VBA Handout

References, tutorials, books

- Excel and VBA tutorials
- Excel VBA Made Easy (Book)
- Excel 2013 Power Programming with VBA (online library reference)
- VBA for Modelers (Book on Amazon)

Code basics

- Comments start with a single quote: '
- All variables require a type declaration
  - **Long** for integers
  - **Double** for floating point numbers
  - **String** for strings
  - **Boolean** for true or false variables
  - **Variant** for a flexible type (anything but string)
- Declare variables as:

```
  Dim myVar As <Type>
```

- Variables can be declared inside or outside of functions or subroutines. Variables declared outside have a global scope.
- Indentation doesn't matter.

Conditional statements

- If statements

```
If condition Then
    ' do stuff
ElseIf condition Then
    ' do stuff
Else
    ' do stuff
End If
```

- Conditionals
  - Equals, =
  - Greater than, >
  - Less than, <
  - Greater than or equal to, >=
  - Less than or equal to, <=
  - `Or`
  - `And`
  - `Not`
  - `True`
  - `False`

Ranges and Arrays

- Range variables come from cells in the worksheet.
  - Declare a range variable

    ```
    Dim myRange As Range
    Set myRange=Range("A1:A20")
    ```

  - Get or set the values contained in the range.

    ```
    Range("<Val>").Value
    ```

  - Get a specific cell in a range at `Row`, `Col`.

    ```
    Range("<Val>").Cells(Row, Col)
    ```

  - Get or set the value in a given cell in a range.

    ```
    Range("<Val>").Cells(Row, Col).Value
    ```

  - Set the formula of a cell.

    ```
    Range("<Val>").Cells(Row, Col).Formula
    ```

- Array variables
  - Declare 1D Array:

    ```
    Dim myArray(100) As Double
    ```

  - Declare 2D Array:

    ```
    Dim myArray(5, 8) As Double
    ```

  - Access elements with (...):

```
' 1D array
Dim myArray(5) As Double
myArray(3) = 2
' 2D array
Dim myArray(5,5) As Double
myArray(2,3) = 2
```

– Initialize array directly from Range Variable as below. These are *always* 2D arrays with an index that starts counting at one.

```
Dim  myArray as Variant
myArray = myRange.Value
' Always a 2D array with indicies starting at 1
' Ex: myArray(1,1)
```

– These are

– Get array size from upper and lower bound

```
UBound(myArray)-LBound(myArray)+1
```

Loops

- `For` loop

```
Dim i As Long
Dim iStart As Long
Dim iEnd As Long
For i = iStart To iEnd
    ' do stuff
    If i=iStart+2 Then
        Exit For ' break the loop early
    End If
Next i
```

- `Do While` loop

```
i = 0
Do While i < 100
    i = i+1
    ' do stuff
Loop
```

- `Do Until` loop

```
i = 0
Do Until i = 100
    i = i+1
    ' do stuff
Loop
```

Subroutines and Functions

- Functions and subroutines are similar except functions can return a value and subroutines cannot.
- Arguments can be passed by *value* or by *reference*.
  - Pass by value means a copy of a passed variable is made, so changes to it don't affect the orignal variable.
  - Pass by reference means that if you pass a variable and change that argument inside the routine or function, then it changes the corresponding variable that was passed. This is useful for getting data out of a subroutine
  - **In VBA, pass by reference is the default**
  - **Keywords:** `ByVal` and `ByRef`. Also `Optional` for optional arguments
- Functions
  - A function is called using its name, e.g. `sum(5, 4)`
  - There is no explicit *return* statement in VBA. Instead we assign to the value to be returned to name of the function.
  - The type of value that the Function returns also needs to be declared (after the parentheses).
  - The function will exit and return at the end, but you can exit early with `Exit Function`

```
Function sum(x As Double, y As Double) As Double
    sum = x + y
End Function
```

- Subroutines
  - A subroutine is called with `Call mySubName(parameters)`. If there are no parameters, just use `Call mySubName`.

```
Sub add_2_vals(x As Double, y As Double, _
                 sum As Double)
    ' x and y passed by reference. If changed in
    ' here the variables passed in will change
    ' (same for sum, but that is by design).
    '
    sum = x + y
End Sub
```

```
Sub add_2_vals(ByVal x As Double, ByVal y As Double, _
             sum As Double)
    ' x, y passed by value. If changed here, the
    ' variables passed in won't be changed.
    sum = x+y
End Sub
```

Debugging

- The "Immediate Window" (View–>Immediate Window) is useful for Debugging VBA code. Using the `Debug.Print` Command one can examine the value of variables or run functions. One can also run subroutines (via the `Call` command) in this window.
- In the Immediate Window, "?" is equivalent to `Debug.Print`

```
Debug.Print myFunction(3)
? myFunction(3)
Call mySub(4)
```

Useful Builtin VBA functions

- A pop-up message box:

  ```
  msgBox("A window will open with this message")
  msgBox("The value of myVar is " & myTemp & " (K)")
  ```

- Input box to get user input:

  ```
  myVariable = InputBox("Some message")
  ```

- Accessing individual cells

  ```
  ' Get data from excel cell B23 and store it in myVar
  myVar = Range("B23").Value
  ' Put data from myVar into cell B23
  Range("B23").Value = myVar
  ```

- Test if cell is empty, return true or false:

  ```
  IsEmpty(Range("B23"))
  ```

- Clear range of cells:

  ```
  Range("B5:B25").Clear
  ```

- Grab the value in the cell that is up 3 and right 4 from the currently active cell.

  ```
  ActiveCell.Offset(−3, 4).Value
  ```

- Math functions: `Abs`, `Int`, `Sqrt` `Exp`, `Log` (natural log), `Cos`, `Sin`, `Tan`
- To get access to Excel worksheet functions:

  ```
  WorkSheetFunction.funcName(x,y, etc.)}
  ```