EXCEL WORKSHOP III
INTRODUCTION TO VBA IN EXCEL
Background

- You are an actuary for Bruin Auto Liability Insurance, LLC, a Los Angeles based company insuring personal automobile casualty losses in Southern California.
- Prior to your team's analysis of rates and reserves, you notice that the policy data is not the way you want it.
- Each policy is only listed once, regardless of how many times it has been renewed.
- This is extremely problematic for your team, since they need to be able to identify the specific 1-year term they're looking at when conducting their analyses.
Background

- Your task is to update the way the company's data is stored.
- If a policy has been renewed twice (i.e. it was inforce 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).
Background

- Write a VBA macro to accomplish this task with the short excerpt of data.
- Then, once you've ensured the macro works correctly, run it on the full set of policy data, splitting the 13,018 policies into 30,395 rows.
Our Goal

<table>
<thead>
<tr>
<th>PolicyNumber</th>
<th>PolicyStartDate</th>
<th>NumberOfTerms</th>
<th>AnnualMilesDriven</th>
<th>County</th>
</tr>
</thead>
<tbody>
<tr>
<td>P100256</td>
<td>1/19/2014</td>
<td>5</td>
<td>24000</td>
<td>Riverside</td>
</tr>
<tr>
<td>P100256</td>
<td>1/19/2014</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P100256</td>
<td>1/19/2015</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P100256</td>
<td>1/19/2016</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P100256</td>
<td>1/19/2017</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P100256</td>
<td>1/19/2018</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 the Developer Tab

- Under File ➔ Options:
- Customize Ribbon
- Check “Developer”
The VBA Environment
VBA in Microsoft 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:
- Objects have properties and methods, the most important of which is:
  - Range(“A1”).Value
Getting started: Option Explicit

- This is always a good idea to put in the **declarations** at the top of your module
- This forces you to declare all your variables
Subroutines

- We enclose our code in the following:
  
  Sub Myroutine()

  End Sub

- This identifies our subroutine as being named “Myroutine()”

- We can call our subroutine in several ways:
Subroutines

Adding a button or some other tool

Clicking “Macros” in the Developer tab
Macro Recorder: The easiest way to “write” a macro

- Click “record macro”
- All your actions will be translated into code
- You can look at and modify this code to suit your purposes
- Often slow and clunky, but can be a very powerful tool if used right (e.g. no one remembers how to filter/sort data in VBA, but this tool helps!)
Variables

- We declare a variable with the Dim keyword:
  - Dim mystring As String
  - Dim dbl As Double
  - Dim num As Integer
  - Dim rng As Range

- We assign them with “=”
  - mystring = “string”
  - num = 5

- Except with ranges:
  - Set rng = Range(“A1:B3”)
Variables

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

- **Syntax:** `MsgBox(“Prompt”)`
- Can be very useful in debugging to display the value of variables
Another useful debugging tool: The Immediate Window

- Short lines of code can be run here
- You can also ask questions:
  - ?Worksheets.Count
- **If you use** `Debug.Print` **in your code, the output goes to this window**
If statements

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

- If condition is true, runs statements. If not, but elseifcondition is true, runs elseifstatements. Otherwise, runs elsestatements.

- There can be as many Elses 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
Arrays

- Arrays are created with parentheses to indicate size:
  - `Dim array(5) as Integer`

- The size can be changed:
  - `ReDim array(10)`

- But this will delete the data. To preserve the data inside, use
  - `ReDim Preserve array(15)`

- Individual elements can be accessed and modified with parentheses:
  - `array(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.
Functions

- Syntax:
  ```vba
  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!