Sub Programs

To Solve a Problem, First Make It Simpler
Top Down Design

- Start with overall goal.
- Break Goal into Sub Goals
- Break Sub Goals into Sub Sub Goals
- Until the Sub-Sub-Sub-Sub-Sub Goals are Easy.
Functions

- The VB Function is derived from a mathematical function:
  - \( f(x) = x^2 + 3x + 1 \)
- We use the function to calculate a single value.
A function called “factorial” takes in a number and returns its factorial. 

\[ \text{fact}(5) \text{ returns 120} \]
How Functions are Used

- Functions are defined somewhere.
- They are then used in another place.
- The place they are used is called the Calling Function or Calling Sub Procedure or Calling Routine.
Defining Our Own Functions

Functions have several parts to them:

- The Header
- The Body
- The End Statement
Function even (num as integer) as Integer

If((num mod 2) = 0) Then
    even = true
    Return
End If

even = false

End Function
The Function Header

The function header has:

- the word Function,
- the name of the function,
- the list of arguments it will accepts
- the function type

Function Norb (fleeb as Integer) as Integer
Function Headers Must Have

- The Word **Function** at their beginning
- A **Name** - The names are restricted.
- An **Argument List** (possibly empty)
  - What values will be sent in?
- A **Type**
  - The function is going to return a value, what type will it be?
Function Bodies

- A series of statements.
- There is one statement that must be in that series,
  - `function_name = expression`
- This indicates what value should be returned from the function.
Early Out

The function body can contain an *Exit Function* statement.  
This returns control immediately to the calling function.

Useful for Finding Something and Quitting
This contains the words End Function on a line by itself:

End Function

This is used to indicate the last statement in the Function Definition.
Example Function

Two Numbers In, Smaller One Out
Function Smaller (A as Integer, B as Integer)
As Integer
If A < B Then
    Smaller = A
Else
    Smaller = B
End If
End Function
How do we get numbers in?

- What if we want to send a value in?
- We use an argument to receive values into the function.
- The arguments are used like variables and have whatever value was sent to them by the calling program.
- Arguments are described in the Header of the function.
The argument list will have the following form:

( ByVal var_name as type , ByVal next_var as type)

Parts in italics are user defined, those in bold are optional, everything else is required.
Argument List

- Allows the function to operate on differing data.
- Variables followed by type (as in Dim)
- In Excel these are likely to be ranges.
The number of arguments are up to you.

Each argument must have a type

Each argument can be plain or ByVal

Arguments are named just like variables

( Fleeb As Integer)
Using the Argument List

Values are transferred to the arguments in order.

- The first value in the calling routine become the value of the first argument
- The second value becomes the second argument and so on.
Connecting the Function

\[ x = 2 \]
\[ \text{For } i = 1 \text{ to } 10 \]
\[ y = \text{pow}(i, x) \]
\[ \text{next } i \]
\[ \text{MsgBox CStr}(y) \]

Function \textit{Pow} (m as integer, ex as single) as Single
\[ \text{Pow} = m ^ \text{ex} \]
End Function
Argument Examples

Function Fleeb (R1 As Range) As Integer
- Takes in a range, returns an integer.

Function Norb(R1 As Range, num As Integer) As Single
- Takes in a range and an integer, returns a single.
The optional ByVal modifier means that doing things to the argument in the function will have no effect on them in the calling function.

This is good if we have several people writing a program and we want to make sure our function doesn’t interfere with what they are doing.
Argument Communication

- Plain arguments are two way communication.
- ByVal arguments are one way communications between the calling function and the function.
How ByVal Effects the Connection

Before ByVal

After ByVal
Side Effects

- Functions are not supposed to have any effect other than returning a value.
- If they do anything else it is known as a side effect.
- Side effects are usually not a good idea.
- Someone using your function has to be very, very careful so that your function doesn’t mess up their program.
Sub Procedures

Sometimes we want to do something more than just pass back a single value.

We can use a Sub Procedure to modify the value of several arguments.

This is not a side effect with a Sub Procedure, it is what they are designed for.
Defining Sub Procedures

- Sub Procedures also have a header, body, and an end statement.
- The header looks like this:
  ```vbnet
  SUB sub_name (ByVal var_name as type , ...) 
  ```
- It doesn’t have a return type because it doesn’t “return” a value.
- Values are passed back through the argument list.
Unlike Functions, Sub Procedures are called using a special command.

\textit{Call} is used in front of the Sub:

\texttt{Call Sort_Array(Arg1, Arg2)}

This is done because the Sub is not evaluated to some value - it returns its value through the parameters in its argument list.
The body of a Sub Procedure can have a series of statements.

There is no need to set the name of the Sub to a value, after all, we’re returning values via the argument list.

A Exit Sub statement can be used to return control to the calling subprogram.
The Sub Procedure is terminated by the statement End Sub.
Scoping

- Variables are only visible in a certain area - this area is known as their scope.
- The ability to encapsulate functionality in sub programs is enhanced by the limited scope of the variables in the sub-programs.
Scoping, Con’t

Places where a variable can’t be seen, it can’t be modified. So we don’t have to worry about it. (Out of sight, out of mind...)

Variables declared within a Sub or Function are only visible within them.

Sub Fleeb ()
Dim a, b
a = 0
End Sub

Sub Norb ()
Dim a, b
a = 3
End Sub
Scoping Within a Module

- Variables declared at the module level are visible anywhere in the module.
- Module level variables are specified at the top of the page.
- Except where they are blocked by a local variable of the same name.
## Illustration of Scoping

### Module’s General Declaration

Dim `b,c,d`

<table>
<thead>
<tr>
<th>Sub Fleeb ()</th>
<th>Sub Norb ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim <code>a</code></td>
<td>Dim <code>b</code></td>
</tr>
<tr>
<td><code>a = 0</code></td>
<td><code>b = 3</code></td>
</tr>
<tr>
<td>‘`a,b,c,d are visible’</td>
<td>‘`b (the local one), c, d ‘are visible’</td>
</tr>
<tr>
<td>End Sub</td>
<td>End Sub</td>
</tr>
</tbody>
</table>

'`a,b,c,d are visible' |

'`b (the local one), c, d ‘are visible' |
Modules in VB

- Modules (files containing just code) allow us to pass code between spreadsheets.

- Sometimes you want to give out your code in such a way that it can be added to an existing spreadsheet.
Global Variables

- Modules in a Spreadsheet can have variables accessible in two modules.

- This is done by declaring it “Global”, in which case it can be accessed from anywhere in the spreadsheet’s VBA code.
Scoping in a Project

Module 1
Global A, B
Dim C, D
Sub A
Dim A, E
Function B
Dim B, F

Form 1
Dim B, G
Sub C
Dim C, H
Function D
Dim B, J

Form 2
Dim B, G, K
Sub E()
Dim B, A, L
Function F
Dim L, M, J, E
Variable Lifetime

- Variables only last as long as the code block they are in.
- After the code ends they evaporate.
- Unless:
  - The values can be preserved by passing them back.
  - The variables and their values can be preserved by declaring them static.
Summary

 » Over the last few weeks we have seen how VB supports the decomposition of functionality.

 » This included:

   » the ability to declare a Sub or Function
   » the ability to pass values from one Sub/Function to another,
   » the scoping of variables,
   » and the lifetime of variables.