Introduction to VBA in Excel

Agenda

1. Introduce Case Study
2. Getting Started with VBA
3. Subroutines (Macros)
4. Writing VBA Code
5. Demo

01.

Case Study Introduction

Background

- You are an actuary for Bruin Health Insurance, LCC, a Los Angeles based company insuring personal health claims in Southern California
- Prior to your team's analysis of coverage and reserves, you notice that the policy data is not formatted in the desired manner
- Each membership is only listed once, regardless of how many times it has been renewed

- Problem: You need to be able to identify the specific 1-year term you're looking at when conducting the analyses

Background

- Task: Update the way the company's data is stored

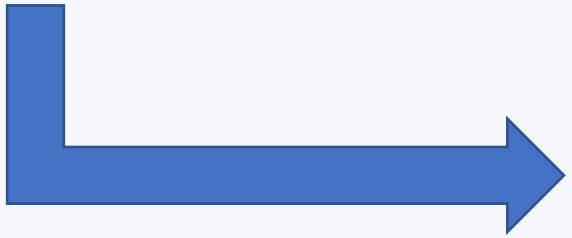
- If a membership has been renewed twice (i.e. it was in force for 3 terms), there should be 3 rows, one for each term
- The listed start date should be the start date of each term (i.e. a new row is created upon renewal)

Objective

1. Write a VBA macro to accomplish this task with the short excerpt of data
2. Then, once you've ensured the macro works correctly, run it on the full set of membership data, splitting the 13,041 membership plans into 39,215 rows

Your Goal

	A	B	C	D	E
1	MemberID	MembershipStartDate	NumberOfTerms	AvgAnnualClaims	MedicalPlan
2	P100458	3/17/14	3	$964.82	HMO
3	P105944	9/15/16	3	$692.34	HMO
4	P108055	9/21/17	5	$516.26	PPO
5	P108257	10/24/17	2	$696.74	HMO
6	P108429	11/20/17	1	$533.93	POS
7	P108776	1/15/18	2	$276.34	PPO
8	P108873	2/2/18	2	$786.09	HMO
9	P111792	6/14/19	2	$316.17	POS



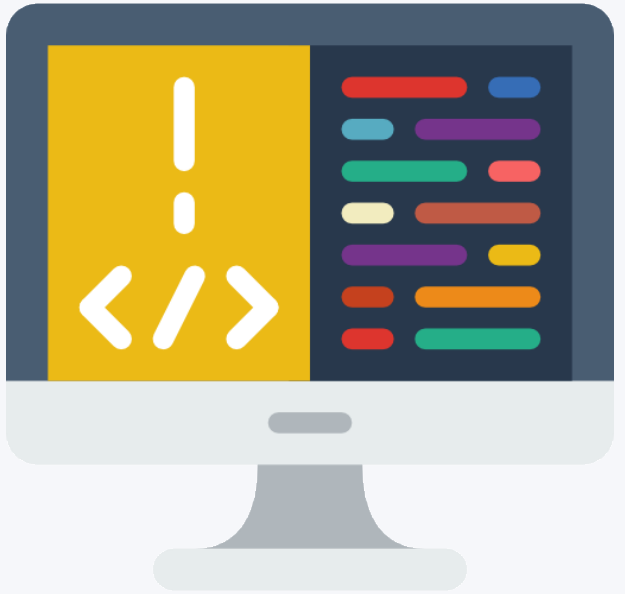
	A	B	C	D
1	MemberID	MembershipStartDate	AvgAnnualClaims	MedicalPlan
2	P100458	3/17/14	$964.82	HMO
3	P100458	3/17/15	$964.82	HMO
4	P100458	3/17/16	$964.82	HMO
5	P105944	9/15/16	$692.34	HMO
6	P105944	9/15/17	$692.34	HMO
7	P105944	9/15/18	$692.34	HMO
8	P108055	9/21/17	$516.26	PPO
9	P108055	9/21/18	$516.26	PPO
10	P108055	9/21/19	$516.26	PPO
11	P108055	9/21/20	$516.26	PPO
12	P108055	9/21/21	$516.26	PPO
13	P108257	10/24/17	$696.74	HMO
14	P108257	10/24/18	$696.74	HMO
15	P108429	11/20/17	$533.93	POS
16	P108776	1/15/18	$276.34	PPO
17	P108776	1/15/19	$276.34	PPO
18	P108873	2/2/18	$786.09	HMO
19	P108873	2/2/19	$786.09	HMO
20	P111792	6/14/19	$316.17	POS
21	P111792	6/14/20	$316.17	POS

02.

Getting Started with VBA

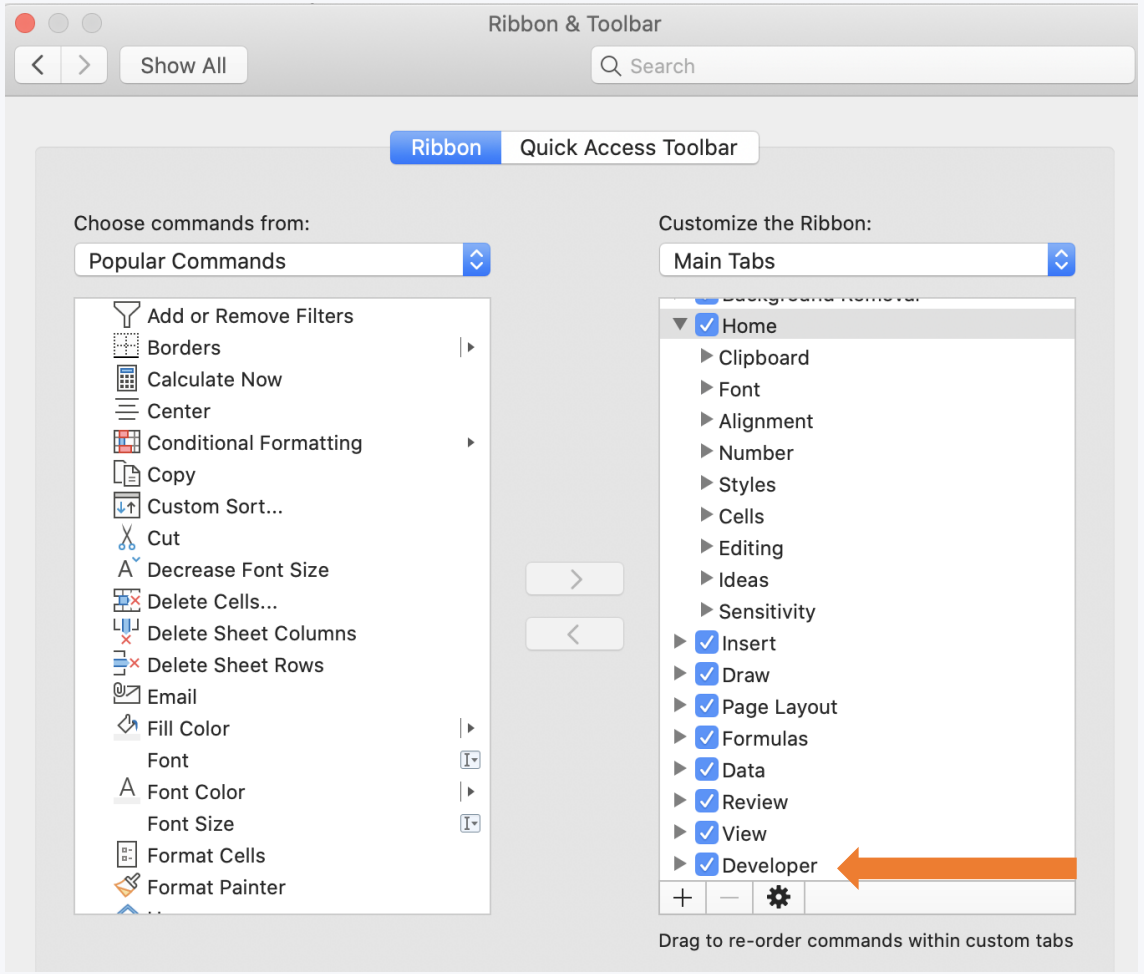
What is VBA?

- VBA stands for Visual Basic with Applications
- Closely related to Microsoft's Visual Basic
- Object-based language (similar to object-oriented languages)
- Used in Excel to simplify repetitive or complex tasks



Enabling Developer Tab

- On the menu bar, click:
  - Excel ->
  - Preferences ->
  - Ribbon & Toolbar ->
  - Developer



The VBA Environment

- Multiple ways to open the VBA Editor:

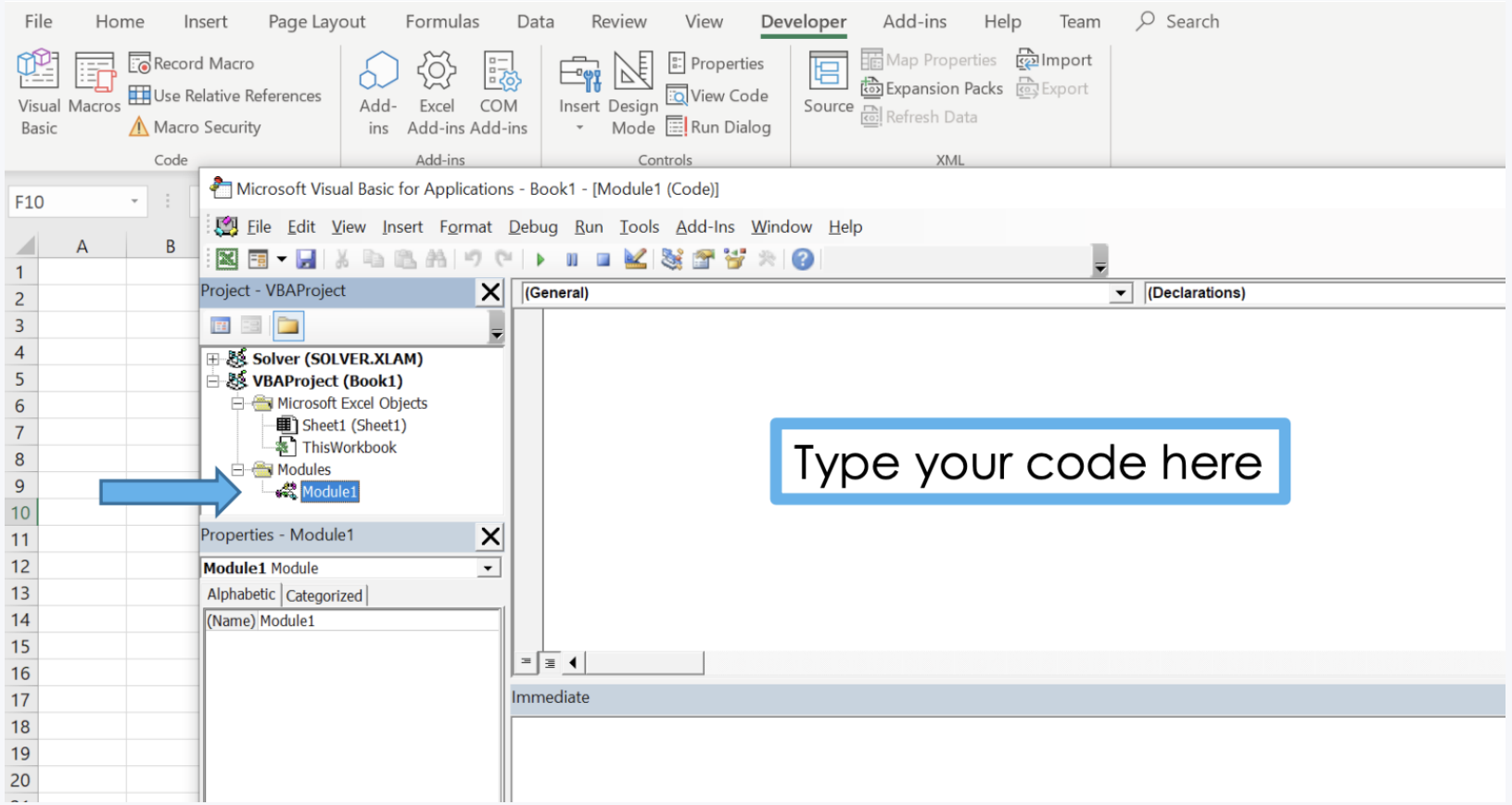
1) Go to Developer tab → Select Visual Basic

1) Keyboard shortcut:
    1) Windows: Alt + F11
    2) Mac: Opt + F11

2) Right-click on a worksheet → Select View Code

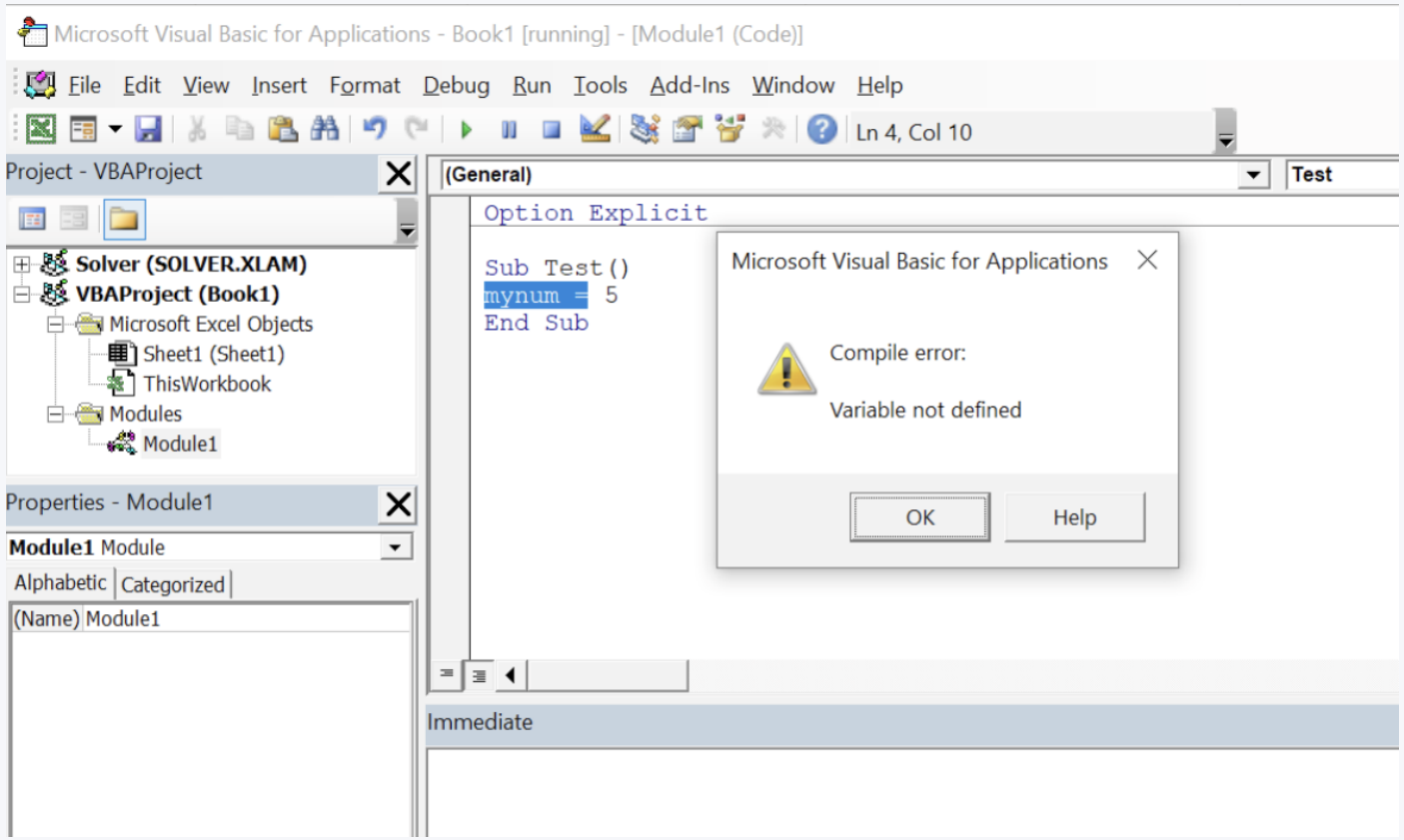
Getting Started

- A module is where you can write VBA code



Getting Started

- It is always a good idea to put "Option Explicit" in the declarations at the top of your module
- This forces you to declare all your variables
- You can either:
  - Manually type this, or
  - Go to Tools → Options → Check "Require Variable Declaration"



03.

Subroutines (Macros)

VBA in Excel

- We write subroutines (denoted Sub) to accomplish tasks that do not return values
- We write functions to take in inputs and return some output
- VBA objects include workbooks, worksheets, and ranges:
  - Application.Workbooks("Book1.xlsm").Worksheets("Sheet1").Range("A1")
- Objects have properties and methods, the most important of which is: Range("A1").Value

Subroutines

- Most of your VBA code will be in subroutines (also called macros)
- Subroutines can be used to perform calculations, change formatting, and copy and paste among other repetitive tasks
- We enclose our code in the following:

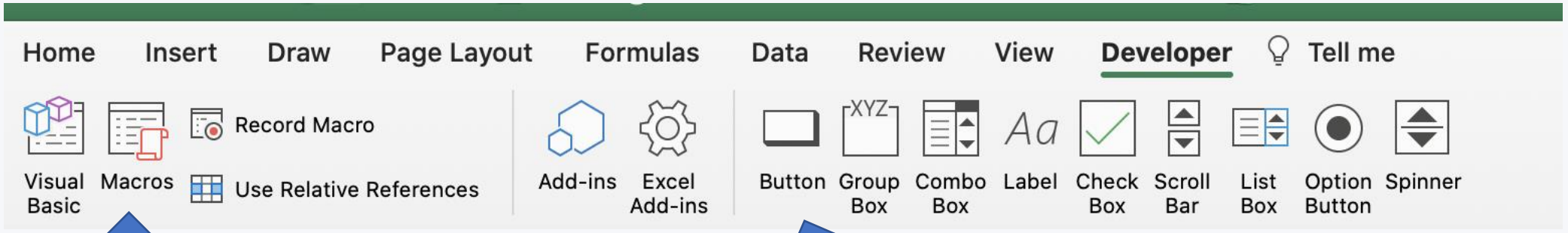
```
Sub Myroutine()


End Sub
```

- In this case, we have named our subroutine “Myroutine()” (but we can name this whatever we want)

Subroutines

There are multiple ways to call a subroutine



Click “macros” in the developer tab

Add a button from the developer tab

04.

Writing VBA Code

VBA in Excel

- 2 commonly used methods in VBA: Select and Activate
- Select allows you to select one or more objects
  - Worksheets("Sheet1").Select
  - Range("A1:A3,C1:C3,E1:E3").Select
- Activate allows you to select one object. If you already have multiple objects selected, this allows you to select one object within them.
  - Range("A1").Activate
- Ex: You can select a worksheet where you want your code to run and then activate the first cell in that worksheet that you want to apply changes to.

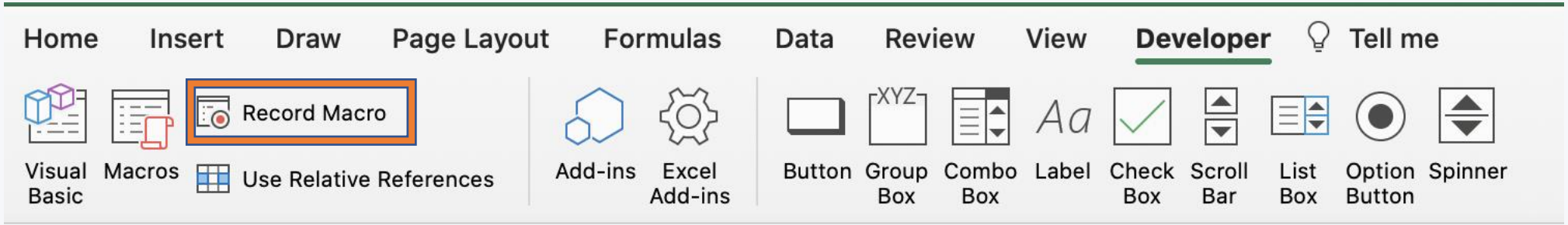
Functions

- **Syntax**

```
Function MyFunction(param1 As dtype, param2 As dtype,…) As dtype
    statements
    MyFunction = value
End Function
```

- **Ultimately, you assign a return value by setting the name of the function to some value**
- **These functions can be called from workbook cells, like how you use COUNTIF(), VLOOKUP(), and other functions**

Macro Recorder



- This is the easiest way to "write" a macro
- Under the Developer tab, click "Record Macro"
- All your actions will be translated into code, which you can find under Modules in the VBA Editor
- You can look at and modify this code to suit your purposes
- Often slow and clunky, but can be very powerful tool if used right (e.g. no one remembers how to filter/sort data in VBA, but this tool helps!)

Variables

- Variable types:
  - Integer, Double, String, Boolean, Date, Currency, and more
- We use the keyword Dim to declare a variable
  - Dim mystring As String
  - Dim dbl As Double
  - Dim num As Integer
  - Dim rng As Range
- We assign them with "="
  - mystring = "Hello"
  - num = 5
- However, to assign a range, we use:
  - Set rng = Range("A1:B3")

Note: All highlighted words in sample code are variables that you can name whatever you want.

Variables

- If the data type is not specified, the variable will be declared as a variant
- A variant can contain any kind of data, and the data type can change at any point
- Try to avoid these if possible (they require a lot of memory)

Arrays

- Arrays are created with parentheses to indicate size:
  - Dim myarray(5) as Integer
- The size can be changed:
  - ReDim myarray(10)
- But this will delete the data. To preserve the data inside, use:
  - ReDim Preserve myarray(15)
- Individual elements can be accessed and modified with parentheses:
  - myarray(0) = 5
- There's a lot more to be learned with arrays, but you can look online for more details. We'll work primarily with Workbook objects instead

If Statements

- Syntax

```
If condition Then
    [statements]
[ElseIf elseifcondition Then
    elseifstatements]
[Else
    elsestatements]
End If
```

Reminder: All highlighted words in sample code are statements that you can change to whatever you want.

All other code is part of the fixed syntax.

- If condition is true, runs statements. If condition is false but elseifcondition is true, runs elseifstatements. Otherwise, runs elsestatements.
- There can be as many ElseIfs as needed

While Loops

- Syntax

```
While condition
    statements
Wend
```

- Runs statements until condition evaluates to FALSE.
- Make sure that condition will not be TRUE forever, or you will have an endless while loop

For Loops

- Syntax

  For counter = start To end [Step increment]

  statements

  Next [counter]

- **Typically, counter is an Integer that we increment**
  - "For i = 0 to 3" **will run 4 times (i = 0, 1, 2, 3)**
  - "For i = 2 to 7 Step 2" **will run 3 times (i = 2, 4, 6)**
  - "For i = 5 to 0 Step -3" **will run 2 times (i = 5, 2)**

For Each Loops

- Syntax

```
For Each cell In range
    statements
Next
```

- A quick and simple way to loop through all the cells in a range
- cell and range must both be Range objects
- range should be initialized to some Range, cell need not be

Message Boxes

- **Syntax:** MsgBox(“Prompt”)

- You can replace “Prompt” with whatever you want to appear in a message box

- Can be very useful in debugging to display the value of variables

Word of Caution

! Once you run a macro, you can’t undo it !

Make sure to save your workbook before running a macro to avoid losing your work in case the macro doesn’t work as intended.

Questions?

05.

Demo Time!