Displaying the Last Saved Date in Excel: A VBA Tutorial

Authored by

Mohammed looti

November 11, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Displaying the Last Saved Date in Excel: A VBA Tutorial*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=16861

For professionals and analysts who manage complex datasets within Microsoft Excel, maintaining an accurate and visible audit trail is not merely a convenience--it is a critical requirement for data governance. One of the most frequently requested pieces of information to display directly on a worksheet is the precise date and time the file was last saved. Unlike standard, volatile functions such as `NOW()` or `TODAY()`, which constantly update and recalculate based on the current system time, the objective here is to capture and display a static, reliable timestamp that documents the last successful save operation. This fixed record is fundamental for robust version control and ensuring data integrity across collaborative projects.

While Excel's standard formula library does not include a simple, direct method for retrieving this specific piece of file information, the solution lies in leveraging the extensibility of Visual Basic for Applications (VBA). Although the prospect of writing code might seem intimidating, the implementation process is surprisingly straightforward, involving the integration of a small, pre-written custom function into your workbook. This comprehensive guide details the precise, step-by-step methodology required to successfully implement this functionality, ensuring you can display the reliable last saved date in any cell you choose.

Accessing Internal File Properties Using VBA

Every time an Excel workbook is saved, the application automatically stores a set of properties related to the file, including the author, the total editing time, the creation date, and most critically, the timestamp of the most recent save action. These properties are collectively known as **document metadata**. Standard Excel formulas are inherently limited to manipulating data contained within cells or performing numerical and logical calculations; they do not possess the necessary access permissions to query these internal file properties directly.

This limitation is exactly why VBA becomes an indispensable tool. VBA empowers users to create custom functions, referred to as **User-Defined Functions (UDFs)**, which are capable of interfacing directly with Excel's powerful object model. By developing a UDF, we can instruct the application to look up the specific property related to the last save time, retrieve that value from the system, and return it to the worksheet, where it behaves exactly like a native built-in formula.

The technical foundation of our solution relies on accessing the BuiltinDocumentProperties collection, which is an integral part of the `ActiveWorkbook` object. This collection holds all the predefined properties we need, including the exact timestamp of the last save operation. Before we can write or paste the necessary code, however, we must first ensure that the essential development tools required for working with VBA are visible and accessible within the Excel interface.

Step 1: Enabling the Developer Tab for Custom Coding

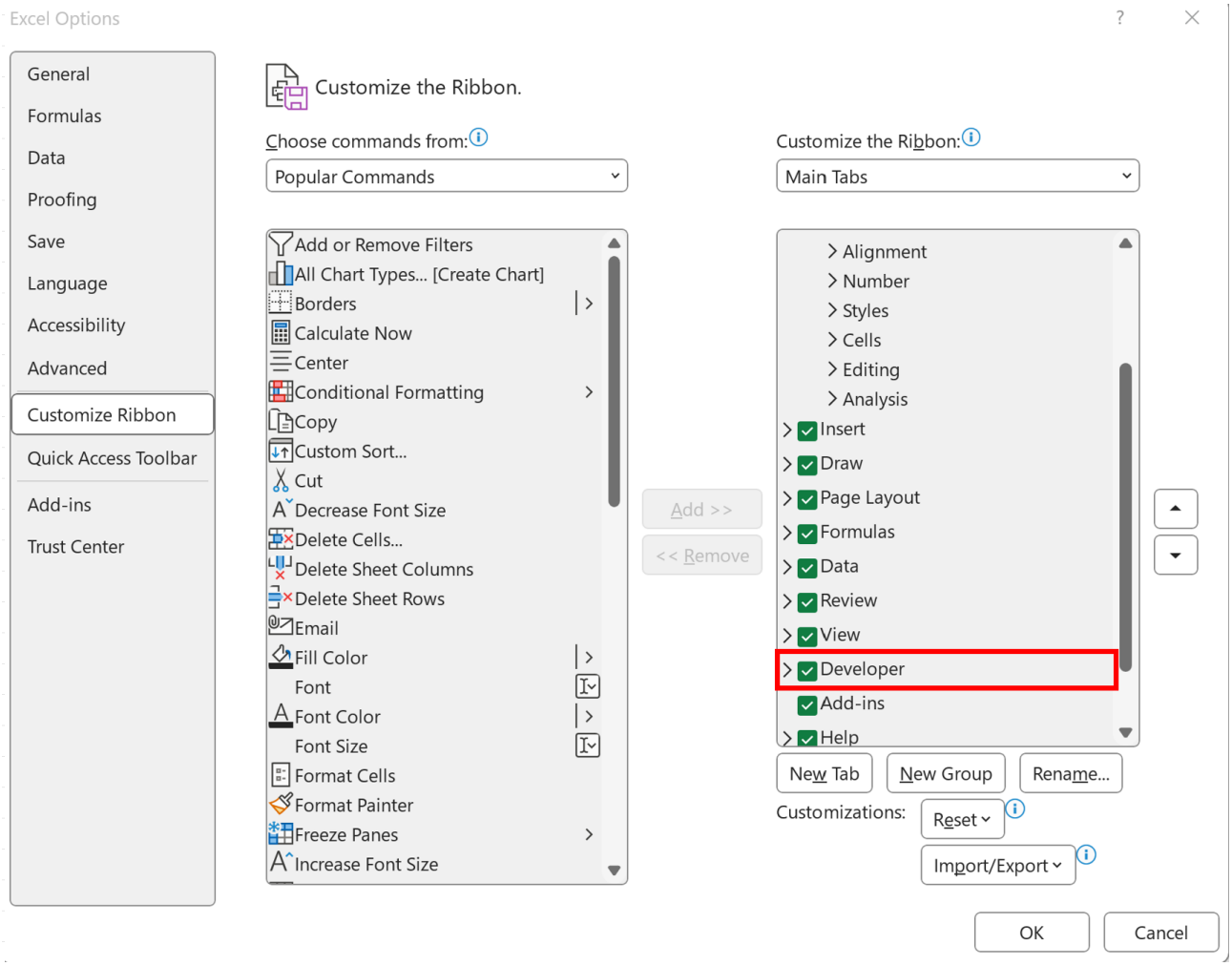
The initial requirement for integrating any custom code into an Excel workbook is making the **Developer** tab visible on the main Excel ribbon. Microsoft intentionally hides this tab by default because it provides access to advanced tools necessary for development, such as recording a macro, inserting form controls, and launching the crucial Visual Basic Editor (VBE)--the environment where all VBA code resides.

To enable the **Developer** tab, please follow these precise configuration steps:

Click on the **File** tab, which is located in the upper-left corner of the Excel window.
From the resulting backstage view, select **Options** near the bottom of the left-hand navigation pane; this action opens the comprehensive Excel Options dialog box.
Within the dialog box, click on the **Customize Ribbon** category.
In the column that lists the **Main Tabs** (typically situated on the right side of the window), scroll down to locate the entry for **Developer**. Ensure that the corresponding checkbox is checked.
Click **OK** to confirm your changes and dismiss the dialog box.

Once this configuration is complete, the **Developer** tab will immediately appear on your main ribbon interface. This tab grants instant access to the necessary tools for the subsequent steps, and it is a one-time setup process for your specific installation of Excel.

The image below illustrates the process of ensuring the **Developer** option is selected within the Customize Ribbon menu:

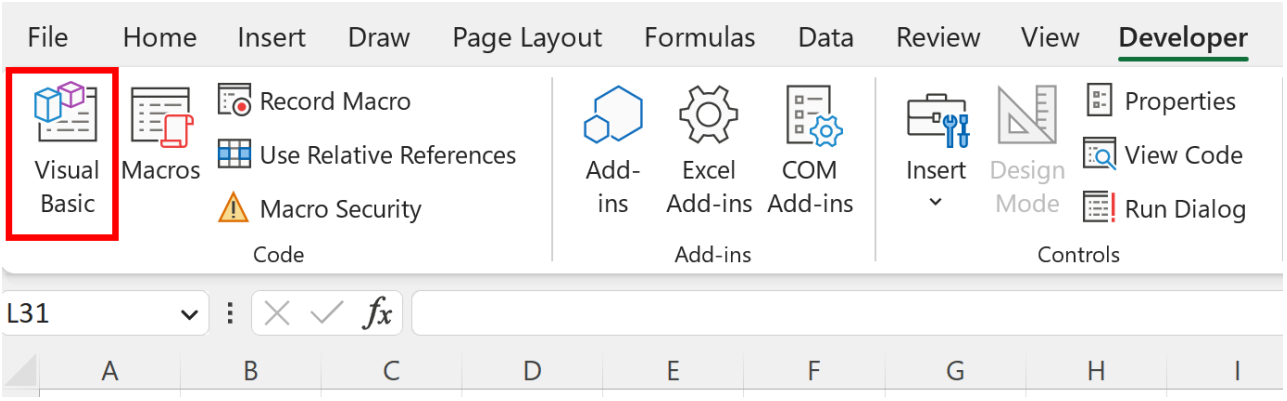


Step 2: Implementing the LastSavedDate Function in the VBE

With the **Developer** tab now accessible, the next step involves opening the Visual Basic Editor (VBE) to insert and define our custom function. The VBE serves as the integrated development environment (IDE) for Excel, where all procedures and functions are written, managed, and stored. Our goal in this step is to define a new function that efficiently reads the specific property holding the last save time.

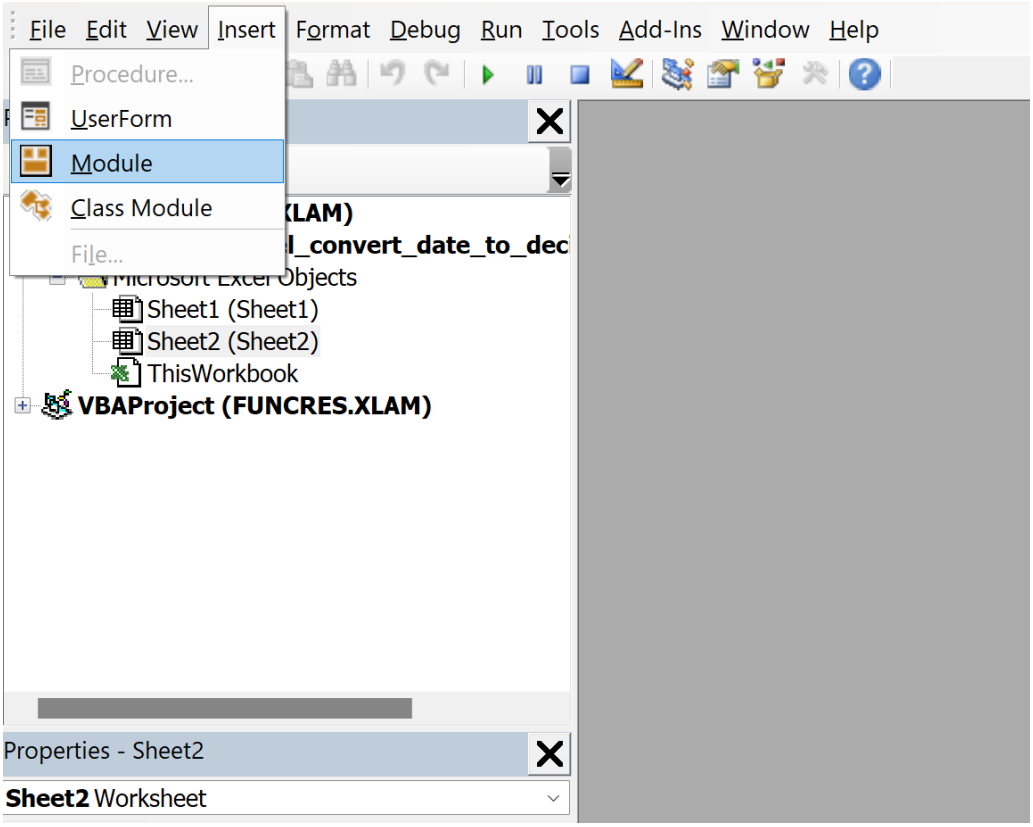
To launch the VBE:

Navigate to the recently enabled **Developer** tab on the Excel ribbon.
Click the **Visual Basic** icon, which is usually the first button on the far left of the ribbon. This command will launch the VBE in a separate, dedicated window.



Inside the VBE, we must insert a new standard **Module**. Modules are specific containers designed to hold general procedures and functions that can be executed from anywhere within the current Excel workbook.

Click the **Insert** menu tab within the VBE menu bar.
Select **Module** from the resulting dropdown menu.



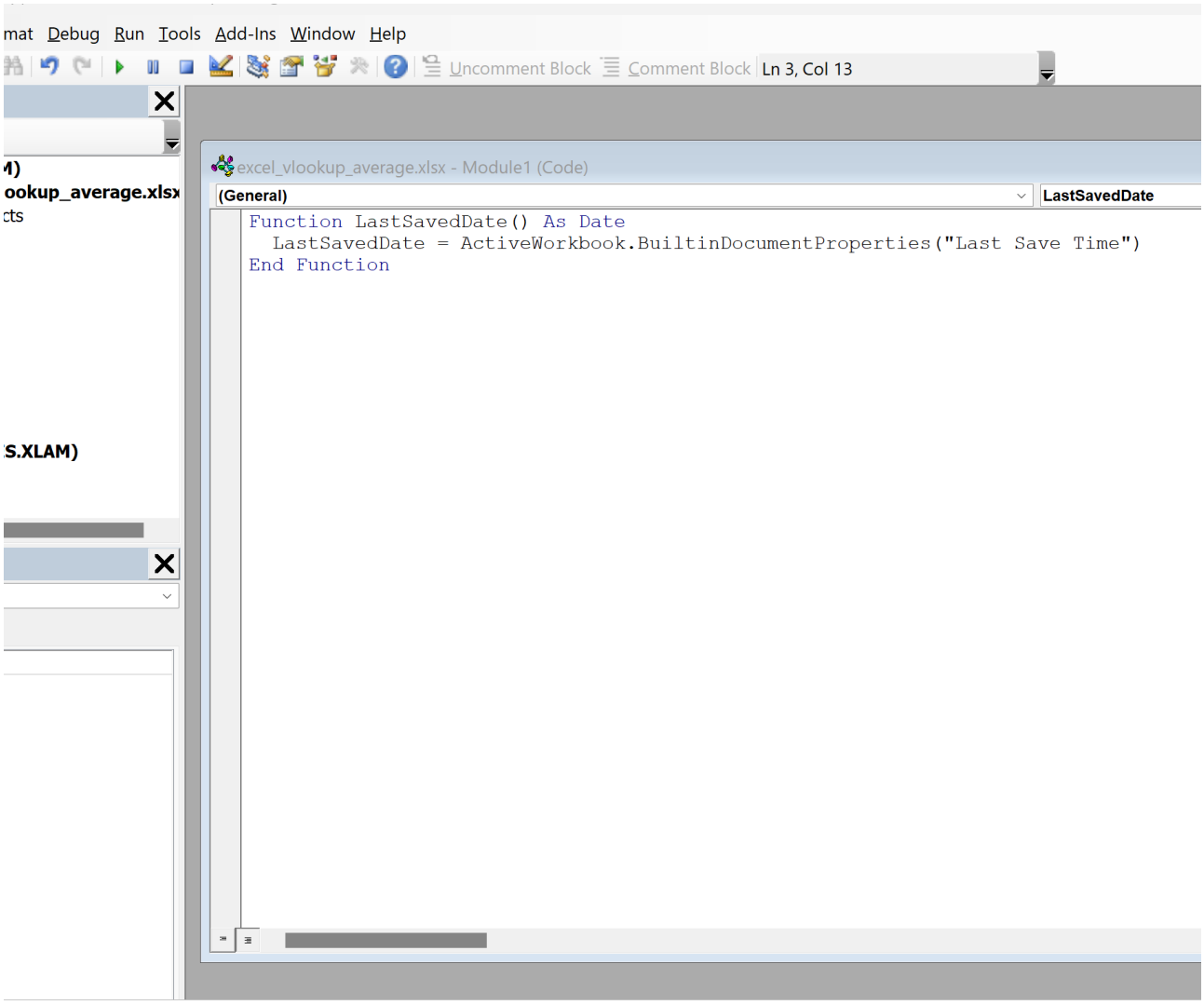
A blank code editor window, titled with the new module's name (e.g., Module1), will appear. Into this window, carefully paste the following concise VBA code, which defines our desired custom

function:

```
Function LastSavedDate() As Date
LastSavedDate = ActiveWorkbook.BuiltinDocumentProperties("Last Save Time")
End Function
```

This function, explicitly named `LastSavedDate`, is designed to return a data type of `Date`, ensuring proper handling of the timestamp. The critical line of code accesses the BuiltinDocumentProperties collection of the `ActiveWorkbook` object and specifically retrieves the value corresponding to the property named "Last Save Time". This property holds the exact date and time stamp recorded during the last successful save operation.

The screenshot below illustrates the correct placement of this function code within the module editor window:



Once the code has been successfully pasted, you may close the Visual Basic Editor and return to your main Excel worksheet. The custom function is now defined and ready for immediate use.

Step 3: Executing the Custom Function in the Worksheet

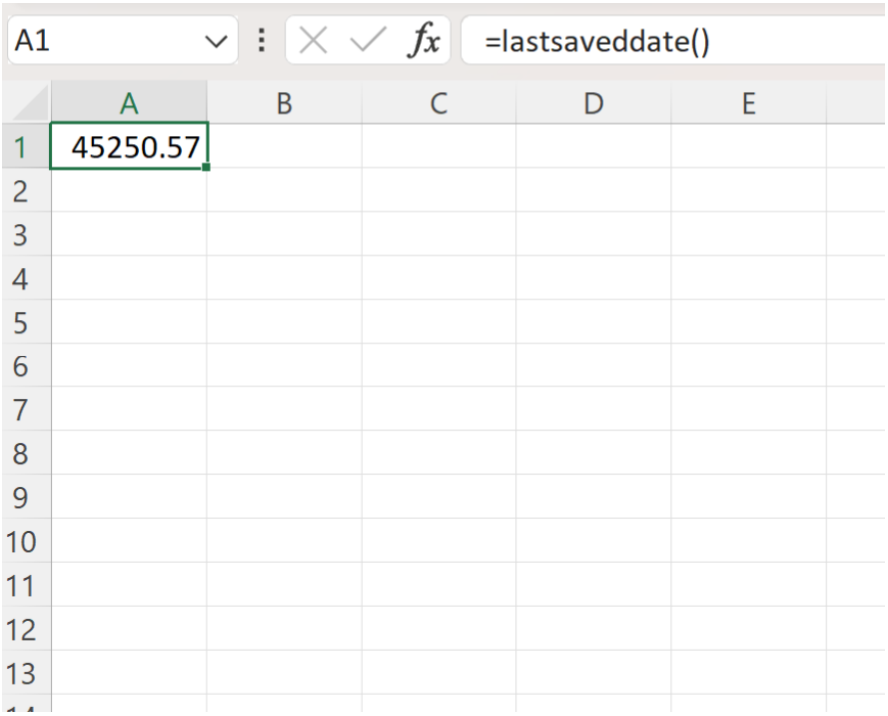
After the `LastSavedDate` function has been defined within a standard module, it integrates seamlessly into Excel, functioning exactly like any of Excel's native formulas. This means you can call it directly from the formula bar in any cell within the active workbook. Calling the function is the straightforward final step needed to retrieve and display the saved date.

To implement the function, navigate to the specific cell where you want the last saved date to appear (for illustration purposes, we will use cell **A1**). In the formula bar, type the following formula:

=LastSavedDate()

Upon pressing the Enter key, the cell will execute the underlying macro created in the VBE, retrieve the "Last Save Time" property from the document metadata, and display the resulting value.

Here is an example of placing this custom formula into cell **A1**:



It is crucial to observe that when the formula is first executed, the result returned by the function will typically appear as a large, often five-digit, numeric value rather than a recognizable date

format. This is Excel's standard internal mechanism for storing dates and times, known as a **serial date number**. This number represents the total count of days elapsed since the reference date of January 1, 1900.

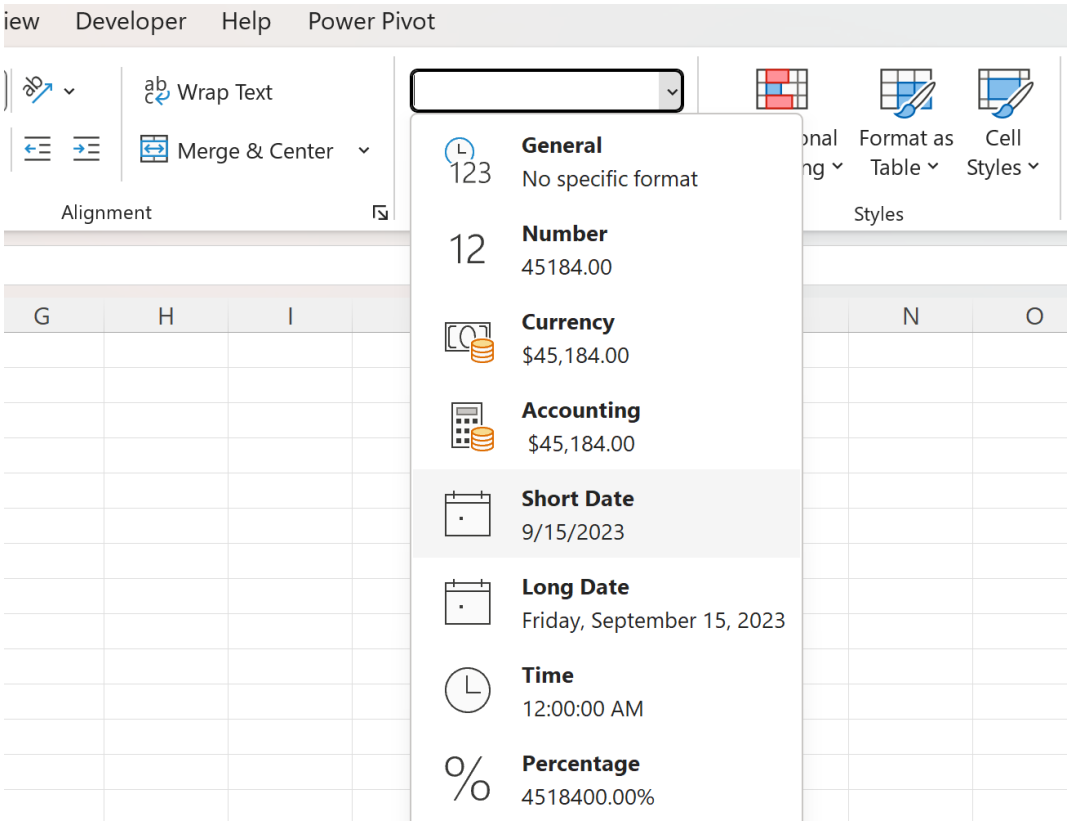
Step 4: Formatting the Serial Number Output as a Readable Date

Since the raw serial numeric output is not intuitive for end-users, the final and essential step involves applying the correct cell formatting. This process instructs Excel to interpret the underlying serial number as a calendar date and time. Crucially, applying formatting does not alter the actual value returned by the function; it only changes how that value is visually displayed.

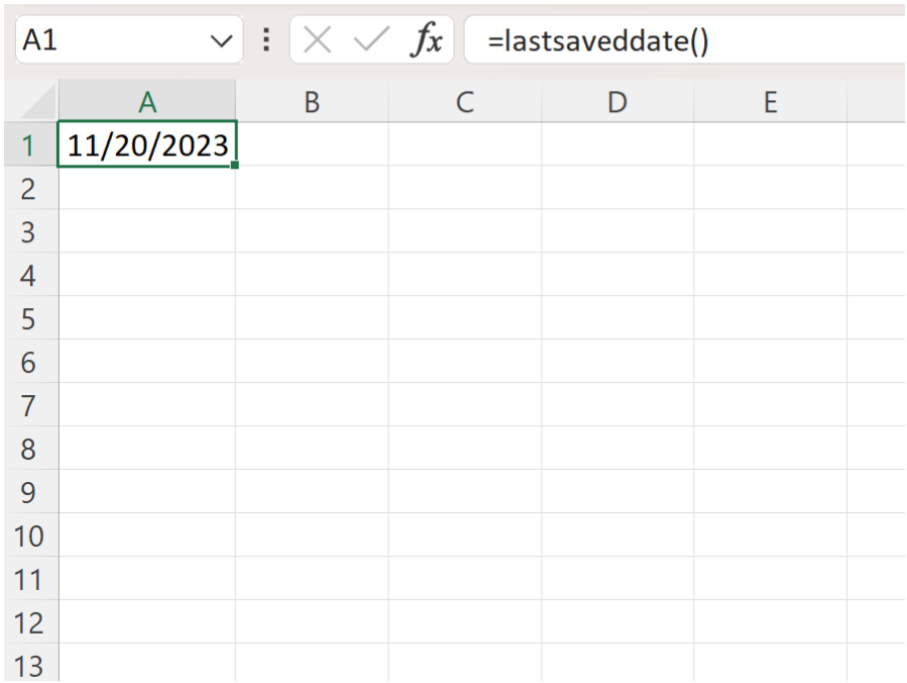
To correctly format the output:

Ensure that the cell containing the formula (e.g., **A1**) remains selected.
Navigate back to the **Home** tab on the Excel ribbon interface.
Locate the **Number** group, typically found near the center of the ribbon.
Click the **Number Format** dropdown menu (which likely currently displays "General").
From the extensive list of options, select **Short Date** or **Long Date**, based on your specific presentation requirements. For most auditing and tracking purposes, **Short Date** is generally the preferred choice.

The following image illustrates the process of selecting the **Short Date** format option:



Immediately after the formatting is applied, the numeric value in cell **A1** is transformed into a clear, easily readable date format:



As clearly demonstrated in this final example, the Excel workbook was last saved on **11/20/2023**. This date will automatically update the very next time the user saves the file, establishing a dynamic yet reliable audit timestamp permanently embedded within the worksheet content.

Conclusion and Essential Macro Considerations

The technique of implementing a User-Defined Function using VBA to dynamically display the last saved date is an exceptionally effective method for incorporating crucial metadata tracking into any serious spreadsheet application. This solution guarantees that key documents always reflect their latest version history, which is invaluable for streamlining collaboration and meeting compliance requirements. However, because this functionality relies entirely on a macro, it is absolutely vital to remember that the workbook must be saved using a macro-enabled file format (e.g., `.xlsm`). If saved in the standard `.xlsx` format, the custom function and all associated code will be permanently stripped from the file.

Furthermore, users should maintain a precise understanding of the function's behavior: the displayed date will only update upon a physical save action performed by the user. If significant changes are made to the data but the file is not explicitly saved, the displayed date will remain unchanged. This behavior is intentional, as it accurately reflects the file's last recorded saved state, providing true version control rather than displaying the current, potentially unsaved, state.

Additional Resources for Advanced Excel Development

To further expand your proficiency in advanced Excel operations and automation, we recommend exploring related topics focusing on extending Excel's capabilities through code and object manipulation:

How to automatically run a macro when a workbook opens.
Advanced techniques for using conditional formatting based on date properties.
Tutorials explaining how to perform other common file property operations using the Excel object model.