overview of values and formulas

Values and formulas will focus on what to do with a Range after you have it. This typically falls into a couple of categories:

- Do some control logic based on the value of the Cell
- Apply some formatting to the cell
- Modify the formulas of the cell
- Manipulate the cell or its neighbors in order to produce a more useful result
- Transform the cells based on their content
- Do something specific to Excel with the `Range`: conditional formatting, data validation, comment, hyperlink etc.
- adding or deleting a `Range` or possibly just using one of the clear function

In addition to those basic tasks, also include:

- Working with Conditional Formatting
- Combining some more advanced topics like using the data in a `Range` to manipulate something about a `Chart`

TODO: run through bUTL and see what other category of things there are

chapter 2 - 1, introduction to manipulations

This chapter will focus on the actual work of using `Ranges` for some purpose. This chapter is predicated on the previous one which focused on obtained a `Range`. When talking of "using" a `Range`, the goal usually takes one of the following forms:

WHY BULLET

- Work through a spreadsheet of data, processing it from one format into another. This can be to pull data out, do calculations on a subset of data, change the formatting, aggregate data, summarize data into a new spreadsheet, etc. The options here are really endless, but the main idea is that you have an existing spreadsheet of data to do something with.

The next category of work is to process some small amount of data in place, typically to clean up data or convert it to some other form. A lot of this type of work is providing some functionality that would be great to have in Excel by default. This work also includes a lot of very specific types of functions that only make sense with your data. In that sense, these types of actions can be the quickest hitters; they are specific to your task and easy to program.

Another category of work is to run through an existing worksheet and perform some amount of checking on it. These checks do not necessarily need to modify the spreadsheet, they can be checking for formula errors, bad values, etc.

Another type of work that can be done is to modify the spreadsheet to make it easier to do work or to manage a workflow. These types of things are often implemented as events, but they can just be stray macros as well. When modifying the spreadsheet, you are often showing/hiding columns or worksheets. You can also be sorting `Worksheets`. You might be moving some number of `Worksheets` over from a "template" and setting up a common work environment. You may also be d

As we progress down this list, things are becoming increasingly complicated. At some point, the work involved will progress from a couple of simple tasks to a much more involved workflow. It's generally the nature of a complicated workflow that it is simply doing a long string of simple tasks ~~all at once~~ IN SERIES. In that sense, if you can learn these techniques, you can start to become comfortable combining them in more complicated fashions.

simple manipulations (one steppers)

This section will focus on simple manipulations. Simple manipulations generally take a two step process:
1. identify a `Range` to work with and then do something to that `Range`. In a lot of cases, the `Range` contains multiple cells and may be iterated through a cell at a time to apply the action.

2. These simple manipulations truly are simple. It includes things like:

- Change the value of a cell
- Change the value of a group of cells
- Change the formula of a cell
- Change the formatting of a cell
- Clear the formatting or value from a cell
- Return some piece of information about a cell

TODO: deal with these later

Name a group of cells

Add a hyperlink to the current cell

TODO: add a couple examples of the simple manipulations

slightly more complicated manipulations (the two steppers)

This section will on the so called "two steppers". I call them that because these manipulations typically involve two commands after identifying a `Range`. the first command is usually a logic or loop, and the second command is the actual work to be done. Two steppers are important because a large number of complicated tasks involve nesting and combining these two steps.

Some examples of two step manipulations includes;

- Run through a list of cells, if the text is numeric, convert to a number
- Run through a list of cells, if the cell is blank, fill with the value from above
- Run through some cells, check if the row is odd or even, and color the row from one of two colors
- Run through one list of cells, apply the formatting to the same cell in a different column

TODO: find some better examples for these as well

strategy #1, do something if

This strategy really is the core of all advanced VBA development. It's simple enough: "do something, if". The endless possibilities come from the choices for "do something" and the things that could be checked in the "if". There are a handful of common scenarios that are best covered by storing some utility code (e.g. convert to a number if numeric). Most of these two step solutions though are specific to the task at hand.

In this section, the goal is to show the general form of this strategy with a couple of examples.

TODO: add a couple examples of this

strategy #2, work through one Range and apply to another Range

This strategy comes up frequently when working through `Ranges` that are related somehow. The general idea is that you want to apply an action in one `Range` based on something about another `Range`. The simplest case of this is to move a value from `Range` to another. This simple case sometimes reduces to not much more than copying and pasting. Having said that, once you get past the simplest version of it, you will be doing something that copy and paste cannot handle.

TODO: add a couple examples of this

ARE THERE MORE TWO STEPPERS ?

things to change and check

This section will focus on the common properties that are checked and changed with these types of manipulations.

properties of the Range

The common properties of the Range to work with include:

- Value
- Text
- Formula
- Font
- Interior
- NumberFormat

TODO: add some examples of working with these

commonly used VBA functions

In addition to the properties of the Range, there are a handful of common VBA functions that come up when working with simple to moderate manipulations. These include:

- Split - split a string into an array based on a delimiter (the reverse of Join)
- Join - join an array into a string with a delimiter (this reverse of Split)
- Asc - determine the ASCII code for a character
- Chr - return a character for an ASCII code (the reverse of Asc)
- InStr - determine if a string is in another one (called Substring in other languages)
- Left, Mid, Right - grab parts of a string
- Trim - remove any whitespace from the start or end of a string
- Len - determine the length of a string
- UCase, LCase - used to force a string to upper or lower case
- UBound, LBound - determine the bounds of any array
- WorksheetFunction - get access to any Excel functions in VBA
- IsNumeric, IsEmpty - check if a number – TODO: add the others here
- CDbl, CLng, CBool, CDate - convert a value of one type to another – TODO: add any others

ADD Examples

FORMATS

- Replace - replace one string in another
- Application.Index, Application.Match - these are the VBA versions of the Excel functions
- Application.Transpose - convert a 1D array from vertical to horizontal and back
- Is Nothing - check if a reference has been set
- TypeName - check the type of an object (useful if working with `Variant`)
- RGB - useful way to build colors from known red, green, and blue values
- Count - common way to get the size of a group, used often to resize an input/output or to check logic

TODO: search through bUTL for other common functions

CategoricalColoring.md

REMOVE

```vba
Public Sub CategoricalColoring()


    '+Get User Input
    Dim targetRange As Range
    On Error GoTo errHandler
    Set targetRange = GetInputOrSelection("Select Range to Color")

    Dim coloredRange As Range
    Set coloredRange = GetInputOrSelection("Select Range with Colors")

    '+Do Magic
    Application.ScreenUpdating = False
    Dim targetCell As Range
    Dim foundRange As Variant

    For Each targetCell In targetRange
        foundRange = Application.Match(targetCell, coloredRange, 0)
        '+ Matches font style as well as interior color
        If IsNumeric(foundRange) Then
            targetCell.Font.FontStyle = coloredRange.Cells(foundRange).Font.FontStyle
            targetCell.Font.Color = coloredRange.Cells(foundRange).Font.Color
            '+Skip interior color if there is none
            If Not coloredRange.Cells(foundRange).Interior.ColorIndex = xlNone Then
```

```
                targetCell.Interior.Color = coloredRange.Cells(foundRange).
                   Interior.Color
            End If
        End If
    Next targetCell
    '+ If no fill, restore gridlines
    targetRange.Borders.LineStyle = xlNone
    Application.ScreenUpdating = True
    Exit Sub
errHandler:
    MsgBox "No Range Selected!"
    Application.ScreenUpdating = True
End Sub
```

ColorForUnique.md — Remove

```
Public Sub ColorForUnique()

    Dim dictKeysAndColors As New Scripting.Dictionary
    Dim dictColorsOnly As New Scripting.Dictionary

    Dim targetRange As Range

    On Error GoTo ColorForUnique_Error

    Set targetRange = GetInputOrSelection("Select column to color")
    Set targetRange = Intersect(targetRange, targetRange.Parent.UsedRange)

    'We can colorize the sorting column, or the entire row
    Dim shouldColorEntireRow As VbMsgBoxResult
    shouldColorEntireRow = MsgBox("Do you want to color the entire row?",
       vbYesNo)

    Application.ScreenUpdating = False

    Dim rowToColor As Range
    For Each rowToColor In targetRange.Rows

```

```
        'allow for a multi column key if initial range is multi-column
        'TODO: consider making this another prompt... might (?) want to color multi range based on single column key
        Dim keyString As String
        If rowToColor.Columns.Count > 1 Then
            keyString = Join(Application.Transpose(Application.Transpose(rowToColor.Value)), "||")
        Else
            keyString = rowToColor.Value
        End If

        'new value, need a color
        If Not dictKeysAndColors.Exists(keyString) Then
            Dim randomColor As Long
createNewColor:
            randomColor = RGB(Application.RandBetween(50, 255), _
                            Application.RandBetween(50, 255), Application.RandBetween(50, 255))
            If dictColorsOnly.Exists(randomColor) Then
                'ensure unique colors only
                GoTo createNewColor 'This is a sub-optimal way of performing this error check and loop
            End If

            dictKeysAndColors.Add keyString, randomColor
        End If

        If shouldColorEntireRow = vbYes Then
            rowToColor.EntireRow.Interior.Color = dictKeysAndColors(keyString)
        Else
            rowToColor.Interior.Color = dictKeysAndColors(keyString)
        End If
    Next rowToColor

    Application.ScreenUpdating = True

    On Error GoTo 0
    Exit Sub
```

```

ColorForUnique_Error:
    MsgBox "Select a valid range or fewer than 65650 unique entries."

End Sub
```

This colors cells the same that contain the same value. It alternates between two colors creating a banded effect for unique values

Colorize.md

```
Public Sub Colorize()

    Dim targetRange As Range
    On Error GoTo errHandler
    Set targetRange = GetInputOrSelection("Select range to color")
    Dim lastRow As Long
    lastRow = targetRange.Rows.Count
    Dim interiorColor As Long
    interiorColor = RGB(200, 200, 200)

    Dim sameColorForLikeValues As VbMsgBoxResult
    sameColorForLikeValues = MsgBox("Do you want to keep duplicate values the same color?", vbYesNo)

    If sameColorForLikeValues = vbNo Then

        Dim i As Long
        For i = 1 To lastRow
            If i Mod 2 = 0 Then
                targetRange.Rows(i).Interior.Color = interiorColor
            Else: targetRange.Rows(i).Interior.ColorIndex = xlNone
            End If
        Next
    End If


    If sameColorForLikeValues = vbYes Then
        Dim flipFlag As Boolean
        For i = 2 To lastRow
```

```
            If targetRange.Cells(i, 1) <> targetRange.Cells(i - 1, 1) Then
                flipFlag = Not flipFlag
            If flipFlag Then
                targetRange.Rows(i).Interior.Color = interiorColor
            Else: targetRange.Rows(i).Interior.ColorIndex = xlNone
            End If
        Next
    End If
    Exit Sub
errHandler:
    MsgBox "No Range Selected!"
End Sub
```

CombineCells.md

```
Public Sub CombineCells()

    'collect all user data up front
    Dim inputRange As Range
    On Error GoTo errHandler
    Set inputRange = GetInputOrSelection("Select the range of cells to
        combine")

    Dim delimiter As String
    delimiter = Application.InputBox("Delimiter:")
    If delimiter = "" Or delimiter = "False" Then GoTo delimiterError

    Dim outputRange As Range
    Set outputRange = GetInputOrSelection("Select the output range")

    'Check the size of input and adjust output
    Dim numberOfColumns As Long
    numberOfColumns = inputRange.Columns.Count

    Dim numberOfRows As Long
    numberOfRows = inputRange.Rows.Count

    outputRange = outputRange.Resize(numberOfRows, 1)

    'Read input rows into a single string
    Dim outputString As String
    Dim i As Long
    For i = 1 To numberOfRows
        outputString = vbNullString
        Dim j As Long
        For j = 1 To numberOfColumns
            outputString = outputString & delimiter & inputRange(i, j)
        Next
        'Get rid of the first character (delimiter)
        outputString = Right(outputString, Len(outputString) - 1)
        'Print it!
        outputRange(i, 1) = outputString
    Next
    Exit Sub
delimiterError:
    MsgBox "No Delimiter Selected!"
    Exit Sub
errHandler:
    MsgBox "No Range Selected!"
End Sub
```

ConvertToNumber.md

```
Public Sub ConvertToNumber()

    Dim targetCell As Range
    Dim targetSelection As Range

    Set targetSelection = Selection

    Application.ScreenUpdating = False
    Application.Calculation = xlCalculationManual

    For Each targetCell In Intersect(targetSelection, ActiveSheet.UsedRange)
        If Not IsEmpty(targetCell.Value) And IsNumeric(targetCell.Value) Then
             targetCell.Value = CDbl(targetCell.Value)
    Next targetCell

    Application.ScreenUpdating = True
    Application.Calculation = xlCalculationAutomatic

End Sub
```

CopyTranspose.md REMOVE

```
Public Sub CopyTranspose()

    'If user cancels a range input, we need to handle it when it occurs
    On Error GoTo errCancel
    Dim selectedRange As Range

    Set selectedRange = GetInputOrSelection("Select your range")

    Dim outputRange As Range
    'Need to handle the error of selecting more than one cell
    Set outputRange = GetInputOrSelection("Select the output corner")

    Application.ScreenUpdating = False
    Application.EnableEvents = False
    Application.Calculation = xlCalculationManual

    Dim startingCornerCell As Range
    Set startingCornerCell = selectedRange.Cells(1, 1)

    Dim startingCellRow As Long
    startingCellRow = startingCornerCell.Row
    Dim startingCellColumn As Long
    startingCellColumn = startingCornerCell.Column

    Dim outputRow As Long
    Dim outputColumn As Long
    outputRow = outputRange.Row
    outputColumn = outputRange.Column

    Dim targetCell As Range

    'We check for the intersection to ensure we don't overwrite any of the original data
    'There's probably a better way to do this than For Each
    For Each targetCell In selectedRange
        If Not Intersect(selectedRange, Cells(outputRow + targetCell.Column - startingCellColumn, outputColumn + targetCell.Row - startingCellRow)) Is Nothing Then
            MsgBox "Your destination intersects with your data"
            Exit Sub
        End If
    Next targetCell

    For Each targetCell In selectedRange
        ActiveSheet.Cells(outputRow + targetCell.Column - startingCellColumn, outputColumn + targetCell.Row - startingCellRow).Formula = targetCell.Formula
    Next targetCell

errCancel:
    Application.ScreenUpdating = True
    Application.EnableEvents = True
    Application.Calculation = xlCalculationAutomatic
    Application.Calculate
End Sub
```

CreateConditionalsForFormatting.md – KEEP ELSEWHERE?

```
Public Sub CreateConditionalsForFormatting()

    On Error GoTo errHandler
    Dim inputRange As Range
    Set inputRange = GetInputOrSelection("Select the range of cells to convert")
    'add these in as powers of 3, starting at 1 = 10^0
    Const ARRAY_MARKERS As String = " ,k,M,B,T,Q"
    Dim arrMarkers As Variant
    arrMarkers = Split(ARRAY_MARKERS, ",")

    Dim i As Long
    For i = UBound(arrMarkers) To 0 Step -1

        With inputRange.FormatConditions.Add(xlCellValue, xlGreaterEqual, 10 ^ (3 * i))
            .NumberFormat = "0.0" & Application.WorksheetFunction.Rept(",", i) & " "" " & arrMarkers(i) & """"
        End With

    Next
    Exit Sub
errHandler:
    MsgBox "No Range Selected!"
End Sub
```

ExtendArrayFormulaDown.md Remove

```
Public Sub ExtendArrayFormulaDown()

    Dim startingRange As Range
    Dim targetArea As Range


    Application.ScreenUpdating = False

    Set startingRange = Selection

    For Each targetArea In startingRange.Areas

        Dim targetCell As Range
        For Each targetCell In targetArea.Cells

            If targetCell.HasArray Then

                Dim formulaString As String
                formulaString = targetCell.FormulaArray

                Dim startOfArray As Range
                Dim endOfArray As Range

                Set startOfArray = targetCell.CurrentArray.Cells(1, 1)
                Set endOfArray = startOfArray.Offset(0, -1).End(xlDown).
                   Offset(0, 1)

                targetCell.CurrentArray.Formula = vbNullString

                Range(startOfArray, endOfArray).FormulaArray = formulaString

            End If

        Next targetCell
    Next targetArea


    'Find the range of the new array formula
    'Save current formula and clear it out
    'Apply the formula to the new range
    Application.ScreenUpdating = True
End Sub
```

MakeHyperlinks.md

```
Public Sub MakeHyperlinks()

    '+Changed to inputbox
    On Error GoTo errHandler
    Dim targetRange As Range
    Set targetRange = GetInputOrSelection("Select the range of cells to
       convert to hyperlink")

    'TODO: choose a better variable name
    Dim targetCell As Range
    For Each targetCell In targetRange
        ActiveSheet.Hyperlinks.Add Anchor:=targetCell, Address:=targetCell
    Next targetCell
    Exit Sub
errHandler:
    MsgBox "No Range Selected!"
End Sub
```

OutputColors.md

[illegible] Remove

```
Public Sub OutputColors()

    Const MINIMUM_INTEGER As Long = 1
    Const MAXIMUM_INTEGER As Long = 10
    Dim i As Long
    For i = MINIMUM_INTEGER To MAXIMUM_INTEGER
        ActiveCell.Offset(i).Interior.Color = Chart_GetColor(i)
    Next i

End Sub
```

SelectedToValue.md

```
Public Sub SelectedToValue()

    Dim targetRange As Range
    On Error GoTo errHandler
    Set targetRange = GetInputOrSelection("Select the formulas you'd like to
        convert to static values")

    Dim targetCell As Range
    Dim targetCellValue As String
    For Each targetCell In targetRange
        targetCellValue = targetCell.Value
        targetCell.Clear
        targetCell = targetCellValue
    Next targetCell
    Exit Sub
```

```
errHandler:
    MsgBox "No selection made!"
End Sub
```

Selection_ColorWithHex.md

```
Public Sub Selection_ColorWithHex()

    Dim targetCell As Range
    Dim targetRange As Range
    On Error GoTo errHandler
    Set targetRange = GetInputOrSelection("Select the range of cells to color")
    For Each targetCell In targetRange
        targetCell.Interior.Color = RGB( _
                                    WorksheetFunction.Hex2Dec(Mid(targetCell.Value, 2, 2)), _
                                    WorksheetFunction.Hex2Dec(Mid(targetCell.Value, 4, 2)), _
                                    WorksheetFunction.Hex2Dec(Mid(targetCell.Value, 6, 2)))

    Next targetCell
    Exit Sub
errHandler:
    MsgBox "No selection made!"
End Sub
```

SplitAndKeep.md

```
Public Sub SplitAndKeep()

    On Error GoTo SplitAndKeep_Error

    Dim rangeToSplit As Range
    Set rangeToSplit = GetInputOrSelection("Select range to split")
```

```vba

    If rangeToSplit Is Nothing Then
        Exit Sub
    End If

    Dim delimiter As Variant
    delimiter = InputBox("What delimiter to split on?")
    'StrPtr is undocumented, perhaps add documentation or change function
    If StrPtr(delimiter) = 0 Then
        Exit Sub
    End If

    Dim itemToKeep As Variant
    'Perhaps inform user to input the sequence number of the item to keep
    itemToKeep = InputBox("Which item to keep? (This is 0-indexed)")

    If StrPtr(itemToKeep) = 0 Then
        Exit Sub
    End If

    Dim targetCell As Range
    For Each targetCell In Intersect(rangeToSplit, rangeToSplit.Parent.
       UsedRange)

        Dim delimitedCellParts As Variant
        delimitedCellParts = Split(targetCell, delimiter)

        If UBound(delimitedCellParts) >= itemToKeep Then
            targetCell.Value = delimitedCellParts(itemToKeep)
        End If

    Next targetCell

    On Error GoTo 0
    Exit Sub

SplitAndKeep_Error:
    MsgBox "Check that a valid Range is selected and that a number was
       entered for which item to keep."
```

```
End Sub
```

SplitIntoColumns.md

```
Public Sub SplitIntoColumns()

    Dim inputRange As Range

    Set inputRange = GetInputOrSelection("Select the range of cells to split"
        )

    Dim targetCell As Range

    Dim delimiter As String
    delimiter = Application.InputBox("What is the delimiter?", , ",",
        vbOKCancel)
    If delimiter = "" Or delimiter = "False" Then GoTo errHandler
    For Each targetCell In inputRange

        Dim targetCellParts As Variant
        targetCellParts = Split(targetCell, delimiter)

        Dim targetPart As Variant
        For Each targetPart In targetCellParts

            Set targetCell = targetCell.Offset(, 1)
            targetCell = targetPart

        Next targetPart

    Next targetCell
    Exit Sub
errHandler:
    MsgBox "No Delimiter Defined!"
End Sub
```

SplitIntoRows.md

```
Public Sub SplitIntoRows()

    Dim outputRange As Range

    Dim inputRange As Range
    Set inputRange = Selection

    Set outputRange = GetInputOrSelection("Select the output corner")

    Dim targetPart As Variant
    Dim offsetCounter As Long
    offsetCounter = 0
    Dim targetCell As Range

    For Each targetCell In inputRange.SpecialCells(xlCellTypeVisible)
        Dim targetParts As Variant
        targetParts = Split(targetCell, vbLf)

        For Each targetPart In targetParts
            outputRange.Offset(offsetCounter) = targetPart

            offsetCounter = offsetCounter + 1
        Next targetPart
    Next targetCell
End Sub
```

TrimSelection.md

```
Public Sub TrimSelection()

    Dim rangeToTrim As Range
    On Error GoTo errHandler
    Set rangeToTrim = GetInputOrSelection("Select the formulas you'd like to
       convert to static values")

    'disable calcs to speed up
    Application.ScreenUpdating = False
    Application.EnableEvents = False
    Application.Calculation = xlCalculationManual

    'force to only consider used range
    Set rangeToTrim = Intersect(rangeToTrim, rangeToTrim.Parent.UsedRange)

    Dim targetCell As Range
    For Each targetCell In rangeToTrim

        'only change if needed
        Dim temporaryTrimHolder As Variant
        temporaryTrimHolder = Trim(targetCell.Value)

        'added support for char 160
        'TODO add more characters to remove
        temporaryTrimHolder = Replace(temporaryTrimHolder, chr(160),
           vbNullString)

        If temporaryTrimHolder <> targetCell.Value Then targetCell.Value =
           temporaryTrimHolder

    Next targetCell

    Application.Calculation = xlCalculationAutomatic
    Application.EnableEvents = True
    Application.ScreenUpdating = True

    Exit Sub
errHandler:
    MsgBox "No Delimiter Defined!"
    Application.ScreenUpdating = False
    Application.EnableEvents = False
    Application.Calculation = xlCalculationManual
End Sub
```