'-----------------------------------------------------------------------------------------------------------------------------------

'Last Update: Oct 3, 2013

'-----------------------------------------------------------------------------------------------------------------------------------

Option Explicit

Option Base 1

Option Compare Binary

'Option Compare Text

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8: String Functions

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'A string is often times just a thing that we print out or pass from one part of our program to another without regard for its

'contents. Other times, however, we need to know what's inside. We might need to verify its contents, modify it in some way, or

'extract a specific piece of information from it. When dealing with options quotes, for example, we sometimes need to parse out

'the stock symbol, the expiration month and the strike price, which are all strung together in one long option symbol. The String

'functions summarized below. To see the behavior of each sub open the Immediate Window, click inside each Sub, and press F8:

'Please notice:

' 1) To use " (same as Chr(34) see worksheet "WSHEET\_STRING\_CHARACTERS") as string in VBA you have to use """" or Chr(34)

' 2) Use & \_ to continue your code on a new line

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_CONVERT()

Dim TEXT\_STR As String

Debug.Print "Convert to string, where expression is a string, a string literal, a string constant"

Debug.Print "TEXT\_STR = CStr(3.155)"

TEXT\_STR = CStr(3.155)

Debug.Print TEXT\_STR

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_LOWER\_UPPER\_CASE()

Dim TEXT\_STR As String

Debug.Print "TEXT\_STR = " & """" & "Converting strings to upper/lower case" & """"

TEXT\_STR = "Converting strings to upper/lower case"

Debug.Print "Converting strings to uppercase: UCase(TEXT\_STR) = " & UCase(TEXT\_STR)

Debug.Print "Converting strings to lowercase: LCase(TEXT\_STR) = " & LCase(TEXT\_STR)

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.1: Stringing more than two words together

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'It does seem strange that the Excel function Concatenate(), provides no additional functionality

'over using the & operator. The RNG\_TEXT\_STRING\_FUNC function is a simple User Defined Function (UDF)

'which provides the following additional features:

'1. The cells to be concatenated are defined as a single row range, rather than DELIM\_STR cells.

'2. An optional separator character may be specified.

'3. Numbers, including dates, times and currency are concatenated as displayed

Function RNG\_TEXT\_STRING\_FUNC(ByRef DATA\_RNG As Excel.Range, \_

Optional ByVal DELIM\_STR As String = " ") As String

'Concatenates a range of strings and/or values, with an optional separator string.

'DATA\_RNG: must be a range with at least two columns. Rows after the first are ignored.

'All blank cells are ignored

'Default separator is one space; for no separator enter "" as the delimiter string.

'To display correctly, cells containing values must not be formatted as general.

Dim i As Long

Dim j As Long

Dim NSIZE As Long 'NumCells

Dim TEMP\_MATRIX() As Variant

On Error GoTo ERROR\_LABEL

j = 0

NSIZE = DATA\_RNG.Columns.Count

If NSIZE = 1 Then

NSIZE = DATA\_RNG.Rows.Count

j = 1

End If

ReDim TEMP\_MATRIX(1 To 2, 1 To NSIZE) 'We use this matrix (array) to save the values and formats in

'cells A3:D8. We will cover arrays in Lesson 4

If j = 0 Then

For i = 1 To NSIZE

TEMP\_MATRIX(1, i) = DATA\_RNG(1, i).Value

TEMP\_MATRIX(2, i) = DATA\_RNG(1, i).NumberFormat

Next i

Else

For i = 1 To NSIZE

TEMP\_MATRIX(1, i) = DATA\_RNG(i, 1).Value

TEMP\_MATRIX(2, i) = DATA\_RNG(i, 1).NumberFormat

Next i

End If

'See how the function name "RNG\_TEXT\_STRING\_FUNC" can be used as a string variable for its return value.

'Remember how we define the function RNG\_TEXT\_STRING\_FUNC: ... As String

If TEMP\_MATRIX(1, 1) <> "" Then

RNG\_TEXT\_STRING\_FUNC = Format(TEMP\_MATRIX(1, 1), TEMP\_MATRIX(2, 1))

End If

For i = 2 To NSIZE

If TEMP\_MATRIX(1, i) <> "" Then

RNG\_TEXT\_STRING\_FUNC = RNG\_TEXT\_STRING\_FUNC & DELIM\_STR & Format(TEMP\_MATRIX(1, i), TEMP\_MATRIX(2, i))

End If

Next i

Exit Function

ERROR\_LABEL:

RNG\_TEXT\_STRING\_FUNC = Err.Number

End Function

Sub TEST\_RNG\_TEXT\_STRING\_FUNC()

Dim i As Long

Dim j As Long

Dim k As Long

Dim NSIZE As Long

Dim ii As Long

Dim jj As Long

Dim FORMATS\_STR(1 To 6) As String

Dim FORMULAS\_STR As String

Dim TEMP\_ARR As Variant

Dim FORMAT\_RNG As Range

Dim DST\_RNG As Range

'-------------------------------------------------------------------------------------------------------

'Setting up the tables in the WSHEET\_TEST

'-------------------------------------------------------------------------------------------------------

With Worksheets("WSHEET\_TEST")

.Select

With .Cells

.Clear

.ColumnWidth = 10

End With

End With

FORMATS\_STR(1) = "General"

FORMATS\_STR(2) = "0"

FORMATS\_STR(3) = "0.000"

FORMATS\_STR(4) = "0.00E+00"

FORMATS\_STR(5) = "$#,##0.00"

FORMATS\_STR(6) = "m/d/yyyy"

FORMULAS\_STR = "abc|1|=PI()|=10000000000|1|40238|def|2|=+A5+1|=+A6\*$A$6| |=+A8+1|ghi|c|=+B5+1|=+B6\*$A$6|" & \_

"3|=+B8+1|jkl|4|=+C5+1|=+C6\*$A$6|5| |=RNG\_TEXT\_STRING\_FUNC(A3:E3)|=RNG\_TEXT\_STRING\_FUNC(A4:E4, " & """" & \_

"," & """" & ")|=RNG\_TEXT\_STRING\_FUNC(A5:E5," & """" & ", " & """" & ")|=RNG\_TEXT\_STRING\_FUNC(A6:E6," & \_

"""" & "," & """" & ")|=RNG\_TEXT\_STRING\_FUNC(A7:E7)|=RNG\_TEXT\_STRING\_FUNC(A8:E8," & """" & "; " & """" & \_

")|=CONCATENATE(A3,B3,C3,D3)|=CONCATENATE(A4,B4,C4,D4)|=CONCATENATE(A5,B5,C5,D5)|=CONCATENATE(A6,B6,C6,D6)|" & \_

"=CONCATENATE(A7,B7,C7,D7)|=CONCATENATE(A8,B8,C8,D8)|=PI()|=+A11+1|=+A12+1|=+A13+1|" & \_

"=RNG\_TEXT\_STRING\_FUNC(A11:A14," & """" & ", " & """" & ")|"

j = Len(FORMULAS\_STR)

k = 0

For i = 1 To j

If Mid(FORMULAS\_STR, i, 1) = "|" Then: k = k + 1

Next i

'Debug.Print k 'No. Formulas

Set DST\_RNG = [A1] 'Same as Range("A1")

With DST\_RNG

.Value = "RNG\_TEXT\_STRING\_FUNC Function"

Set FORMAT\_RNG = Range(.Cells(3, 1), .Cells(8, 4)): GoSub FORMAT\_LINE

ii = 1

For j = 1 To 4

For i = 1 To 6

jj = InStr(ii, FORMULAS\_STR, "|") 'see Lesson 3.8.4: String Position

With FORMAT\_RNG.Cells(i, j)

.Formula = Mid(FORMULAS\_STR, ii, jj - ii)

.NumberFormat = FORMATS\_STR(i)

End With

ii = jj + 1

Next i

Next j

With FORMAT\_RNG

.HorizontalAlignment = xlCenter

.VerticalAlignment = xlBottom

.WrapText = False

.Orientation = 0

.AddIndent = False

.IndentLevel = 0

.ShrinkToFit = False

.ReadingOrder = xlContext

.MergeCells = False

With .Font

.Color = -16776961

.TintAndShade = 0

End With

End With

End With

Set DST\_RNG = [F1]

With DST\_RNG

.Value = "RNG\_TEXT\_STRING\_FUNC Function"

Set FORMAT\_RNG = Range(.Cells(3, 1), .Cells(8, 1)): GoSub FORMAT\_LINE

For i = 1 To 6

jj = InStr(ii, FORMULAS\_STR, "|") 'see Lesson 3.8.4: String Position

FORMAT\_RNG.Cells(i, 1).Formula = Mid(FORMULAS\_STR, ii, jj - ii)

ii = jj + 1

Next i

With FORMAT\_RNG

.HorizontalAlignment = xlLeft

.VerticalAlignment = xlBottom

.WrapText = False

.Orientation = 0

.AddIndent = False

.IndentLevel = 0

.ShrinkToFit = False

.ReadingOrder = xlContext

.MergeCells = False

With .Font

.ThemeColor = xlThemeColorLight1

.TintAndShade = 0

End With

.EntireColumn.AutoFit

End With

End With

Set DST\_RNG = [H1]

With DST\_RNG

.Value = "Output from Concatenate() function"

Set FORMAT\_RNG = Range(.Cells(3, 1), .Cells(8, 1)): GoSub FORMAT\_LINE

For i = 1 To 6

jj = InStr(ii, FORMULAS\_STR, "|")

FORMAT\_RNG.Cells(i, 1).Formula = Mid(FORMULAS\_STR, ii, jj - ii)

ii = jj + 1

Next i

With FORMAT\_RNG

.HorizontalAlignment = xlLeft

.VerticalAlignment = xlBottom

.WrapText = False

.Orientation = 0

.AddIndent = False

.IndentLevel = 0

.ShrinkToFit = False

.ReadingOrder = xlContext

.MergeCells = False

With .Font

.ThemeColor = xlThemeColorLight1

.TintAndShade = 0

End With

.EntireColumn.AutoFit

End With

End With

'-------------------------------------------------------------------------------------------------------

Set DST\_RNG = [A10]

With DST\_RNG

.Value = "Output from text in a column"

Set FORMAT\_RNG = Range(.Cells(2, 1), .Cells(5, 1)): GoSub FORMAT\_LINE

For i = 1 To 4

jj = InStr(ii, FORMULAS\_STR, "|")

With FORMAT\_RNG.Cells(i, 1)

.Formula = Mid(FORMULAS\_STR, ii, jj - ii)

.NumberFormat = FORMATS\_STR(3)

End With

ii = jj + 1

Next i

With FORMAT\_RNG

.HorizontalAlignment = xlCenter

.VerticalAlignment = xlBottom

.WrapText = False

.Orientation = 0

.AddIndent = False

.IndentLevel = 0

.ShrinkToFit = False

.ReadingOrder = xlContext

.MergeCells = False

With .Font

.Color = -16776961

.TintAndShade = 0

End With

End With

Set FORMAT\_RNG = .Cells(2, 3)

jj = InStr(ii, FORMULAS\_STR, "|")

FORMAT\_RNG.Formula = Mid(FORMULAS\_STR, ii, jj - ii)

End With

'--------------------------------------------------------------------------------------------------------------

Exit Sub

'--------------------------------------------------------------------------------------------------------------

FORMAT\_LINE:

'--------------------------------------------------------------------------------------------------------------

On Error Resume Next 'May not work in Mac

With FORMAT\_RNG 'Format borders

.Borders(xlDiagonalDown).LineStyle = xlNone

.Borders(xlDiagonalUp).LineStyle = xlNone

With .Borders(xlEdgeLeft)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeTop)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeBottom)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlEdgeRight)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlInsideVertical)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

With .Borders(xlInsideHorizontal)

.LineStyle = xlContinuous

.ColorIndex = 0

.TintAndShade = 0

.Weight = xlThin

End With

End With

On Error GoTo 0

Return

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.2: String Length

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_LENGTH()

Dim i, j, k As Long

Dim START\_TIMER1, START\_TIMER2 As Single

Dim END\_TIMER1, END\_TIMER2 As Single

Dim nLOOPS As Long

Dim CHR\_STR As String

Dim TEXT\_STR As String

'Is Len an efficient method for checking the length of a string. Lets write a simple non-scientific procedure to compare

'Len with a loop procedure to see which runs faster.

Debug.Print "Methods for checking the length of a string"

nLOOPS = 100000

'The following procedure uses two loops:

'1. a loop using Len to calculate string length

'2. a Do loop to measure the string's length

TEXT\_STR = "Hello World!"

' 1. use Len

Debug.Print "Use Len(" & """" & "Hello World!" & """" & ") = " & Len(TEXT\_STR)

START\_TIMER1 = Timer

For i = 1 To nLOOPS

j = Len(TEXT\_STR)

Next i

END\_TIMER1 = Timer

' 2. Count Loop

START\_TIMER2 = Timer

For i = 1 To nLOOPS 'Lets be creative to avoid using Len as the loop control

j = 0: k = 0

Do 'here we use a Mid function call, an If-Then statement and a string comparison,

'which at least partially explains why it is slower.

k = k + 1

CHR\_STR = Mid(TEXT\_STR, k, 1) 'Function Mid(String, Start As Long, [Length])

'Mid returns a string containing a specified number of characters from a string.

'http://msdn.microsoft.com/en-us/library/05e63829(v=vs.90).aspx

If CHR\_STR <> "" Then j = j + 1

Loop While CHR\_STR <> ""

Next i

Debug.Print "Use Loop(" & """" & "Hello World!" & """" & ") = " & j

END\_TIMER2 = Timer

TEXT\_STR = "Number of iterations: " & nLOOPS & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Len: " & Format(END\_TIMER1 - START\_TIMER1, "#.###") & " seconds" & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Loop: " & Format(END\_TIMER2 - START\_TIMER2, "#.###") & " seconds" & vbCrLf

Debug.Print TEXT\_STR

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.3: String Remove Spaces

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_REMOVE\_SPACES()

Dim TEXT\_STR As String

Debug.Print "Remove Spaces from Strings"

TEXT\_STR = "Hello World"

Debug.Print "TEXT\_STR = " & """" & "Hello World" & """"

Debug.Print "Simple way to remove spaces from any given input string without looping: REMOVE\_SPACES\_FUNC(TEXT\_STR) = " & \_

REMOVE\_SPACES\_FUNC(TEXT\_STR)

Debug.Print "Remove extra spaces from any given input string with looping: REMOVE\_EXTRA\_SPACES\_FUNC(TEXT\_STR) = " & \_

REMOVE\_EXTRA\_SPACES\_FUNC(TEXT\_STR)

End Sub

'remove spaces from any given input string without looping

Function REMOVE\_SPACES\_FUNC(ByVal TEXT\_STR As String) As String

REMOVE\_SPACES\_FUNC = Replace(TEXT\_STR, " ", "") 'For further reference see: http://www.fontstuff.com/vba/vbatut05.htm#removespaces

End Function

'Note that REMOVE\_SPACES\_FUNC function will blindly remove all spaces from the input string, which is different from removing

'excess spaces. Take a look at this great article: http://msdn.microsoft.com/en-us/library/aa201315%28v=office.11%29.aspx

Function REMOVE\_EXTRA\_SPACES\_FUNC(ByVal TEMP\_STR As String) 'DATA\_STR

Dim i As Long

'Dim TEMP\_STR As String

On Error GoTo ERROR\_LABEL

'TEMP\_STR = DATA\_STR

i = InStr(1, TEMP\_STR, Space(2), 0)

Do Until i = 0

TEMP\_STR = Replace(TEMP\_STR, Space(2), Space(1), 1, -1, 0)

i = InStr(1, TEMP\_STR, Space(2), 0) 'see Lesson 3.8.4: String Position

Loop

REMOVE\_EXTRA\_SPACES\_FUNC = TEMP\_STR

Exit Function

ERROR\_LABEL:

REMOVE\_EXTRA\_SPACES\_FUNC = Err.Number

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.4: String Position

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_POSITION()

Dim TEXT\_STR As String

Debug.Print "Find the position of a sub-string within another string"

'InStr VBA function can be used when you would like to find the position of a sub-string within another string. The format

'of the InStr function is:

'INSTR(position\_to\_start, string\_to\_search, string\_to\_find,comparison\_type)

'Where:

'position\_to\_start = The position where you would like to function to begin the comparison (optional)

'string\_to\_search = The string which is being searched

'string\_to\_find = The string or sub-string which needs to be located in the above string

'comparison\_type = Specifies whether the comparison should be carried out as a binary (byte level, case sensitive) or

'text (byte level, case insensitive) (optional)

'The function returns the position of the string\_to\_find within the string\_to\_search starting from the first position. If it

'is not able to locate the sub-string, it will return 0. If any of the strings is NULL, it returns a NULL. If the string\_to\_find

'is a zero length, the function would return the position\_to\_start numerical value.

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "InStr(5, " & """" & "This is a string" & """" & ", " & """" & "string" & """" & \_

") would give the result as 11 (the given sub-string starts at position 11)."

Debug.Print InStr(5, "This is a string", "string")

Debug.Print "InStr(5, " & """" & "This is a string" & """" & ", " & """" & "String" & """" & \_

") would give the result as 0 (the given substring could not be located)."

Debug.Print InStr(5, "This is a string", "String")

Debug.Print "InStr(5, " & """" & "This is a string" & """" & ", " & """" & "" & """" & \_

") would give the result as 5 (the given substring is of 0 length and hence return the number specified as the position\_to\_start)."

Debug.Print InStr(5, "This is a string", "")

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.5: String Find

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_FIND\_EXTRACT()

Dim TEXT\_STR As String

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

TEXT\_STR = "How long is a piece of string?"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "TEXT\_STR = " & """" & "How long is a piece of string?" & Chr(34) '""""

Debug.Print "Return the first word from a string: LEFTW\_FUNC(TEXT\_STR) = " & LEFTW\_FUNC(TEXT\_STR)

Debug.Print "Return the first three words from a string: LEFTW\_FUNC(TEXT\_STR, 3) = " & LEFTW\_FUNC(TEXT\_STR, 3)

Debug.Print "Return the first word from a comma delimited string: LEFTW\_FUNC(TEXT\_STR," & """" & \_

"," & """" & ") = " & LEFTW\_FUNC(TEXT\_STR, , ",")

Debug.Print "Return the last word from a string: RIGHTW\_FUNC(TEXT\_STR) = " & RIGHTW\_FUNC(TEXT\_STR)

Debug.Print "Return the last three words from a string: RIGHTW\_FUNC(TEXT\_STR, 3) = " & RIGHTW\_FUNC(TEXT\_STR, 3)

Debug.Print "Return the second word from a string: MIDW\_FUNC(TEXT\_STR, 2) = " & MIDW\_FUNC(TEXT\_STR, 2)

Debug.Print "Return the second to fith words from a string: MIDW\_FUNC(TEXT\_STR, 2, 4) = " & MIDW\_FUNC(TEXT\_STR, 2, 4)

Debug.Print "Find from the left hand end (VBA function): FIND\_REV\_FUNC(" & """" & "o" & """" & ", TEXT\_STR) = " & \_

FIND\_REV\_FUNC("o", TEXT\_STR)

Debug.Print "Find from the left hand end (Excel Built-In function): WorksheetFunction.Find(" & """" & "o" & """" & \_

", TEXT\_STR, 1) = " & WorksheetFunction.Find("o", TEXT\_STR, 1)

Debug.Print "Find from the right, starting at character 20: FIND\_REV\_FUNC(" & """" & "o" & """" & ", TEXT\_STR, 20) = " & \_

FIND\_REV\_FUNC("o", TEXT\_STR, 20)

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

TEXT\_STR = "5 squared = 25"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "TEXT\_STR = " & """" & "5 squared = 25" & """"

Debug.Print "Extract a value from the left end of a string: LEFT\_VAL\_FUNC(TEXT\_STR) = " & LEFT\_VAL\_FUNC(TEXT\_STR)

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

TEXT\_STR = "Pi = 3.1415"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "TEXT\_STR = " & """" & "Pi = 3.1415" & """"

Debug.Print "Extract a value from the right end of a string: RIGHT\_VAL\_FUNC(TEXT\_STR) = " & RIGHT\_VAL\_FUNC(TEXT\_STR)

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

TEXT\_STR = "Hello World"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "TEXT\_STR = " & """" & "Hello World" & """"

Debug.Print "Reverse a string: REVERSE\_FUNC(TEXT\_STR) = " & REVERSE\_FUNC(TEXT\_STR)

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

'Note: LEFTW\_FUNC(), RIGHTW\_FUNC(), MIDW\_FUNC(), LEFT\_VAL\_FUNC(), and RIGHT\_VAL\_FUNC() all have

'an optional delimiter as the final argument. Default delimiter is a space, remember LEFTW\_FUNC() example

'3 for a comma delimiter. See worksheet "WSHEET\_STRING\_FUNCTIONS"

End Sub

'For the Categories and Descriptions of the following functions

'see http://newtonexcelbach.wordpress.com/2009/05/21/adding-function-categories-and-descriptions/

'LEFTW\_FUNC: Return the first word from a string

Function LEFTW\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal NO\_WORDS As Long = 1, \_

Optional ByVal DELIM\_STR As String = " ") As String

Dim i As Long

Dim j As Long 'SpacePos

'LEFTW\_FUNC: Return words from the start of a string

'TEXT\_STR: String from which the words are returned

'NO\_WORDS: Number of words to return, Optional, default 1

'DELIM\_STR: Character separating words, Optional, default space

TEXT\_STR = Trim(TEXT\_STR)

j = 0

For i = 1 To NO\_WORDS

j = InStr(j + 1, TEXT\_STR, DELIM\_STR)

If NO\_WORDS > 0 And j = 0 Then j = Len(TEXT\_STR)

Next i

LEFTW\_FUNC = Left(TEXT\_STR, j - 1)

End Function

'RIGHTW\_FUNC: Return the last word from a string

Function RIGHTW\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal NO\_WORDS As Long = 1, \_

Optional ByVal DELIM\_STR As String = " ") As String

Dim i As Long

Dim j As Long 'SpacePos

Dim k As Long 'StrLen

'RIGHTW\_FUNC: Return words from the end of a string

'TEXT\_STR: String from which the words are returned

'NO\_WORDS: Number of words to return, Optional, default 1

'DELIM\_STR: Character separating words, Optional, default space

TEXT\_STR = Trim(TEXT\_STR)

k = Len(TEXT\_STR): j = Len(TEXT\_STR) + 1

For i = 1 To NO\_WORDS

j = InStrRev(TEXT\_STR, DELIM\_STR, j - 1)

'InStrRev: Returns the position of the first occurrence of one string within another, starting from the right side of the string.

'For further reference see: http://msdn.microsoft.com/en-us/library/t2ekk41a(v=vs.90).aspx

Next i

RIGHTW\_FUNC = Right(TEXT\_STR, k - j)

End Function

'MIDW\_FUNC: Return the second word from a string

Function MIDW\_FUNC(ByVal TEXT\_STR As String, \_

ByVal INDEX\_NO As Long, \_

Optional ByVal NO\_WORDS As Long = 1, \_

Optional ByVal DELIM\_STR As String = " ") As String

Dim i As Long

Dim RESULT\_ARR As Variant

'MIDW\_FUNC: Return words from the middle of a string

'TEXT\_STR: String from which the words are returned

'INDEX\_NO: Index of first word to return, base 1

'NO\_WORDS: Number of words to return, Optional, default 1

'DELIM\_STR: Separator, Optional, default space

RESULT\_ARR = Split(TEXT\_STR, DELIM\_STR) 'Lesson 3.8.8: String Split

INDEX\_NO = INDEX\_NO - 1

MIDW\_FUNC = Trim(RESULT\_ARR(INDEX\_NO))

If NO\_WORDS > 1 Then

If INDEX\_NO + NO\_WORDS - 1 > UBound(RESULT\_ARR) Then NO\_WORDS = UBound(RESULT\_ARR) - INDEX\_NO + 1

For i = 1 To NO\_WORDS - 1

MIDW\_FUNC = MIDW\_FUNC & DELIM\_STR & Trim(RESULT\_ARR(INDEX\_NO + i))

Next i

End If

End Function

'LEFT\_VAL\_FUNC: Extract a value from the left end of a string

Function LEFT\_VAL\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal DELIM\_STR As String = " ") As Double

Dim i As Long

'LEFT\_VAL\_FUNC: Return a value from the start of a string

'TEXT\_STR: String from which the value is returned

'DELIM\_STR: Character separating words, Optional, default space

LEFT\_VAL\_FUNC = CDbl(LEFTW\_FUNC(TEXT\_STR, 1, DELIM\_STR))

End Function

'RIGHT\_VAL\_FUNC: Extract a value from the right end of a string

Function RIGHT\_VAL\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal DELIM\_STR As String = " ") As Double

Dim i As Long

'RIGHT\_VAL\_FUNC: Return a value from the end of a string

'TEXT\_STR: String from which the value is returned

'DELIM\_STR: Character separating words, Optional, default space

RIGHT\_VAL\_FUNC = CDbl(RIGHTW\_FUNC(TEXT\_STR, 1, DELIM\_STR))

End Function

'FIND\_REV\_FUNC: Find from the left hand end

Function FIND\_REV\_FUNC(ByVal FIND\_TEXT\_STR As String, \_

ByVal WITHIN\_TEXT\_STR As String, \_

Optional ByVal l As Long = -1) As String

'FIND\_REV\_FUNC: Find from the right hand end

'FIND\_TEXT\_STR: Text string to be found

'WITHIN\_TEXT\_STR: Text to be searched

'l: Index of starting character, Optional, default -1 = last character

FIND\_REV\_FUNC = InStrRev(WITHIN\_TEXT\_STR, FIND\_TEXT\_STR, l)

'For further reference see: http://msdn.microsoft.com/en-us/library/t2ekk41a(v=vs.80).aspx

End Function

'REVERSE\_FUNC: Reverse a string

Function REVERSE\_FUNC(ByVal TEXT\_STR As String) As String

'TEXT\_STR: Text string to reverse

REVERSE\_FUNC = StrReverse(TEXT\_STR)

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.6: String to Array

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_TO\_ARRAY()

Dim i As Long

Dim TEXT\_STR As String

Dim TEXT\_ARR As Variant

Debug.Print "Convert a String to an Array"

'Lets look at a function to tokenize a string into discrete array elements.

TEXT\_STR = "The quick brown fox jumps over the lazy dog." 'After passing the string through this function you will end up with

'To calculate how many delimiters exist in the given string, use this function:

Debug.Print "To calculate how many delimiters exist in the given string, use this function: NUMBER\_DELIM\_FUNC(TEXT\_STR, " & """" & " " & \_

"""" & ") = " & NUMBER\_DELIM\_FUNC(TEXT\_STR, " ")

Debug.Print "In the TEXT\_STR there are eight spaces, so there will be nine array elements: TEXT\_ARR = STRING\_ARRAY\_FUNC(TEXT\_STR, " & """" & " " & """" & ")"

TEXT\_ARR = STRING\_ARRAY\_FUNC(TEXT\_STR, " ") 'The number of array elements will always be the delimiter count + 1.

For i = LBound(TEXT\_ARR) To UBound(TEXT\_ARR) 'i = Lower Index to Upper Index

Debug.Print TEXT\_ARR(i)

Next i

End Sub

Function STRING\_ARRAY\_FUNC(ByVal TEXT\_STR As String, \_

Optional DELIM\_STR As String = ",") As Variant

STRING\_ARRAY\_FUNC = Split(TEXT\_STR, DELIM\_STR) 'Lesson 3.8.8: String Split

'Pass in a delimited string and (optional) the delimiter and it will return the string tokenized into a zero-based

'array based on that delimiter. By default the delimiter is a comma, but you can specify any character. The input

'string must be delimited by the character specified as the delimiter.

End Function

Function NUMBER\_DELIM\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal DELIM\_STR As String = ",") As Long

'To calculate how many delimiters exist in the given string

NUMBER\_DELIM\_FUNC = (Len(TEXT\_STR) - Len(Replace(TEXT\_STR, DELIM\_STR, "")))

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.7: Building a string from an array

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lets now build a string from the contents of an array.

Function ARRAY\_BUILD\_STRING\_FUNC(ByRef DATA\_ARR() As Variant, \_

Optional ByVal DELIMITER\_STR As String = ",") As String

' Loop through array and build text string consisting

' of all the array elements, delimited as specified

Dim j As Long

For j = LBound(DATA\_ARR) To UBound(DATA\_ARR)

ARRAY\_BUILD\_STRING\_FUNC = ARRAY\_BUILD\_STRING\_FUNC & DELIMITER\_STR & DATA\_ARR(j)

Next j

' remove leading DELIMITER\_STR and return string

ARRAY\_BUILD\_STRING\_FUNC = Right$(ARRAY\_BUILD\_STRING\_FUNC, Len(ARRAY\_BUILD\_STRING\_FUNC) - Len(DELIMITER\_STR))

End Function

Sub TEST\_ARRAY\_BUILD\_STRING\_FUNC()

Dim DATA\_ARR() As Variant

DATA\_ARR = Array("MSFT", "GOOG", "TD")

Debug.Print ARRAY\_BUILD\_STRING\_FUNC(DATA\_ARR, "|")

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.8: String Split

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_SPLIT\_OR\_INSTRREV()

Dim i, j As Long

Dim START\_TIMER1, START\_TIMER2 As Single

Dim END\_TIMER1, END\_TIMER2 As Single

Dim nLOOPS As Long

Dim FILE\_NAME As Variant

Dim DELIM\_STR As String

Dim TEXT\_STR As String

Debug.Print "Split or InstrRev"

'InStrRev returns the position of the first occurrence of one string within another, starting from the right side of the string.

'For further reference see: http://msdn.microsoft.com/en-us/library/t2ekk41a(v=vs.80).aspx

'Lets compare parsing a string using Split and InstrRev to see which is faster.

DELIM\_STR = "\"

TEXT\_STR = "C:\EUM\Documents\LECTURE3.xls"

nLOOPS = 100000

' use Split

START\_TIMER1 = Timer

For i = 1 To nLOOPS

'Dim FILE\_NAME As Variant

FILE\_NAME = Split(TEXT\_STR, DELIM\_STR) 'File Name as an array: FILE\_NAME = {C:,EUM,DOCUMENTS,LECTURE3.xls}

j = UBound(FILE\_NAME) 'Index of the Last Entry in the Array: 3

'0=C:; 1=EUM; 2=DOCUMENTS; 3=LECTURE3.xls

FILE\_NAME = FILE\_NAME(j) 'The Variable FILE\_NAME is no re-assigned but no as a string!

Next i

Debug.Print j, FILE\_NAME

END\_TIMER1 = Timer

' use InstrRev

START\_TIMER2 = Timer

For i = 1 To nLOOPS

'The InstrRev Function returns only a number, which requires the Mid Function to do the actual parsing.

FILE\_NAME = Mid(TEXT\_STR, InStrRev(TEXT\_STR, DELIM\_STR) + 1)

'InStrRev: Returns the position of the first occurrence of one string within another, starting from the right side of the string.

'For further reference see: http://msdn.microsoft.com/en-us/library/t2ekk41a(v=vs.90).aspx

Next i

END\_TIMER2 = Timer

TEXT\_STR = "Number of iterations: " & nLOOPS & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Split: " & Format(END\_TIMER1 - START\_TIMER1, "#.###") & " seconds" & vbCrLf

TEXT\_STR = TEXT\_STR & "Using InstrRev: " & Format(END\_TIMER2 - START\_TIMER2, "#.###") & " seconds" & vbCrLf

Debug.Print TEXT\_STR

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.9: Append/Prepend a String/Value to Each Member of an Array

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Let's say we have an existing array and need to append a value to each element. For example, an ID number or a date

'value. This function loops through a passed array and appends the value.

Function ARRAY\_APPEND\_FUNC(ByRef DATA\_ARR As Variant, \_

ByVal VALUE\_STR As Variant, \_

Optional ByVal DELIM\_STR As String = ",") As Variant

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

TEMP\_ARR = DATA\_ARR

If IS\_2D\_ARRAY\_FUNC(TEMP\_ARR) Then

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

For j = LBound(TEMP\_ARR, 2) To UBound(TEMP\_ARR, 2)

TEMP\_ARR(i, j) = TEMP\_ARR(i, j) & DELIM\_STR & VALUE\_STR

Next j

Next i

Else

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

TEMP\_ARR(i) = TEMP\_ARR(i) & DELIM\_STR & VALUE\_STR

Next i

End If

ARRAY\_APPEND\_FUNC = TEMP\_ARR

End Function

'This function uses the IS\_2D\_ARRAY\_FUNC function we introduced in a previous section. It assumes that the array being passed is

'either one or two dimensional.

'We chose the simplest way to do this. We could have also declared an empty array, resized it to the same size as the original

'array, then wrote each array value to the new array (no word on which method is faster).

Sub TEST\_ARRAY\_APPEND\_FUNC() 'let's append a value to each element in an array.

Dim i As Long, j As Long

Dim DATA\_ARR As Variant

DATA\_ARR = ARRAY\_CREATE\_FUNC(5, 2, False)

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1), LBound(DATA\_ARR, 2) & ":" & UBound(DATA\_ARR, 2)

DATA\_ARR = ARRAY\_FILL\_FUNC(DATA\_ARR, "MSFT")

For i = LBound(DATA\_ARR, 1) To UBound(DATA\_ARR, 1)

For j = LBound(DATA\_ARR, 2) To UBound(DATA\_ARR, 2)

Debug.Print i, j, DATA\_ARR(i, j)

Next j

Next i

DATA\_ARR = ARRAY\_APPEND\_FUNC(DATA\_ARR, " GOOG")

For i = LBound(DATA\_ARR, 1) To UBound(DATA\_ARR, 1)

For j = LBound(DATA\_ARR, 2) To UBound(DATA\_ARR, 2)

Debug.Print i, j, DATA\_ARR(i, j)

Next j

Next i

End Sub

'Prepend a Value to Each Member of an Array: Prepending a value is almost the same procedure, except the value is placed

'at the beginning of each element instead of at the end.

Function ARRAY\_PREPEND\_EACH\_FUNC(ByRef DATA\_ARR As Variant, \_

ByVal VALUE\_STR As Variant, \_

Optional ByVal DELIM\_STR As String = ",") As Variant

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

TEMP\_ARR = DATA\_ARR

If IS\_2D\_ARRAY\_FUNC(TEMP\_ARR) Then

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

For j = LBound(TEMP\_ARR, 2) To UBound(TEMP\_ARR, 2)

TEMP\_ARR(i, j) = VALUE\_STR & DELIM\_STR & TEMP\_ARR(i, j)

Next j

Next i

Else

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR)

TEMP\_ARR(i) = VALUE\_STR & DELIM\_STR & TEMP\_ARR(i)

Next i

End If

ARRAY\_PREPEND\_EACH\_FUNC = TEMP\_ARR

End Function

'As with the previous function, this one assumes that the array is one or two dimensional.

Sub TEST\_ARRAY\_APPEND\_EACH\_FUNC() 'let's append a value to each element in an array.

Dim i As Long, j As Long

Dim DATA\_ARR As Variant

DATA\_ARR = ARRAY\_CREATE\_FUNC(5, 2, False)

Debug.Print LBound(DATA\_ARR, 1) & ":" & UBound(DATA\_ARR, 1), LBound(DATA\_ARR, 2) & ":" & UBound(DATA\_ARR, 2)

DATA\_ARR = ARRAY\_FILL\_FUNC(DATA\_ARR, "MSFT")

For i = LBound(DATA\_ARR, 1) To UBound(DATA\_ARR, 1)

For j = LBound(DATA\_ARR, 2) To UBound(DATA\_ARR, 2)

Debug.Print i, j, DATA\_ARR(i, j)

Next j

Next i

DATA\_ARR = ARRAY\_PREPEND\_EACH\_FUNC(DATA\_ARR, " GOOG")

For i = LBound(DATA\_ARR, 1) To UBound(DATA\_ARR, 1)

For j = LBound(DATA\_ARR, 2) To UBound(DATA\_ARR, 2)

Debug.Print i, j, DATA\_ARR(i, j)

Next j

Next i

End Sub

'Add Values to the Beginning of an Array: The following VBA function increases the size of the array and inserts new

'values into the empty elements.

Function ARRAY\_PREPEND1\_FUNC(ByRef DATA\_ARR As Variant, \_

ParamArray VALUES\_TO\_PREPEND\_ARR() As Variant) As Variant

'The input array must be one dimensional. You can also add values using a ParamArray. We could have also used a

'traditional array and simply looped through it to get the values to add, but looping through a ParamArray works

'exactly the same (and provides another tool for our use).

'For further reference see:

'http://msdn.microsoft.com/en-us/library/ct363x9h%28v=vs.80%29.aspx

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

' make temp array the same size as Parram Array (Always Base 0)

ReDim TEMP\_ARR(LBound(DATA\_ARR) To (UBound(DATA\_ARR) + UBound(VALUES\_TO\_PREPEND\_ARR) + 1))

' put new elements at the beginning of the new array

j = LBound(DATA\_ARR)

For i = LBound(VALUES\_TO\_PREPEND\_ARR) To UBound(VALUES\_TO\_PREPEND\_ARR)

TEMP\_ARR(j) = VALUES\_TO\_PREPEND\_ARR(i)

j = j + 1

Next i

For i = LBound(DATA\_ARR) To UBound(DATA\_ARR)

TEMP\_ARR(j) = DATA\_ARR(i)

j = j + 1

Next i

ARRAY\_PREPEND1\_FUNC = TEMP\_ARR

End Function

Function ARRAY\_PREPEND2\_FUNC(ByRef DATA\_ARR As Variant, \_

ParamArray VALUES\_TO\_PREPEND\_ARR() As Variant) As Variant ' This function does the same thing as my code above, albeit as a one-liner

ARRAY\_PREPEND2\_FUNC = Split(Join(VALUES\_TO\_PREPEND\_ARR, Chr(0)) & Chr(0) & Join(DATA\_ARR, Chr(0)), Chr(0))

End Function

Sub TEST\_ARRAY\_PREPEND12\_FUNC()

Dim i As Long

Dim DATA\_ARR As Variant

Dim DATA1\_ARR As Variant

Dim DATA2\_ARR As Variant

DATA\_ARR = Split("TD GS RY C")

DATA1\_ARR = ARRAY\_PREPEND1\_FUNC(DATA\_ARR, "ORCL", "GOOG", "AAPL")

DATA2\_ARR = ARRAY\_PREPEND2\_FUNC(DATA\_ARR, "ORCL", "GOOG", "AAPL")

For i = LBound(DATA1\_ARR, 1) To UBound(DATA1\_ARR, 1)

Debug.Print i, DATA1\_ARR(i), DATA2\_ARR(i)

Next i

'Parram Array always returns a zero base array: to Change it to base 1 use the function

'ARRAY\_CHANGE\_BASE\_ONE\_FUNC

DATA2\_ARR = ARRAY\_CHANGE\_BASE\_ONE\_FUNC(DATA2\_ARR)

Debug.Print LBound(DATA1\_ARR) & ":" & UBound(DATA1\_ARR) & " to " & LBound(DATA2\_ARR) & ":" & UBound(DATA2\_ARR)

End Sub

Function ARRAY\_CHANGE\_BASE\_ONE\_FUNC(ByRef DATA\_RNG As Variant)

Dim i As Long

Dim j As Long

Dim SROW As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim DATA\_VECTOR As Variant

Dim TEMP\_VECTOR As Variant

On Error GoTo ERROR\_LABEL

DATA\_VECTOR = DATA\_RNG

SROW = LBound(DATA\_VECTOR, 1)

NROWS = UBound(DATA\_VECTOR, 1)

If SROW = 1 Then

ARRAY\_CHANGE\_BASE\_ONE\_FUNC = DATA\_VECTOR

Exit Function

End If

If IS\_2D\_ARRAY\_FUNC(DATA\_VECTOR) = True Then

NCOLUMNS = UBound(DATA\_VECTOR, 2)

ReDim TEMP\_VECTOR(1 To NROWS + 1, 1 To NCOLUMNS)

For i = 0 To NROWS

For j = 1 To NCOLUMNS

TEMP\_VECTOR(i + 1, j) = DATA\_VECTOR(i, j)

Next j

Next i

ElseIf IS\_1D\_ARRAY\_FUNC(DATA\_VECTOR) = True Then

ReDim TEMP\_VECTOR(1 To NROWS + 1, 1 To 1)

For i = 0 To NROWS

TEMP\_VECTOR(i + 1, 1) = DATA\_VECTOR(i)

Next i

End If

ARRAY\_CHANGE\_BASE\_ONE\_FUNC = TEMP\_VECTOR

Exit Function

ERROR\_LABEL:

ARRAY\_CHANGE\_BASE\_ONE\_FUNC = Err.Number

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.10: String Empty Check

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_EMPTY\_CHECK()

Dim i As Long

Dim START\_TIMER1, START\_TIMER2 As Single

Dim END\_TIMER1, END\_TIMER2 As Single

Dim CHECK\_FLAG As Boolean

Dim nLOOPS As Long

Dim TEXT\_STR As String

Dim VARIABLE\_STR As String

Dim FIXED\_STR As String \* 10

Debug.Print "Check if a String is Empty"

'While testing out some operations on a Variant, you may run across an interesting problem with fixed-length string.

'Remember you can declare a string to be a fixed length with a declaration like

'Dim TEXT\_STR As String \* 10 (for further reference see http://www.cpearson.com/excel/sizestring.htm)

'Turns out, this type of variable fails the Len test: it returns the predetermined length of the variable even if you

'don't set it to any value! Let's start by figuring out by the definition of "empty". Lets define it as a string equal

'to "". The next function determines if a string is empty and returns True or False accordingly.

'There are four possible conditions:

'1. A variable-length string that hasn't been set to any value: Len(TEXT\_STR) = 0

'2. A variable-length string that has been set to some value, ex: "Hello World" : Len(TEXT\_STR) > 0

'3. A fixed-length string that hasn't been set to any value: Len(TEXT\_STR) > 0

'4. A fixed-length string that has been set to some value, ex: "Hello World" : Len(TEXT\_STR) > 0

'In other words, variable-length strings behave according to common sense, whereas fixed-length strings

'always return a length even if they haven't been set to anything.

'What we could find is that in condition #3, the value of string is actually a series of Chr(0) characters, equal

'to the fixed-length number.

'In only one case is the length of the string actually ever zero: a variable-length string that hasn't been set to any value.

'In that case, the length is zero and we can exit the function. This satisfies condition #1. Otherwise, we need to loop.

'To speed things up, we could short circuit the IS\_STRING\_EMPTY\_FUNC function inside the For loop on Version 0 Ü if it

'runs into a character other than Chr(0), the function immediately exits and returns False because the string cannot

'possibly be empty. This handles condition #2 and #4.

'The reason we total the number of Chr(0) values and compare it to the length of the string at the end is to avoid cases

'where Chr(0) is actually a valid part of the input string. Only cases where the entire string consists of Chr(0) will be

'caught, which meets condition #3. This of course is an assumption, because somewhere out there a bunch of Chr(0) characters

'could be a valid string to someone.

'I hoped there was a better way to do this but IsEmpty returns false for both types of strings. I tried TEXT\_STR = Empty but

'it works the same as Len: returns True for variable length strings and False for all other cases. Empty returns the same

'response as Len when checking variable- and fixed-length strings.

'According to Excel's Visual Basic Help, Empty is a Variant subtype, so we could expect some implicit type conversion here when

'comparing string values to Empty. There is a dramatic difference in speed! Len is much faster!!

Debug.Print "Dim VARIABLE\_STR As String"

Debug.Print "Dim FIXED\_STR As String \* 10"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

VARIABLE\_STR = "Hello World"

FIXED\_STR = VARIABLE\_STR

Debug.Print "VARIABLE\_STR = " & VARIABLE\_STR 'Hello World

Debug.Print "FIXED\_STR = " & FIXED\_STR 'Hello Worl

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

VARIABLE\_STR = ""

FIXED\_STR = ""

Debug.Print "Len(VARIABLE\_STR) = " & Len(VARIABLE\_STR) '0

Debug.Print "Len(FIXED\_STR) = " & Len(FIXED\_STR) '10

Debug.Print "VARIABLE\_STR = Empty: " & (VARIABLE\_STR = Empty) 'True

Debug.Print "FIXED\_STR = Empty: " & (FIXED\_STR = Empty) 'False

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

VARIABLE\_STR = Chr(0)

FIXED\_STR = VARIABLE\_STR

Debug.Print "VARIABLE\_STR = Chr(0)"

Debug.Print "FIXED\_STR = VARIABLE\_STR"

Debug.Print "Len(VARIABLE\_STR) = " & Len(VARIABLE\_STR) '1

Debug.Print "Len(FIXED\_STR) = " & Len(FIXED\_STR) '10

Debug.Print "VARIABLE\_STR = Empty: " & (VARIABLE\_STR = Empty) 'False

Debug.Print "FIXED\_STR = Empty: " & (FIXED\_STR = Empty) 'False

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 0) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 0)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 1) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 1)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 2) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 2)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 3) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 3)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 4) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 4)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 5) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 5)

Debug.Print "IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 6) = " & IS\_STRING\_EMPTY\_FUNC(VARIABLE\_STR, 6)

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

nLOOPS = 100000

VARIABLE\_STR = "Hello World"

START\_TIMER1 = Timer

For i = 1 To nLOOPS

CHECK\_FLAG = (Len(VARIABLE\_STR) > 0)

Next i

END\_TIMER1 = Timer

' use Empty

START\_TIMER2 = Timer

For i = 1 To nLOOPS

CHECK\_FLAG = (VARIABLE\_STR = Empty)

Next i

END\_TIMER2 = Timer

TEXT\_STR = "Number of iterations: " & nLOOPS & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Len: " & Format(END\_TIMER1 - START\_TIMER1, "#.###") & " seconds" & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Empty: " & Format(END\_TIMER2 - START\_TIMER2, "#.###") & " seconds" & vbCrLf

Debug.Print TEXT\_STR

'It seems that Len is the fastest method for checking string length (and therefore, whether strings are empty), and it scales well.

End Sub

Function IS\_STRING\_EMPTY\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal VERSION As Integer = 0) As Boolean

Dim i As Long

Dim j As Long 'number of blanks

Dim CHECK\_FLAG As Boolean

CHECK\_FLAG = False

'-----------------------------------------------------------------------------------------------------------

Select Case VERSION

'-----------------------------------------------------------------------------------------------------------

Case 0

'-----------------------------------------------------------------------------------------------------------

If Len(TEXT\_STR) = 0 Then ' if len is zero, must be empty

CHECK\_FLAG = True

Else

' check if every character is Chr(0) if so, assume it is an uninitialized fixed-length string

For i = 1 To Len(TEXT\_STR)

If Mid$(TEXT\_STR, i, 1) = Chr(0) Then

j = j + 1

Else

GoTo 1983 ' a valid char, cannot be empty

End If

Next i

CHECK\_FLAG = (j = Len(TEXT\_STR))

End If

'-----------------------------------------------------------------------------------------------------------

Case 1

'-----------------------------------------------------------------------------------------------------------

If Len(TEXT\_STR) = 0 Then ' if len is zero, must be empty

CHECK\_FLAG = True

Else

CHECK\_FLAG = (TEXT\_STR = String(Len(TEXT\_STR), Chr(0)))

End If

'-----------------------------------------------------------------------------------------------------------

Case 2 'One-liner routine

'-----------------------------------------------------------------------------------------------------------

CHECK\_FLAG = Len(Replace(TEXT\_STR, Chr(0), "")) = 0

'-----------------------------------------------------------------------------------------------------------

Case 3

'-----------------------------------------------------------------------------------------------------------

If Len(Replace(TEXT\_STR, Chr(0), "")) Then

CHECK\_FLAG = False ' TEXT\_STR is NOT empty, so do what you want to with its contents

Else

CHECK\_FLAG = True ' TEXT\_STR is really empty, so react accordingly

End If

'-----------------------------------------------------------------------------------------------------------

Case 4 'Another one-liner that is some two-and-a-half times faster executing than Case 2

'-----------------------------------------------------------------------------------------------------------

CHECK\_FLAG = Not TEXT\_STR Like "\*[!" & Chr$(0) & "]\*" 'Please refer to the OPTIONAL\_TEST\_STRING\_PATTERNS section

'-----------------------------------------------------------------------------------------------------------

Case 5

'-----------------------------------------------------------------------------------------------------------

CHECK\_FLAG = (Len(Application.WorksheetFunction.Clean(TEXT\_STR)) = 0)

'-----------------------------------------------------------------------------------------------------------

Case Else

'-----------------------------------------------------------------------------------------------------------

'You could prefer using Trim(TEXT\_STR) because of the problems faced with residual blanks in strings and cells.

'Speaking of spaces, you could also encounter problems with higher order ASCII characters such as the non-breaking

'space character (ASCII 160), especially with data cut and pasted into Excel. Note that the worksheet CLEAN function

'only removes ASCII values 0-31.

CHECK\_FLAG = (Len(Application.WorksheetFunction.Clean(Trim(TEXT\_STR))) = 0)

'-----------------------------------------------------------------------------------------------------------

End Select

'-----------------------------------------------------------------------------------------------------------

1983:

IS\_STRING\_EMPTY\_FUNC = CHECK\_FLAG

Exit Function

ERROR\_LABEL:

IS\_STRING\_EMPTY\_FUNC = False

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.11: String Comparisons

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

Sub TEST\_STRING\_COMPARISONS()

Debug.Print "Types of String Comparisons"

'The simplest way to compare two strings is use =, > and < operators. By default VBA will use the binary comparison.

'However, if you want to do a case insensitive comparison, you will need to explicitly set the comparison option to

'Text at the beg. of your module: Option Compare Text

'If the option is not set (or is set to Option Compare Binary), the following code will show FALSE because the binary

'equivalent of \_Textî is not the same as \_thisî.

'However then the binary comparison has been explicitly set to Text, the code shows TRUE

Debug.Print "(" & """" & "This" & """" & " = " & """" & "this" & """" & ") --> " & ("This" = "this")

'-> Binary String Comparison (Case sensitive) in VBA

'For any formula If you want to compare two string in such a manner that each individual characters is compared

'with its counterpart in a case sensitive manner (Ex. This is not equal to thisî because T is not equal to ´t),

'you can do either of the two things:

'1. Declare the statement Option Compare Binary at the very onset of VBA code or

'2. Specify the comparison to be made as Binary in the formula (as an argument)

'In case you were wondering why A has a lesser value than a, it is so because ´A is represented by a lower byte than ´a

'and occurs earlier in the list when starting from 0. The binary comparison is the one that VBA would use

'if nothing else has been explicitly specified

'-> Text Comparison in VBA (Case insensitive comparison)

'For any formula if you want to compare two string in such a manner that each individual characters is compared

'with its counterpart in a case insensitive manner (Ex. THis is would be equal to this even though some of

'the characters dont have the same case as their counterparts in the other string), you can do either of the

'two things:

'1. Declare the statement Option Compare Text at the very onset of VBA code or

'2. Specify the comparison to be made as Text in the formula (as an argument)

'In this case ´A is equal to ´a. This helps in comparing strings when they have the same sequence of

'characters but each may have a case different from its counterpart.

'Lets compare a strings using strComp:

'StrComp VBA function can be used when you would like to compare two strings and return a value indicating whether both

'are identical or which one is greater/smaller than the other. The syntax of the VBA StrComp function is:

'StrComp(first\_string\_to\_compare, second\_string\_to\_compare,comparison\_type)

'Where:

'first\_string\_to\_compare = The first string that you would like to compare.

'second\_string\_to\_compare = The second string that you would like to compare.

'string\_to\_find = The string or sub-string which needs to be located in the above string

'comparison\_type = Specifies whether the comparison should be carried out as a binary (byte level, case sensitive) or

'text (byte level, case insensitive) (optional)

'Please bear in mind that if the comparison type is not explicitly specified within the function, the value specified in the

'Option Compare Statement at the module level will be used.

'The function returns:

'-1 when second\_string\_to\_compare is greater than first\_string\_to\_compare

'0 when second\_string\_to\_compare is equal to the first\_string\_to\_compare

'1 when second\_string\_to\_compare is less than first\_string\_to\_compare

'Null when either of the strings is NULL

'For example the following function would show -1 if no comparison\_type is explicitly specified.

Debug.Print "StrComp(" & """" & "This" & """" & ", " & """" & "this" & """" & ") = " & StrComp("This", "this")

'Need to declare Option Compare Binary

'However when we explicitly specify the comparison type to be text, the function would show 0.

Debug.Print "StrComp(" & """" & "This" & """" & ", " & """" & "this" & """" & ",1) = " & StrComp("This", "this", 1)

'Compare as text while ignoring case

End Sub

'Since you 're actually comparing ANSI values (see Sub Test\_Ansi\_values above) when you compare two strings, you can use the same comparison operators that

'you would use with numeric expressionsãgreater than (>), less than (<), equal to (=), and so on. In addition to these numeric

'comparison operators, you can also use the Like operator, which is specifically for use in comparing strings, including

'strings that contain wildcard characters.

Sub Test\_Ansi\_values()

Dim i As Long

Dim DST\_RNG As Range

Dim FORMULA\_STR As String

Dim SRC\_WSHEET As Worksheet

Set SRC\_WSHEET = Worksheets("WSHEET\_TEST")

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

For i = 1 To 255

.Cells(i, 1) = i 'See http://en.wikipedia.org/wiki/ASCII

.Cells(i, 2).Formula = "=CHAR(" & .Cells(i, 1).Address(False, False) & ")"

Next i

End With

End Sub

' When you use comparison operators such as the greater than (>) and less than (<) operators to compare two strings, the result

' you get depends on the string-comparison setting for the module. Consider the following example:

' "vba" > "VBA"

' If the string-comparison setting is Option Compare Binary, the comparison returns True.

' When VBA performs a binary text comparison, it compares the binary values for each corresponding position in the string until it

' finds two that differ. In this example, the lowercase letter "v" corresponds to the ANSI value 118, while the uppercase letter "V"

' corresponds to the ANSI value 86. Since 118 is greater than 86, the comparison returns True.

'Note: For tables of ANSI values and the characters they represent, search the Column 1 and 2 of the worksheet "WSHEET\_STRING\_CHARACTERS".

' If the string-comparison setting is Option Compare Text, "vba" > "VBA" returns False, since the strings are equivalent apart from case.

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.12: Finding strings/values in an array

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Unfortunately , there 's no built-in way to check if a value is in array.

'We might be inclined to loop through the array and using a function like Instr to match each value in the array against

'the selected value to see if it exists.

Function ARRAY\_FINDING\_VALUES\_FUNC(ByRef DATA\_ARR As Variant, \_

ByVal FIND\_STR As Variant)

Dim i As Long

Dim MATCH\_FLAG As Boolean

MATCH\_FLAG = False

For i = LBound(DATA\_ARR) To UBound(DATA\_ARR)

If InStr(DATA\_ARR(i), FIND\_STR) > 0 Then

MATCH\_FLAG = True

Exit For

End If

Next i

ARRAY\_FINDING\_VALUES\_FUNC = Array(MATCH\_FLAG, i)

End Function

'But there is also another way to do it that doesn't require looping. Let's used a combination of UBound and Filter to

'reach a solution that fully replaces a loop. So here 's a function that checks if a given value is in an array:

'Suggested reading Using the Filter Function to Search String Arrays

'http://msdn.microsoft.com/en-us/library/office/aa164525(v=office.10).aspx

Function ARRAY\_SEARCH\_STRING\_FUNC1(ByRef DATA\_ARR As Variant, ByVal FIND\_STR As Variant) As Boolean 'Checking arrays for exact matches

' checks if FIND\_STR is found in DATA\_ARR, no loop!

ARRAY\_SEARCH\_STRING\_FUNC1 = (UBound(Filter(DATA\_ARR, FIND\_STR)) > -1)

End Function

'Let's test it out by looking for a stock symbol in an array:

Sub TEST\_ARRAY\_FINDING\_VALUES\_FUNC\_AND\_ARRAY\_SEARCH\_STRING\_FUNC1()

Dim DATA\_ARR As Variant

Dim RESULT\_ARR As Variant

DATA\_ARR = Split("GS,TD,GOOD,MSFT", ",")

RESULT\_ARR = ARRAY\_FINDING\_VALUES\_FUNC(DATA\_ARR, "MSFT")

Debug.Print RESULT\_ARR(LBound(RESULT\_ARR)), RESULT\_ARR(UBound(RESULT\_ARR))

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "MSFT")

End Sub

'In checking arrays for exact matches we showed how a simple one-line function can check if a given value is in an array

'and its variation for exact match. However, there is a problem when a value is found only in a substring of an

'array element, which is usually not what we want. We need to match the whole array element!!!

Sub PROBLEMS\_WITH\_ARRAY\_SEARCH\_STRING\_FUNC1()

Dim DATA\_ARR As Variant

DATA\_ARR = Array("Patrick", "Mike", "Nico Fermin")

'Load some names into an array, then check if a given name is in the array "Nico". We would expect

'this to return false. But it doesn't. Since Nico is a substring of "Nico Fermin", this will return

'true. It depends on the situation, but most times this is not what we want.

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "Nico")

'How do we fix this without looping through the array?

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "c")

'Even though "c" is not a discrete element (element of the array) it is a substring, so this function returns TRUE

End Sub

'To solve this problem we could join the array with a comma delimiter, then parse the resulting string for

'the occurrence of the alleged "match" (according to Filter). Then we do a string comparison to see if it is

'truly a match.

Sub SOLVE\_THE\_PROBLEM\_WITH\_ARRAY\_SEARCH\_STRING\_FUNC1()

Dim i As Long 'Start Position

Dim j As Long 'Next comma position

Dim DATA\_ARR As Variant

Dim FIND\_STR As Variant

Dim NAMES\_STR As String

Dim MATCH\_STR As String

DATA\_ARR = Array("Patrick", "Mike", "Nico Fermin")

FIND\_STR = "Nico"

If ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, FIND\_STR) Then

NAMES\_STR = Join(DATA\_ARR, ",") & ","

' start from the name ....

i = InStr(NAMES\_STR, FIND\_STR)

' get position of the comma after the name, starting from after the name ...

j = InStr(i + 1, NAMES\_STR, ",")

' the alleged "match" is in between

MATCH\_STR = Mid$(NAMES\_STR, i, j - i)

If StrComp(FIND\_STR, MATCH\_STR) = 0 Then

Debug.Print "found it!"

Else

Debug.Print "didn't find it!"

End If

End If

End Sub

'Of course, we don't want to do all of this in a procedure. So we'll have to modify our nice one-line function and

'turn it into a function that joins the array elements, looks for delimiters, then checks if the target

'string is the same value as the parsed substring

Function ARRAY\_SEARCH\_STRING\_FUNC2(ByRef DATA\_ARR As Variant, \_

ByVal FIND\_STR As Variant, \_

Optional ByVal EXACT\_FLAG As Boolean = True) As Boolean

'EXACT\_FLAG =False option of a substring match, but if we want an exact match set it = TRUE

Dim i As Long

Dim j As Long

Dim MATCH\_STR As String

Dim NAMES\_STR As String

If UBound(Filter(DATA\_ARR, FIND\_STR)) > -1 Then

NAMES\_STR = Join(DATA\_ARR, ",") & ","

' start from the allegedly matched term ....

i = InStr(NAMES\_STR, FIND\_STR)

' get position of the comma after the allegedly matched term ...

j = InStr(i + 1, NAMES\_STR, ",")

' the alleged "match" is in between

MATCH\_STR = Mid$(NAMES\_STR, i, j - i)

'This code assumes that the matched term is only located once in the source array. In other words, it only matches

'the first occurrence because it only searches once (using InStr). This is trivial because we don't care how many

'times the string appears, only whether it appears at all.

If EXACT\_FLAG Then

ARRAY\_SEARCH\_STRING\_FUNC2 = (StrComp(FIND\_STR, MATCH\_STR) = 0)

Else

ARRAY\_SEARCH\_STRING\_FUNC2 = (StrComp(FIND\_STR, MATCH\_STR) <> 0)

End If

'The StrComp function compares two strings and returns a value that represents the result

'of the comparison. The StrComp function can return one of the following values:

'-1 (if string1 < string2)

'0 (if string1 = string2)

'1 (if string1 > string2)

'Null (if string1 or string2 is Null)

'Parameter | Description

'string1 | Required. A string expression

'string2 | Required. A string expression

'compare | Optional. Specifies the string comparison to use. Default is 0

'Can have one of the following values:

'0 = vbBinaryCompare - Perform a binary comparison

'1 = vbTextCompare - Perform a textual comparison

End If

End Function

'Here is another solution which simply loops through the array (not as intelligent, but simpler):

Function ARRAY\_SEARCH\_STRING\_FUNC3(ByRef DATA\_ARR As Variant, FIND\_STR As Variant) As Boolean

Dim i As Long

For i = LBound(DATA\_ARR) To UBound(DATA\_ARR)

If StrComp(DATA\_ARR(i), FIND\_STR) = 0 Then

ARRAY\_SEARCH\_STRING\_FUNC3 = True

Exit For

End If

Next i

End Function

'As an alternative, if we know that the array element is in a specific position, we can use an offset (j) like this:

Function ARRAY\_SEARCH\_STRING\_FUNC4(ByRef DATA\_ARR As Variant, \_

ByVal FIND\_STR As Variant, \_

Optional ByVal j As Long) As Boolean

Dim i As Long

For i = (LBound(DATA\_ARR) + j) To UBound(DATA\_ARR)

If StrComp(DATA\_ARR(i), FIND\_STR) = 0 Then

ARRAY\_SEARCH\_STRING\_FUNC4 = True

Exit For

End If

Next i

End Function

Sub PROBLEMS\_WITH\_ARRAY\_SEARCH\_STRING\_FUNC234()

Dim DATA\_ARR As Variant

DATA\_ARR = Array("Patrick", "Mike", "Nico Fermin")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "c")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC2(DATA\_ARR, "c")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC3(DATA\_ARR, "c")

'Lets assume that DATA\_ARR is a large array and we know that the string we are searching

'for is somewhere in the second half. We could call ARRAY\_SEARCH\_STRING\_FUNC4 like:

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC4(DATA\_ARR, "c", (UBound(DATA\_ARR) / 2))

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "Nico")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC2(DATA\_ARR, "Nico")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC3(DATA\_ARR, "Nico")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC4(DATA\_ARR, "Nico", (UBound(DATA\_ARR) / 2))

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC1(DATA\_ARR, "Nico Fermin")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC2(DATA\_ARR, "Nico Fermin")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC3(DATA\_ARR, "Nico Fermin")

Debug.Print ARRAY\_SEARCH\_STRING\_FUNC4(DATA\_ARR, "Nico Fermin", (UBound(DATA\_ARR) / 2))

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.13: Arrays for unique strings/values and differences

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lets now look at some functions for checking the intersections of two arrays and returning unique values.

'Duplicates from an Array: If you have an array with one or more duplicates, the following function gets rid of them and

'returns the array without duplicates. Currently the function only supports one dimensional arrays, you should all try

'coding the version that supports two-dimensional arrays.

Function ARRAY\_UNIQUE\_VALUES\_FUNC(ByRef DATA\_ARR As Variant) As Variant

Dim i As Long

Dim j As Long

Dim TEMP\_ARR As Variant

' start the temp array with one element and

' populate with first value

j = LBound(DATA\_ARR)

ReDim TEMP\_ARR(j To j)

TEMP\_ARR(j) = DATA\_ARR(LBound(DATA\_ARR))

For i = LBound(DATA\_ARR) To UBound(DATA\_ARR)

If Not ARRAY\_SEARCH\_STRING\_FUNC2(TEMP\_ARR, DATA\_ARR(i)) Then ' not in destination array

'We use the ARRAY\_SEARCH\_STRING\_FUNC2 function to check if the value being inspected is in the destination array.

ReDim Preserve TEMP\_ARR(UBound(TEMP\_ARR) + 1)

TEMP\_ARR(UBound(TEMP\_ARR)) = DATA\_ARR(i)

End If

Next i

'In Lecture 3 we will learn how to code this function using a Collection Object for Speed.

ARRAY\_UNIQUE\_VALUES\_FUNC = TEMP\_ARR

End Function

Sub TEST\_ARRAY\_UNIQUE\_VALUES\_FUNC()

Dim i As Long

Dim DATA\_ARR As Variant

DATA\_ARR = Split("11,3,8,2,10,12,14,11,9,10,11,5,6,15,2,8,6,4,4,1,13,4,3,9,14,11,2,14,5,2,4,3,7,7,5,15,4,5,12,12,15,12,12,7,4,14", ",")

DATA\_ARR = ARRAY\_UNIQUE\_VALUES\_FUNC(DATA\_ARR)

For i = LBound(DATA\_ARR) To UBound(DATA\_ARR): Debug.Print DATA\_ARR(i): Next i

End Sub

'The following function takes two arrays and returns the values in the first array that are present in both.

'We will start by looping through an array and check if any of its values are present in the second array.

'If so, the value is added to a temporary array which is passed back to the calling function.

Function ARRAYS\_INTERSECT\_VALUES\_FUNC(ByRef DATA1\_ARR As Variant, \_

ByRef DATA2\_ARR As Variant) As Variant 'This function finds the values in the first array that exist in the second array.

Dim i As Long

Dim TEMP\_ARR As Variant

' start with a single element

ReDim TEMP\_ARR(LBound(DATA1\_ARR) To LBound(DATA1\_ARR))

' if element in first array exists in second array, keep it

For i = LBound(DATA1\_ARR) To UBound(DATA1\_ARR)

If ARRAY\_SEARCH\_STRING\_FUNC1(DATA2\_ARR, DATA1\_ARR(i)) Then

ReDim Preserve TEMP\_ARR(UBound(TEMP\_ARR) + 1)

TEMP\_ARR(UBound(TEMP\_ARR)) = DATA1\_ARR(i)

End If

Next i

For i = LBound(TEMP\_ARR) To UBound(TEMP\_ARR) - 1

' first element is Empty, so shift all elements one position up

TEMP\_ARR(i) = TEMP\_ARR(i + 1)

Next i

ReDim Preserve TEMP\_ARR(LBound(TEMP\_ARR) To UBound(TEMP\_ARR) - 1)

ARRAYS\_INTERSECT\_VALUES\_FUNC = TEMP\_ARR

'If there are no values in the second array that exist in the first, the function will return an empty array.

'Make sure to test if an empty array is returned.

End Function

Sub TEST\_ARRAYS\_INTERSECT\_VALUES\_FUNC()

Dim i As Long

Dim DATA1\_ARR As Variant

Dim DATA2\_ARR As Variant

Dim DATA3\_ARR As Variant

DATA2\_ARR = Split("11,3,8,2,10,12,14,11,9,10,11,5,6,15,2,8,6,4,4,1,13,4,3,9,14,11,2,14,5,2,4,3,7,7,5,15,4,5,12,12,15,12,12,7,4,14", ",")

DATA1\_ARR = ARRAY\_UNIQUE\_VALUES\_FUNC(DATA2\_ARR)

DATA3\_ARR = ARRAYS\_INTERSECT\_VALUES\_FUNC(DATA1\_ARR, DATA2\_ARR)

For i = LBound(DATA3\_ARR) To UBound(DATA3\_ARR): Debug.Print DATA1\_ARR(i), DATA3\_ARR(i): Next i

End Sub

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.14: Wildcard string comparisons (Optional)

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

' You can perform wildcard string comparisons by using the Like operator. The following table shows the wildcard characters supported

' by VBA.

'\* Any number of characters. t\* matches any word beginning with "t."

'? Any single character. t??t matches any four-letter word beginning and ending with "t."

'# Any single digit (0\_9). 1#3 matches any three-digit number beginning with "1" and ending with "3."

'[charlist] Any single character in charlist. [a-z] matches any letter that falls between "a" and "z"

'(case-sensitivity depends on Option Compare setting).

'[!charlist] Any single character not in charlist. [!A-Z] excludes the uppercase alphabetic characters (case-sensitivity depends

'on Option Compare setting).

'You can use the Like operator to perform data validation or wildcard searches. For example, suppose you want to ensure that a user has

'entered a telephone number in the format nnn-nnn-nnnn. You can use the Like operator to check that the entry is valid, as the following

'procedure does: strPhone Like "###-###-####". This line checks that the passed-in value is a valid, properly formatted telephone number.

'It compares characters in a string to make sure that certain positions contain numeric characters. In order to return True, all

'characters must be digits between 0 and 9 or hyphens, and the hyphens must be present at the correct position in the string.

Sub OPTIONAL\_TEST\_STRING\_PATTERNS()

Dim i As Long

Dim START\_TIMER1, START\_TIMER2 As Single

Dim END\_TIMER1, END\_TIMER2 As Single

Dim TEXT\_STR As String

Dim CHECK\_FLAG As Boolean

Dim nLOOPS As Long

Dim REGEX\_OBJ As Object

'As we have seen above, the wildcards allows partial sub-strings to matched to strings while providing some amount of

'flexibility. However there is a constraint \_ when you specify \*, # or ?, you can not specify which of the characters

'you would like to exclude from the list of characters that generate a positive match.

'VBA provides the option to specify and narrow down the a list of values that can generate a positive match when a

'compared with a wildcard. The same mechanism can also be used to exclude some other characters from being matched.

'In the previous example, suppose we wanted to use only A,B and C as the matching characters and not others we could

'have written \_\*[A-C]oneî (which essentially means \_(A or B or C)+oneî). By enclosing the list in brackets [], we

'ensure that only those characters are used to return a positive match. You hyphen (-) specifies that all the

'characters lying between the staring and the end character can be used. You can also specify a custom list within

'the parenthesis such as [a,b,c,t,r] etc.

'Now suppose rather than including a list of characters, what If you wanted to exclude a certain set? Well you can specify

'an exclusion set by using the ! symbol within the bracket [!]. For example, you can write \*[!a-c] to exclude the characters

'a,b and c from being used as replacements in the wildcard.

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "String Patterns: existence of a word within another."

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "EXACT\_WORD\_IN\_STRING\_FUNC(" & """" & "Existence of a word within another" & """" & ", " & """" & "wrd" & """" & \_

") = " & EXACT\_WORD\_IN\_STRING\_FUNC("Existence of a word within another", "wrd")

Debug.Print "EXACT\_WORD\_IN\_STRING\_FUNC(" & """" & "Existence of a word within another" & """" & ", " & """" & "wrd" & """" & \_

") = " & EXACT\_WORD\_IN\_STRING\_FUNC("Existence of a word within another", "word")

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "String Patterns: Get Word Count"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

'Instead of looping and looking for spaces to calculate number of words in a string, lets use the Split Function to

'tokenize the input String into an array. We'll examine the first letter of each array member. If it is a capital

'letter (A through Z), we increase the word count by one. The Like Function is used to compare the first letter

'of each array member with the alphabet. This way, we don't need to rely on a perfectly trimmed String Ü only

'the first letter of each word is needed.

Debug.Print "WORD\_COUNT\_FUNC(" & """" & "Advanced End User Modeling" & """" & ") = " & \_

WORD\_COUNT\_FUNC("Advanced End User Modeling")

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "String Patterns: Remove unwanted characters from a string"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

'Let's now analyze a function to remove unwanted characters from a string, it involves looping through a string

'and using the Like Operator to determine if any given character is wanted.

Debug.Print "CLEAN\_STRING\_FUNC(" & """" & "abcdefgeuhuep9Y\*&#PO#M@##}{{:><?><}" & """" & ") = " & \_

CLEAN\_STRING\_FUNC("abcdefgeuhuep9Y\*&#PO#M@##}{{:><?><}")

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

Debug.Print "String Patterns: Is Letter and Is Alpha"

Debug.Print "-------------------------------------------------------------------------------------------------------------------------------"

'Lets test the relative speeds of the IS\_LETTER\_FUNC (loop) and IS\_ALPHA\_FUNC (RegEx) functions.

'-----------------------------------------------------------------------------------------------------------------------------------

'Regular expressions (RE) are used for pattern matching purpose in programming. The RegExp object provides a powerful

'parsing tool to efficiently handle string searches or string replacements. Microsoft Word provides a watered down version

'for text searching via Edit'Find'Special.

'The RegExp Object is accessed in Excel (all versions) via either Early or Late Binding.

'Early Binding requires that a VBA reference is set from the Visual Basic Editor via Tools'References'Microsoft VBScript

'Regular Expression 5.5.

'The reference can be set programmatically by running the following:

'ActiveWorkbook.VBProject.References.AddFromGuid "{3F4DACA7-160D-11D2-A8E9-00104B365C9F}", 5, 5

'or: Tools -> References -> Microsoft VbScript Regular Expression 5.5 -> Ok

'The advantage of Early Binding is that it provides VBA intellisense. But as there are only three RegExp Methods, four

'RegExp Properties and two RegExp Collections I prefer Late Binding Set REGEX\_OBJ = CreateObject("vbscript.regexp")

'as it eliminates the need for a user to set the Reference.

'-----------------------------------------------------------------------------------------------------------------------------------

Set REGEX\_OBJ = CreateObject("vbscript.regexp") ' create the RegExp Object with early binding

nLOOPS = 10000

TEXT\_STR = "EventhoughRegularExpressioncanevaluatetheentirestringinonego,theloopwasstillfaster"

'Use Loop

START\_TIMER1 = Timer

For i = 1 To nLOOPS

CHECK\_FLAG = IS\_LETTER\_FUNC(TEXT\_STR, 0)

Next i

END\_TIMER1 = Timer

'Use RegEx

START\_TIMER2 = Timer

For i = 1 To nLOOPS

CHECK\_FLAG = IS\_ALPHA\_FUNC(TEXT\_STR, REGEX\_OBJ)

Next i

END\_TIMER2 = Timer

TEXT\_STR = "Number of iterations: " & nLOOPS & vbCrLf

TEXT\_STR = TEXT\_STR & "Using Loop: " & Format(END\_TIMER1 - START\_TIMER1, "#.###") & " seconds" & vbCrLf

TEXT\_STR = TEXT\_STR & "Using RegEx: " & Format(END\_TIMER2 - START\_TIMER2, "#.###") & " seconds" & vbCrLf

Debug.Print TEXT\_STR

'Even though Regular Expression can evaluate the entire string in one go, the loop was still faster. Refactoring the IS\_ALPHA\_FUNC

'function to create the RegExp object only once, however, and passing it to another function, closed the gap. Keep in mind that the

'RegExp object should be created in a parent procedure and passed to a child function for it to use RegExp parsing methods

End Sub

Function EXACT\_WORD\_IN\_STRING\_FUNC(ByVal TEXT\_STR As String, ByVal WORD\_STR As String) As Boolean

EXACT\_WORD\_IN\_STRING\_FUNC = " " & UCase(TEXT\_STR) & " " Like "\*[!A-Z]" & UCase(WORD\_STR) & "[!A-Z]\*"

'The like operator returns NULL if either of the TEXT\_STR or WORD\_STR strings happen to be blank. The additional spaces before

'and after the strings, therefore, prevent NULL from being returned if the TEXT\_STR or WORD\_STR happens to be blank. The spaces

'before and after the TEXT\_STR string are provided so that the searched for word can be identified when it is the first or

'last word in the TEXT\_STR string. The first word would not have a character in front of it that the \_\*[!A-Z]î part of the

'Like pattern string could match.

End Function

Function WORD\_COUNT\_FUNC(ByVal TEXT\_STR As String) As Long

Dim i As Long

Dim FIRSTL\_STR As String 'firstLetter

Dim WORDS\_ARR() As String

WORDS\_ARR = Split(TEXT\_STR)

For i = LBound(WORDS\_ARR) To UBound(WORDS\_ARR)

FIRSTL\_STR = UCase$(Left$(WORDS\_ARR(i), 1)) ' if it's alphabetic, +1 word

If FIRSTL\_STR Like "[A-Z]" Then 'ignores extra spaces found in between words, as well as punctuation

WORD\_COUNT\_FUNC = WORD\_COUNT\_FUNC + 1

End If

Next i

End Function

Function CLEAN\_STRING\_FUNC(ByVal TEXT\_STR As String) As String

Dim i As Long

Dim CHR\_STR As String

For i = 1 To Len(TEXT\_STR) ' grab each character in turn

CHR\_STR = Mid$(TEXT\_STR, i, 1)

If CHR\_STR Like "[0-9,A-Z,a-z,"" ""]" Then

CLEAN\_STRING\_FUNC = CLEAN\_STRING\_FUNC & CHR\_STR

End If

Next i

'This function doesn't require any workarounds we may tempted to use, like creating an array of "bad" characters

'and comparing each character in the input string with each "bad" character to see if there's a match. That could

'be a lot of looping.

End Function

'Lets take a look at a function for checking if the entire string, regardless of its length, is comprised of letters.

'It doesn't hardcode the string length Ü the loop makes the length of the string superficial. If there is only one

'character, it simply loops once. It uses the Asc function to determine the character code for each character in the

'string, taking advantage of the fact that the letters A-Z and a-z are consecutive.

Function IS\_LETTER\_FUNC(ByVal TEXT\_STR As String, \_

Optional ByVal VERSION As Integer = 0) As Boolean

Dim i As Integer

'This function assumes that the input string contains only letters (IS\_LETTER\_FUNC = True), then set IS\_LETTER\_FUNC

'to False (and immediately exit) if we encounter a non-letter (is that a word?). This way, we only set IS\_LETTER\_FUNC

'to True or False once, instead of wasting processor time setting it to True N times (explicitly sets the return value

'to False, and it repeatedly sets the return value to True).

Select Case VERSION

Case 0

IS\_LETTER\_FUNC = True ' assume True

For i = 1 To Len(TEXT\_STR)

Select Case Asc(Mid(TEXT\_STR, i, 1))

Case 65 To 90, 97 To 122

Case Else ' not a letter, return False

IS\_LETTER\_FUNC = False

Exit For

End Select

Next i

Case Else

IS\_LETTER\_FUNC = Len(TEXT\_STR) > 0 And Not TEXT\_STR Like "\*[!A-Za-z]\*" 'protect against it returning True for the empty string

End Select

End Function

'We can also use Regular Expressions to check if a string contains only letters. This would mean only

'one pass through a string, instead of looping. We need two functions from the Regular Expressions page: GetRegEx and

'TestRegex. Once you copy those functions to a standard module, the following function may be used to check if a given

'string consists only of letters a-z (case insensitive):

Function IS\_ALPHA\_FUNC(ByVal TEXT\_STR As String, \_

ByRef REGEX\_OBJ As Object) As Boolean

Dim MATCH\_OBJ As Object ' MatchCollection

'By Passing REGEX\_OBJ ByRef we avoid instantiating the RegExp object every time it is called

If Not REGEX\_OBJ Is Nothing Then

With REGEX\_OBJ

.Pattern = "^[a-zA-Z]+$" '^[a-zA-Z]+$ --> translates to from the start of the

'string, match any number/combination of upper or lower case letters until the end of the string.

'The test must match the entire string (set by the ^ at the start and the $ at the finish).

.MultiLine = False

End With

' match test string against regex string

Set MATCH\_OBJ = REGEX\_OBJ.Execute(TEXT\_STR)

IS\_ALPHA\_FUNC = (MATCH\_OBJ.Count > 0)

End If

End Function

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Lesson 3.8.15: Regular Expressions in VBA

'-----------------------------------------------------------------------------------------------------------------------------------

'-----------------------------------------------------------------------------------------------------------------------------------

'Regular expressions (RE) are used for pattern matching purpose in programming. The RegExp object provides a powerful

'parsing tool to efficiently handle string searches or string replacements. Microsoft Word provides a watered down version

'for text searching via Edit'Find'Special.

'The RegExp Object is accessed in Excel (all versions) via either Early or Late Binding.

'Early Binding requires that a VBA reference is set from the Visual Basic Editor via Tools'References'Microsoft VBScript

'Regular Expression 5.5.

'The reference can be set programmatically by running the following:

'ActiveWorkbook.VBProject.References.AddFromGuid "{3F4DACA7-160D-11D2-A8E9-00104B365C9F}", 5, 5

'The advantage of Early Binding is that it provides VBA intellisense. But as there are only three RegExp Methods, four

'RegExp Properties and two RegExp Collections I prefer Late Binding Set REGEX\_OBJ = CreateObject("vbscript.regexp")

'as it eliminates the need for a user to set the Reference.

'RegExp Function for testing a given string(s) against a given test pattern

Function REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(ByVal PATTERN\_STR As String, \_

ByVal INPUTS\_STR As String, \_

Optional ByVal DELIM\_STR As String = ",", \_

Optional ByVal HEADINGS\_STR As String = "Source String,Numeric Portion", \_

Optional ByVal MULTILINE\_FLAG As Boolean = True, \_

Optional ByVal GLOBAL\_FLAG As Boolean = True, \_

Optional ByVal IGNORE\_FLAG As Boolean = False, \_

Optional ByVal USED\_STR As String = "", \_

Optional ByVal VERSION As Integer = 0, \_

Optional ByRef REGEX\_OBJ As Object)

Dim i As Long

Dim j As Long

Dim k As Long

Dim l As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim INPUT\_STR As String

Dim OUTPUT\_STR As String

Dim INPUTS\_ARR As Variant

Dim TEMP\_MATRIX() As String

'On Error GoTo ERROR\_LABEL

If InStr(1, INPUTS\_STR, DELIM\_STR) > 1 Then

INPUTS\_ARR = Split(INPUTS\_STR, DELIM\_STR)

Else

ReDim INPUTS\_ARR(1 To 1)

INPUTS\_ARR(1) = INPUTS\_STR

End If

NROWS = UBound(INPUTS\_ARR) - LBound(INPUTS\_ARR) + 1

If Right(HEADINGS\_STR, Len(DELIM\_STR)) <> DELIM\_STR Then: HEADINGS\_STR = HEADINGS\_STR & DELIM\_STR

NCOLUMNS = 0

For i = 1 To Len(HEADINGS\_STR)

If Mid(HEADINGS\_STR, i, Len(DELIM\_STR)) = DELIM\_STR Then: NCOLUMNS = NCOLUMNS + 1

Next i

ReDim TEMP\_MATRIX(0 To NROWS, 1 To NCOLUMNS)

i = 1

For k = 1 To NCOLUMNS

j = InStr(i, HEADINGS\_STR, DELIM\_STR)

TEMP\_MATRIX(1, k) = Mid(HEADINGS\_STR, i, j - i)

i = j + Len(DELIM\_STR)

Next k

'Dim REGEX\_OBJ As Object

'Dim REGEX\_OBJ As VBScript\_RegExp\_55.RegExp 'Reference must be set to Microsoft VbScript Regular Expression 5.5

If REGEX\_OBJ Is Nothing Then: Set REGEX\_OBJ = CreateObject("vbscript.regexp") ' create the RegExp Object with late binding

'-------------------------------------------------------------------------------------

'RegExp Method Properties

'-------------------------------------------------------------------------------------

With REGEX\_OBJ

'-------------------------------------------------------------------------------------

'Global True or False (default).

'-------------------------------------------------------------------------------------

'True finds all matches in a string

'False finds the first match only

.Global = GLOBAL\_FLAG 'look for global matches

'-------------------------------------------------------------------------------------

'IgnoreCase True or False (default).

'-------------------------------------------------------------------------------------

'True ignores case

'False is case sensitive

.IgnoreCase = IGNORE\_FLAG

'-------------------------------------------------------------------------------------

'MultiLine True or False (default).

'-------------------------------------------------------------------------------------

'True will search each line of a multline string

'False searches only the current line

.MultiLine = MULTILINE\_FLAG

'-------------------------------------------------------------------------------------

'Patterns see table below

'-------------------------------------------------------------------------------------

.Pattern = PATTERN\_STR

'Character --> Description

'\ Marks the next character as either a special character or a literal. For example, "n" matches the character "n".

'"\n" matches a newline character. The sequence "\\" matches "\" and "\(" matches "(".

'^ Matches the beginning of input.

'$ Matches the end of input.

'\* Matches the preceding character zero or more times. For example, "zo\*" matches either "z" or "zoo".

'+ Matches the preceding character one or more times. For example, "zo+" matches "zoo" but not "z".

'? Matches the preceding character zero or one time. For example, "a?ve?" matches the "ve" in "never".

'. Matches any single character except a newline character.

'(pattern) Matches pattern and remembers the match. The matched substring can be retrieved from the resulting Matches

'collection, using Item [0]...[n]. To match parentheses characters ( ), use "\(" or "\)".

'x|y Matches either x or y. For example, "z|wood" matches "z" or "wood". "(z|w)oo" matches "zoo" or "wood".

'{n} n is a nonnegative integer. Matches exactly n times. For example, "o{2}" does not match the "o" in "Bob," but matches

'the first two o's in "foooood".

'{n,} n is a nonnegative integer. Matches at least n times. For example, "o{2,}" does not match the "o" in "Bob" and

'matches all the o's in "foooood." "o{1,}" is equivalent to "o+". "o{0,}" is equivalent to "o\*".

'{n,m} m and n are nonnegative integers. Matches at least n and at most m times. For example, "o{1,3}" matches the first

'three o's in "fooooood." "o{0,1}" is equivalent to "o?".

'[xyz] A character set. Matches any one of the enclosed characters. For example, "[abc]" matches the "a" in "plain".

'[^xyz] A negative character set. Matches any character not enclosed. For example, "[^abc]" matches the "p" in "plain".

'[a-z] A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase

'alphabetic character in the range "a" through "z".

'[^m-z] A negative range characters. Matches any character not in the specified range. For example, "[m-z]" matches any

'character not in the range "m" through "z".

'\b Matches a word boundary, that is, the position between a word and a space. For example, "er\b" matches the "er" in "never"

'but not the "er" in "verb".

'\B Matches a non-word boundary. "ea\*r\B" matches the "ear" in "never early".

'\d Matches a digit character. Equivalent to [0-9].

'\D Matches a non-digit character. Equivalent to [^0-9].

'\f Matches a form-feed character.

'\n Matches a newline character.

'\r Matches a carriage return character.

'\s Matches any white space including space, tab, form-feed, etc. Equivalent to "[ \f\n\r\t\v]".

'\S Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]".

'\t Matches a tab character.

'\v Matches a vertical tab character.

'\w Matches any word character including underscore. Equivalent to "[A-Za-z0-9\_]".

'\W Matches any non-word character. Equivalent to "[^A-Za-z0-9\_]".

'\num Matches num, where num is a positive integer. A reference back to remembered matches. For example, "(.)\1"

'matches two consecutive identical characters.

'\n Matches n, where n is an octal escape value. Octal escape values must be 1, 2, or 3 digits long. For example, "\11"

'and "\011" both match a tab character. "\0011" is the equivalent of "\001" & "1". Octal escape values must not exceed

'256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.

'\xn Matches n, where n is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For

'example, "\x41" matches "A". "\x041" is equivalent to "\x04" & "1". Allows ASCII codes to be used in regular expressions.

'For further reference see: http://msdn.microsoft.com/library/default.asp?url=/library/en us/script56/html/vsmthreplace.asp

End With

'-------------------------------------------------------------------------------------------------------------

j = 1

For i = LBound(INPUTS\_ARR) To UBound(INPUTS\_ARR)

INPUT\_STR = INPUTS\_ARR(i)

'-------------------------------------------------------------------------------------------------------------

Select Case VERSION

'-------------------------------------------------------------------------------------------------------------

Case 0 'The Test Method is used to test whether a Regular Expression pattern is matched in a string. The Test

'Method returns True or False. It is equivalent to testing whether the number of matches found is greater than 0.

OUTPUT\_STR = REGEX\_OBJ.Test(INPUT\_STR)

'-------------------------------------------------------------------------------------------------------------

Case 1 'The Replace method is used to find a match or matches of a Regular Expression pattern in a string and then

'replace them with a new string.

OUTPUT\_STR = REGEX\_OBJ.Replace(INPUT\_STR, USED\_STR) 'USED\_STR -> Replace Str

'-------------------------------------------------------------------------------------------------------------

Case Else 'The Execute method is used to extract a match or matches of a Regular Expression pattern from a string.

'RegExp Collections: A Matches collection contains individual Match objects, and can be only created using the Execute

'method of the RegExp object. There are three useful properties of Regular Expressions Methods. (1) FirstIndex: The

'string position at which the match occurs; (2) Count: The number of matches in the string; (3) Length: The length of

'the match.

Dim COLLECTION\_OBJ As Object 'Late Binding

'Dim COLLECTION\_OBJ As VBScript\_RegExp\_55.MatchCollection 'Reference must be set to Microsoft VbScript Regular Expression 5.5

Dim MATCH\_OBJ As Object 'Late Binding

'Dim MATCH\_OBJ As VBScript\_RegExp\_55.Match 'Reference must be set to Microsoft VbScript Regular Expression 5.5

Set COLLECTION\_OBJ = REGEX\_OBJ.Execute(INPUT\_STR) ' match test string against REGEX\_OBJ string

If COLLECTION\_OBJ.Count > 0 Then 'Test to see if a match exists and if it does then create a new string by looping

OUTPUT\_STR = ""

If VERSION = 2 Then

For Each MATCH\_OBJ In COLLECTION\_OBJ 'Loop through each match in the collection and concatenate them

OUTPUT\_STR = OUTPUT\_STR & USED\_STR & MATCH\_OBJ 'USED\_STR -> Separator String

Next MATCH\_OBJ

ElseIf VERSION = 3 Then 'Reverse

For k = COLLECTION\_OBJ.Count - 1 To 0 Step -1 'The first match is number 0

OUTPUT\_STR = OUTPUT\_STR & USED\_STR & COLLECTION\_OBJ(k) 'USED\_STR -> Separator String

Next k

Else 'SubMatches --> A SubMatches collection contains individual submatch strings, and can only be created using the

'Execute method of the RegExp object. Note that the submatch strings can also be accessed but not created via the

'Match method of the RegExp object.

Dim SUBMATCH\_OBJ As Object 'Late Binding

'Dim SUBMATCH\_OBJ As VBScript\_RegExp\_55.SubMatches 'Reference must be set to Microsoft VbScript Regular Expression 5.5

Dim ITEM\_OBJ As Variant

For Each MATCH\_OBJ In COLLECTION\_OBJ 'Loop through each match in the collection and concatenate them

l = l + 1

Set SUBMATCH\_OBJ = MATCH\_OBJ.SubMatches ' create the Sub Match collection

For Each ITEM\_OBJ In SUBMATCH\_OBJ ' loop through each Sub Match

OUTPUT\_STR = OUTPUT\_STR & "Match " & l & " contains submatch " & ITEM\_OBJ & vbCrLf

Next ITEM\_OBJ

Next MATCH\_OBJ

End If

Else

OUTPUT\_STR = ""

End If

'-------------------------------------------------------------------------------------------------------------

End Select

'-------------------------------------------------------------------------------------------------------------

TEMP\_MATRIX(j, 1) = INPUT\_STR

TEMP\_MATRIX(j, 2) = OUTPUT\_STR

j = j + 1

Next i

GoSub RESET\_LINE

REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC = TEMP\_MATRIX

'--------------------------------------------------------------------------------------------------------------------

Exit Function

'--------------------------------------------------------------------------------------------------------------------

RESET\_LINE:

'--------------------------------------------------------------------------------------------------------------------

Set REGEX\_OBJ = Nothing

Set MATCH\_OBJ = Nothing

Set COLLECTION\_OBJ = Nothing

'--------------------------------------------------------------------------------------------------------------------

Return

'--------------------------------------------------------------------------------------------------------------------

ERROR\_LABEL:

'--------------------------------------------------------------------------------------------------------------------

GoSub RESET\_LINE

REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC = Err.Number

End Function

'The following function provides examples that cover the RegExp Execute, Replace and Test Methods. The examples

'also show how to create and access submatches and backreferences.

Sub TEST1\_REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC()

Dim i As Long

Dim j As Long

Dim k As Long

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim PRINT\_STR As String

Dim HEADINGS\_STR As String

Dim INPUTS\_STR As String

Dim PATTERN\_STR As String

Dim INPUTS\_ARR As Variant

Dim PATTERNS\_ARR As Variant

Dim TEMP\_MATRIX() As String

'Quick view on basic principles for understanding Patterns:

'\*\*\* : means a range.

' E.g., a-z means all the letters a to z.

'[] : means exactly one of the letters quoted.

' E.g., [abc] means exactly one of a, b or c.

' [A-Z] means exactly one of A, B, ..., Z

'() : just for grouping purpose.

'| : means "or".

' E.g. a|b means a or b.

'+ : means one or more of the pattern before it.

' E.g., a+ means one or more consecutive a. (ab)+ means ab, abab, ababab, ..

'\* : means zero or more.

' E.g., [a-z]\* means empty string, or any string consisting of a, .., z.

'? : means zero or one.

' E.g., [a-z]? means empty string, or any one of a, b, .., z.

'--> Single-character patterns:

'. : a single dot. Matches any character except newline (\n).

' E.g., "x." matches a string of length 2, first letter is 'x', second is any letter except '\n'.

'[] : character class, a list of characters enclosed by '[' and ']'.

' Exactly one of these characters is used.

' E.g., [abcdef], [a-z], [A-Z], [0123456789], [0-9],

' [a-zA-Z], [a-zA-Z0-9\_].

'^ : used at the beginning of a character class, means the negated character class.

' E.g., [^0-9], [^abcdABCD], [^a-z], [^\^].

'Some predefined character class abbreviations:

'abr: same as | meaning

'------------------------------

'\d : [0-9] : a digit

'\D : [^0-9] : non-digits

'\w : [a-zA-Z0-9\_] : word char

'\W : [^a-zA-Z0-9\_] : non-word char

'\s : [ \r\t\n\f] : space char

'\S : [^ \r\t\n\f] : non-space char

'\n : [\n] : new line

'--> Grouping patterns:

'\*\*\* Sequence: a continuous series of patterns.

' E.g., abc, 123456

'\*\*\* Multiplier, repeated copies of the immediately previous pattern

' {min, max} : general multiplier: min to max copies

' {n} : exactly n copies.

' \* : zero or more. same as {0,}.

' + : one or more. same as {1,}.

' ? : zero or one. same as {0,1}.

' E.g., [a]\*, a+, a?, [a]{0,}, a{1,}, a{0,1}.

'\*\*\* Memory: remember the pattern quoted by '(' and ')', the remembered

' patterns are refered to as \1, \2, \3, ... in the pattern string,

' and also can be refered to by $1, $2, $3, ... later.

' E.g., fred(.)barney\1, this matches "fredxbarneyx".

' and now the value of $1 is "x".

'\*\*\* Alternation: match exactly one of the alternatives.

' |

' E.g., red|green|blue

'--> Anchoring patterns:

' ^ : marks the beginning of a pattern.

' E.g., ^a, the 'a' must appear at the beginning of the matched string.

' $ : marks the end of a pattern.

' E.g., a$, the 'a' must appear at the end of the matched string.

' \b : marks a word boundary (a spot between 2 characters that has a

' \w on one side of it, and a \W on the other side).

' \B : marks a non-boundary of word.

'--> Other issues:

'\*\*\* Escape sequence:

' To escape special charaters, e.g., those used above.

' E.g., \., \[, \], \(, \), \\, \\*, \+, \?, \^, \$.

'\*\*\* Case sensitivity: Specify whether case (upper case or lower case) matters.

'\*\*\* Global v.s. Local: Mostly used for replacing: replace the first match (local)

' or all the matches (global)?

'\*\*\* Precedence: which combination of the patterns match first?

' e.g., a|b\* - is this (a|b)\* or a|(b\*) ?

' The precedence table of most common regex grouping patterns is:

' -----------------------------------------------------

' Name | Representation

' -----------------------------------------------------

' 1) Parentheses | ( )

' 2) Multipliers | ? + \* {m,n} {m, n}?

' 3) Sequence and anchoring | abc ^ $

' 4) Alternation | |

' -----------------------------------------------------

'--> WARNING: Limitations of regular expression

'\*\*\* Some patterns cannot be matched by regular expressions.

' One example involves counting the number of occurences of a pattern.

' e.g. It can 't match a pattern anbn, where a and b are

' both repeated n times. This is because regular expression can't

' keep track of the number of occurences.

' Similarly, anbncn ... also cannot be matched.

'\*\*\* When there are lots of data to process, improper use of the regular

' expression pattern may lead to low efficiency of the program.

'------------------------------------------------------------------------------------------------------------------------

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "1. Extracting the alpha-numeric portion of a string" & vbCrLf

INPUTS\_STR = "1a2b 3c4d"

HEADINGS\_STR = "Source String,Numeric Portion"

PATTERN\_STR = "(\d+)([a-z]+)"

'(\d+) --> means one or more consecutive digits

'([a-z]+) --> means one or more consecutive letters a to z

PRINT\_STR = "Execute Method." & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, False, True, True, "", 4)

'This routine loops through the INPUTS string and uses the RegExp Execute method to extract any groups

'of numbers as matches. Then concatenates the RegExp Match Collection match for each Input cell

'into the corresponding Output cell. Note that the numbers could have been extracted individually or

'in their groups rather than as a total.

'If GLOBAL\_FLAG = False then only the first match would be extracted

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "2. Regular Expressions Examples" & vbCrLf

PATTERNS\_ARR = Split("[0-9]{1,3} ^abc x. [a-z][A-Z][0-9] [a-z]+ (a|b)? (a|b)+ [^abc] [^abc]$ ^abc \$$ (.)(.)(.).+\3\2\1", " ")

'[0-9]{1,3} --> look for 1 to 3 numeric digits

'^abc --> Finds abc that occurs at the beginning of a line

'x. --> Matches any character except newline (\n) after x

'[a-z][A-Z][0-9] --> A range of 3 characters (or more) starting with a lower case, uppercase, and a digit.

'[a-z]+ --> one or more consecutive characters

'(a|b)? --> zero or one 'a' or 'b'

'(a|b)+ --> one or more consecutive 'a' or 'b'

'[^abc] '--> Finds any single character except a, b, c

'[^abc]$ '--> Finds any single character except a, b, c at the end of a line.

'^abc --> Finds abc that occurs at the beginning of a line

'\$$ '--> Finds $ character at the end of a line.

INPUTS\_ARR = Split("12,abc123,xy,dA3,abc123,b,abcd,a,b,abc,($),12abcdcba56", ",")

For k = LBound(PATTERNS\_ARR) To UBound(PATTERNS\_ARR) - 1

PRINT\_STR = ""

INPUTS\_STR = CStr(INPUTS\_ARR(k))

HEADINGS\_STR = "Source String,Numeric Portion"

PATTERN\_STR = CStr(PATTERNS\_ARR(k))

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, False, "", 0) 'RegExp Test method

GoSub PRINT\_LINE

Next k

PRINT\_STR = ""

INPUTS\_STR = CStr(INPUTS\_ARR(k))

HEADINGS\_STR = "Source String,Numeric Portion"

PATTERN\_STR = CStr(PATTERNS\_ARR(k))

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, False, "$1+$2", 1)

GoSub PRINT\_LINE

'Input Str: 12abcdcba56

'Pattern: (.)(.)(.).+\3\2\1

'Matched pattern: abcdcba

'(.)(.)(.) --> 'abc' --> Matches any 3 character except newline (\n)

'.+ --> 'dcba' --> means one or more consecutive character except newline (\n).

'Remember the pattern quoted by '(' . ')', the remembered

'patterns are refered to as \1, \2, \3, ... in the pattern string,

'and also can be refered to by $1, $2, $3, ... later.

'$1 -> a

'$2 -> b

'$3 -> c

'Replace with $1+$2

'Replace matched pattern with $1+$2 --> abcdcba --> a+b

'Replace Result: 12a+b56

'Replace matched pattern with $1+$2+$3 --> abcdcba --> a+b+c

'Replace Result: 12a+b+c56

'Remember from the basic principles:

'a single dot matches any character except newline (\n).

'E.g., fred(.)barney\1, this matches "fredxbarneyx".

'and now the value of $1 is "x".

Exit Sub

PRINT\_LINE:

SROW = LBound(TEMP\_MATRIX, 1): NROWS = UBound(TEMP\_MATRIX, 1)

SCOLUMN = LBound(TEMP\_MATRIX, 2): NCOLUMNS = UBound(TEMP\_MATRIX, 2)

For i = SROW + 1 To NROWS 'Exclude Headings

PRINT\_STR = PRINT\_STR & "Pattern: " & PATTERN\_STR & vbCrLf

PRINT\_STR = PRINT\_STR & "Input Str: " & TEMP\_MATRIX(i, SCOLUMN) & vbCrLf

PRINT\_STR = PRINT\_STR & "Output Str: "

For j = SCOLUMN + 1 To NCOLUMNS

PRINT\_STR = PRINT\_STR & TEMP\_MATRIX(i, j) & Chr(9)

Next j

PRINT\_STR = PRINT\_STR & vbCrLf & vbCrLf

Next i

Debug.Print PRINT\_STR

Return

End Sub

Sub TEST2\_REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC()

'1. Extracting the numeric portion of a string.

'2. Reversing the order of a persons name.

'3. Testing whether the first three characters at the start of a string are found at the end of a string in reverse order

'4. Testing validity of a numeric string input

'5. Removing back to back alphanumeric strings

Dim i As Long

Dim j As Long

Dim k As Long

Dim SROW As Long

Dim NROWS As Long

Dim SCOLUMN As Long

Dim NCOLUMNS As Long

Dim PRINT\_STR As String

Dim HEADINGS\_STR As String

Dim INPUTS\_STR As String

Dim PATTERN\_STR As String

Dim TEMP\_MATRIX() As String

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "1. Extracting the numeric portion of a string" & vbCrLf

INPUTS\_STR = "12 FredBlog 2004,1234 aaa 78,A19,No Number,1/2,0.01,99.9,A12:B10,1a2b3c4d,2\*6"

HEADINGS\_STR = "Source String,Numeric Portion"

PATTERN\_STR = "[^\d]+" 'look for strings that are not numeric

PRINT\_STR = "Replace Method for strings that are not numeric." & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, False, "", 1)

'The routine loops through the INPUTS string and uses the RegExp Replace method to find any characters that are

'not numeric and replaces them with "" . If the RegExp Global property was set to False then only the

'first set of non-numeric data would be removed. This method is more efficient in this case than Execute

'as there is no need to handle the Match Collection.

'If GLOBAL\_FLAG = False then only the first set of non-numeric data would be removed.

GoSub PRINT\_LINE

PATTERN\_STR = "\d+" 'look for numeric strings of any length

PRINT\_STR = "Execute Method for numeric strings of any length." & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, False, "", 2)

'This routine loops through the INPUTS string and uses the RegExp Execute method to extract any groups

'of numbers as matches. Then concatenates the RegExp Match Collection match for each Input cell

'into the corresponding Output cell. Note that the numbers could have been extracted individually or

'in their groups rather than as a total.

'If GLOBAL\_FLAG = False then only the first match would be extracted

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "2. Reversing the order of a persons name." & vbCrLf

INPUTS\_STR = "Fermin Nicolas Mr,Miller Andrew Mr,Fletcher Tamara Ms,VanderHoeven Alex Mr"

HEADINGS\_STR = "Source String,Name Reversed"

'These routines loop through the INPUTS string and uses the RegExp Execute method to extract any alphabetical group

'matches. It then concatenates the RegExp Match Collection match for each B cell into the corresponding C cell.

'Note that the numbers could have been extracted individually or in their groups.

'If GLOBAL\_FLAG = False then only the first match would be extracted.

PATTERN\_STR = "([a-z]+)(\s)([a-z]+)(\s)([a-z]+)" 'look for 5 substrings consisting of word-space-word-space-word

PRINT\_STR = "Replace Method for 5 substrings consisting of word-space-word-space-word." & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, True, "$5$4$3$2$1", 1)

'Since REPLACE\_STR = $5$4$3$2$1 submatches can be referred to as $X where X is the substring position

'Reverse submatch order starting with 5th match and ending with 1st match.

GoSub PRINT\_LINE

PATTERN\_STR = "[a-z]+" 'look for any word string

PRINT\_STR = "Execute Method for any word string." & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, True, True, True, " ", 3)

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "3. Testing whether the first three characters at the start of a string are found at the end of a string in reverse order." & vbCrLf

INPUTS\_STR = "12 Fred 212,221 Gray 122,167 Paul 5167,abc def cba,yesasey,notme,ghighi,ACDC,Glenelg,1010001"

HEADINGS\_STR = "String, Test"

PATTERN\_STR = "^(.)(.)(.).+(\3\2\1)$" 'Create sub matches of the first three characters and then look for them in reverse

'order at the end of the string. Ignore anything in between the first three and last three characters. The test must match

'the entire string (set by the ^ at the start and the $ at the finish)

PRINT\_STR = "Test Method - Create sub matches of the first three characters and then look for them in reverse " & \_

"order at the end of the string. Ignore anything in between the first three and last three characters." & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, False, False, True, "", 0)

'This routine loops through INPUTS string and uses the RegExp Test method to find if the first three characters are repeated in

'reverse order at the end of the string. It uses Back References to match the first three Sub Matches. It returns False if

'order is not reversed or if the string is less than six characters long.

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "4. Testing the validity of a required numeric string input." & vbCrLf

INPUTS\_STR = "123.12|124.|-21.344|2,2222|3a|.012|0.012|-111222.2|12.34|1,234,212.33" 'Here we change the delimiter character. Why?

HEADINGS\_STR = "Source String|Test Numeric Format"

PATTERN\_STR = "^-?(\d{1,3})+(\,?\d{3})\*(\.\d+|)$" 'Accept either a negative sign or no sign at the start of the string.

'Then look for 1 to 3 numeric digits; if commas exist in the string then the integer portion must have commas every 3

'significant digits; if a decimal point exists then it must have a number before it (0-9) and a fractional remainder.

'The test must match the entire string (set by the ^ at the start and the $ at the finish).

PRINT\_STR = "Test Method - Accept either a negative sign or no sign at the start of the string. " & \_

"Then look for 1 to 3 numeric digits; if commas exist in the string then the integer portion must have commas every 3 " & \_

"significant digits; if a decimal point exists then it must have a number before it (0-9) and a fractional remainder" & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, "|", HEADINGS\_STR, False, False, True, "", 0)

'The routine loops through INPUTS string and uses the RegExp Test method to find if the number format is valid.

'To return True:

'- The string must be numeric.

'- If the number string contains a decimal point then it must have a number preceding and following the decimal point.

'- If the string contains a comma then the whole integer portion must be formatted with commas every 3 significant digits.

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Debug.Print "5. Removing duplicated alphanumeric strings." & vbCrLf

INPUTS\_STR = "the the great great great students,everybody is getting 10 10 out of 10,No not more classes,Nico nico fermin,1 2 3 3 4,3 1 2 3,No no No Mr Fawlty,the fast faster"

HEADINGS\_STR = "Source String,Error Tested String"

PATTERN\_STR = "\b(\w+)\b(\s+\1\b)+" 'Look for a word boundary, then an alphanumeric string, then a word boundary.

'Then look for a space, then test if the alphanumeric word is repeated.

PRINT\_STR = "Replace Method for a word boundary, then an alphanumeric string, then a word boundary. " & \_

"Then look for a space, then test if the alphanumeric word is repeated" & vbCrLf & vbCrLf

TEMP\_MATRIX = REGEXP\_EXECUTE\_VS\_REPLACE\_FUNC(PATTERN\_STR, INPUTS\_STR, ",", HEADINGS\_STR, False, True, True, "$1", 1)

'This routine loops through INPUTS string and uses the RegExp Replace method with Back References to remove any consecutive

'alphanumeric strings.

GoSub PRINT\_LINE

'------------------------------------------------------------------------------------------------------------------------

Exit Sub

PRINT\_LINE:

SROW = LBound(TEMP\_MATRIX, 1): NROWS = UBound(TEMP\_MATRIX, 1)

SCOLUMN = LBound(TEMP\_MATRIX, 2): NCOLUMNS = UBound(TEMP\_MATRIX, 2)

For i = SROW + 1 To NROWS 'Exclude Headings

PRINT\_STR = PRINT\_STR & "Pattern: " & PATTERN\_STR & vbCrLf

PRINT\_STR = PRINT\_STR & "Input Str: " & TEMP\_MATRIX(i, SCOLUMN) & vbCrLf

PRINT\_STR = PRINT\_STR & "Output Str: "

For j = SCOLUMN + 1 To NCOLUMNS

PRINT\_STR = PRINT\_STR & TEMP\_MATRIX(i, j) & Chr(9)

Next j

PRINT\_STR = PRINT\_STR & vbCrLf & vbCrLf

Next i

Debug.Print PRINT\_STR

Return

End Sub

'The following example uses the RegExp Execute method to convert the cell reference in a range of Excel

'Formulae from relative to absolute reference. It specifies what cell reference to modify and whether the

'reference should be Relative, Absolute, Row Absolute or Column Absolute.

'For further reference see:

'http://en.wikipedia.org/wiki/Regular\_expression

Sub TEST\_MODIFIED\_CELL\_REFERENCE\_FUNC()

Dim i As Long

Dim j As Long

Dim l As Long

Dim NROWS As Long

Dim NCOLUMNS As Long

Dim FORMULAS\_STR As String

Dim DST\_RNG As Range

Dim SRC\_WSHEET As Worksheet

Set SRC\_WSHEET = Worksheets("WSHEET\_TEST")

With SRC\_WSHEET

With .Cells

.Clear

.ColumnWidth = 10

End With

.Cells(1, 1) = "MODIFIED\_CELL\_REFERENCE\_FUNC"

.Cells(2, 1) = "Type"

.Cells(2, 2) = "Index"

FORMULAS\_STR = "=$A$4|=SUMPRODUCT($H$11:I19,K11:L19)|=$B$5+1/A10|=($A$3)+$B$3+F4|" & \_

"=10^$B$5+A11|=+SUM($A$4:B4,C4:E4)|=INDIRECT(" & """" & "AB123456" & """" & ")|" & \_

"=INDIRECT(" & """" & "AB99999" & """" & ")|=INDIRECT(" & """" & "$AB$12345" & """" & ")|" & \_

"=" & """" & "NICO" & """" & "|"

.Cells(5, 1) = "Formula"

Set DST\_RNG = Range(.Cells(6, 1), .Cells(15, 1))

i = 1

For l = 1 To 10

j = InStr(i, FORMULAS\_STR, "|")

DST\_RNG.Cells(l, 1).Formula = Mid(FORMULAS\_STR, i, j - i)

i = j + 1

Next l

With Range(.Cells(5, 3), .Cells(15, 4))

.FormulaArray = "=MODIFIED\_CELL\_REFERENCE\_FUNC(" & DST\_RNG.Address & "," & \_

SRC\_WSHEET.Cells(3, 1).Address & "," & SRC\_WSHEET.Cells(3, 2).Address & ")"

.ColumnWidth = 25

End With

.Cells(3, 1) = 0

.Cells(3, 2) = 1

End With

End Sub

Function MODIFIED\_CELL\_REFERENCE\_FUNC(ByRef DATA\_RNG As Excel.Range, \_

Optional ByVal REFERENCE\_TYPE As Integer = 1, \_

Optional ByVal REFERENCE\_INDEX\_VAL As Integer = 1, \_

Optional ByVal REGEX\_OBJ As Object)

'REFERENCE\_TYPE: Cell Reference Type

'0: Abs

'1: Rel

'2: Row

'3: Col

'REFERENCE\_INDEX\_VAL: Formula Precedent to change

Dim i As Integer

Dim j As Integer

Dim k As Integer

Dim NSIZE As Integer

Dim DATA\_CELL As Excel.Range

Dim PATTERN\_STR As String

Dim REPLACE\_STR As String

Dim INPUT\_STR As String

'Dim REGEX\_OBJ As Object

Dim SUBMATCH\_OBJ As Object

Dim COLLECTION\_OBJ As Object

Dim END\_FLAG As Boolean

Dim LONG\_FLAG As Boolean

Dim NEW\_TEXT\_STR As String

'i: PosMatch

'j: LenMatch

'END\_FLAG: EndString

'LONG\_FLAG: LongString

On Error GoTo ERROR\_LABEL

GoSub INIT\_LINE

NSIZE = DATA\_RNG.Cells.Count

ReDim TEMP\_MATRIX(0 To NSIZE, 1 To 2)

TEMP\_MATRIX(0, 1) = "Original Formula"

TEMP\_MATRIX(0, 2) = "Modified Formula"

For k = 1 To NSIZE

Set DATA\_CELL = DATA\_RNG.Cells(k)

If DATA\_CELL.HasFormula = True Then

INPUT\_STR = DATA\_CELL.Formula

NEW\_TEXT\_STR = ""

GoSub REGEX\_LINE

TEMP\_MATRIX(k, 1) = INPUT\_STR

If NEW\_TEXT\_STR = "" Then

TEMP\_MATRIX(k, 2) = NEW\_TEXT\_STR

Else 'Make the change to the formula using the position and length of the match

TEMP\_MATRIX(k, 2) = Application.WorksheetFunction.Replace(INPUT\_STR, i, j, NEW\_TEXT\_STR)

End If

Else

TEMP\_MATRIX(k, 1) = ""

TEMP\_MATRIX(k, 2) = ""

End If

Next k

GoSub RESET\_LINE

MODIFIED\_CELL\_REFERENCE\_FUNC = TEMP\_MATRIX

'--------------------------------------------------------------------------------------------------------------------

Exit Function

'--------------------------------------------------------------------------------------------------------------------

INIT\_LINE:

'--------------------------------------------------------------------------------------------------------------------

If REGEX\_OBJ Is Nothing Then: Set REGEX\_OBJ = CreateObject("vbscript.regexp") ' create the RegExp Object with early binding

If REFERENCE\_INDEX\_VAL < 1 Then: REFERENCE\_INDEX\_VAL = 1

If REFERENCE\_INDEX\_VAL > 4 Then: REFERENCE\_INDEX\_VAL = 4

Select Case LCase(REFERENCE\_TYPE)

Case 0

REPLACE\_STR = "abs"

Case 1

REPLACE\_STR = "rel"

Case 2

REPLACE\_STR = "row"

Case Else

REPLACE\_STR = "col"

End Select

PATTERN\_STR = "(?:[\^\])-/+\*:,=" & """" & "[(])(\$?)([A-Z]{1,2})(\$?)(\d{1,5})" '-> (?:[\^\])-/+\*:,="[(])(\$?)([A-Z]{1,2})(\$?)(\d{1,5})([^\d]|$)

With REGEX\_OBJ

.Global = True 'Match case to avoid matching lower case alphabetic charcaters

.IgnoreCase = False 'Look for cell reference

.Pattern = PATTERN\_STR 'Look for all formula references

End With

'--------------------------------------------------------------------------------------------------------------------

Return

'--------------------------------------------------------------------------------------------------------------------

REGEX\_LINE:

'--------------------------------------------------------------------------------------------------------------------

Set COLLECTION\_OBJ = REGEX\_OBJ.Execute(INPUT\_STR)

'If the formula is not long enough then dont look at submatches

If COLLECTION\_OBJ.Count >= REFERENCE\_INDEX\_VAL Then

'Parse the appropriate formula reference

Set SUBMATCH\_OBJ = COLLECTION\_OBJ(REFERENCE\_INDEX\_VAL - 1).SubMatches

If SUBMATCH\_OBJ(3) < 65537 Then 'If row number exceeds 65536 then the parsed string is not a valid formula

' Check to see whether the the first character after the SUBMATCH\_OBJ of the row number

' is the end of the string or is numeric

' If the first character is numeric then the row match is invalid

i = COLLECTION\_OBJ(REFERENCE\_INDEX\_VAL - 1).FirstIndex + 2

j = COLLECTION\_OBJ(REFERENCE\_INDEX\_VAL - 1).Length - 1

LONG\_FLAG = False

END\_FLAG = Len(INPUT\_STR) = i + j - 1

If Not (END\_FLAG) Then

LONG\_FLAG = Not (IsNumeric(Mid(INPUT\_STR, i + j, 1)))

End If

If END\_FLAG Or LONG\_FLAG Then

' Change the formula to Relative, Absolute, Column Absolute or Row Absolute

Select Case REPLACE\_STR

' Four valid arguments:

Case "rel" 'Relative Referenced

NEW\_TEXT\_STR = SUBMATCH\_OBJ(1) & SUBMATCH\_OBJ(3)

Case "abs" 'Absolute Referenced

NEW\_TEXT\_STR = "$" & SUBMATCH\_OBJ(1) & "$" & SUBMATCH\_OBJ(3)

Case "col" 'Absolute Column Referenced

NEW\_TEXT\_STR = "$" & SUBMATCH\_OBJ(1) & SUBMATCH\_OBJ(3)

Case "row" 'Absolute Row Referenced

NEW\_TEXT\_STR = SUBMATCH\_OBJ(1) & "$" & SUBMATCH\_OBJ(3)

Case Else

'conversion input was invalid so return initial string

End Select

End If

End If

End If

'--------------------------------------------------------------------------------------------------------------------

Return

'--------------------------------------------------------------------------------------------------------------------

RESET\_LINE:

'--------------------------------------------------------------------------------------------------------------------

Set REGEX\_OBJ = Nothing

Set COLLECTION\_OBJ = Nothing

Set SUBMATCH\_OBJ = Nothing

'--------------------------------------------------------------------------------------------------------------------

Return

'--------------------------------------------------------------------------------------------------------------------

ERROR\_LABEL:

'--------------------------------------------------------------------------------------------------------------------

GoSub RESET\_LINE

MODIFIED\_CELL\_REFERENCE\_FUNC = Err.Number

End Function