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

'Last Update: Sept 25, 2013

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

Option Explicit

Option Base 1

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

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

'Lesson 3.4: Calculation Mode and Excel Optimization

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

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

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

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

'Lesson 3.4.1: How to properly set calculation mode in Excel VBA

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

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

'Most people set calculation mode to 'manual' at the beginning of the code, then set it to 'automatic' at the

'end. This assumes that the setting was 'automatic' in the first place and the user wants the calculation mode

'to be 'automatic' after the code finishes. While I believe that in 99.9% of cases, calculation should always

'be set to automatic (see http://www.ozgrid.com/Excel/ExcelSpreadsheetDesign.htm for explanation), your code

'should never make unrequested changes without notifying the user. Here is how to properly set the calculation

'mode so it is restored to whatever it was previously.

Sub Test\_CalcMode()

Dim CalcMode As Long

' get current calculation state, save for later

CalcMode = Application.Calculation

' set to manual for the duration of the code

Application.Calculation = xlCalculationManual

'your code here

' set it back to whatever it was before the code was run.

Application.Calculation = CalcMode

End Sub

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

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

'Lesson 3.4.2: Optimize your macros

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

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

'This can be cut and pasted into your routines to make them run faster. It will stop events, disable automatic

'calculation and screen updating and, if any dialog boxes pop up, it gives the default answer silently. Keep

'in mind that in some cases, you may actually want some of these things to happen (i.e. your code may depend

'on calculations made in the workbook during processing). In that case, simply remove the parts you don't

'want. In keeping with the spirit of storing and restoring settings to avoid disrupting the user, the code

'uses variables to store the current states of each property, turns them off for the duration of the code,

'then restores them to whatever they were previously. That being said, sometimes you want calculation, events,

'etc to occur. I recommend leaving calculation at 'automatic' under most circumstances, otherwise you are

'asking for trouble.

'Paste this in right after the initial Sub line in your macro (or you might want to create a separate procedure

'just for this):

Sub Test\_Optimize\_Macros()

Dim bEvents As Boolean

Dim bAlerts As Boolean

Dim CalcMode As Long

Dim bScreen As Boolean

' save current settings

bEvents = Application.EnableEvents

bAlerts = Application.DisplayAlerts

CalcMode = Application.Calculation

bScreen = Application.ScreenUpdating

' disable events, alerts, automatic calculation &amp; screen updating

With Application

.EnableEvents = False

.DisplayAlerts = False

.Calculation = xlCalculationManual

.ScreenUpdating = False

End With

' your code here

'Then right before the End Sub, paste this:

' restore previous settings

With Application

.EnableEvents = bEvents

.DisplayAlerts = bAlerts

.Calculation = CalcMode

.ScreenUpdating = bScreen

End With

End Sub

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

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

'Lesson 3.4.3: Minimize interaction between VBA and Excel

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

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

'Believe it or not, VBA and Excel are actually separate entities. There is a time penalty when reading worksheet

'data into VBA, so you want to minimize the amount of times you have to touch the worksheet. Ideally, you should

'touch the worksheet once to read in data, then once to write back.

'This is why cell-by-cell loops are dangerous, because you are hitting the worksheet multiple times.

'Here's a typical loop:

' Set SRC\_RNG = Range("A1:A10000")

' For Each DCELL In SRC\_RNG

' If DCELL.Value = 3 Then

' totals = totals + 1

' End If

' Next DCELL

'If there are 500 cells in the selection, then VBA has to touch the worksheet 500 times. As you can imagine,

'this code will be extremely slow (relative to, say, the amount of time it takes to make microwave popcorn).

'It will be even slower as you try to do more complicated things like insert formulas, update another worksheet,

'make more intense calculations, etc. A better way is to read the entire range into an array, and do your

'looping there. An array in memory will loop much faster.

'Here is some code that can be used to read any selection into memory and manipulate it very quickly.

'Set SRC\_RNG = Selection

'lRows = SRC\_RNG.Rows.Count

'lCols = SRC\_RNG.Columns.Count

'ReDim DATA\_MATRIX(1 To lRows, 1 To lCols)

' once

'DATA\_MATRIX = SRC\_RNG.Value

'For j = 1 To lCols

' For i = 1 To lRows

' DATA\_MATRIX(i, j) will contain the data from the cell you want to check

' Next i

'Next j

' twice

'SRC\_RNG.Value = DATA\_MATRIX

'Set SRC\_RNG = Nothing

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

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

'Lesson 3.4.4: Functions Volatile

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

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

'Normally ExcelÕs smart recalculation engine only recalculates formulas that either have been changed/entered

'or depend on a cell or formula that has been changed somewhere higher up the chain of precedents for the formula.

'This makes for very efficient calculation speed since in a typical workbook only a small faction of the formulas

'will be dependent on any particular cell or piece of data.

'But some functions need to recalculate at every recalculation. For example NOW() should always give you the

'current time at the last calculation, and RAND() should give you a different random number each time it is

'calculated. These functions are called Volatile Functions, and any formula that uses one of them is a Volatile formula.

'You can see more discussion of ExcelÕs built-in volatile functions and the volatile actions that trigger a

'recalculation at http://www.decisionmodels.com/calcsecretsi.htm.

'-> How does ExcelÕs smart recalc engine know when to recalculate a function or a formula?

'Excel maintains its dependency trees by looking at what other cells a function or a formula refers to, and the smart

'recalc engine uses these dependency trees to work out which formulas to recalculate.

'For Functions Excel only looks at the arguments to the function to determine what the function depends on. So if

'you write a function like this:

Function Depends1(theCell As Range)

Depends1 = ActiveSheet.Range("Z9") + theCell + theCell.Offset(0, 1)

End Function

'and call it in a formula =Depends("A1") then Excel will only recalculate your function when A1 changes, and

'not when B1 or Z9 changes. This could give you incorrect results.

'Note: During a recalculation if Excel does evaluate the UDF it determines which cell references are actually being

'used inside the function to affect the function result, and if those cells have not yet been finally calculated it

'will reschedule the Function for later calculation. This is required to make the UDF be finally calculated in the

'correct dependency sequence.

'-> How to fix this problem?

'There are several ways to fix this problem, but only one good one! Make the function Volatile!!!

'If you add Application.Volatile to the function it will always recalculate:

Function Depends2(theCell As Range)

Application.Volatile

Depends2 = ActiveSheet.Range("Z9") + theCell + theCell.Offset(0, 1)

End Function

'But this will slow down the calculation, so generally its a bad idea unless, like RAND() or NOW() the function really

'needs to be Volatile. Use Ctrl/Alt/F9 to trigger a full calculation

'If you press Ctrl/Alt/F9 then Excel will recalculate every single formula in all the open workbooks, regardless of

'what has changed or is volatile. Of course this can be very slow.

'Make sure the Arguments to the UDF refers to ALL the cells the UDF uses.

'You could Change the UDF to

Function Depends3(theCell1 As Range, theCell2 As Range)

Depends3 = theCell1.Resize(1, 1) + theCell1.Resize(1, 1).Offset(0, 1) + theCell2

End Function

'This is the best solution.

'Call it using =Depends(A1:B1,Z9) so that Excel knows that B1 is being referenced by theCell1.Offset(0,1).

'Now Excel knows all the cells that the function depends on and it will be recalculated correctly and efficiently.

'-> Detecting whether a Function or Formula is Volatile

'You can download VolatileFuncs.zip from http://www.DecisionModels.com/Downloads/VolatileFuncs.zip

'This contains tests for the volatile Excel built-in functions, using a function to increment a counter each time

'the referenced cell changes.