How to Generate the Alphabet in Google Sheets Using CODE and CHAR Functions

Authored by

Mohammed looti

November 10, 2025

Automating Sequential Data Generation in Spreadsheets

In the realm of data management and sophisticated spreadsheet operations, the requirement to populate columns or rows with sequential identifiers is exceptionally common. Whether compiling complex reports, creating lookup tables, or assigning unique IDs, the necessity for automatically generating the letters of the alphabet, typically spanning from **A** to **Z**, is a frequent task. Attempting to manually input these characters, especially across extensive ranges within a large Google Sheets document, is inefficient, time-consuming, and highly susceptible to human error. Fortunately, the powerful functional capabilities built into Google Sheets offer an elegant, formula-driven solution that completely automates this process. This method leverages the fundamental principles of character encoding, transforming text manipulation from a manual chore into a repeatable, dynamic system.

The core challenge in automating alphabetical sequences lies in the nature of text data within a spreadsheet environment. Characters like "A" and "B" are not inherently numerical and cannot be incremented arithmetically in the way that numbers are. To overcome this limitation and enforce a sequential pattern, we must introduce a crucial intermediary step: converting the character into its corresponding numerical value, increasing that value by one, and then converting the resulting number back into the next character in the sequence. This intricate yet seamless process requires the combined power of two specialized functions: **CODE** and **CHAR**. By integrating these functions into a recursive formula, we can establish a dynamic chain that, when extended via the autofill feature, effortlessly generates the complete alphabetical sequence, dramatically improving data preparation workflow.

This comprehensive tutorial serves as a guide to implementing this sophisticated solution. We will move beyond simple, repetitive manual entry and establish a robust, formula-driven generation method. Our focus will not only be on the practical application of the formula but also on a detailed exploration of how the **CODE** and **CHAR** functions interact with standard character encoding systems. This deep dive into the underlying mechanism ensures you gain a transferable skill set, highly valuable for automating many other complex sequencing and text manipulation tasks within Google Sheets and similar spreadsheet platforms.

The Foundational Mechanism: Bridging Text and Numbers with CODE and CHAR

Successful automated generation of the alphabet hinges upon utilizing specific functions that translate between human-readable characters and machine-readable numerical codes. This critical translation is managed by the **CODE** and **CHAR** functions, which operate based on the ASCII standard, or the more extensive Unicode system used universally in modern computing environments. The **CODE** function performs the vital first conversion: it accepts a text character as

its argument and returns the corresponding decimal numerical value defined by the underlying character set. For instance, according to the standard ASCII table, the uppercase letter "A" is assigned the decimal code **65**, "B" is 66, "C" is 67, and so on. This numerical assignment is paramount because, unlike text strings, numbers are inherently sequential and are easily manipulated using standard mathematical operations.

Once the numerical representation of the starting character is acquired via the **CODE** function, standard arithmetic can be applied directly to drive the sequence. To move from one character to the next, we simply add 1 to the numerical result produced by **CODE**. This simple mathematical addition transforms the code for 'A' (65) into the code for 'B' (66), ensuring progression. Without this essential numerical conversion step, attempting to apply arithmetic, such as adding 1 directly to a text string like "A," would inevitably result in a formula error. Spreadsheet applications are designed to perform mathematical operations exclusively on numerical values, making the **CODE** function an indispensable translator in this process.

The final stage in this transformation loop is executed by the **CHAR** function. The **CHAR** function acts as the inverse of **CODE**: it takes a numerical value (the specific character code) as input and outputs the corresponding text character. By feeding the incremented numerical code (e.g., 66) into the **CHAR** function, the spreadsheet accurately renders the next letter in the sequence (e.g., "B"). The true power emerges when these two functions are nested: the incremented output of **CODE** is immediately used as the input for **CHAR**. This nesting creates a self-referencing, dynamic formula that forms the foundation for automatically generating the entire alphabet sequence within any spreadsheet environment that supports these basic character manipulation functions.

Step-by-Step Implementation: Building the Recursive Formula

Implementing the alphabetical autofill functionality requires a precise setup, starting with the initial character placement and followed by the strategic insertion of the recursive formula. This straightforward process ensures that users of all experience levels can swiftly achieve the desired alphabetical sequence. We begin by anchoring the sequence, define the core formula, and then utilize the powerful autofill feature to complete the task.

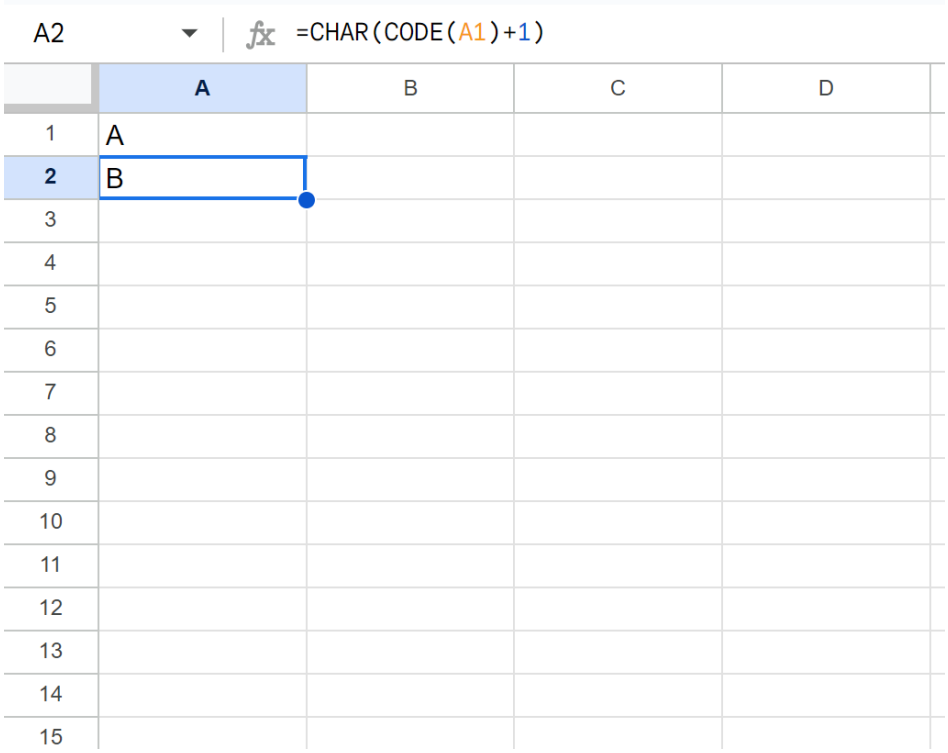
The initial requirement is to manually input the character that will serve as the starting point of your sequence. For generating the standard uppercase alphabet, we must type the letter "A" into the designated starting cell, which will serve as the required reference point for all subsequent calculations. For clarity in this tutorial, we will place the starting letter into cell **A1**. This initial input acts as the indispensable anchor for the entire series, as the formula placed in the cell immediately below it will reference this specific location to determine its own calculated value.

	A	B	C	D
1	A			
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				

With the starting character secured, the next crucial step involves inputting the core sequencing formula into the cell immediately succeeding the starting point--cell **A2** in our example. This nested formula must execute the entire conversion cycle: it must convert the character in the preceding cell to its numerical code, increment that code by one, and subsequently convert the resulting number back into a character. The precise structure of this powerful nested formula is as follows:

=CHAR(CODE(A1)+1)

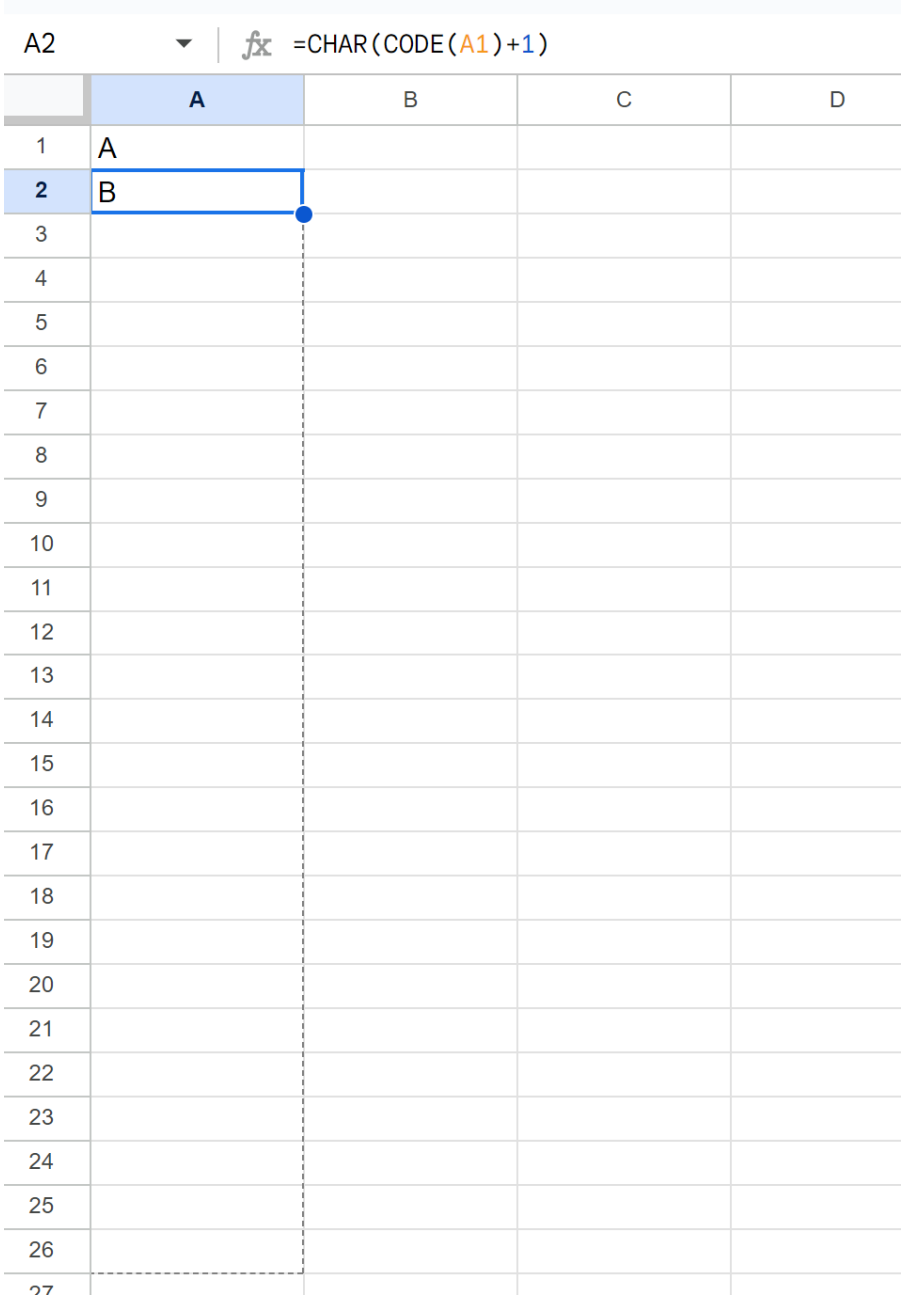
Upon confirming the entry of this formula into cell **A2**, the cell will instantly display the letter "B." This outcome provides immediate confirmation of success: the **CODE** function correctly retrieved the numerical value of "A" (65), the formula successfully incremented this value to 66, and the CHAR function accurately returned the character corresponding to 66, which is "B." This successful implementation of the core logic validates the mechanism necessary for continuous sequence generation.



Leveraging Efficiency: Automating the Sequence with Autofill

The true efficiency and time-saving benefit of this method are unlocked through the utilization of the autofill feature, a standard functionality across all modern spreadsheet applications, including Google Sheets. Because the formula entered in cell **A2** employs a relative reference to the cell directly above it (**A1**), dragging this formula downwards ensures that every subsequent cell calculates the next character based on its own immediate predecessor. This relational link is what makes the sequence dynamic. For instance, when the formula is extended to cell **A3**, the formula structure automatically updates to `=CHAR(CODE(A2)+1)`. Since A2 contains 'B', the formula references 'B', increments its code, and returns 'C'.

To initiate this rapid autofill process, the user must position the mouse cursor precisely over the bottom-right corner of cell **A2**. When correctly positioned, a small, distinctive dark cross symbol (**+**), commonly known as the fill handle, will materialize. By clicking and continuously dragging this fill handle downwards, the formula is copied sequentially across the required range of cells. Dragging the formula 25 cells down from A2 will generate the complete uppercase alphabet, spanning from A through Z, as each cell successfully references the preceding character, determines its numerical code, increments it by one, and displays the subsequent character.



The culmination of this streamlined process is a perfectly aligned, clean list containing every letter of the alphabet, generated entirely using a single, robust formula deployed instantaneously via the autofill mechanism. This technique proves invaluable for various administrative and analytical tasks, such as swiftly establishing column headers for categorical data, generating internal sequential identifiers, or preparing foundational lookup tables. While demonstrated here for the standard A-Z range, the methodology is highly scalable. Users can extend this sequence far past Z, although continued increments past the standard alphabetical range (Z's code is 90) will result in the generation of non-alphabetic symbols, based on the continuation of the character encoding set.

	A	B	C	D
1	A			
2	B			
3	C			
4	D			
5	E			
6	F			
7	G			
8	H			
9	I			
10	J			
11	K			
12	L			
13	M			
14	N			
15	O			
16	P			
17	Q			
18	R			
19	S			
20	T			
21	U			
22	V			
23	W			
24	X			
25	Y			
26	Z			
27				

Decoding the Logic: A Deep Dive into Character Encoding

A thorough examination of the formula `=CHAR(CODE(A1)+1)` reveals the foundational dependency on character encoding standards, such as ASCII. The success of this technique is entirely reliant on the fact that characters are stored and processed internally as numerical values within the computer's memory, thereby allowing for the necessary mathematical manipulation. When this recursive formula is initially placed in cell **A2**, the process begins by analyzing the content of its referenced cell, **A1**, which holds the letter "A."

The inner function, `CODE(A1)`, is executed first. Since the uppercase letter "A" corresponds directly to the decimal ASCII code **65**, the function accurately returns this numerical value. Subsequently,

the arithmetic operation defined by the formula is carried out: `+1` is added to 65, resulting in the new numerical code **66**. This singular, simple increment is the engine that drives the entire alphabetical sequence forward.

Finally, the outer function, `CHAR(66)`, takes the stage. The **CHAR** function efficiently consults the character set and retrieves the character associated with the numerical code 66. According to the standard encoding, the decimal value 66 corresponds precisely to the character "B." This final character is then displayed in cell **A2**. As the entire formula is dragged down the column, the relative reference point dynamically shifts (A1 becomes A2, A2 becomes A3, and so forth), guaranteeing a continuous, self-correcting sequence. For example, in cell A3, the formula references A2 (which contains 'B', code 66), increments the code to 67, and successfully returns the character 'C'. This efficient sequential process continues until the designated range is fully populated.

Case Sensitivity and Extending the Character Range

When implementing the **CODE** and **CHAR** solution, a critical consideration is the inherent sensitivity to character case, dictated by both the ASCII and Unicode systems. These standards assign distinctly separate numerical values to uppercase and lowercase letters. Specifically, the range for uppercase letters (A-Z) occupies codes 65 through 90, while the range for lowercase letters (a-z) begins significantly later, occupying codes 97 through 122. These are non-contiguous numerical blocks, meaning that a sequence generated from 'Z' will not automatically transition to 'a'; it will instead proceed to the character defined by code 91 (which is typically a symbol like '[').

This structural distinction offers flexibility. If the user initiates the sequence in cell **A1** with a lowercase "a" instead of an uppercase "A," the **CODE** function will return 97. Consequently, the formula will accurately generate the entire sequence of lowercase letters (a, b, c, ...) until it naturally reaches "z." Therefore, the desired case for the output sequence is determined solely by the initial character manually entered into the anchor cell (A1).

While this technique excels at generating simple A-Z or a-z sequences, extending the sequence beyond the single alphabet (e.g., creating double-letter sequences like AA, AB, AC) requires significantly more complex formulas. These advanced sequences necessitate combining the **CHAR(CODE()+1)** logic with other functions, such as `COLUMN()` or `ROW()`, alongside arithmetic functions like `MOD` and `QUOTIENT`, to correctly manage the "roll-over" logic (similar to how a base-26 number system works). However, for the fundamental task of producing a contiguous alphabetical list, the recursive **CHAR(CODE()+1)** method remains the most direct, elegant, and efficient approach available.

Additional Resources for Advanced Spreadsheet Mastery

Mastering the fundamental manipulation of text data using functions like **CODE** and **CHAR** unlocks tremendous potential for automating complex tasks within Google Sheets. These character functions are frequently incorporated as foundational components within larger, more sophisticated array formulas designed for comprehensive data transformation and reporting.

To further enhance data handling capabilities and deepen understanding of spreadsheet dynamics, the following resources provide guidance on related advanced tasks:

How to handle complex text manipulations using the **CONCATENATE** function or the streamlined ampersand operator (**&**) for string joining.
Techniques for dynamic range generation and creating sophisticated numerical or date sequences using the powerful **SEQUENCE** function, which offers alternatives to manual dragging.
Detailed, official documentation providing the full list of Google Sheets functions, including their specific syntax, arguments, and practical examples.