SAS: Remove First Character from String

Authored by

Mohammed looti

November 16, 2025

Introduction: Mastering String Manipulation in SAS for Data Cleaning

Working extensively with textual or categorical data is an inevitable part of modern data analysis. The SAS system provides an exceptionally robust suite of functions designed specifically to handle and modify character strings efficiently. A frequently encountered requirement during data preparation involves standardizing these strings by removing extraneous or identifying characters, particularly those prefixed to the value. This detailed guide focuses on the most direct and efficient technique for discarding the very first character of a string, utilizing the highly flexible SUBSTR function.

The ability to precisely control and modify string values is foundational for crucial data tasks such as cleaning, preparation, and standardization across various analytical projects. Whether a programmer is tasked with correcting erroneously included prefixes, stripping identifying markers, or simply reformatting data structures to meet specific output requirements, gaining proficiency in manipulating individual characters within a string is an indispensable skill for any proficient SAS developer. Effective string management ensures data integrity and simplifies subsequent analyses.

This tutorial is meticulously structured to guide the user through the entire process, starting with a fundamental understanding of the core function and progressing to practical, implementable examples. Our primary goal is to empower readers to confidently use the SUBSTR function to achieve this specific, targeted string transformation with maximum efficiency and clarity, thereby streamlining data workflows considerably.

The Power of the SUBSTR Function in SAS Programming

The SUBSTR function is a cornerstone utility in the SAS system, primarily designed for extracting a specific substring from a larger character value. Its design versatility allows the user to define both the exact starting position and the desired length of the segment being extracted. While its nomenclature suggests only extraction, the clever manipulation of its arguments enables it to perform a variety of text manipulation tasks, including the effective removal of unwanted leading or trailing components.

Instead of relying on complex conditional logic or specialized removal functions, the most elegant and efficient solution for character omission often involves exploiting how `SUBSTR` handles starting positions. By simply instructing the function to begin extraction immediately after the character(s) intended for discard, we effectively isolate and create a new string that inherently omits those initial characters. This transformation is **clean**, highly readable, and generally performs very quickly, making it the preferred method for high-volume data cleaning tasks.

The fundamental structure of this function requires the original source string and a starting position. Crucially, when the optional third argument--specifying the length of the extracted segment--is

omitted, SUBSTR intelligently defaults to extracting all remaining characters from the specified starting point right up to the end of the original string. This specific behavior is the critical mechanism we leverage to achieve our goal of discarding only the first character without needing to calculate the string's total length manually.

Implementing the Basic Syntax in a SAS DATA Step

The most concise and widely adopted technique for removing the first character from a target string within the SAS environment utilizes the `SUBSTR` function directly within a DATA step. This approach is highly favored for its clarity, efficiency, and seamless integration into standard data processing workflows, making it a **best practice** for this transformation.

The standard programming syntax required to execute this common transformation is remarkably simple and involves a single assignment statement, as shown below:

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

In this fundamental syntax structure, the object labeled `new_data` refers to the resulting dataset, which will contain the successfully modified strings, while `original_data` represents the source dataset feeding the process. The operational core of the entire procedure is contained within the assignment statement: `string_var = substr(string_var, 2);`. Here, `string_var` is the name of the character variable whose values are intended for modification.

By inputting the numeric value `2` as the starting position argument, we explicitly instruct the `SUBSTR` function to initiate the extraction process at the **second position** of the source string. Because the length parameter is deliberately omitted, the function executes its default behavior: it automatically captures all subsequent characters from that position until it reaches the string's terminus. This mechanism efficiently generates a new substring that is functionally identical to the original input, yet conspicuously missing the first character, thereby achieving the desired data cleansing outcome in a single line of code.

Practical Demonstration: Applying SUBSTR to Real-World Data

To fully grasp the practical utility of the `SUBSTR` function in a controlled analytical environment, let us proceed with a concrete, real-world example. We will simulate a scenario commonly encountered during data import where a dataset containing identification or descriptive fields inadvertently includes a uniform, leading prefix that must be systematically removed. For this

illustration, we will use a dataset detailing basketball teams where every team name begins with an unnecessary 'x'.

Our first step involves the creation of a sample dataset named `my_data`. This dataset will feature two key variables: `team`, which holds the prefixed character strings, and `points`, a numeric variable representing scores. We use an inline DATA step with `DATALINES` to populate this initial data structure, demonstrating the source data preparation:

```
/* Create initial dataset with unwanted prefix */
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;

/* Display the contents of the original dataset */
proc print data=my_data;
```

Upon the successful execution of this SAS code block, the initial dataset can be inspected. A visual check using the PROC PRINT output confirms that every entry within the `team` column indeed begins with the unnecessary 'x' prefix. This confirmation is vital, as it verifies the starting condition and the necessity of the subsequent manipulation step.

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

To systematically eradicate this leading 'x' from all team names, we now introduce the core transformation logic using `SUBSTR(team, 2)`. We will create a distinct output dataset, **`new_data`**, ensuring that the integrity of our original **`my_data`** remains preserved, adhering to **best practices** in data manipulation:

```
/* Apply SUBSTR function to remove the first character */
data new_data;
set my_data;
team = substr(team, 2);
run;

/* View the transformed dataset */
proc print data=new_data;
```

The execution of this code generates **`new_data`**. The subsequent PROC PRINT statement allows for immediate verification of the results. As the output demonstrates, the first character has been successfully and universally removed, resulting in clean team names such as "Mavs", "Pacers", and "Lakers". This visible transformation underscores the simplicity and effectiveness of the `SUBSTR` approach for **targeted character removal**.

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

As clearly established by the output review, the leading prefix 'x' has been entirely stripped from every string contained within the `team` column. This demonstration serves as compelling evidence of the power and inherent simplicity of utilizing the `SUBSTR` function for managing and cleaning character data in a standardized manner.

Deconstructing the SUBSTR Function's Arguments for Precision

To truly appreciate why the operation `substr(team, 2)` works so precisely and efficiently for our specific requirement, it is essential to internalize the complete syntax and the functional role of each argument within the SUBSTR function. This function generally accepts up to three parameters, following the structure:

```
SUBSTR(Source, Position, N)
```

Let us analyze the role of each component in detail:

`Source`: This is the mandatory input argument, representing the original character variable or literal string from which the subsequent substring extraction is intended. In our previous practical example, this parameter was assigned the value of the `team` variable.
`Position`: This is a required numeric value that dictates the exact starting index within the `Source` string where the function should commence its extraction. It is critically important to recall that SAS string indexing operates on a **1-based** system, meaning the very first character occupies position 1, the second occupies position 2, and so forth. Consequently, using the value `2` instructs SAS to ignore the initial character and begin reading from the second position, achieving the removal objective.

`N`: This final parameter is an optional numeric argument that specifies the precise number of characters to be extracted starting from the defined `Position`. When this argument is intentionally omitted, as demonstrated in our primary example, the `SUBSTR` function automatically extracts all remaining characters from the designated starting point to the absolute end of the `Source` string, ensuring that the resulting substring is complete, minus the discarded prefix.

The combination of these parameters--specifically, supplying the variable name and the starting position 2, while omitting the length N--provides a highly effective and concise solution. This single function call, `substr(team, 2)`, instructs SAS to extract the remainder of the string beginning at the second character, producing a **flawless output string** without the unwanted leading element.

Handling Edge Cases and Considering Alternatives

While the `SUBSTR` function is undoubtedly the most robust and preferred method for removing the first character when dealing with structured data, a seasoned programmer must account for potential edge cases to ensure the code's resilience and stability across diverse data inputs. A crucial consideration involves how the function processes extremely short strings or values that are entirely missing (null or blank).

If the source string is either entirely empty (a null string) or contains only a single character, the expression `substr(string_var, 2)` will gracefully return an empty string. This behavior occurs because the function cannot locate a second character from which to begin extraction, and it generally aligns with the expected outcome: if there is no character to preserve after the first one is removed, the result should be nothing. However, if business logic dictates that single-character strings should be treated differently (perhaps preserved or assigned a specific default), you must implement **defensive conditional logic**.

For example, you might introduce an `IF` statement utilizing the `LENGTH` function to check the string size before applying the transformation. A typical conditional structure might look like this: `IF length(string_var) > 1 THEN string_var = substr(string_var, 2); ELSE string_var = 'Default';`. It is considered a best practice to rigorously test your `SUBSTR` implementation against various boundary conditions, including zero-length strings, single-character strings, strings with special characters, or entirely missing values, guaranteeing that the result is consistently predictable.

While `SUBSTR` offers the most direct solution for this specific task, it is worth noting that the SAS system provides alternative functions that can achieve character removal in different contexts. Functions such as `SCAN` (useful when characters are separated by delimiters), `COMPRESS` (for removing specific lists of characters anywhere in the string), or `TRANWRD` (for replacing specific substrings) exist. However, for the straightforward requirement of unconditionally stripping the leading character, `SUBSTR` remains the most **idiomatic**, efficient, and computationally

lightweight choice available to the DATA step programmer.

Conclusion: An Essential Tool for SAS Data Preparation

Successfully removing the first character from a string in SAS is a foundational string manipulation task, and the solution is elegantly encapsulated within the `SUBSTR` function. By expertly utilizing its capability to extract a substring starting precisely at the second position and continuing to the end, data professionals can achieve this necessary transformation with exceptional efficiency and minimal required code.

The concise syntax, `string_var = substr(string_var, 2);`, should be recognized as a powerful and indispensable statement within every SAS programmer's toolkit for data cleansing. It is imperative, however, to consistently review and account for potential edge cases--specifically short or empty strings--to guarantee that the code remains **robust** and yields predictable outcomes across all possible data conditions.

Mastery of these fundamental string functions is absolutely critical for performing effective and reliable data preparation and subsequent analysis in the SAS environment. As you further your programming expertise, continuous exploration of the full range of string functions and their potential combinations will unlock increasingly sophisticated text processing capabilities. Diligent practice in handling these common manipulation tasks will quickly lead to proficiency in managing complex character data structures.

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