# How do I use the INDEXC Function in SAS?

Authored by

**stats writer**

November 19, 2025

The INDEXC function in SAS is one of the most useful tools for character manipulation and text analysis within the programming environment. Unlike other search functions that look for an entire phrase, **INDEXC** specializes in searching a primary character string for the presence of any single character contained within a specified set of search characters. This capability is essential when you need to quickly locate the position of any special character, punctuation mark, or a specific group of letters within a variable.

Fundamentally, the function operates by returning a numeric value indicating the starting position of the very first match found. If the specified search characters are not present anywhere within the source string, the function predictably returns a value of zero (0). This makes it highly effective for data validation, cleaning processes, and conditional logic creation in your **SAS** programs. Understanding the exact syntax and differentiating it from similar functions, such as the standard INDEX function, is crucial for leveraging its full power in complex data tasks.

## The Syntax and Arguments of INDEXC

The primary purpose of the **INDEXC function** is to efficiently identify the location of any individual character defined in a search set within a larger source string. This allows for flexible pattern matching, focusing on character inclusion rather than exact substring matching. When utilized correctly, the function provides a precise numeric index indicating the byte position of the first matching character encountered.

This powerful **SAS** function utilizes a straightforward syntax that requires two mandatory arguments: the source string to be analyzed and the collection of characters to search for. The function's structure is defined clearly, ensuring ease of use across various programming contexts within the **SAS Data Step**.

The basic syntax structure for implementing **INDEXC** is as follows:

**INDEXC(source, excerpt)**

Where these arguments fulfill specific roles:

**source**: This is the mandatory **character string** or variable that you intend to analyze. This variable holds the text in which the search operation will take place.
**excerpt**: This is the mandatory string containing the specific set of characters the function will search for within the *source* string. Crucially, **INDEXC** looks for any single character from this set, not the entire string as a contiguous block.

It is important to remember that **SAS** character functions are case-sensitive by default, meaning 'A' and 'a' are treated as distinct characters unless case-insensitive options are employed using

related functions like `UPCASE` or `LOWCASE` on the source string prior to execution. For the purpose of the primary **INDEXC** function, the search is executed based on the exact characters provided in the *excerpt* argument.

## Practical Example: Using the INDEXC Function in SAS

To demonstrate the practical utility of the **INDEXC function**, consider a scenario where we have a dataset containing customer names, and we need to identify which names contain potentially unusual or specific characters, such as 'x', 'y', or 'z'. This type of analysis is common when performing quality checks or preliminary data exploration.

We begin by defining our sample **dataset** named `original_data`. This set includes a single character variable, `name`, which holds various employee names. The code block below illustrates the creation and initial display of this source data:

```
/*create dataset containing names for analysis*/
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 contents*/
proc print data=original_data;
```

| 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 |

Once the source data is confirmed, we can apply the **INDEXC** function within a new **Data Step**. Our goal is to search the `name` variable for any occurrence of the characters 'x', 'y', or 'z'. We define the search characters as the string 'xyz' in the second argument of the function. The resulting position will be stored in a newly created variable, which we name `first_xyz`.

The following **SAS** code executes this search operation and then prints the resulting **dataset**, allowing us to inspect the generated index values:

```
/*find position of first occurrence of either x, y or z in name*/
data new_data;
set original_data;
first_xyz = indexc(name, 'xyz');
run;

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

| Obs | name | first_xyz |
|---|---|---|
| 1 | Andy Lincoln Bernard | 4 |
| 2 | Barren Michael Smith | 0 |
| 3 | Chad Simpson Arnolds | 0 |
| 4 | Derrick Smith Henrys | 19 |
| 5 | Eric Millerton Smith | 0 |
| 6 | Frank Giovanni Goode | 0 |

## Interpreting the Output of the INDEXC Function

The resulting column, `first_xyz`, provides a precise index (position count) of the first instance where any of the searched characters ('x', 'y', or 'z') appeared in the corresponding `name` field. Analyzing this output is key to understanding how **INDEXC** works on a character-by-character basis, contrasting sharply with substring search functions.

A fundamental aspect of interpreting the output is recognizing the significance of the return value zero (**0**). If **INDEXC** scans the entire source character string and finds none of the characters specified in the search set, it returns **0**. This result is particularly useful for filtering or subsetting data records that do not contain specific invalid or desired characters. Conversely, any positive integer returned signifies the byte position where the match was first identified.

Examining the output table above confirms this behavior. Consider the first observation: "Andy Lincoln Bernard." The search characters were 'x', 'y', and 'z'. The first occurrence of any of these is 'y' in "Andy," which is the fourth character in the string, thus `first_xyz` correctly returns **4**. For the second observation, "Barren Michael Smith," none of the characters 'x', 'y', or 'z' are present, resulting in a return value of **0**. This simple mechanism allows for powerful logical checks against text variables.

## The Critical Difference Between INDEX and INDEXC Functions

When working with character manipulation in **SAS**, developers frequently encounter two similar-sounding functions: **INDEXC** and INDEX function. While both locate characters within a larger string, their mechanism of operation is fundamentally different. Understanding this distinction is vital to ensure you use the correct tool for your analytical goal.

The standard **INDEX function** is designed to search for the first occurrence of an entire substring--a contiguous sequence of characters--within the source string. For example, if you search for 'Smith', the **INDEX** function will only return a positive value if the characters S-m-i-t-h appear consecutively in that exact order. It treats the search term as a single, inseparable unit.

In stark contrast, the **INDEXC function** (where 'C' typically stands for Characters or Contains) searches for the first occurrence of **any individual character** found within the search string argument. If the search string is 'Smith', **INDEXC** scans the source string and stops at the very first occurrence of 'S', 'm', 'i', 't', or 'h', regardless of their order or whether they appear together. This distinction dictates which function is appropriate for pattern matching versus exact phrase matching.

## Comparative Example: INDEX vs. INDEXC in Practice

To clearly illustrate this functional difference, we will apply both the **INDEX** and **INDEXC** functions to our original **dataset** using the same search term, 'Smith'. The resulting output variables will highlight how each function interprets the search argument.

We will create two new variables: `index_smith` using the **INDEX function**, and `indexc_smith` using the **INDEXC function**. Both functions will search the `name` variable for the string 'Smith'.

```
/*create new dataset comparing both functions*/
data new_data;
set original_data;
index_smith = index(name, 'Smith');
indexc_smith = indexc(name, 'Smith');
run;

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

| Obs | name | index_smith | indexc_smith |
|---|---|---|---|
| 1 | Andy Lincoln Bernard | 0 | 7 |
| 2 | Barren Michael Smith | 16 | 9 |
| 3 | Chad Simpson Arnolds | 0 | 2 |
| 4 | Derrick Smith Henrys | 9 | 5 |
| 5 | Eric Millerton Smith | 16 | 3 |
| 6 | Frank Giovanni Goode | 0 | 8 |



Observing the results in the table provides definitive clarity. The column `index_smith` utilized the **INDEX function** and therefore displays the position of the first occurrence of the complete, contiguous **substring** 'Smith' in the `name` column. Notice that for the first row, "Andy Lincoln Bernard," the value is **0** because the full word 'Smith' is not present.

In contrast, the `indexc_smith` column, powered by the **INDEXC function**, displays the position of the first occurrence of any of the individual letters: 's', 'm', 'i', 't', or 'h'. In the first row, "Andy Lincoln Bernard," the letter 'i' is the seventh character encountered, causing `indexc_smith` to return **7**. This row perfectly encapsulates the functional difference: one looks for the whole word, the other looks for any component letter.

## Advanced Considerations and Related Character Functions

While **INDEXC** is highly effective for simple character searches, its utility extends into more complex data manipulation tasks. For instance, it is often paired with other **SAS** functions like `SUBSTR` to extract specific portions of a character string based on the index position returned, or used within conditional statements (`IF-THEN`) to categorize records based on the presence of certain characters.

It is also worthwhile to briefly mention related functions that offer variations on this search logic. The `INDEXW` function, for example, is designed to search for a word rather than a **substring** or individual characters, using delimiters to define word boundaries. Furthermore, `INDEXANY` and `INDEXALPHA` are specialized functions that leverage similar character set logic to locate specific character classes (like alphabetic characters or numeric digits) without explicitly listing every character in the search argument. Choosing between **INDEXC**, **INDEX**, and these specialized variations depends entirely on whether your goal is to find a contiguous block, any character from a set, or a predefined character class.

## Summary and Next Steps in SAS Programming

The **INDEXC function** is an indispensable tool in the **SAS** programmer's toolkit, particularly when performing data cleaning, validation, or pattern identification where the presence of individual characters is more important than the location of a complete phrase. By returning the numeric position of the first matched character, or zero if no match is found, it provides a simple yet powerful mechanism for conditional processing and text parsing.

Mastering character functions is a cornerstone of effective **SAS** programming. The following tutorials explain how to use other common functions in SAS: