 Excel stored cell colours as one of 56 standard values (ColorIndex). This still works.
- 2007 introduced themes, but can store RGB (red, green, blue) values too
  - `.Color` is a Long integer, derived from R, G, B components (each 0..255)
  - code = 1*red + 256*green + 256*256*blue
- Example: dark brown is R=200, G=100, B=0
  - COL_DKBROWN = 200+256*100 = 256200
- better: use RGB function
  - `Selection.Font.Color = RGB(200, 100, 0)`
Humans can code iteration, Recorder can't
Humans can use local variables, Recorder can't
Humans can use Const definitions and meaningful names, Recorder can't

The Macro Recorder makes extensive use of the Selection object (since the user must select elements before using them)

Humans can refer directly to the affected object:
Const COL_DATA = 2
  With Columns(COL_DATA) ‘ = Columns("B:B")
  .HorizontalAlignment = xlRight

• Coordinate system is Cartesian (which way is Y?)
• Units are points (1/72 inch, approx 28 pts per cm)
• Drawing elements are members of the Shapes collection belonging to the active sheet
• New shapes are created using methods (built-in procedures) with distinctive args for shape type and characteristics
• Many shape types (type AutoShapeType to Help, then click on msoAutoShapeType for list)
• Changes to properties with user interface is via Selection object (though not required in VBA)

Shapes in VBA

Create most shapes with
Dim shp As Shape
Set shp = ActiveSheet.Shapes.AddShape(type, _
  left, top, width, height)
Set shp = ActiveSheet.Shapes.AddTextBox(_
  then use shp to set properties

Each shape is given a name and number according to its type ("Rectangle 1", "AutoShape 5"), and each has an index, equal to the collection size after Add
lastShape = ActiveSheet.Shapes.Count
Can later use the index to reference the shape:
ActiveSheet.Shapes(lastShape)
Keeping track of all shapes this way can be tricky

Shape Operations

Applying stroke (outline)
With object.Line
  .Weight = 1.5 ‘ in points/pixels
  .Style = msoLineSingle ‘ default
  .DashStyle = msoLineSquareDot
  .ForeColor.RGB = RGB(0,0,128) ‘ dark blue stroke
  .EndArrowheadStyle = msoArrowheadTriangle
  .EndFillArrowheadLength = msoArrowheadLengthMedium
End With

Applying fills
  .Fill.ForeColor.RGB = RGB(red, green, blue) ‘0..255
  .Fill.Transparency = @0 ‘ @0 = 0.0 (VB idiom)
  .Fill.Visible = False ‘ to hide (also .Line)

Text Boxes

Text is held in boxes (with stroke and fill)
object.Characters.Text = "Power Station"
With object.Characters.Font
  .Name = "Arial Narrow"
  .FontStyle = "Bold"
  .Size = 11
End With

Font properties of substrings
object.Characters(Start:=3, Length:=2) _
  .Font.Subscript = True

Remove box:
  .Fill.Visible = False
  .Line.Visible = False
Examples (Wed 4pm)

1. Colour Picker (using RGB codes from a sheet)
2. Programming with shapes (time permitting):
   (a) Chessboard – just alternating black/white squares
   (b) Venn Diagram

Algorithm is simple (but caics are fiddly)
Draw canvas
Calculate geometry
Draw first set
Draw second set
Draw labels (4 sub calls)
Draw pointer

Summary

- Reference parameters allow a subprogram to change the value of variables declared outside
- Collections store a set or sequence of objects, referenced by either name or index
- Objects enable access to complex data managed by the application
- A Range object represents part of a worksheet, can also be viewed as a collection of cells
- Macro recorder creates editable VBA from user interaction with the application