Learning the INDEX Function in SAS: A Step-by-Step Guide

Authored by

Mohammed looti

October 27, 2025

In the realm of data processing and analysis, particularly within the powerful **SAS** environment, the ability to efficiently search and analyze text-based information is paramount. **String manipulation** functions are essential tools for cleaning, standardizing, and extracting data from character variables. Among the most fundamental of these tools is the **INDEX function**, which serves a critical purpose: identifying the location of a specific substring within a larger character sequence.

This article provides a comprehensive guide to utilizing the **INDEX function** in **SAS**. We will delve into its precise syntax, examine how it handles various search scenarios, and demonstrate practical applications using executable code examples. Mastering this function is key to performing advanced conditional logic and data extraction tasks based on textual content in your datasets.

Understanding the SAS INDEX Function: Syntax and Parameters

The primary purpose of the **INDEX function** is to return the starting position of the first occurrence of a search string (the excerpt) within a source **character string**. If the substring is successfully located, the function returns a positive integer corresponding to the position where the match begins. Crucially, **SAS** counts character positions starting at one (1), not zero.

If the specified excerpt cannot be found anywhere within the source string, the function returns a value of zero (0). This zero return value is highly significant, as it often acts as a flag in subsequent conditional processing--such as an `IF-THEN/ELSE` block--indicating that a specified pattern or keyword is absent from the observed data record.

The **INDEX function** employs a straightforward and intuitive syntax structure, requiring only two essential arguments to operate effectively:

INDEX(source, excerpt)

The components of this syntax are defined as follows:

source: This argument represents the character variable or literal string that the function is instructed to analyze. This is the larger string being searched.
excerpt: This argument represents the specific substring of characters that the function is attempting to locate within the *source* string.

It is vital to remember that both the source and the excerpt must be character values. Attempting to pass numeric variables to the **INDEX function** without explicit conversion will result in a syntax error or unexpected behavior, necessitating the use of functions like `PUT` if numeric values must be searched as text.

Practical Demonstration: Setting Up the Sample Dataset

To fully illustrate the utility of the **INDEX function**, we will first create a small, representative dataset. This dataset, named `original_data`, contains a single column of full names, which will serve as our source variables for string analysis. This creation process typically occurs within a **DATA step** in **SAS**, utilizing `DATALINES` to quickly input the sample data records.

The following code block executes the creation of our initial dataset, followed by a simple `PROC PRINT` to verify that the data has been correctly loaded into the **SAS** environment:

```
/*create dataset*/
data original_data;
input name $25.;
datalines;
Andy Lincoln Bernard
Barren Michael Smith
Chad Simpson Arnolds
Derrick Smith Henrys
Eric Millerton Smith
Frank Giovanni Goode
;
run;

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

Once this **DATA step** is executed, the resulting output table confirms the structure of our sample data, showing six records, each containing a name that is up to 25 characters long:

Obs	name
1	Andy Lincoln Bernard
2	Barren Michael Smith
3	Chad Simpson Arnolds
4	Derrick Smith Henrys
5	Eric Millerton Smith
6	Frank Giovanni Goode

This dataset provides the foundation for our analysis. We are now prepared to apply the **INDEX**

function to the `name` variable to locate specific substrings, demonstrating how this function returns positional data that can be used for further data manipulation or quality checks.

Executing a Case-Sensitive Search using INDEX

Our initial objective is to locate the position of the surname "Smith" within each record of the `name` column. We will use the **INDEX function** within a new **DATA step** to create a fresh variable, `first_smith`, which will store the starting position of the string 'Smith' for every observation. If 'Smith' is not present in a given record, the new variable will correctly hold a value of 0.

The code below demonstrates this operation. Notice that we are searching specifically for 'Smith' with an initial capital letter, which is crucial because, by default, the **INDEX function** operates in a **case-sensitive** manner. If the case does not match exactly, the function will not find the match.

```
/*find position of first occurrence of 'Smith' in name*/
data new_data;
set original_data;
first_smith = index(name, 'Smith');
run;

/*view results*/
proc print data=new_data;
```

Upon reviewing the output generated by the `PROC PRINT` statement, we can clearly observe the positional results. For rows where 'Smith' is present (records 2, 4, and 5), the `first_smith` variable returns a positive integer indicating the starting position. For instance, in the record "Barren Michael Smith," the word 'Smith' begins at the 17th character position (counting spaces as characters), so the function returns 17. Conversely, for all other records where the substring 'Smith' is absent, the function correctly returns the value 0.

Obs	name	first_smith
1	Andy Lincoln Bernard	0
2	Barren Michael Smith	16
3	Chad Simpson Arnolds	0
4	Derrick Smith Henrys	9
5	Eric Millerton Smith	16
6	Frank Giovanni Goode	0

The resulting new column, `first_smith`, provides a powerful metric. A non-zero value confirms the presence of the desired string and its location, which can then be used with other functions, such as `SUBSTR`, to extract the string, or used in filtering operations via a `WHERE` clause in a subsequent procedure. This ability to quickly identify presence or absence based on a zero or non-zero return is central to many text processing routines.

The Importance of Case Sensitivity in String Searches

One of the most frequent pitfalls encountered when using the **INDEX function** is forgetting its default behavior: strict case sensitivity. If the case of the characters in the search excerpt does not perfectly match the case of the characters in the source string, the function will fail to register a match, regardless of the characters themselves being identical.

To demonstrate this concept, consider if we were to mistakenly search for the lowercase string 'smith' instead of the correctly capitalized 'Smith'. Even though the records contain the word, the case mismatch causes the **INDEX function** to behave as if the word were not there at all. This strict matching rule is a design choice in **SAS** to ensure precision, but it requires the user to proactively manage case variations if a broader search is intended.

Executing the same **DATA step** as before, but changing the excerpt to lowercase 'smith', yields significantly different results:

```
/*find position of first occurrence of 'smith' in name*/
data new_data;
set original_data;
first_smith = index(name, 'smith');
run;

/*view results*/
proc print data=new_data;
```

As anticipated, because the search is **case-sensitive**, every record in the dataset returns a value of 0 in the `first_smith` column. This outcome confirms that no instance of the exact string 'smith' was found. This highlights a critical need in real-world data cleaning: before performing string comparison, data often must be standardized to a single case (either all uppercase or all lowercase) to ensure all intended matches are captured.

Obs	name	first_smith
1	Andy Lincoln Bernard	0
2	Barren Michael Smith	0
3	Chad Simpson Arnolds	0
4	Derrick Smith Henrys	0
5	Eric Millerton Smith	0
6	Frank Giovanni Goode	0

To successfully search data that may have inconsistent capitalization, we must employ an additional function to normalize the case of the source string before passing it to the **INDEX function**. This preparatory step is vital for robust **string manipulation** in **SAS**.

Achieving Case-Insensitive Searching with LOWCASE

To overcome the limitations imposed by the default case-sensitive nature of the **INDEX function**, **SAS** programmers typically combine **INDEX** with one of the case conversion functions, such as the **LOWCASE function** (or `UPCASE`). The **LOWCASE function** takes a character string as input and returns the exact same string with all characters converted to lowercase.

The strategy for a case-insensitive search is straightforward: first, convert the `source` variable (in our case, `name`) to lowercase using **LOWCASE**. Second, ensure the `excerpt` variable (the string we are searching for) is also specified in lowercase. By feeding the **INDEX function** two inputs that are guaranteed to be in the same case, we eliminate the sensitivity issue and ensure all potential matches are registered.

The following code implements this combined approach, effectively replicating the success of our initial search for 'Smith', but doing so robustly, regardless of the original capitalization in the `original_data`:

```
/*find position of first occurrence of 'smith' in name*/
data new_data;
set original_data;
first_smith = index(lowcase(name), 'smith');
run;

/*view results*/
proc print data=new_data;
```

When this code is executed, the results perfectly match the first successful search. The **INDEX function** now returns a non-zero position for records 2, 4, and 5. It is important to note a subtle detail here: the position returned (e.g., 17) corresponds to the position in the *temporarily created* lowercase string, not the original string. However, since case conversion does not change the number of characters or the location of words, the position remains accurate for the purposes of identifying where the substring begins.

Obs	name	first_smith
1	Andy Lincoln Bernard	0
2	Barren Michael Smith	16
3	Chad Simpson Arnolds	0
4	Derrick Smith Henrys	9
5	Eric Millerton Smith	16
6	Frank Giovanni Goode	0

By skillfully pairing the **INDEX function** with the **LOWCASE function**, analysts can perform comprehensive and reliable searches across datasets where data entry or source formats may vary in capitalization. This technique is indispensable for robust data quality checks and text parsing operations.

Additional Resources

While the **INDEX function** is highly effective for finding the starting position of a substring, **SAS** offers a rich library of related functions that provide alternatives or slightly different capabilities for **string manipulation**. Functions such as `FIND` (which offers modifiers for case and trimming), `INDEXC` (which searches for any character in a list), and `INDEXW` (which searches for whole words) are often used in conjunction with or as alternatives to **INDEX** depending on the complexity of the pattern matching required.

For those interested in expanding their proficiency in **SAS** text processing, we recommend exploring the documentation for these related functions:

Tutorials explaining how to use other common functions in **SAS**.
Detailed guides on advanced techniques for searching based on regular expressions.

Mastery of the **INDEX function** is a fundamental step toward becoming a proficient **SAS** user, enabling powerful and efficient character data analysis.