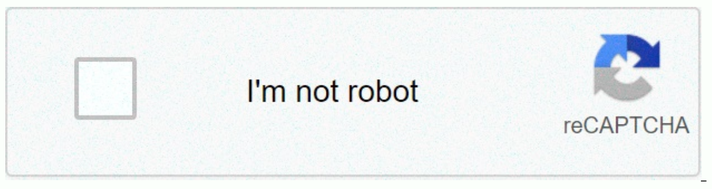

Continue

Dim as worksheet vba excel

If the worksheet you want to retrieve exists from compile-time in ThisWorkbook (i.e. the workbook which contains the VBA code you are looking for in), then the simplest and most reliable way to refer to that Worksheet Object is to use its code name: Debug.Print1 Range(A1). Value you can set the code name to anything you need (as long as it's a valid VBA identifier), independently read the tab name (which the user can modify at any time), by changing the (Name) property of the Properties tool (F4): The name of the property refers to the table name that the user can change on a crow; (Name) the property refers to the code name in the worksheet, and the user cannot change it without accessing the Visual Basic Editor. VBA use this code name to automatically declare a global-dimensional worksheet object which code you become using anywhere to refer to this sheet, for free. In other words, if the sheet exists in ThisWorkbook at compile-time, there's never a need to declare a variable for it – the variable is already there! If the worksheet is created at run-time (inside ThisWorkbook or not), then you need to declare & assign a Worksheet Variable for it. Use the Worksheets property of a Recover Workbook object: Dim wb as Workbook Set wb = Application.Workbooks.Open(Path) Dim ws As Worksheet Set ws = wb. Sheet (nameOrIndex) important note... Both the name and index of a worksheet can easily be edited by the user (accidentally or not), unless workbook structure is protected. If workbook is not protected, you simply cannot assume that the name or index alone will give you the specific worksheet you're after – it's always a good idea to validate the formatting of the sheet (example verify that cell A1 contains some specific text, or that there's a table with a specific name , which has some specific column headings). Using the sheets collection contains Worksheet objects, but can also include Chart Case, and a half-dozen more types of legacy sheets which are not worksheets. Assigning a worksheet reference from whatever Sheets (nonOrIndex) returns, risk sends a type of mismatch run-time error for this reason. Not qualifying the Worksheets collection is an implicit reference to ActiveWorkbook – meaning the Worksheets collection is pulling out of all that active workbook at the time the instruction is executed. These implicit references make the frame code and bulk-prone, especially if the user can navigate and interact with the Excel UI while the code is running. Unless you mean to activate a specific sheet, you never need to call ws. Activate in order to do 99% of what you want to do with a worksheet. Just use your variable variables instead. This article will discuss the ActiveSheet object in VBA. It will also discuss how to activate, select, and go to Worksheets (> Much More). ActiveSheet in VBA, ActiveSheet refers to the currently active worksheet. Only one worksheet can be active at a time. Activate worksheet (Insert ActiveSheet) to insert the using Worksheet.Activate: Worksheets (Insert). Activate the Activate Worksheets command will actually go to the sheet, change the visible worksheet. The example above uses the Sheet (Tab) name. Instead you can use the VBA code name for the worksheet: ActiveSheet Name to get the ActiveSheet name: Select sheets vs ActiveSheet at any point in time, only one worksheet can be the ActiveSheet. However, multiple worksheets can be selected at once. When multiple worksheets are selected only the top-most worksheet is considered active (the ActiveSheet). Select Worksheet If you would like to select a worksheet instead of activating it. Use .select instead. Select Worksheet by this Tab name to select a worksheet based on it in Sheet Tab Name Select Worksheet name by Index This number selects a worksheet based on its relative position of other selected worksheet tabs and VBA code Select worksheets by code name can prevent errors caused by worksheet name changes. Select current worksheet to select current worksheet, use the ActiveSheet object: More activated/Selected Worksheet Instance set ActiveSheet to Dim Ws As Worksheetset Ws = ActiveSheet Change Active ActiveSheet.Name Name = NewName With ActiveSheet Using allows you to streamline your code when working with objects (such as sheets or ActiveSheet). With ActiveSheet . Name = StartFresh . Cell.Clear . Range (A1). Value = . Nameend and Notice how you don't need to repeat ActiveSheet before each line of code. This can be a great time saver when working with a long list of commands. Loop in selected sheets these macros will loop through all selected sheets, display the names. Sub GetSelectedSheetsName() Dim ws As Worksheet For Each ws In ActiveWindow.SelectedSheets MsgBox ws. Next name wsend Sub Goto Next Sheet This code will go to the next worksheet. If the ActiveSheet is the last sheet, then it will go to the first sheet of the workbook. If ActiveSheet.Index = Worksheets.Count Then Worksheets(1). ActiveSheet.Next.ActivateEnd If stop searching VBA code online. Learn more about AutoMacro – A VBA Builder code that enables beginners to scratch code procedures and minimal knowledge code and many time-saving features for all users! learn more! <<Return to VBA Example apart from cells and ranges, working with worksheets is another area you should know about to use VBA efficiency in Excel. Just like any object in VBA, worksheets have different properties and methods associated with it that you can use while automating your work with VBA in Excel. In this tutorial, I will cover 'Worksheets' in detail and also show you some convenient examples. So let's start. All the codes I mentioned in this tutorial need to be placed in the VB Editor. Go to the 'Where to insert the VBA code' section to know how it works. If you are interested in learning VBA the easiest way, check out my Online Excel Training. Differences between worksheets and sheets in VBA in VBA, you have collection that can be somewhat confusing at times. In a workbook, you can have worksheets and as well as chart sheet. The example below contains three worksheets with one chart sheet. In Excel VBA: 'Worksheets' collection would refer to the collection of all the worksheet objects in a workbook. In the above example, the Worksheets collection would consist of three worksheets. The 'sheets' collection would refer to all the worksheets as well as chart sheets in the workbook. In the above example, it would have four elements – 3 Worksheets + 1 Sheet Tabs. If you have a workbook which only contains worksheets and no chart sheets, then 'Sheets' and 'Sheets' collection are the same. But when you have one or more chart sheets, the 'Sheets' collection should be larger than the 'Sheets' collections = Sheet + Sheets Sheets Now, with this distinction, I recommend being as specific as possible when writing a VBA code. So if you have to refer to worksheets only, use the 'Worksheets' collection, and if you have to refer to all sheets (including chart sheets), use the 'sheets' collection. Reference a worksheet in VBA There are many different ways you can use to refer to a worksheet in VBA. Understanding how to refer to worksheets would help you write better code, especially when using loop in your VBA code. Using the Worksheet Name is the easiest way to refer to a worksheet is to use its name. For example, suppose you have a workbook with three worksheets – Sheet 1, Sheet 2, Sheet 3. And you want to activate Sheet 2. You can do this by using the following code: Sub ActivateSheet() Sheet(Sheet2). Activate the End Sub code above requires VBA to refer to Sheet2 in the Worksheets collection and activate it. Since we are using the exact sheet name, you can also use the sheets collection here. So the following code would do the same. Sub ActivateSheet() Sheets(Sheet2). Activate End Group By using the index number while using the sheet name is an easy way to refer to a worksheet, sometimes, you may not know the exact name in the worksheet. For example, if you are using a VBA code to add a new worksheet in the workbook, and you don't know how many worksheets are already there, you wouldn't know the name of the new worksheet. In this case, you can use the index number of the worksheets. Supposing you have the following sheets in a workbook: The below code would activate Sheet2: Sub ActivateSheet() sheet(2). Activate End Sub Note that we used the index number 2 in Worksheets(2). This would refer to the second object in the collection of worksheets. Now what happens when you use 3 as the index number? It will select Sheet3. If you are wondering why it selects Sheet3, as it's clearly the fourth object. This occurs because a chart sheet is not part of the sticker collection. So when we use the index numbers in the Worksheets collection, it will only refer to the in the workbook (and ignore the chart sheets). On the contrary, if you are using sheets, Sheets(1) would refer to Sheets1, Sheets (2) would refer to Sheet2, Sheets (3) would refer to Chart1 and Sheets (4) would refer to Sheet3. This technique of using index numbers is useful when you want to loop through all the worksheets in a workbook. You can count the number of worksheets and then loop through these using this count (we will see how to do this later in this tutorial). Note: The index number goes from left to right. So if you shift Sheet2 to the left of Sheet1, then Worksheets (1) would refer to Sheet2. Using the Worksheet Code Name One of the cons of using the sheet name (as we saw in the section above) is that a user can change it. But if the sheet name has been changed, your code would not work until you change the name of the worksheet in the VBA code as well. To attack this problem, you can use the code name in the worksheet (instead of the regular name that we have used so far). A code name can be assigned in the VB Editor and do not change when you change the name of the sheet in the worksheet area. To give your worksheet a code name, follow the below steps: Click the Developer tab. Click the Visual Basic button. This will open the VB Editor. Click the View option in the menu and click Project Pane. This will make the Properties window visible. If the Properties pane is already visible, skip this step. Click the sheet name of the project explorer that you want to rename. In the Properties pane, change the name of the front field (Name). Note that you can't have spaces in the name. The above steps would change the name of your worksheet to the VBA backend. In the Excel worksheet, you can name the worksheet whatever you want, but in the backend, it will reply to both the names – the sheet name and the code name. In the image above, the sheet name is 'SheetName' and the code name is 'CodeName'. Even if you change the sheet name on the worksheet, the code name always stays the same. Now you can use either the Worksheets collection to refer to the worksheet or use the code. For example, both the line will activate the worksheet. Sheet (Sheetname). Activating CodeName.Activate the difference in these two is that if you changed the name of the worksheet, the first one would not work. But the second line will continue to work even with the name changed. The second line (using the StringName) is also shorter and easier to use. Refer to a worksheet in a different workbook If you want to refer to a worksheet in a different workbook, which workbook needs to be opened while running them code, and you need to specify the name of the workbook and the worksheet that you want to refer to. For example, if you have a workbook with the example names and want to activate Sheet1 in the Example workbook, you need to use the below code: Sub SheetActivate() Workbooks (e.g. xlsx). Sheet (Sheet1). Activate Note End if the workbook has been saved, you need to use the file name along with the extension. If you aren't sure which name to use, take help in the Project Explorer. In case the workbook has not been saved, you do not need to use the file extension. Add a worksheet below would add a worksheet (as the first worksheet – i.e., as the leftmost sheet in the sheet tab). Sub AddSheet() Worksheets.Add Sub End It takes the default name Sheet2 (or any other number based on how many sheets are already there). If you want a worksheet to be added before a specific worksheet (says Sheet2), then you can use the below code. Sub AddSheet() Worksheets.Add Before: =Worksheets(Sheet2) Finish the above code telling VBA to add a sheet and then use the 'Before' statement to specify the previous worksheet which the new worksheet should be inserted. Similarly, you can also add a sheet after a worksheet (says Sheet2), using the below code: Sub AddSheet() Worksheets.Add After: =Worksheets(Sheet2) End Sub If you want the new sheet to be added at the end of the sheets, you need to first know how many sheets are there. This code first counts the number of sheets, and adds the new sheet after the last sheet (which we refer by using the index number). Sub AddSheet() Dim SheetCount As Integer SheetCount = Worksheets.Count Worksheets.Add After: =Worksheets(SheetCount) End Of Group Delete A Worksheet Below Would Delete The Active Sheet In The Workbook. Sub DeleteSheet() ActiveSheet.Delete The End Sub code above would display a warning prompt before deleting the worksheet. If you don't want to view the prompt warning, use the code below: Sub DeleteSheet() Application.Displayalerts = False ActiveSheet.Delete ActiveWindow.SelectedSheets.Delete End Sub When Application.Displayalerts set to false, it will not show you the warning prompt. If you use it, remember to set it back to True at the end of the code. Remember that you can't undo this delete, so use the above code when you're absolutely sure. If you want to delete a specific sheet, you can do so by using this code: Sub DeleteSheet() Sheet(Sheet2). Delete End Sub Group You can also use the code name in the sheet to delete it. Sub DeleteSheet() Sheet5.Delete End Sub Rename Groups You can modify the name property of the worksheet to rename it. The following code will change the name of Sheet1 to 'Summary'. Sub RenameSheet() Worksheets(Sheet1). Name=Summary End Sub You can combine this with the add sheet method to have a range of sheets with specific names. For example, if you want to insert four sheets with the name 2018 Q1, 2018 Q2, 2018 Q3,

and 2018 Q4, you can use the below code. Sub RenameSheet() Dim Countsheets As Integer Countsheets = Worksheets.Count for i = 1 To 4 Worksheets.Add after: = Worksheets(Countsheets + i - 1) Worksheets(Countsheets+ me). Name =2018 Q> Next I Finish Sub In the above code, we first count the number of sheets and then use a Next Loop new sheets at the end. As the sheet is added, the code also renames it. Assign workbook Object to a Variable when working with worksheet, you can assign a worksheet to an object variable, and then use the variable instead of the worksheet references. For example, if you want to add a year prefix to all the worksheets, instead of counting the sheets and the loop running that many number of times, you can use the object variable. Here is the code that will add 2018 as a prefix to all names of the worksheet. Sub RenameSheet() Dim Ws As Worksheet For Each Wss In Worksheet Ws.Name = 2018 - &Ws.Name Next Ws End Class The above code declare a variable Wss as the worksheet type (using the 'Dim Ws Line As Worksheet'). Now we don't need to count the number of the loop sheets of these. Instead, we can use 'For each Wss in Worksheets' loop. This will allow us to go through all the sheets in the worksheets collection. It doesn't matter if there are 2 sheets or 20 sheets. While the above code allows us to loop through all the sheets, you can also assign a specific sheet to a variable. In the code below, we assign the Variable Ws to Sheet2 and use it to access the entire Sheet2 property. Sub RenameSheet() Dim Ws As Worksheet Insert Ws = Sheet(Sheet2) Ws.Name = Summarize Ws.Protect End Sub once you insert a worksheet reference to an object variable (using the SET statement), which can be used instead of the worksheet reference. This can be useful when you have a long complicated code and want to change the reference. Instead of making the switch everywhere, you can simply make the switch to the SET statement. Note that the code declares the Ws object as the Worksheet type (using the Dim Ws line as Worksheet). Hiding worksheets by using VBA (Hide + Very Hidden) Hide and unhiding worksheets in Excel is a simple task. You can hide a worksheet and the user would not see it when he/she opens the workbook. However, they can easily prevent the worksheet by right-clicking on any sheet tab. But what if you don't want them to be able to unhide the worksheet(s). You can do this by using VBA. The below code would hide all the worksheets in the workbook (except the active sheet), such that you cannot unhide it by right-clicking the sheet name. Sub HideAllExeactiveSheet() Dim Ws As Worksheet For Each Ws In ThisWorkbook.Worksheets If Ws.Name <> ActiveSheet.Name Then Ws.Visible = xlSheetVeryHidden Next Ws End Sub In the above code, the Ws.Visible property is changed to xlSheetVeryHiden. When the visible property is set to xlSheetVisible, the sheet is visible in the worksheet area (as sheet tabs). When the visible property is set to xlSheetHidden, the sheet is hidden but the user can prevent it by right-clicking on any sheet tab. When the visible property is set to xlSheetVeryHidden, the sheet is hidden and cannot be unhidden from worksheet area. You need to use a VBA code or the properties window to unhide it.optional simply hide sheets, which may be unusually easy, use the below code: Sub HideAllExeptActiveSheet() Dim Ws As Worksheet Per Ws In ThisWorkbook.Worksheets If Ws.Name <> ActiveSheet.Name Then Ws.Visible = xlSheetHidden Next Ws End Class The below code would hide all the worksheets (both hidden and very hidden). Sub UnhideAllWoksheets() Dim Ws As Worksheet For Each Ws In ThisWorkbook.Worksheets Ws.Visible = xlSheetVisible Next Ws End Sub Related Article: Unhide To uncheck all sheets in Excel (in one go) Hide sheets based on the Text in it Supposing you have multiple sheets with the name in different departments or years and want to hide all the sheets except the ones which are the 2018 year in it. You can do this by using a VBA FUNCTION INSTR. The following code would hide all the sheets except those with the 2018 text in it. Sub HideWithMatchingText() Dim Ws As Worksheet For Each Wss In Worksheet If InStr(1, Ws.Name, 2018, vbBinaryCompare) = 0 Then Ws.Visible = xlSheetHidden End If Next Ws End Class In The Above Code, the INSTR function returns the position of the character where it finds the matching string. If it doesn't find the matched string, it returns 0. The above code checks whether the name contains the 2018 text in it. If it does, nothing happens, other things the worksheet is hidden. You can take this one step further by having the text in a cell and using this cell in the code. This will allow you to have a value in the cell and then when you run the macro, all the sheets, except the one with the matching text in it, would remain visible (along with the sheets where you are entering the value in the cell). To sort the worksheets in an Alphabetical Order Using VBA, you can quickly sort the worksheets based on their names. For example, if you have a workbook which contains sheets for different departments or years, then you can use the below code to quickly sort these sheets in a voting order. Sub TrirtSheetsTabName() Application.ScreenUpdating = False Dim Shcount As Integer, i As Integer, j As Integer Shcount = Sheets.Count To i = 1 ShCount - 1 To j = 1 + 1 Shcount If sheets(j). Name < Sheets(i). Name then sheets(j). Move before: = Sheets(i) End If Next Next i Application.ScreenUpdating = True End Note That this code works well with text names and at most of the cases and years and numbers as well. But it can give you the wrong results in case you have the sheet names as 1,2,11. It will sort and give you sequences 1, 11, 2. This is because it reads the comparison as text with regards 2 larger than 11. Protect / Unprotect all the sheets in one Go if you have a lot of worksheets in a workbook, and you want to protect all the sheets, you can use the below VBA code. It allows you to specify the password in the code. You will need this password to unprotect the worksheet. Sub ProtectAllSheets() Dim ws As Worksheet Dim password As String Password = Test123 ' Replace Test123 with the password you want for each ws in ws worksheet. Protect password:=password ws Finish Sub This code would unprotect all the sheets in one go. Sub ProtectAllSheets() Dim ws As Worksheet Dim password As String Password = Test123 ' Replace Test123 with the password you use while protecting for each ws in your worksheet. Unprotect Password: = Password Next Ws End Class Create a table of Contents of all sheets (with Hyperlinks) If you have a range of worksheets in the workbook and you want to quickly insert a summary sheet which contains links to all sheets, you can use the below code. Sub AddIndexSheet() Worksheets.Add ActiveSheet.Name = Index To i = 2 To Worksheets.ActiveSheet.Hyperlinks.Add Anchor :=Cells(i - 1, 1), _ Address:=, SubAddress:=Worksheets(i). Names >; ! A1, _ TextToDisplay:=Worksheets(i). My Next Name Finish Group the code above delve a new worksheet and name it Index. It then loop through all the worksheets and create a hyperlink for all the worksheets in the index sheet. Where to insert the VBA code to ask where the VBA code goes in your Excel workbook? Excel has a VBA backend called the VBA editor. You need to copy and paste the code into the VB Editor code window. Here are the steps to do so: Go to the Developer tab. Click the Visual Basic option. This will open the VB editor in the backend. In the Project Explorer pane in the VB Editor, right-click any object for the workbook in which you want to insert the code. If you don't see the Project Explorer go to the View tab and click Project Explorer. Go to Insert and click Module. This will set a module object for your workbook. Copy and paste the code into the Module window. You may also like the following Excel tutorials: Tutorials: