Click to verify

In Excel VBA, you can rename a sheet by setting its ‘Worksheet.Name’ property to the desired text string value. In this tutorial, I will present to you several examples of Excel VBA code that you can customize and use to rename a worksheet or worksheets in a workbook. In this section, I present to you examples of Excel VBA code that you can use to rename one sheet in a workbook. Suppose you want to rename the active sheet in the current workbook as ‘Annual Budget.’ You can use the VBA code below to accomplish the task: Sub RenameActiveSheet() ActiveSheet.Name = "Annual Budget" End Sub Excel assigns an index number to each sheet you add to a workbook. For example, Excel assigns index 1 to the first sheet you add to a workbook, index 2 to the second sheet, and so on. You can use the sheet’s index number to identify the sheet you want to rename. Suppose you want to rename the second sheet you added to the current workbook as ‘Annual Sales.’ You can use the following code to do the task: Sub RenameSheetByIndex() ThisWorkbook.Sheets(2).Name = "Annual Sales" End Sub Suppose you want to rename the sheet ‘Schools’ in the current workbook as ‘Colleges.’ You can use the subroutine below to achieve that: Sub RenameSheetByCurrentName() ThisWorkbook.Sheets("Schools").Name = "Colleges" End Sub You can rename a sheet to a cell’s value in VBA. Suppose the value in cell A1 of the active sheet is ‘Progress Report.’ You can rename the active sheet as the value in that cell using the subroutine below: Sub RenameSheetToCellValue() ActiveSheet.Name = Range("A1").Value End Sub Note: If cell A1 is empty, the code will trigger the error message ‘Run-time error 1004:application-defined or object-defined error’ shown below: If the value in cell A1 has any of the disallowed special characters mentioned in the section ‘Guidelines for Renaming Sheets,’ you will get the error message ‘Run-time error 1004: You typed an invalid name for a sheet or chart’ shown below: Suppose you want to rename Sheet1 in another open workbook, ‘ Experiments.xlsx,’ as ‘Instructions.’ You can use the VBA code below to accomplish the task: Sub RenameSheetAnotherWorkbook() Workbooks("Experiments.xlsx").Sheets("Sheet1").Name = "Instructions" End Sub The code below renames the active sheet as the current date: Sub RenameSheetByDate() ActiveSheet.Name = Format(Now(), "dd-mm-yyyy") End Sub The code uses the ‘Now’ function to get the current date and time and then the ‘Format’ function to apply the dd-mm-yyyy format to the date. The code then sets the ‘ActiveSheet.Name’ property to the formatted date string, thus renaming the sheet as the current date. If you run the code and the workbook already has a sheet named the current date, you will get the error message ‘Run-time error 1004: That name is already taken. Try a different one’ shown below: Each sheet in a workbook must have a unique name. You can add the time component to the date format to make each sheet renamed as the current date unique. You can use the subroutine below to rename the active sheet as the current date and time: Sub RenameSheetByDateTime() ActiveSheet.Name = Format(Now(), "dd-mm-yyyy_hh_mm_ss") End Sub Notice that I have used underscores instead of full colons in the time component of the date and time format because Excel does not allow full colons in sheet names. Sometimes, you may need to rename a sheet as the workbook name. Suppose the current workbook is named ‘Performance.xlsx.’ You can rename the active sheet as the current workbook name using the VBA code below: Sub RenameSheetAsWorkbookName() Dim wbName As String wbName = Mid(ThisWorkbook.FullName, InStrRev(ThisWorkbook.FullName, "\") + 1) wbName = Left(wbName, InStrRev(wbName, ".") - 1) ActiveSheet.Name = wbName End Sub The code uses the ‘Mid,’ ‘InStrRev,’ and ‘Left’ functions to extract the workbook name from the file’s full path and remove its file extension. It then assigns the resultant string to the ‘wbName’ variable and finally renames the active sheet as the value in the ‘wbName’ variable. Note: This code only works if you have saved the workbook. If you run it in a new workbook that you have not yet saved, you will get the error message ‘Run-time error 5: Invalid procedure call or argument’ shown below: Suppose you want to add the prefix ‘Annual_’ before the sheet name of the ‘Budget’ worksheet in the current workbook. The following code will do it for you: Sub PrefixSheet() Dim ws As Worksheet Dim preFix As String Set ws = ThisWorkbook.Sheets("Budget") preFix = "Annual_" ws.Name = preFix & ws.Name End Sub If you want to append a suffix to the end of the worksheet name, you can modify the code as follows: Sub SuffixSheet() Dim ws As Worksheet Dim Suffix As String Set ws = ThisWorkbook.Sheets("Colleges") Suffix = "_Annual" ws.Name = ws.Name & Suffix End Sub If you attempt to rename a non-existent sheet using VBA, you will get the error message ‘Run-time error 9: Subscript out of range’ shown below: So that you don’t get error 9, it is recommended that the code first check whether the sheet exists. If it does, the code then renames the sheet or graciously exits if it doesn’t. The code below checks whether the sheet ‘Evaluation’ exists in the current workbook. If it does, it renames it ‘Appraisal Report’ and displays a message box indicating that the sheet has been renamed. If it does not, the code displays a message box stating it does not exist. Sub CheckSheetExistsRename() Dim wb As Workbook Dim ws As Worksheet Dim sheetExists As Boolean Set wb = ThisWorkbook sheetExists = False For Each ws In wb.Sheets If ws.Name = "Evaluation" Then sheetExists = True Exit For End If Next ws If sheetExists Then wb.Sheets("Evaluation").Name = "Appraisal Report" MsgBox "Sheet has been renamed." Else MsgBox "Sheet does not exist." End If End Sub Also read: VBA to Copy Sheet to New Workbook In this section, I present to you examples of Excel VBA code that you can use to rename multiple sheets in a workbook. Suppose you want to rename each sheet in the current workbook as the value in cell A1. You can use the subroutine below to accomplish the task: Sub RenameSheetsAsCellValues() Dim ws As Worksheet Dim cellValue As String For Each ws In ThisWorkbook.Sheets cellValue = ws.Range("A1").Value If cellValue "" Then ws.Name = cellValue End If Next ws End Sub The code uses the ‘For Each Next’ construct to loop through all the sheets in the current workbook and renames each sheet as the value in its cell A1. The code checks whether cell A1 is empty; if it is, the code leaves the sheet’s name as is. Note: Before running the code, ensure that the values in cell A1 of each sheet do not contain any disallowed special characters mentioned in the section ‘Guidelines for Renaming Sheets.’ Additionally, ensure that the cells do not contain duplicate values. With VBA, you can easily rename sheets in a workbook based on values on a list. Suppose you have the following list on Sheet1 of the current workbook: You can use the VBA code below to rename the sheets in the workbook as values in the list: Sub RenameSheetsAsListValues() Dim ws As Worksheet Dim i As Long i = 1 For Each ws In Worksheets If Range("A1").Offset(i - 1, 0) "" Then ws.Name = ActiveSheet.Range("A1").Offset(i - 1, 0) End If i = i + 1 Next ws End Sub When you execute the code, it renames all the sheets in the workbooks as values on the list: The code uses the i counter variable and the ws worksheet variable. The code uses the ‘For Each Next’ construct to loop through each workbook. Inside the loop, the code uses the ‘If End If’ construct to check whether the cell at an offset from cell A1 (identified using the counter variable) is empty. If it is not, the code renames the target worksheet as the value in the cell, increments the counter variable by one, gets the following worksheet, and repeats the process until all the worksheets in the workbook have been renamed. Suppose you want to add the prefix ‘Regional_’ before the names of all the sheets in the current workbook. You can use the VBA code below to do the task: Sub PrefixAllSheets() Dim ws As Worksheet Dim preFix As String preFix = "Regional_" For Each ws In ThisWorkbook.Worksheets ws.Name = preFix & ws.Name Next ws End Sub The code uses a ‘ws’ Worksheet variable and a ‘preFix’ String variable. The above code uses the ‘For Each Next’ construct to loop through each sheet, adding the text string ‘Regional_’ in the ‘preFix’ variable before the sheet name. You can use the code below to append a suffix after the names of all the sheets in the current workbook: Sub SuffixAllSheets() Dim ws As Worksheet Dim Suffix As String Suffix = "_Regional" For Each ws In ThisWorkbook.Worksheets ws.Name = ws.Name & Suffix Next ws End Sub The above code uses the ‘For Each Next’ construct to loop through each sheet, appending the text string ‘_Regional’ in the ‘Suffix’ variable after the sheet name. Also read: Protect Excel Workbook Using VBA You must adhere to the following guidelines when renaming sheets: The name of a sheet should not exceed 31 characters. Excel does not allow the following special in sheet names: / (forward slash), \ (backslash), : (full colon),? (question mark), * (asterisk), [(left square bracket), or] (right square bracket). A sheet name cannot be an empty string. You cannot rename two or more sheets in a workbook with the same name; each sheet must have a unique name. A sheet name cannot begin with a space character. Sheet names are case-insensitive. For instance, Excel interprets SHEET1, Sheet1, and sheet1 as referring to the same worksheet in a workbook. In this tutorial, I gave you several Excel VBA code examples that you can modify and use to rename a sheet or sheets in a workbook. I hope you found the tutorial helpful. Other Excel articles you may also like: Home / VBA / How to Rename a Sheet using VBA in Excel - Written by Puneet Gogia When you add a new sheet in a workbook, you have the option to name it. But you can also rename it any time using the name property of the worksheet. In this tutorial, we will look at different ways to rename a sheet or multiple sheets using a VBA code. First, define a sheet or a worksheet with its name “Sheets(“Sheet1”)” that you want to rename using the worksheet object. After that, you need to use (.Name) to access the name property that allows you to rename a sheet with a new name. Next, type an equal sign (=) to that to tell VBA the value you want to use to rename the sheet. In the end, type the name of the sheet that you want to use. But make sure to use specify a name using the double quotation marks “mySheet”. If you want to rename the active sheet, in that case, you don’t need to define the sheet name, instead, you need to use the ActiveSheet object that tells VBA to refer to the sheet that is active right now. Here’s the code. Activesheet.Name = "mySheet" Note: To rename a sheet you don’t need to activate it. As you know every sheet has a number based on its position in the workbook. Let’s say you want to rename the sheet that you have on the fifth number, the code would be. Sheets(5).Name = "mySheet5" When you run the above macro, it renames the sheet that is on the fifth number. If you try to rename a worksheet that doesn’t exist, VBA will show you an error, just like below. The solution to this problem is the following code that uses FOR EACH, which can loop through all the worksheets to find the sheet that you have defined and then rename that sheet. Sub check_sheet_rename()'This line declares a variable called ws to represent a WorksheetDim ws As Worksheet object.'These lines declare two variables, mySheet and SheetName, to represent strings.Dim mySheet As StringDim SheetName As String'This line prompts the user to enter a new name for the sheet. The input is stored in the SheetName variable.mySheet = InputBox("enter the name of the sheet that you want to rename.")SheetName = InputBox("Enter new name for the sheet.")'This line starts a loop that will iterate through each worksheet in the current workbook.For Each ws In ThisWorkbook.Worksheets'This line checks if the name of the current worksheet in the loop matches the name entered by the user. If mySheet = ws.Name Then'If the condition in step 6 is met, this line renames the worksheet to the new name entered by the user. ws.Name = SheetName'This line ends the If statement. End If'This line ends the For Each loop.Next wsEnd Sub You can also rename a sheet by taking value from a cell. Let’s say the value is in cell A1. Sheets("Sheet1").name = Range("A1").Value But let’s say you want to name multiple sheets based on the values in a range of cells. In that case, you need to have code like the following. Sub vba_sheet_rename_multiple()'These lines define the variables used in the code. wsCount and rCount are used to store counts, ws for a Worksheet object, name for a Range object, and i for a loop counter.Dim wsCount As LongDim rCount As LongDim ws As WorksheetDim name As RangeDim i As Long'This line counts the number of worksheets in the current workbook and assigns it to wsCount.wsCount = ThisWorkbook.Worksheets.Count'This line counts the number of rows in the range A1 to A10 and assigns it to rCount.rCount = Range("A1:A10").Rows.Count'This checks if the number of worksheets does not match the number of names provided in the range A1 to A10. If they don't match, a message box is displayed, saying "There's some problem with the names provided," and the macro is stopped.If wsCount rCount Then MsgBox "There's some problem with the names provided." Exit SubElse'This loop checks each cell in the range A1 to A10. If any cell is left blank, the variable i is incremented by 1. For Each name In Range("A1:A10") If IsEmpty(name) = True Then i = i + 1 End If Next name'If any blank cells were found in the range A1 to A10, a message box is shown, saying "There's a blank cell in the names range," and the macro is stopped. If i > 0 Then MsgBox "There's is a blank cell in the names range." Exit Sub End IfEnd Ifi = 1'This loop goes through each worksheet in the workbook. It assigns the value of the cell in the range A1 to A10 (corresponding to the loop counter i) to the name of the worksheet. The loop counter i is then incremented by 1.For Each ws In ThisWorkbook.Worksheets ws.name = Range("A1:A10").Cells(i, 1).Value i = 1 + iNext wsEnd Sub When you run this VBA code, first it will check if the cells in the range are equal to the number of sheets that you have in the workbook. After that, it will check if all the cells in the range that you have specified have values or not. And in the end, rename all the sheets using those names. It will verify two conditions using IF THEN ELSE and then rename all the sheets. Excel Vba Rename Worksheet In the realm of data manipulation and analysis, Microsoft Excel is an indispensable tool for professionals across various industries. Its versatility and power are further enhanced through the use of Visual Basic for Applications (VBA), a programming language that allows users to automate tasks, create custom functions, and develop powerful macros. One of the essential tasks that VBA can simplify is the renaming of worksheets, a process that can be particularly useful when dealing with large datasets or complex spreadsheet projects. Excel VBA provides a comprehensive toolkit for automating tasks and enhancing the functionality of your spreadsheets. Among its many capabilities, VBA enables you to manipulate worksheets, cells, and data with precision and efficiency. Renaming worksheets is a straightforward process that can be accomplished with just a few lines of code, yet it offers significant benefits in terms of organization and clarity, especially when working with multiple sheets. The first step in using VBA to rename worksheets is to understand the basic syntax and structure of the code. The core of the process involves using the Name property of the Worksheet object, which allows you to specify the new name for the sheet. This property can be accessed and modified using the With statement, which provides a convenient way to work with multiple properties of an object. Here's a simple example of VBA code to rename a worksheet: Sub RenameWorksheet() With Sheets("OldSheetName") .Name = "NewSheetName" End With End Sub In this code snippet, the Sub procedure defines the start and end of the macro. The With statement selects the worksheet to be renamed, in this case, "OldSheetName". The .Name property is then assigned a new value, "NewSheetName", effectively renaming the sheet. Finally, the End With statement concludes the block of code for the worksheet. Advanced Techniques for Renaming Worksheets While the basic VBA code for renaming worksheets is straightforward, there are several advanced techniques that can enhance the functionality and efficiency of this process. One such technique involves using the ActiveWorkbook object to rename worksheets. This is particularly useful when you want to rename sheets in the current workbook without having to explicitly specify the workbook name. Here's an example: Sub RenameActiveWorkbookSheet() ActiveWorkbook.Sheets("OldSheetName").Name = "NewSheetName" End Sub In this code, the ActiveWorkbook object refers to the currently active workbook, and the Sheets collection allows you to access and manipulate the worksheets within that workbook. By using this approach, you can simplify your code and make it more dynamic. Another advanced technique involves using a loop to rename multiple worksheets at once. This can be especially beneficial when dealing with large datasets or when you need to apply a consistent naming convention to a series of worksheets. Here's an example using a For Each loop: Sub RenameMultipleSheets() Dim ws As Worksheet For Each ws In ActiveWorkbook.Worksheets If ws.Name = "OldSheetName" Then ws.Name = "NewSheetName" End If Next ws End Sub In this code, the For Each loop iterates through each worksheet in the active workbook. For each sheet, the code checks if the current worksheet's name matches "OldSheetName", and if so, it renames it to "NewSheetName". This technique allows you to efficiently rename multiple sheets with a single macro. When using VBA to rename worksheets, it’s essential to follow best practices and consider potential pitfalls to ensure your macros are efficient, reliable, and user-friendly. One crucial aspect is error handling. Always anticipate potential errors, such as a worksheet with the target name already existing, and handle these errors gracefully using VBA's error handling mechanisms. For instance, you might want to check if the new name is already in use and provide a user-friendly message if an error occurs. Additionally, it's good practice to provide clear and descriptive names for your worksheets, as this can significantly enhance the readability and maintainability of your spreadsheets. Avoid using generic names like "Sheet1" or "Sheet2," and instead, opt for names that are meaningful and indicative of the sheet's content or purpose. Finally, consider the impact of your VBA code on the overall performance of your spreadsheet. While VBA can greatly enhance productivity, it's essential to optimize your code to ensure it doesn't slow down your workbook, especially if you're working with large datasets or complex macros. Best Practice Description Error Handling Implement robust error handling mechanisms to gracefully manage unexpected situations. Descriptive Names Use meaningful names for worksheets to enhance clarity and readability. Performance Optimization Optimize your VBA code to ensure it doesn't hinder the overall performance of your spreadsheet. Remember, while VBA can automate tasks and enhance your spreadsheet's functionality, it's essential to balance its use with good coding practices and user-friendliness. When working with VBA for worksheet renaming, there are a few common pitfalls that can lead to errors or unexpected outcomes. Being aware of these pitfalls and knowing how to avoid them is crucial for writing reliable and efficient VBA code. One common mistake is forgetting to activate the target worksheet before renaming it. VBA code often assumes that the desired worksheet is already active, but this may not always be the case. Always ensure that you select or activate the correct worksheet before attempting to rename it. Here's an example of how to activate a worksheet: Sub ActivateAndRenameWorksheet() Sheets("OldSheetName").Select ActiveSheet.Name = "NewSheetName" End Sub Another pitfall is trying to rename a worksheet that doesn't exist. VBA will raise an error if you attempt to rename a non-existent worksheet, which can disrupt the flow of your macro. Always ensure that the worksheet you're targeting actually exists in the workbook. You can use the On Error Resume Next statement in combination with an If statement to gracefully handle this situation. Sub RenameWorksheetGracefully() On Error Resume Next Sheets("OldSheetName").Name = "NewSheetName" On Error GoTo 0 If Err.Number 0 Then MsgBox "Worksheet not found. Operation aborted." End If End Sub Additionally, be cautious when renaming worksheets that have references in other parts of the workbook or in other workbooks. Renaming a worksheet can break these references, leading to errors or incorrect calculations. Always check for and update any references to the worksheet before renaming it. Excel VBA’s worksheet renaming functionality finds practical applications in various real-world scenarios, especially in industries where data management and analysis are crucial. Consider, for instance, a financial analyst working with a large dataset containing quarterly financial reports for multiple companies. Each quarter's data is stored in a separate worksheet, and the analyst needs to regularly update and reorganize these sheets. By using VBA to rename worksheets, the analyst can quickly and efficiently update the sheet names to reflect the latest quarter, ensuring that the data remains organized and easily accessible. In another scenario, a data scientist working on a machine learning project may have multiple worksheets containing different datasets, features, and models. By using VBA to rename worksheets, the data scientist can easily track the progress of their project, keep datasets organized, and ensure that the naming conventions are consistent and meaningful. These real-world applications demonstrate how VBA's worksheet renaming capability can streamline data management, enhance productivity, and improve the overall user experience when working with complex Excel spreadsheets. Excel VBA for Dynamic Worksheet Renaming One of the most powerful aspects of Excel VBA is its ability to handle dynamic data and situations. This capability is particularly useful when it comes to renaming worksheets, as it allows you to create macros that can adapt to changing data or user input. For example, imagine you have a worksheet where each row represents a different product, and you want to automatically rename the worksheet based on the product name in the first row. By using VBA's dynamic referencing capabilities, you can create a macro that reads the product name from the worksheet and uses it to rename the sheet. Sub RenameWorksheetDynamically() Dim productName As String productName = Sheets("Sheet1").Range("A1").Value Sheets("Sheet1").Name = productName End Sub In this code, the productName variable is assigned the value from cell A1 of the "Sheet1" worksheet. This value is then used to rename the worksheet. This dynamic approach ensures that the worksheet is always named after the product in the first row, even if the data changes. Similarly, you can use dynamic references to rename worksheets based on user input. For instance, you could create a user form where the user enters a new name for the worksheet, and your VBA code then uses this input to rename the sheet. Sub RenameWorksheetWithUserInput() Dim newSheetName As String newSheetName = InputBox("Enter the new name for the worksheet:") Sheets("OldSheetName").Name = newSheetName End Sub In this example, the InputBox function displays a dialog box where the user can enter a new name for the worksheet. The entered value is then used to rename the sheet. This approach provides a user-friendly way to dynamically rename worksheets based on user preferences. Future Implications and Trends in Excel VBA Worksheet Renaming As Excel VBA continues to evolve, the future of worksheet renaming and other related tasks looks promising, offering enhanced capabilities and improved user experiences. One notable trend is the increasing integration of Excel VBA with other programming languages and frameworks. For instance, Excel VBA can now be used in conjunction with Python, R, and other languages to perform advanced data analysis and visualization tasks. This integration opens up new possibilities for dynamic worksheet renaming, allowing users to leverage the strengths of multiple languages within a single Excel workbook. Additionally, the ongoing development of Excel's Object Model and VBA engine promises to bring more robust and efficient worksheet manipulation capabilities. Future updates may include improved error handling, better support for dynamic references, and enhanced performance, making VBA an even more powerful tool for Excel users. Furthermore, the rise of cloud-based collaboration tools and services is expected to influence the future of Excel VBA. With more teams working collaboratively on shared Excel workbooks, there will likely be an increased demand for features that facilitate real-time worksheet renaming and management, ensuring that collaborative projects remain organized and efficient. Future Trend Description Language Integration Excel VBA's integration with other languages will enhance its capabilities and open up new possibilities for worksheet renaming and manipulation. Enhanced Object Model Improvements to Excel's Object Model will lead to more robust and efficient worksheet renaming and management. Cloud Collaboration The rise of cloud-based collaboration tools will drive the development of features that support real-time worksheet renaming and management. As Excel VBA continues to evolve, staying updated with the latest trends and capabilities will ensure that you can leverage its full potential for worksheet renaming and other tasks. While the basic VBA techniques for renaming worksheets are powerful in their own right, there are even more advanced methods that can further enhance your Excel automation capabilities. One such technique is the use of arrays and loops to batch-rename multiple worksheets based on a predefined list of names. This approach is particularly useful when you have a large number of worksheets and need to rename them all at once according to a specific naming convention. Here's an example using a loop to rename worksheets: Sub BatchRenameWorksheets() Dim newNames(1 To 10) As String ' Assuming you have 10 worksheets to rename Dim i As Integer newNames(1) = "Sheet1NewName" newNames(2) = "Sheet2NewName" ' ... For i = 1 To 10 Sheets(i).Name = newNames(i) Next i End Sub In this code, the newNames array is used to store the new names for the worksheets. The loop iterates through the array, renaming each worksheet with the corresponding name. This technique allows you to batch-rename worksheets quickly and efficiently. Another advanced technique involves using Excel's built-in naming conventions to dynamically generate new worksheet names based on specific rules or patterns. For example, you could use the Cells.Add method to automatically name new worksheets based on the last used cell in a column or row. This approach can be particularly useful when you're working with large datasets and need to create new worksheets on the fly. Sub RenameWorksheetBasedOnLastCell() Dim lastCell As Range Set lastCell = Cells(Rows.Count, 1).End(xlUp) Sheets.Add.Name = "NewWorksheet" & lastCell.Address End Sub In this code, the lastCell variable is set to the last used cell in column A. The Sheets.Add method is then used to add a new worksheet, and its name is set to a dynamic value based on the address of the last used cell. This ensures that new worksheets are named consistently and in a predictable manner. Conclusion: Excel VBA Worksheet Renaming - A Powerful Tool for Data Management Excel VBA’s worksheet renaming capabilities offer a powerful tool for data management and automation, enabling users to streamline their workflows and enhance productivity. Whether you’re dealing with large datasets, complex spreadsheet projects, or simply looking to keep your worksheets organized, VBA provides a versatile and efficient solution. From basic worksheet renaming to advanced techniques like batch renaming and dynamic naming conventions, VBA offers a range of functionalities that can be tailored to specific needs. By leveraging these capabilities, users can make their Excel workbooks more dynamic, efficient, and user-friendly. As Excel VBA continues to evolve, its future implications and trends point towards even greater integration with other programming languages, improved object models, and enhanced support for cloud-based collaboration. These advancements will undoubtedly open up new possibilities for worksheet renaming and other related tasks, further cementing Excel VBA's role as a critical tool for data professionals. In conclusion, Excel VBA's worksheet renaming functionality is a testament to the power and versatility of this programming language. By mastering these techniques, users can unlock new levels of efficiency and control over their data, ultimately leading to more effective and productive outcomes. How do I use VBA to rename multiple worksheets at once? + You can use a loop to iterate through each worksheet and rename them. For example, you can use a For Each loop to go through all worksheets in the active workbook and rename them based on a specific naming convention. What is the best way to handle errors when renaming worksheets using VBA? + Implement robust error handling mechanisms to gracefully manage unexpected situations. For instance, use the On Error Resume Next statement in combination with an If statement to check for errors and provide user-friendly messages. Can I use VBA to rename worksheets based on user input? + Yes, you can create a user form where the user enters a new name for the worksheet, and your VBA code then uses this input to rename the sheet. This provides a user-friendly Return to VBA Code Examples This tutorial will cover interacting with Sheet names in VBA. Get Sheet Name Sheet names are stored in the Name property of the Sheets or Worksheets object. The Sheet Name is the “tab” name that’s visible at the bottom of Excel: Get ActiveSheet Name This will display the ActiveSheet name in a message box: MsgBox ActiveSheet.Name Get Sheet Name by index Number This will display the first worksheet name in a message box: MsgBox Sheets(1).Name This will display the name of the last worksheet in the workbook: MsgBox Sheets(Sheets.Count).Name Get Sheet Name by Code Name In the VBA Editor, there is an option to change the “code name” of a Sheet. The code name is not visible to the Excel user and can only be seen in the VBA Editor: In VBA, when working with Sheets, you can reference the usual Tab name: Sheets("TabName").Activate or the VBA code name: CodeName.Activate Referencing the code name is desirable in case the Sheet tab name ever changes. If you allow you Excel user access to changing sheet names you should reference the code name in your VBA code so that a Sheet tab name mismatch doesn’t cause an error. Sheet code names are discussed in more detail here. To get the Sheet name using the VBA Code name, do the following: MsgBox CodeName.Name Rename Sheet You can rename Sheets by adjusting the name property of the Sheets or Worksheets object. Rename ActiveSheet ActiveSheet.Name = "NewName" Rename Sheet by Name Sheets("OldSheet").Name = "NewName" Rename Sheet by Sheet Index Number Here we use 1 to rename the first Sheet in the Workbook. Sheets(1).Name = "NewName" Rename Sheet by Code Name This code will rename a sheet using it’s VBA code name (discussed above): Component.Name = "NewName" Check if Sheet Name Exists We created a function to test if a Sheet with a particular name already exists. 'Test if a Range Exists on a Sheet. 'Leave range blank to test if sheet exists 'Inputs: ' WhatSheet - String Name of Sheet (ex "Sheet1") ' WhatRange (Optional, Default = "A1") - String Name of Range (ex "A1") Function RangeExists(WhatSheet As String, Optional ByVal WhatRange As String = "A1") As Boolean Dim test As Range On Error Resume Next Set test = ActiveWorkbook.Sheets(WhatSheet).Range(WhatRange) RangeExists = Err.Number = 0 On Error GoTo 0 End Function The function will return TRUE if the Sheet exists, or FALSE if it does not. Use the function like so: Sub Test_SheetExists() MsgBox RangeExists("setup") End Sub Copy Sheet and Rename This example is from our article on Copying Sheets. After copying and pasting a Sheet, the newly created sheet becomes the ActiveSheet. So to rename a copied Sheet, simply use ActiveSheet.Name: Sub CopySheetRename2() Sheets("Sheet1").Copy After:=Sheets(Sheets.Count) On Error Resume Next ActiveSheet.Name = "LastSheet" On Error GoTo 0 End Sub Note: We added error handling to avoid errors if the Sheet name already exists. Easily access all of the code examples found on our site. Simply navigate to the menu, click, and the code will be inserted directly into your module. .xlam add-in. (No installation required!) Free Download Return to VBA Code Examples Renaming sheets in Excel is a common task, and with VBA, you can do this at scale with some very simple code. While it’s easy to rename one or two worksheets, when you automate it with VBA, it can save you a lot of time. In this article, I will cover some examples where you can use VBA code to rename sheets in Excel. So, grab your coffee, sit back, and let’s get started! It’s time to become an Excel pro! Below is the VBA code that will rename the active sheet to “Sales Data” Sub RenameActiveSheet() ' Change Active sheet name ActiveSheet.Name = "Sales Data" End Sub While this works, it’s not very useful as it will take you less effort to rename the worksheet manually. However, when we extend this concept, we can do some amazing stuff. For example, below is the VBA code which would rename the active sheet by using the value in cell A1: Sub RenameActiveSheet() ' Change Active sheet name based on cell value ActiveSheet.Name = Range("A1").Value End Sub Note: This code will give you an error in case the cell A1 is empty. While you may still think this isn’t a big-time save, when we extend this concept further, it becomes useful. Imagine running a code, and all the sheets in the workbook are renamed using the cell value in the sheet. Now, that’s something you can not you can do faster manually. So, now that the concept of renaming a sheet is clear, let’s look at more practical examples. Also read: How to Rename a Sheet in Excel (Shortcuts) Below is the VBA code that would change sheet name of the first sheet to Sales Data” Sub RenameFirstSheet() ' Rename the worksheet ThisWorkbook.Sheets(1).Name = "Sales Data" End Sub In the above VBA code, we have used the sheet index number ThisWorkbook.Sheets(1) to refer to the first sheet. You can change the number in the parenthesis, and it will refer to the worksheet in that position in the workbook. Sub RenameSheetsCellValue() ' Declare variables Dim ws As Worksheet Dim cellValue As String ' Loop through each worksheet in the workbook For Each ws In ThisWorkbook.Worksheets ' Read the value from cell A1 of the current worksheet cellValue = ws.Range("A1").Value ' Check if the cell is not empty If cellValue "" Then ' Rename the worksheet with the cell value ws.Name = cellValue End If ' Move to the next worksheet in the loop Next ws ' End the subroutine End Sub The above code loops through all the sheets in the workbook and renames the sheet using the value in cell A1 of each sheet. It also uses an IF statement to check whether the cell is empty. If the cell is empty, it leaves that worksheet without renaming it. Also read: VBA Clear Sheet The below VBA code will change the name of the active sheet to the file name: Sub RenameSheetToFileName() ' Declare a variable to hold the file name Dim fileName As String ' Extract the file name from the workbook's full path fileName = Mid(ThisWorkbook.FullName, InStrRev(ThisWorkbook.FullName, "\") + 1) ' Remove the file extension from the file name fileName = Left(fileName, InStrRev(fileName, ".") - 1) ' Rename the active sheet to the file name ActiveSheet.Name = fileName End Sub The above code first extracts the file name from the full file path and then assigns it to a variable called fileName. It then renames the active sheet to the value that the fileName variable holds. Note: For this code to work, you must have saved your file somewhere. If it’s a new workbook that has never been saved, it will give you an error (Run-time error 5) If you want to rename your sheets based on a list of names you already have in a range, you can easily do that with VBA. Below, I have some names that I want to use to rename the sheets. So, I want the first sheet to be renamed to Sales, the second to Marketing, and so on. Here is the VBA code to do this: Sub RenameSheetsBasedOnList() ' Declare a variables Dim ws As Worksheet Dim i As Integer ' Initialize the counter variable to 1 i = 1 ' Loop through each worksheet in the workbook For Each ws In Worksheets ' If cell is not empty, rename the worksheet using cell value If Range("A1").Offset(i - 1, 0) "" Then ws.Name = ActiveSheet.Range("A1").Offset(i - 1, 0) End If ' Increment the counter variable i = i + 1 Next ws End Sub The above code uses two variables – counter (i) and a worksheet variable (was). It loops through each worksheet in the workbook. Inside the loop, it checks whether the cell at an offset from A1 (using the counter) is empty. If it’s not empty, the code renames the worksheet (ws.Name) based on the value in that cell. The counter (i) increments by one after each iteration of the loop, allowing you to go through all worksheets and rename them based on the list starting from cell A1 in the active sheet. Also read: Protect and Unprotect Sheet Using VBA Below is the VBA code you can use to rename a sheet in an already open workbook. Sub RenameSheetInOpenWorkbook() ' Declare variable Dim targetwb As Workbook ' Set the workbook variable to the already open workbook Set targetwb = Workbooks("OpenWorkbook.xlsx") ' Rename the worksheet targetwb.Sheets("Sheet1").Name = "Sales Data" End Sub In the above code, I have used a variable targetwb and used it to store the open workbook object in which I want to rename the sheet. Then, I refer to the sheet that I want to rename (which is Sheet1 in this case) and change its name. Also read: VBA to Hide or Unhide Sheets Below is the VBA code that allows you to rename a worksheet in a closed workbook. Since VBA cannot work with closed workbooks, the code has to first open the workbook, rename the sheet, and then close it back again. Sub RenameSheetInClosedWorkbook() ' Declare Workbook and Worksheet variables Dim targetwb As Workbook Dim targetws As Worksheet ' Open the workbook that is currently closed Set targetwb = Workbooks.Open("C:\Users\sumit\Downloads\Example.xlsx") ' Reference the worksheet to be renamed Set targetws = targetwb.Sheets("Sheet1") ' Rename the worksheet targetws.Name = "New Sheet Name" ' Save and close the workbook targetwb.Close SaveChanges:=True End Sub Also read: Rename Files Using VBA Below is the VBA code that first checks whether the sheet exists in the workbook or not, and if it exists, then it will rename it. Sub RenameSheetIfExists() ' Declare variables Dim wb As Workbook Dim ws As Worksheet Dim sheetExists As Boolean ' Set reference to the workbook Set wb = ThisWorkbook ' Initialize the flag as False sheetExists = False ' Loop through each worksheet to check if it exists For Each ws In wb.Sheets If ws.Name = "OldSheet" Then sheetExists = True Exit For End If Next ws ' Rename the sheet if it exists If sheetExists Then wb.Sheets("OldSheet").Name = "NewSheet" MsgBox "Sheet has been renamed" Else MsgBox "Sheet does not exist." End If End Sub Using the above code, I want to rename a sheet with the name “OldSheet” to “NewSheet”. To do this, I’ve used a loop to go through each sheet and check the name of the sheet. If the name of the sheet is “OldSheet”, it gets renamed to “NewSheet”, and a message is shown saying, “Sheet has been renamed” Below is the VBA code that will rename the active sheet to the current date Sub RenameSheetWithTodaysDate() ' Declare a variable to hold the formatted date string Dim currDate As String ' Get today's date and format it as YYYY-MM-DD currDate = Format(Date, "dd-mmm-yyyy") ' Rename the active sheet to the formatted date string ActiveSheet.Name = currDate End Sub In the above code, I have used the Date function to get the current date and then used the Format function to format the date in the dd-mmm-yyyy format. This is to ensure the sheet name will be valid, as Excel sheet names cannot contain certain characters like slashes (/). Finally, the ActiveSheet.Name property is set to this formatted date string, effectively renaming the active sheet to the current date. Note that Excel requires unique names for each sheet, so if a sheet with today’s date already exists, running this macro will result in an error. If you want to rename the active sheet to the current date as well as time, you can use the code below. Sub RenameSheetWithTodaysDateTime() ' Declare a variable to hold the formatted date string Dim currDateTime As String ' Get today's date and format it as YYYY-MM-DD currDateTime = Format(Now, "dd-mmm-yyyy_HH-MM-SS") ' Rename the active sheet to the formatted date string ActiveSheet.Name = currDateTime End Sub Note: You can not use a colon (:) in the sheet name, so here I used the format HH-SS-MM to show the time. Want to add a specific prefix to all the sheets in the workbook? The below code can do it for you. Sub AddPrefixToAllSheets() ' Declare variables for the worksheet and the prefix string Dim ws As Worksheet Dim prefix As String ' Set the prefix string prefix = "Sales_" ' Loop through all worksheets in the workbook For Each ws In ThisWorkbook.Worksheets ' Add prefix to the worksheet name ws.Name = prefix & ws.Name Next ws End Sub The above code uses a simple For Next loop to go through each sheet and add the prefix text (which I have assigned to a variable called prefix). You can use the same logic to append any suffix after the sheet name for all the sheets in the workbook. Below are some rules you need to keep in mind when renaming sheets in Excel: You can’t use a name that exceeds 31 characters. You can not use any of the following characters in the sheet name : / \ ? * [or] You can not leave the worksheet name blank You can not use the same name for more than one worksheet. A sheet name cannot start with a space character. Sheet names are not case-sensitive. So, Sheet1 and sheet1 are considered the same by Excel. What is Excel VBA? Other Excel VBA articles you may also like: End Sub ADJUSTABLE PARAMETERS New Worksheet Name: Select the name that you want to rename an active worksheet to by changing cell C2, in the VBA code, to any cell that captures the new name of an active worksheet. Please note that if the cell that contains the name that you want to rename an active worksheet to is located in another worksheet, you need to reference that worksheet in the VBA code. Worksheet Selection: Select the worksheet where you hold the new worksheet name by changing the Parameters worksheet name in the VBA code. Worksheet Selection: Select any worksheet in the workbook that you want to rename. Below are examples of the VBA Rename Sheet.Look at the below sample code.Code:Sub Rename_Example1() Dim Ws As Worksheet Set Ws = Worksheets("Sheet1") Ws.Name = "New Sheet" End SubFirst, we have declared the variable as Worksheet in the above code.Dim Ws As WorksheetNext, we have set the reference to the variable as "Sheet1" using the Worksheets object.Set Ws = Worksheets("Sheet1")Now, the variable "Ws" holds the reference of the worksheet "Sheet1."Now, using the "Ws" variable, we have renamed the worksheet "New Sheet."This code will change the "Sheet1" name to "New Sheet."If we run the code manually or through the shortcut key F5, we will again get a Subscript Out of Range error.We get this error because, in the previous step itself, we have already changed the worksheet named "Sheet1" to "New Sheet." Since there is no longer a worksheet name, "Sheet1" is unavailable. VBA throws this error.We can get all the worksheet names of the workbook in a single sheet. For example, the below code will extract all the worksheet names.Code:Sub Renmae_Example2() Dim Ws As Worksheet Dim LR As Long For Each Ws In ActiveWorkbook