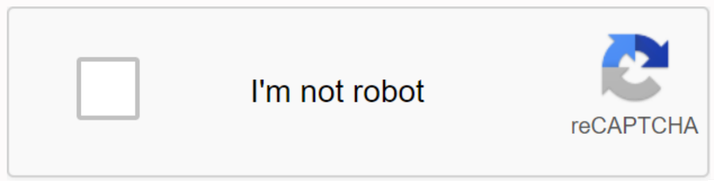

Continue

How to calculate worksheet in excel macro

As the proud owner of several large VBA macros, I spent a considerable amount of time looking for ways to make macros run faster. This article lists my top rules for accelerating VBA. It is easy to expire in bad programming habits when working with small macros, but with large macros and macros running a long time it is essential to use effective coding. This article focuses primarily on Excel vba macros, but many of these rules also apply to Microsoft Access VBA macros. The first five rules generally have the greatest impact on macro performance. Rules from six to 11 have a marginal impact. Please note that my time savings estimates below may vary significantly for your specific application. The analysis used Excel 2007. Rule #1. Turn off automatic spreadsheet calculation This rule is well known, but it is the most important rule. When a new value is entered in a worksheet cell, Excel will recalculate all the cells that relate to it. If the macro writes values to the worksheet, the VBA will have to wait until the worksheet finishes recalculating each entry before it can resume. The impact of leaving the calculation automatically turned on can be dramatic. We recommend that you turn off automatic computing by using the following command at the beginning of the macro. Application.Calculation = xlCalculationManual If you need to recalculate spreadsheet values while the macro is running, you can use any of the following commands. The first command recalculates the entire workbook. The second command recalculates only a specific sheet. The third command recalculates only a specific area. CalculatedWorksheets(sheet1). CalculatedRange(A1:C5). Calculation when the macro is complete, automatic calculation must be enabled again using the following command. If the macro ends prematurely before this command is processed, you will need to manually reset the calculation to automatically in EXCEL. Application.Calculation = xlCalculationAutomatic. Rule #2. Turn off screen updates Each time VBA writes data to the worksheet, refresh esp of the screen you see. Image refresh is a considerable drag per performance. The following command disables screen updates. Application.ScreenUpdating = FALSE At the end of the macro, use the following command to enable screen updates again. Application.ScreenUpdating = #3 TRUE rule. Minimize traffic between vbA and worksheet Once the macro starts, it's important to avoid unnecessary references to the worksheet. Grabbing data from the spreadsheet is a drag on performance. Avoid reading or writing worksheet data in loops whenever possible. It is much more to read the data once and save it to memory rather than reread it each time. In this example, the macro will have to repeatedly take the range named issue_age from the worksheet. This is a common mistake. VBA is much faster when you don't have to stop and interact with the worksheet. For duration = 1 to 100 Attained_Age = Interval(Issue_Age) + + Duration In the following code, the variable Issue_Age is read only once from the worksheet, and traffic between VBA and Excel is minimized. The code below is over 100 times faster than the code above! Issue_Age = Interval(Issue_Age)For duration = 1 to 100 Attained_Age = Issue_Age + DurationNext Duration It is also more efficient to perform all numerical calculations in VBA. It's often tempting to leave formulas in your spreadsheet and call them from the macro. But if speed is important, put all formulas in the macro. This minimizes traffic and does not require spreadsheet recalculation. As a general rule, use the Worksheets, Range, Cells, and Application commands as efficiently as possible outside of loops. Rule #4. Read and write data blocks in one operation This rule is a continuation of the #3 rule. This is another way to minimize traffic between VBA and Excel. Whenever possible, read and write data in pieces. There are several ways to achieve this. Here's an example of reading in a large block of data (2,600 cells) in an array. This example is about 50 times faster than reading in each cell individually in a loop. Dim myArray() As a variant ' note that this must be a variantarrayArray= Worksheets(Sheet1). Range(A1:Z100).value Also, here are examples of writing the array back to the worksheet. All are about 40 times faster than writing each of the 2,600 cells individually in a loop. Worksheets #1 Method(Sheet1). Range(A1:Z100).value = myArray Method #2 With Sheepups(Sheet1) . Range(A1:Z100). Value = myArray End With Method #3 Dim theRange As Range TheRange = Range(A1:Z100)theRange.value = myArray Rule #5. Avoid using certain Excel worksheet functions This rule has been surprising to me. I had naively assumed that the common functions of the worksheet would be processed effectively by the VBA. This is clearly not the case. For example, most VBA users are probably aware that VBA does not have a Max() or Min() function. Excel has these functions. It is common to use the following code that uses the Excel version of Max(): variable1 = Application.Max(Value1, Value2) I found an open source version of a VBA Max() function on the Internet. It was 10 times faster than the Excel-based counterpart above. However, the code below is over 80 times faster! I recognize that the function below only works with two arguments and does not support arrays, but the speed improvement is substantial. Max2 function (Value1, Value2) If Value1 > Value2 Then Max2 = Value1Else Max2 = Value2End IfEnd Function I suggest caution when using worksheet functions in large, time-consuming macros. It should assess the impact of rewriting the function. Note that any command that starts with or Reeu worksheet. refers to an Excel function. I can't say that all Application functions. are slow. But, I have written or downloaded versions of Min(), Max(), Average(), Match(), NormSInv() and StDev() that are much faster than Excel versions. The rule #6. Avoid using Variants in formulas Do not declare a numeric variable as Variant, unless necessary. Note that if you choose not to use Explicit Option at the beginning of the macro, any undefined variable will be a variant. Variants are very flexible because they can be numeric or text, but are slow to process in a formula. The impact on efficiency is not great, but every bit helps. Note that this rule also applies to the functions you write. Based on my tests, the variable types from fastest to slowest in mathematical equations are: Constant, Single, Double, Long, Integer, Variant. Rule #7. Avoid evaluating strings (text) are slow to evaluate. Avoid evaluating strings in code like this: Select Male case like (enter code here)... The Woman case (enter the code here)... Unisex case (enter code here)... End Select Enumeration assigns a constant numeric value to a variable. VBA can quickly process the listed values while keeping the code legible. Enumeration can assign default numeric values or assign specific values. Public Enum enumGender Male = 0 Woman = 1 Unisex = 2End EnumDim Gen as enumGenderSelect Case Gender Male (enter code here)... Woman case (enter code here)... Unisex case (enter code here)... End Select Boolean operators are simply TRUE or FALSE switches that process very quickly. In the example below bMale, bFemale and bUnisex are Boolean variables. Boolean code is about 10 times faster than using strings. If bMale Then (enter code here)... ElseIf bFemale Then (enter code here)... ElseIf bUnisex Then (enter code here)... Ends if #8 rule. Do not select specific worksheets unless necessary In general, you do not need to use the Select command to read or write to a worksheet. It is about 30 times faster not to select a worksheet. Avoid this: Worksheets(sheet1). Select Sum1 = Cells(1, 1)Do this instead: Amount1 = Worksheets(sheet1). Cells(1,1) Rule #9. Avoid excessive use of StatusBar VBA updates can process math faster than the Status Bar can display. Writing to the Status Bar is another example of traffic between VBA and Excel. The following example writes one of every 100 scenarios to the Status Bar. It's about 90 times faster than writing each script at StatusBar. For scenario = 1 to 10000 (enter code here)... If Scenario Mode 100 = 0 Then Application.StatusBar = ScenarioNext Scenario Rule #10. Avoid unnecessary math as actuators we like macros full of formulas. Frequently formulas are not as effective as they should be. In the example below, the macro calculates the monthly value a fund that increases to 5% effective annual interest for 50 years. I frequently see the code like this: Interest_rate = 0.05For i = 1 to 600 fund (i) = fund (i-1) * (1 +interest_rate)^(1/12)Next i It is more efficient to convert the annual interest rate to the monthly rate once after it is shown below. In the loop, VBA uses only one numeric operation (multiplication). (multiplication). the example above uses four numerical operations (a multiplication, a plus, a division and an exponent) within the loop and is therefore about four times slower. Interest rate = .05Interest_factor = (1+interest_rate)^(1/12)For i = 1 to 600 fund(s) = fund(i-1) * interest_factorNext also note that the exponential is slower than adding, subtracting, multiplying or splitting. Rule #11. Do not copy and paste Copy and Paste (or PasteSpecial) functions are slow. It is about 25 times faster to use the following to copy and paste values. Range(A1:Z100.value = Range(A101:Z200value Final Thoughts I found it useful to write a small macro to evaluate the time savings associated with different methods. The macro simply performs a method a million times and records the time spent performing that method. The simple macro below compares the Excel Max() function with the Max2 function displayed in rule #5. '**Assess first functionStart_time = NowFor i = 1 to 1000000 value1 = Application.Max(amt1, amt2)Next iEnd_time = NowWorksheets(sheet1). Cells(1, 2) = End_Time — Start_Time'**Evaluate the second functionStart_time = NowFor i = 1 to 1000000 value1 = Max2(amt1, amt2)Next iEnd_time = NowWorksheets(sheet1). Cells(2, 2) = End_Time — Start_Time Please contact me if you have any other time-saving tips. I also want to point out and thank several excellent articles on the internet that address this topic: by Dermot Balson by Pearson Software Consulting by Diego M. Oppenheimer by Ozgrid Kevin Roper, FSA, MAAA is acting with AEGON USA Inc. Can be contacted at kroper@aegonusa.com kroper@aegonusa.com