Google Sheets: Extract Text Before a Character

Authored by

stats writer

November 17, 2025

Google Sheets stands out as an incredibly powerful and versatile online spreadsheet application, essential for modern data management. It provides robust capabilities for users to create, manipulate, store, and collaboratively share complex datasets across various platforms. In the realm of data processing, a common necessity is the ability to segment or parse textual information, especially when dealing with data imported from external sources where consistency might be lacking. The function allowing users to extract text before a specific character or delimiter is paramount to successful data cleaning and preparation.

When external data is imported, it often contains combined values--such as product codes followed by descriptions, or names separated from dates by a hyphen, comma, or specific phrase. Manually separating these components in a large dataset is inefficient and prone to error. Fortunately, Google Sheets offers highly sophisticated functions, particularly those involving Regular Expressions, to handle these complex parsing tasks automatically. Mastering the technique of extracting the beginning portion of a string--the text occurring before a specified marker--is a foundational skill for any serious spreadsheet user.

This comprehensive article will delve into the technical importance and practical application of extracting text before a defined character in Google Sheets. We will focus specifically on the efficient and flexible function, REGEXEXTRACT, demonstrating how its implementation simplifies complex data cleaning workflows. By the end of this tutorial, readers will possess the expertise to construct robust formulas capable of precise text manipulation, significantly enhancing their data processing capabilities.

The REGEXEXTRACT Function: The Superior Choice

While Google Sheets offers multiple functions for text manipulation, such as **LEFT** and **FIND**, these often become cumbersome when dealing with complex or inconsistent patterns. For reliable, versatile extraction based on a specific character or sequence, the REGEXEXTRACT function is the definitive tool. REGEXEXTRACT is designed to extract matching substrings according to a supplied Regular Expression pattern, offering significantly more flexibility than standard text functions.

To extract all text preceding a specific delimiter--for instance, the word "our"--you utilize a specific Regular Expression syntax. The following structure represents the powerful formula used to perform this crucial data parsing operation in any given cell:

=REGEXEXTRACT(A2,"(.*)our.*")

This specialized formula is engineered to process the content within cell **A2**. Crucially, it isolates and returns the portion of the text that occurs immediately before the target string, which in this

initial demonstration is the sequence "our." The power of this approach lies in the utilization of capturing groups within the Regular Expression pattern.

The core mechanism relies on the REGEXEXTRACT function combined with the specific pattern `"(.*)our.*"`. This pattern directs the engine to look for any sequence of characters (`.*`), specifically capture the characters that precede the delimiter (`(.*)`), and then ensure the delimiter "our" is present (`our`), followed potentially by any remaining characters (`.*`). The parentheses `()` around the first `.*` create the capturing group, which is what REGEXEXTRACT ultimately returns.

Understanding the Regular Expression Pattern

To effectively utilize REGEXEXTRACT for precise text segmentation, a fundamental understanding of the Regular Expression syntax is required. The pattern used, `"(.*)our.*"`, is structured to achieve the precise goal of isolating the prefix of the string. Each component of this pattern plays a critical role in defining the extraction logic, ensuring only the desired text is returned, which is essential for accurate data cleaning.

Let us break down the components of the pattern `"(.*)our.*"` used within the formula `=REGEXEXTRACT(A2, "(.*)our.*")`. The entire pattern is enclosed in double quotes, as it is a literal string argument for the function. Internally, the pattern relies on three key elements: the wildcard dot (`.`), the quantifier asterisk (*), and the capturing parentheses (`()`).

`.` **(Dot):** This is the wildcard character, matching any single character (except newline, typically).
`*` **(Asterisk):** This is a quantifier meaning "zero or more occurrences" of the preceding element. Combined, `.*` matches any sequence of characters of any length.
`()` **(Parentheses):** These create a "capturing group." When REGEXEXTRACT finds a match, it returns only the content enclosed within the first set of parentheses.

Therefore, `(.*)` specifically captures everything from the beginning of the cell contents up until the next part of the pattern is matched. The inclusion of `our` anchors the match, specifying that the capturing group must be followed immediately by this target sequence. Finally, the trailing `.*` simply ensures the rest of the string, regardless of its length, is accounted for in the overall match, allowing the pattern to succeed even if the target delimiter is not the last element in the cell.

Practical Application: Extracting Text Before "our" (Example 1)

To illustrate the efficiency of this method, consider a common scenario where a column of data contains phrases that need to be truncated at a specific marker. Suppose we have the following list of descriptive phrases located in column A of Google Sheets, and our objective is to extract only the text that precedes the delimiter "our."

The dataset includes mixed text, making a simple **LEFT** function inadequate because the position of "our" changes in each row. We rely on the pattern matching capability of REGEXEXTRACT to dynamically identify the cutoff point for every entry, ensuring precise data cleaning across the entire dataset.

	A	B	C
1	**Phrase**		
2	Mike is our best player		
3	Chad is our best player I think		
4	Doug is our worst player		
5	Andy is our head coach		
6	Bob is our assistant coach		
7	Greg is our leader		
8	John is our best backup player		
9			
10			
11			
12			
13			
14			
15			
16			

To execute the extraction, we enter the specific Regular Expression formula into cell **B2**, targeting the content of cell **A2**. This formula is identical to the structure discussed previously, utilizing `(.*)` to capture the prefix and `our.*` to define the endpoint of the capture:

=REGEXEXTRACT(A2,"(.*)our.*")

Once the formula is entered into **B2**, the user can then apply standard spreadsheet practices by clicking and dragging the formula down to the remaining cells in column B. This action automatically adjusts the cell reference (e.g., A2 becomes A3, A4, and so on) and executes the text extraction for every entry in the source column, providing instantaneous results and significantly accelerating the workflow.

B2 =REGEXEXTRACT(A2,"(.*)our.*")

	A	B	C
1	**Phrase**	**All Text Before "our"**	
2	Mike is our best player	Mike is	
3	Chad is our best player I think	Chad is	
4	Doug is our worst player	Doug is	
5	Andy is our head coach	Andy is	
6	Bob is our assistant coach	Bob is	
7	Greg is our leader	Greg is	
8	John is our best backup player	John is	
9			
10			
11			
12			
13			

As clearly demonstrated in the resulting table, Column B now accurately and reliably displays only the extracted text that occurred immediately before the string "our" for each corresponding phrase in Column A. This successful application confirms the formula's effectiveness in isolating dynamic text segments based on a fixed delimiter.

Adapting the Formula for Different Delimiters (Example 2)

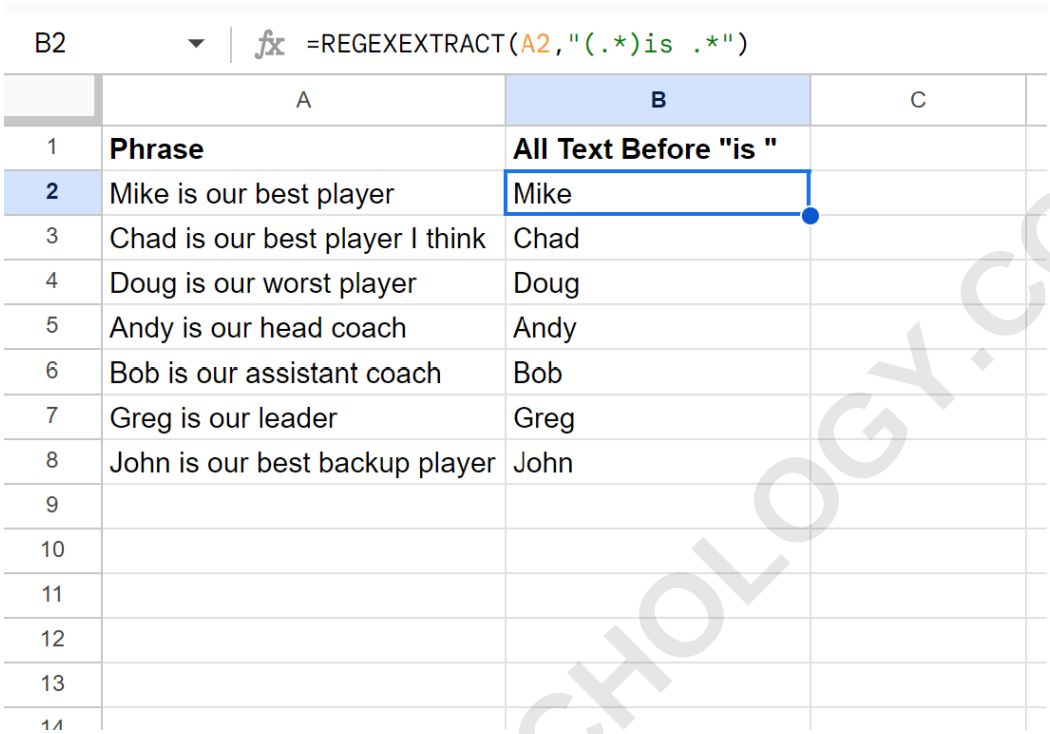
The true versatility of the REGEXEXTRACT method lies in its ease of adaptation. To extract the text before a different specific character, word, or sequence, the user simply needs to modify the delimiter specified within the Regular Expression pattern, replacing the existing "our" with the new target. This flexibility makes the method applicable to virtually any data parsing requirement, whether the delimiter is a comma, a hyphen, a space, or an entire phrase.

Consider a scenario where the desired extraction point is the word "is" followed by a space (represented as `"is "`). Keeping the space in the delimiter is crucial if you want to exclude that space from the final extracted text. By defining the pattern `"(.*)is .*"`, we instruct the function to capture everything (`(.*)`) that precedes the sequence "is " (`is `). The trailing `.*` again accounts for the remainder of the cell's contents.

For example, we input the following modified formula into cell **B2** to extract all text before the occurrence of "is " from the source cell **A2**:

=REGEXEXTRACT(A2,"(.*)is .*")

After applying this formula and dragging it down the column, the output reflects the new delimiter. This demonstrates how minor adjustments to the pattern enable sophisticated and context-specific data cleaning operations, allowing users to rapidly reconfigure their extraction logic without changing the fundamental formula structure.



	A	B	C
1	**Phrase**	**All Text Before "is "**	
2	Mike is our best player	Mike	
3	Chad is our best player I think	Chad	
4	Doug is our worst player	Doug	
5	Andy is our head coach	Andy	
6	Bob is our assistant coach	Bob	
7	Greg is our leader	Greg	
8	John is our best backup player	John	
9			
10			
11			
12			
13			

Column B now successfully displays all text preceding "is " for each entry in Column A, confirming the successful reconfiguration of the Regular Expression pattern.

Handling Edge Cases and Specific Characters

When employing Regular Expressions, it is important to be aware of characters that hold special meaning within the regex syntax, often referred to as metacharacters. If your desired delimiter includes characters like `.`, `*`, `+`, `?`, `^`, `$`, `|`, , `(`, `)`, or , they must be "escaped" using a backslash () so that they are treated as literal characters rather than commands.

For example, if you wished to extract text before a literal question mark (?), the pattern would need to be written as `"(.*)?.*"`. The backslash neutralizes the special meaning of the question mark, allowing the extraction to target the character itself. Failing to escape metacharacters will cause the formula to return an incorrect result or an error, as the regex engine will interpret the character as a command, leading to parsing failures.

Furthermore, managing whitespace is critical. If your delimiter is followed by a space, as in the "is " example, including the space in the delimiter ensures that the extracted text is clean and does not

include unnecessary trailing space. Conversely, if you want the extracted text to include the space before the delimiter, you must ensure your capturing group (`(.*)`) is set up correctly and the delimiter definition is adjusted to omit any preceding whitespace.

Alternatives to REGEXEXTRACT for Simple Extractions

While REGEXEXTRACT is the most robust solution for complex patterns, simpler cases where the delimiter is guaranteed to exist and is a non-regex character can sometimes be handled using a combination of the **LEFT** and **FIND** functions. This approach is less flexible but may offer slightly faster processing times for extremely large datasets or users less familiar with Regular Expressions.

The typical structure involves finding the position of the delimiter using **FIND** and then using **LEFT** to pull back all characters up to that position, minus one character to exclude the delimiter itself. The structure looks like this: `=LEFT(A2, FIND("delimiter", A2) - 1)`. This formula extracts the text in cell A2 up to the first occurrence of "delimiter." Note that **FIND** is case-sensitive, while **SEARCH** is case-insensitive, giving users a choice depending on their data cleaning needs.

However, it is vital to acknowledge the limitations of this traditional approach. If the delimiter is not found, the **FIND** function returns an error (`#VALUE!`), which requires wrapping the entire formula in an **IFERROR** statement to handle exceptions. In contrast, if the pattern is not found, REGEXEXTRACT simply returns `#N/A`, which is often easier to manage, especially when dealing with inconsistent incoming string data.

Conclusion and Review of Best Practices

In conclusion, the REGEXEXTRACT formula, powered by the flexibility of a Regular Expression pattern, is the definitive and most powerful tool available in Google Sheets for extracting text before a specified character or sequence. This function moves far beyond the limitations of simpler text formulas by allowing precise definition of capturing groups and delimiters, regardless of their variable position within the source cell.

By mastering the pattern `"(.*)delimiter.*"`, users gain the ability to quickly and accurately perform complex data parsing tasks, making imported or messy data immediately usable. Always ensure that special regex characters are escaped with a backslash if they are part of your intended delimiter. Consistent application of these techniques ensures high-quality data cleaning and preparation across all your spreadsheet projects.

We encourage readers interested in continuing their mastery of text manipulation to explore related functions, particularly for extracting data that occurs **after** a character.

Google Sheets: How to Extract Text After a Character

The implementation of the REGEXEXTRACT formula represents a significant advancement in spreadsheet data handling efficiency. By understanding this formula and how to apply the capturing group logic, you can quickly and easily extract text before specific characters in Google Sheets, transforming raw data into structured information ready for analysis.