# How to Easily Remove the First Character from a String in SAS

Authored by

**stats writer**

November 21, 2025

In the realm of statistical analysis and data processing, manipulating text data is a routine requirement. When working with the powerful analytical software, SAS, one frequently encounters scenarios where the initial characters of a variable need to be removed or modified. Fortunately, SAS provides robust functions specifically designed for string manipulation, offering efficient ways to cleanse and standardize textual data.

The primary tool for extracting specific portions of text is the **SUBSTR** function. This function is incredibly versatile and allows users to isolate a substring based on defined start and length parameters. Understanding the mechanics of **SUBSTR** is essential for any advanced SAS programmer who needs fine-grained control over their character variables. The SUBSTR function fundamentally operates by taking three potential arguments: the original input string, the starting position for the extraction, and optionally, the number of characters to be extracted.

To effectively remove the first character, we leverage the function's indexing capability. By instructing SUBSTR to begin extraction at the second position (since character indices start at 1 in SAS), we automatically bypass the initial character. If the third argument (number of characters) is omitted, SUBSTR intelligently defaults to extracting all remaining characters until the end of the string. This simple modification--setting the start position to 2--is the most elegant and efficient method for achieving our goal of dropping the leading character.



## Implementing the SUBSTR Function for Leading Character Removal

The most straightforward and widely accepted technique for eliminating the initial character of a string variable within SAS programming is through the utilization of the powerful **SUBSTR** function. This method is preferred due to its clarity, minimal overhead, and direct approach to substring extraction. While other methods exist, such as using regular expressions via the **PRXCHANGE** function, **SUBSTR** remains the core technique for simple positional trimming tasks.

To implement this operation, the code must reside within a DATA step, which is the primary mechanism in SAS for creating, modifying, or combining datasets. Within this step, we read the original data and then assign the modified string back to the variable, effectively overwriting the original value with the truncated version. This ensures that the newly created dataset contains the cleaned text values ready for subsequent analysis.

The following basic syntax demonstrates how to apply the **SUBSTR** function to a character variable named `string_var` within a standard DATA step, creating a new dataset called `new_data`. Notice the critical inclusion of the number `2` as the second argument, which dictates the starting position of the extraction.

**data new_data;**

```
set original_data;
string_var = substr(string_var, 2);
run;
```

The logic within this concise snippet is highly efficient. When `substr(string_var, 2)` is executed, SAS processes the input variable `string_var`. It identifies the index 2 as the starting point, effectively skipping the character at index 1. Since the third optional argument (length of extraction) is omitted, SAS understands the instruction as: "Take everything from position 2 until the very last character." This successful extraction results in a new string value assigned back to `string_var`, where the original leading character has been successfully excised.

This powerful yet simple method is crucial for data preparation tasks, especially when dealing with data imported from external systems that might include unnecessary prefixes, leading identifiers, or formatting characters that hinder proper analysis. Let us now examine a comprehensive, practical example demonstrating the application of this syntax using a sample dataset involving basketball team names.

## Practical Example: Cleaning Team Identifiers

To fully grasp the utility of the **SUBSTR** function, we will work through a practical scenario. Imagine a situation where we have imported data containing information about various professional basketball teams. However, during the import process, a system prefix, represented here by the letter 'x', was erroneously prepended to every team name. This leading character must be systematically removed to ensure the team names are accurate for reporting and joining purposes.

We begin by defining and populating our sample dataset in SAS. We use the **DATALINES** statement within a DATA step to quickly input the data points, specifying that the `team` variable is a character string (indicated by the $ sign) and `points` is numeric.

```
/*create dataset*/
data my_data;
input team $ points;
datalines;
xMavs 113
xPacers 95
xCavs 120
xLakers 114
xHeat 123
xKings 119
xRaptors 105
```

```
xHawks 95
xMagic 103
xSpurs 119
;
run;

/*view dataset*/
proc print data=my_data;
```

| Obs | team | points |
|---|---|---|
| 1 | xMavs | 113 |
| 2 | xPacers | 95 |
| 3 | xCavs | 120 |
| 4 | xLakers | 114 |
| 5 | xHeat | 123 |
| 6 | xKings | 119 |
| 7 | xRaptors | 105 |
| 8 | xHawks | 95 |
| 9 | xMagic | 103 |
| 10 | xSpurs | 119 |



Upon viewing the initial dataset, `my_data`, it is immediately apparent that the **team** column contains flawed data. Every single string value begins with the extraneous character 'x'. This character must be systematically pruned from all records to ensure the integrity of the team names. If this initial 'x' were not uniform (i.e., sometimes it existed, sometimes it didn't), we would need additional conditional logic, but for this specific example, the consistent placement allows for a direct application of the **SUBSTR** function.

## Applying the SUBSTR Transformation

The crucial step involves creating a new dataset, `new_data`, based on `my_data`, where the transformation takes place. We utilize another DATA step and the **SET** statement to iterate through the observations of the existing data. The core of the operation lies in reassigning the value of the `team` variable using the **SUBSTR** function.

Specifically, we instruct SUBSTR to analyze the `team` variable and start reading characters from position 2. This is achieved by the concise expression: `team = substr(team, 2);`. By assigning

the result back to the original `team` variable name, we overwrite the flawed string with the clean, truncated string within the new dataset. It is important to remember that SAS executes these statements row by row, ensuring every single observation is processed correctly.

```
/*create new dataset where first character in each string of team column is removed*/
data new_data;
set my_data;
team = substr(team, 2);
run;

/*view new dataset*/
proc print data=new_data;
```

| Obs | team | points |
|---|---|---|
| 1 | Mavs | 113 |
| 2 | Pacers | 95 |
| 3 | Cavs | 120 |
| 4 | Lakers | 114 |
| 5 | Heat | 123 |
| 6 | Kings | 119 |
| 7 | Raptors | 105 |
| 8 | Hawks | 95 |
| 9 | Magic | 103 |
| 10 | Spurs | 119 |



After executing the above DATA step and running the **PROC PRINT** procedure, we can clearly observe the successful transformation of the data. The output confirms that the leading character 'x' has been systematically removed from every entry in the **team** column. The original values like "xMavs" are now correctly rendered as "Mavs", and "xPacers" is now simply "Pacers". This demonstrates the fundamental power and reliability of the **SUBSTR** function for positional string manipulation.

## Deep Dive into the SUBSTR Function Arguments

Although the example above used a simplified version of the **SUBSTR** function, it is essential to understand its full parameter structure to utilize its complete capabilities for more complex string extraction tasks. The function's design is based on the necessity to specify exactly which part of a

source string should be returned.

The **SUBSTR** function adheres to the following standard syntax pattern, which dictates how the extraction process is executed:

**SUBSTR(Source, Position, N)**

Where each component plays a specific, defined role in the operation:

**Source**: This is the mandatory first argument. It represents the original character variable or literal string that the function will analyze and attempt to extract a substring from. This is the variable containing the data we wish to modify.
**Position**: This mandatory second argument specifies the exact starting point for the extraction. In SAS, character positions are 1-indexed, meaning the first character is at position 1. By setting this value to 2, as we did in our example, we effectively skip the leading character.
**N**: This is the optional third argument. It specifies the number of characters that should be extracted starting from the defined **Position**. If **N** is provided, the resulting substring will have a length equal to **N** (or less, if the string ends before **N** characters are reached).

In our application, `substr(team, 2)`, we intentionally omitted the third argument, **N**. This omission is key to simplifying the task of removing the first character. When **N** is not specified, SAS defaults to extracting the entire remainder of the string, starting from the specified **Position** (which was 2). This ensures that no matter how long the team name is, the entire name, minus the first character, is captured correctly without the need to calculate the exact length using the **LENGTH** function.

## Considerations for Character Length and Truncation

When performing string manipulations in the DATA step, it is critical to consider the defined length of the character variable. SAS variables, by default, often retain the length they inherited when they were first read or calculated. If we are replacing a variable with a shorter string, SAS handles this gracefully. However, if the resulting string generated by **SUBSTR** were longer than the existing variable's defined length, truncation would occur, potentially leading to data loss.

In the case of removing the first character, the resulting string is always shorter by one character, or remains the same length if the original string was null or only one character long. Therefore, **SUBSTR(variable, 2)** poses no risk of truncation when overwriting the original variable. This operation is considered safe and reliable for data cleansing.

For situations where string length manipulation is necessary, programmers often use the **LENGTH** statement within the DATA step before the SET statement to explicitly define the maximum storage size of the character variable, ensuring sufficient capacity for longer results or standardizing character variable sizes across multiple datasets.

## Error Handling and Edge Cases

When utilizing the **SUBSTR** function, especially for trimming the first character, it is important to consider certain edge cases related to the input data. The most common edge case involves variables that contain missing values or are extremely short, such as a string with only one character or an empty string.

If the input variable, `team`, is an empty string (length 0), the expression `substr(team, 2)` will logically fail to find a second position and will return a missing value (or an empty string, depending on SAS settings and version handling of **SUBSTR**). If the variable contains only one character, say "x", the attempt to extract from position 2 onward will also result in an empty string. The key takeaway is that **SUBSTR** is inherently robust; it will not throw an error if the starting position exceeds the string length, but rather returns a null result, which is often desirable in cleaning operations.

For production environments where data quality is paramount, programmers might wrap the **SUBSTR** function within conditional logic to prevent operations on null or very short strings, although for simple trimming, direct application is usually sufficient. An example conditional check might involve using the **LENGTH** function: `IF LENGTH(team) > 1 THEN team = SUBSTR(team, 2);`. This ensures that the operation only proceeds if there is more than one character available to truncate, leaving single characters or empty fields untouched if that is the desired business logic.

## Summary of String Manipulation in SAS

The **SUBSTR** function is an indispensable tool in the SAS programmer's arsenal for handling character data. By simply utilizing the syntax `SUBSTR(variable, 2)` within a DATA step, users can efficiently and reliably remove the first character from any string variable, preparing their data for rigorous analysis. Mastery of this function is fundamental to effective data preparation and cleaning within the SAS environment.

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