# How to Extract Text Before a Specific Character in Google Sheets Using the LEFT Function

Authored by

**Mohammed looti**

November 10, 2025

## Introduction to Dynamic String Truncation in Google Sheets

The ability to manipulate textual data efficiently is fundamental when working with spreadsheets. In Google Sheets, the **LEFT function** is typically employed to extract a specified number of characters starting from the beginning (the left side) of a text string. For instance, if you wanted the first five characters of "Apple_Pie," you would use `=LEFT("Apple_Pie", 5)`, resulting in "Apple."

However, data is rarely uniform. When dealing with variable-length strings, simply specifying a static character count is insufficient and often impractical. Consider a database where entries are structured as `ProductName_SKU`. The product names vary in length, yet they are consistently separated from the SKU by an underscore. To extract only the product name, we need a dynamic solution that tells the **LEFT function** precisely where the relevant text ends, regardless of how long that text may be.

This challenge requires combining the text extraction power of **LEFT** with a positional function that can locate the exact index of a specific character, often referred to as a delimiter. By calculating the position of the delimiter and adjusting that value, we can create a powerful, dynamic formula capable of handling diverse string lengths automatically, leading to cleaner and more reliable data processing workflows.

## Deconstructing the Core Formula: LEFT and FIND

To dynamically extract all characters from the left side of a string up to the point where a particular character is encountered, we must integrate the **LEFT function** with the FIND function. The **LEFT function** requires two arguments: the text string itself and the number of characters to extract. Since we don't know the character count beforehand, the second argument must be calculated dynamically using **FIND**.

The FIND function is designed to locate the starting position of a specified substring within a text string, returning an integer that represents that position. For example, if we search for an underscore ("_") in the string "Boston_Celtics," the **FIND** function will return 7, because the underscore is the seventh character. Crucially, we do not want the underscore itself to be included in our resulting extraction.

This leads us to the complete, generalized formula structure. By taking the position returned by **FIND** and subtracting 1, we ensure that the **LEFT function** stops extracting characters just before the specified delimiter is reached. The formula shown below represents the most efficient way to achieve this dynamic truncation in Google Sheets:

**=LEFT(cell, FIND("specific_character", cell)-1)**

For a concrete illustration, if we apply this logic to a cell, say **A2**, and aim to stop extraction immediately before an underscore ("_"), the resulting formula would look like the second example provided below. This construction is robust because it automatically adjusts the length parameter for the **LEFT function** based on the actual content of the cell.

**=LEFT(A2, FIND("_", A2)-1)**

## Practical Application: Extracting Data Using a Specific Delimiter

To fully appreciate the utility of this combined formula, let us consider a practical scenario involving structured data. Suppose we are managing a list of basketball team affiliations where the full entry includes both the city and the mascot, separated by an underscore. Our objective is to isolate only the city name for a focused analysis.

The data, organized in column A of the spreadsheet, contains entries such as "Boston_Celtics" and "Chicago_Bulls." Since the lengths of the city names vary (e.g., "Boston" has six characters while "Chicago" has seven), a fixed length extraction would fail. We rely entirely on the underscore acting as our positional marker. The following list represents our initial dataset:

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Team** | | | |
| 2 | Mavericks_Team | | | |
| 3 | Rockets_Team | | | |
| 4 | Hornets_Team | | | |
| 5 | Pacers_Team | | | |
| 6 | Raptors_Team | | | |
| 7 | Thunder_Team | | | |
| 8 | Pelicans_Team | | | |
| 9 | Nuggets_Team | | | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |

To perform the required extraction, we initiate the process by inputting the formula into cell **B2**. This formula specifically targets the content of cell **A2**, instructing Google Sheets to find the position of the underscore, subtract one from that position, and then use the resulting number as

the length for the **LEFT function**. This guarantees that only the characters preceding the underscore are returned.

**=LEFT(A2, FIND("_", A2)-1)**

Once the formula is entered into **B2**, it correctly extracts "Boston." The beauty of this dynamic approach is realized when we extend this formula across the remaining dataset, automatically adjusting the extraction length for every entry, regardless of the text length.

## Step-by-Step Implementation and Results

After successfully applying the formula to the first data point in cell **B2**, the next step in efficient spreadsheet management is to propagate this logic down the column. We achieve this by clicking on the small square handle (the fill handle) at the bottom right corner of cell **B2** and dragging it downwards. This action ensures that the relative cell reference (A2) correctly adjusts to A3, A4, and so on, applying the dynamic extraction logic to every team name in our list.

As the formula is dragged down, column B rapidly populates with the extracted city names. For "Chicago_Bulls" in A3, the **FIND function** locates the underscore, and **LEFT** extracts just "Chicago." Similarly, for "LA_Lakers," it extracts "LA." This process demonstrates how effectively the combination of **LEFT** and **FIND** handles varying string lengths, providing a clean column of truncated data ready for further analysis or reporting.

B2 =LEFT(A2, FIND("_", A2)-1)

| | A | B | C |
|---|---|---|---|
| 1 | **Team** | **Team Name Until Underscore** | |
| 2 | Mavericks_Team | Mavericks | |
| 3 | Rockets_Team | Rockets | |
| 4 | Hornets_Team | Hornets | |
| 5 | Pacers_Team | Pacers | |
| 6 | Raptors_Team | Raptors | |
| 7 | Thunder_Team | Thunder | |
| 8 | Pelicans_Team | Pelicans | |
| 9 | Nuggets_Team | Nuggets | |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |

The resulting data in Column B now consistently displays all characters found in each team name up until the first occurrence of the underscore. This outcome validates the effectiveness of using the FIND function to calculate the necessary extraction length dynamically, achieving precise data segmentation without manual intervention for each unique entry.

## Robust Error Handling with the IFERROR Function

While the **LEFT** and **FIND** combination is highly effective, it introduces a potential point of failure. If the specified delimiter (in our case, the underscore) is not present within the text string being analyzed, the FIND function is unable to return a position and instead throws the standard **#VALUE!** error. This error propagates through the entire formula, resulting in a disruptive error message within the spreadsheet cell.

For professional and reliable spreadsheet models, these errors must be handled gracefully. This is where the IFERROR function becomes indispensable. The **IFERROR function** takes two arguments: the primary calculation (the value or formula to check) and the value to return if that calculation results in an error. By wrapping our dynamic extraction formula within **IFERROR**, we can specify a user-friendly or meaningful output instead of the harsh **#VALUE!** indicator.

For instance, if a team name entry (such as "Mavericks") lacks an underscore, the **FIND** operation fails. We can instruct Google Sheets to return a specific text message, like "None Found," or

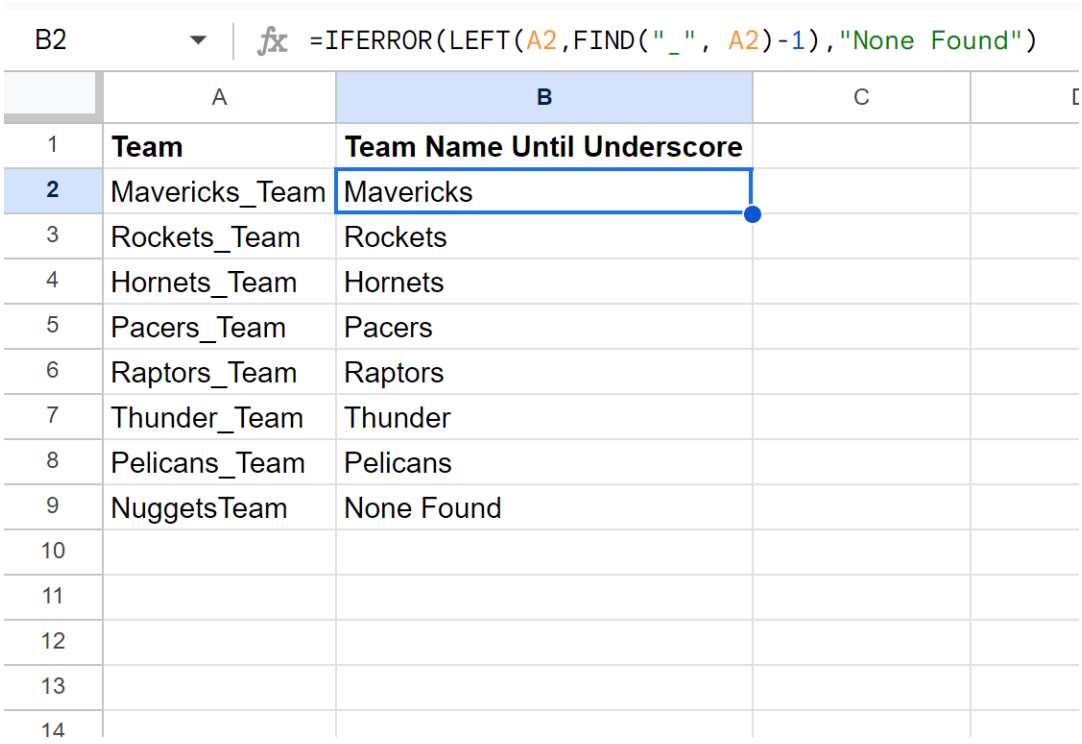
perhaps just an empty cell, ensuring that the spreadsheet remains clean and the error condition is explicitly noted without breaking the entire calculation chain. The structure of the enhanced formula looks like this:

**=IFERROR(LEFT(A2,FIND("_", A2)-1),"None Found")**

## Demonstrating Error Mitigation

Implementing the IFERROR function drastically improves the robustness of our data extraction process. When we apply the modified formula to our existing dataset, including rows that deliberately omit the underscore, the spreadsheet handles the exceptions seamlessly.

As shown in the following screenshot, the formula successfully extracts the city names where the underscore is present. However, for an entry like the one in cell **A9**, which contains "Mavericks" without any separator, the **FIND** function fails, triggering the error handler. Instead of displaying **#VALUE!**, cell **B9** cleanly returns the designated alternative text: "None Found."




| | A | B | C |
|---|---|---|---|
| 1 | **Team** | **Team Name Until Underscore** | |
| 2 | Mavericks_Team | Mavericks | |
| 3 | Rockets_Team | Rockets | |
| 4 | Hornets_Team | Hornets | |
| 5 | Pacers_Team | Pacers | |
| 6 | Raptors_Team | Raptors | |
| 7 | Thunder_Team | Thunder | |
| 8 | Pelicans_Team | Pelicans | |
| 9 | NuggetsTeam | None Found | |
| 10 | | | |
| 11 | | | |
| 12 | | | |
| 13 | | | |
| 14 | | | |

This illustrates a critical best practice in spreadsheet development: anticipating and managing data inconsistencies. Furthermore, it is important to note the flexibility of the **IFERROR** argument. You are not limited to returning "None Found." You can choose to return a numerical value (such as 0), an empty string (by using `""`), or even the original value of the cell itself (by referencing `A2` as the second argument), depending on the requirements of your specific data processing task. This

adaptability ensures maximum control over the output quality.

## Additional Resources for Advanced Text Manipulation

Mastering the combination of **LEFT** and **FIND** is a significant step in advanced string manipulation within Google Sheets. Understanding how to dynamically calculate parameters for text functions allows for complex data cleaning and parsing operations.

For those interested in exploring further techniques for handling text string data, particularly when dealing with more complex patterns or multiple delimiters, exploring regular expression functions like **REGEXEXTRACT** or other extraction functions such as **MID** and **RIGHT** can unlock even greater computational power.

The following tutorials explain how to perform other common tasks in Google Sheets: