Learning SAS: A Comprehensive Tutorial on the INDEXC Function for String Manipulation

Authored by

Mohammed looti

November 14, 2025

Mastering Character-Set Searching with SAS String Functions

Effective manipulation of textual information, commonly referred to as a **string**, stands as a fundamental pillar of modern data analysis and programming. The **SAS** System, a leading platform for advanced analytics and data management, furnishes programmers with an extensive library of specialized functions tailored precisely for sophisticated string processing. These tools are indispensable for critical tasks such as rigorous data cleaning, engineering analytical features, and enforcing data validation rules, enabling users to meticulously extract, modify, and analyze text data with unparalleled precision.

Within this powerful suite of string utilities, the **INDEXC function** offers a unique and highly specific capability: identifying the starting position of the first individual **character** found within a designated collection of characters inside a larger target string. This mechanism fundamentally differs from standard search functions, which are designed to locate entire sequences or fixed substrings. **INDEXC** is engineered to swiftly confirm the presence of *any* character from a specified list, making it exceptionally valuable when the objective is to pinpoint the existence of specific character types--such as punctuation marks, numeric digits, or a custom set of disallowed symbols--rather than searching for exact textual patterns.

This comprehensive guide serves as a deep dive into the practical mechanics of the **INDEXC function**. We will meticulously detail its syntax structure, demonstrate its real-world utility through clear, executable code examples, and--most importantly--establish a precise operational distinction between **INDEXC** and similar **SAS** string tools, particularly the **INDEX function**. Upon completing this article, readers will be equipped with the necessary expertise to confidently integrate **INDEXC** into their data preparation and analytical workflows, significantly enhancing their capability to manage unstructured text data.

Defining the `INDEXC` Function and Its Required Syntax

The core purpose of the **INDEXC function** is to sequentially scan an input **string** and, upon finding a match, return the position of the first **character** that belongs to a secondary, user-defined set of characters. This collective search functionality is tremendously powerful for data validation applications, such as verifying the existence of special symbols, controlling for numeric characters, or checking for any customized list of elements within a large text field. The key efficiency of **INDEXC** lies in its capacity to execute a rapid, unified search across multiple possibilities simultaneously, freeing the user from having to locate a single, fixed sequence.

The syntax required to properly execute the **INDEXC function** is concise and follows a highly accessible structure, making it straightforward for all experience levels of **SAS** users. The function relies on two essential arguments for successful operation:

INDEXC(source, excerpt)

A detailed breakdown of these two components clarifies their respective roles in the character search process:

source: This argument defines the main target **string** that the function will scan. It represents the actual text within which you are attempting to locate the specified characters. This argument can refer to a variable existing within your **dataset** or be provided as a literal string value enclosed in quotes.
excerpt: This critical argument must be a **string** containing all the individual characters the function should search for within the *source* string. Crucially, **INDEXC** seeks the first appearance of *any single character* contained in this *excerpt* string. For example, if *excerpt* is defined as 'AEIOU', the function will return the position of the first 'A', 'E', 'I', 'O', or 'U' it encounters.

Upon successfully finding a match, the **INDEXC function** returns the numerical position of that first matching **character** within the *source* string, with counting conventionally starting at position 1. Conversely, if none of the characters specified in the *excerpt* string are present within the *source* string, the function yields a standardized return value of **0**. This clear, binary return behavior (position greater than 0, or 0) makes **INDEXC** an exceptionally reliable tool for building conditional data validation and complex processing logic.

Practical Implementation: Searching for Custom Character Sets

To vividly illustrate the practical efficacy of the **INDEXC function**, consider a typical scenario encountered in a **SAS** environment. Suppose we are working with a list of names and our primary objective is to quickly ascertain if any name contains specific, potentially non-standard or rare characters, such as 'x', 'y', or 'z'. Flagging such occurrences is vital for ensuring data quality, categorizing linguistic elements, or preparing data for downstream systems that impose strict character limitations.

Our process begins by establishing a foundational **SAS dataset**, named `original_data`, which will hold our list of names. This **dataset** serves as the initial source material for our character manipulation task. The following **SAS** code snippet details the creation of this initial **dataset** using a standard **DATA step**, followed by the necessary **PROC PRINT** procedure to display its contents and confirm our starting point.

```
/*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;
```

The execution of the code above successfully generates the `original_data` table, defining a character variable named `name`. The contents of this table, displayed visually in the image below, confirm the dataset used as the foundation for our forthcoming analysis.

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

Now, we apply the **INDEXC function**. Our goal is to create a new variable that precisely records the position of the first occurrence of 'x', 'y', or 'z' within each `name` field. This calculation is performed within a subsequent **DATA step**, which reads from `original_data` and writes the enhanced results to a new **dataset** named `new_data`.

```
/*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;
```

The core logic resides in the line `first_xyz = indexc(name, 'xyz');`. For every observation, **SAS** utilizes the `name` variable as the *source* string and the literal string 'xyz' as the *excerpt*--the character set to search for. The resulting position of the first match is recorded in the new variable `first_xyz`. The output below clearly illustrates the effective application of this function:

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

The `first_xyz` column accurately reflects the search findings. For example, in the name "Andy Lincoln Bernard", the **character** 'y' is the first match from the set {'x', 'y', 'z'}, appearing at position 4. Therefore, the function returns **4**. Conversely, for the name "Barren Michael Smith", since none of the characters 'x', 'y', or 'z' are present anywhere in the string, the **INDEXC function** correctly yields a result of **0**. This consistent and unambiguous return of **0** for no-match scenarios greatly simplifies subsequent data filtering and conditional processing logic.

Distinguishing `INDEX` from `INDEXC`: Sequence Versus Character Set

A critical distinction for any **SAS** programmer to grasp is the functional separation between **INDEXC** and the closely related **INDEX function**. While **INDEXC** is optimized for finding the position of any single character belonging to a specified collection, the **INDEX function** is strictly designed to locate the starting position of an exact, complete **substring**. The fundamental difference lies in the search methodology: **INDEX** demands a perfect, consecutive, sequential match of characters, whereas **INDEXC** is satisfied merely by the existence of any individual character supplied in its input set, regardless of its surrounding context.

To highlight this crucial disparity, we will execute a simultaneous search using both functions on our original data, searching for the term 'Smith'. This direct comparison will vividly demonstrate how each function interprets the search term--as a five-character sequence in the case of **INDEX**, versus five individual, independent characters (S, m, i, t, h) in the case of **INDEXC**. As a result of this differing interpretation, their return values for the same input data diverge significantly.

/*create new dataset*/
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;
```

In this **DATA step**, we define two new variables: `index_smith` and `indexc_smith`. The `index_smith` variable uses the **INDEX function** to rigorously search only for the exact, consecutive **substring** 'Smith'. Conversely, the `indexc_smith` variable uses the **INDEXC function**, which searches broadly for the first appearance of *any* of the characters S, m, i, t, or h within the name field. The results below confirm the expected behavior:

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

Analyzing the output solidifies the functional difference. Examine the first entry, "Andy Lincoln Bernard". The `index_smith` column returns **0** because the exact sequence 'Smith' is entirely absent. However, the `indexc_smith` column returns **7**. This is because the **character** 'i' (which is part of the 'Smith' character set) is present in the word "Lincoln" at the seventh position of the full string. For the entry "Barren Michael Smith", `index_smith` correctly identifies the beginning of the complete sequence 'Smith' at position 15. For `indexc_smith` on the same entry, the result is **11**, corresponding to the first occurrence of one of the target characters--the letter 'h' in 'Michael'. This compelling illustration confirms that **INDEX** is designed for precise sequence matching, while **INDEXC** is effective for broadly identifying the existence of a set of individual characters.

Advanced Considerations and Best Practices for Optimal Use

To maximize the performance and accuracy of the **INDEXC function** in **SAS**, programmers must be mindful of several advanced behaviors and best practices. A crucial consideration is **case sensitivity**. By default, most **SAS** string functions, including **INDEXC**, treat uppercase and

lowercase letters as fundamentally distinct characters. To achieve a true case-insensitive search--a frequent requirement in real-world text analysis--it is necessary to normalize both the *source* and *excerpt* strings to a uniform case (either all uppercase or all lowercase) using functions like **UPCASE()** or **LOWCASE()** immediately before applying **INDEXC**. For instance, the expression `indexc(upcase(name), 'XYZ')` would reliably search for 'x', 'y', or 'z' regardless of their original casing within the `name` variable.

Another factor to consider is performance, particularly when processing exceptionally large **datasets** or strings of immense length. While **INDEXC** is highly optimized for character-set scanning, for scenarios demanding highly complex pattern matching (such as finding characters that adhere to a specific structural rule), alternative methods like regular expressions (implemented via functions such as **PRXMATCH**) might offer greater flexibility. However, for the vast majority of common data preparation and character existence checks, **INDEXC** remains the superior choice due to its excellent balance of code readability and execution efficiency.

Finally, the full power of **INDEXC** is realized when it is logically combined with other essential **SAS** string and data manipulation functions. For example, once the precise position of a target **character** has been identified using **INDEXC**, that resulting index number can be used in conjunction with **SUBSTR()** to extract the preceding or succeeding text components, or with **LENGTH()** to calculate the remaining string length. This capacity to seamlessly chain functions together facilitates sophisticated parsing, allowing raw, unstructured text to be efficiently transformed into clean, structured variables that are immediately ready for statistical analysis.

Conclusion: The Essential Role of `INDEXC`

The **INDEXC function** is undeniably an essential and distinctive component of the **SAS** programmer's toolkit for handling string manipulation tasks. Its unique ability--to rapidly and accurately identify the first position of *any* character from a specified set within a target string--grants it exceptional versatility across a wide spectrum of data management challenges. Whether the task involves performing mandatory data validation, isolating specific textual components, or simply confirming the presence of certain characters, **INDEXC** consistently provides a robust, highly readable, and exceptionally efficient solution.

By grasping its core functionality, internalizing its concise syntax, and clearly recognizing its operational distinction from the **INDEX function**, users are empowered to confidently select the most appropriate tool for their specific string processing needs. The practical demonstrations provided herein clearly illustrate its straightforward application and the unambiguous interpretation of its results, particularly the critical return value of **0** when no match is successfully located.

Mastering powerful, specialized string functions such as **INDEXC** is paramount for anyone who regularly engages with textual data within the **SAS** environment. This mastery enables analysts to

transform complex, semi-structured text into precisely analyzable data points, thereby facilitating deeper insights and radically streamlining the overall data preparation workflow.

Additional Resources for SAS String Functions

To further solidify your **SAS** programming expertise and explore supplementary string manipulation techniques, we recommend reviewing the following tutorials. These resources cover various common functions and advanced methods that perfectly complement your understanding of **INDEXC** and **INDEX**.

Understanding the COMPRESS Function in SAS

How to use the SUBSTR Function in SAS

Using the SCAN Function in SAS

These supplemental materials will assist you in assembling a comprehensive toolkit for tackling diverse string-related challenges within your **SAS** projects.