How to Easily Convert Text to Numbers in VBA

Authored by

stats writer

January 1, 2026

When developing applications using Visual Basic for Applications (VBA), developers frequently encounter situations where numeric information is presented as a text String. To perform calculations, comparisons, or storage in a dedicated numeric variable, this text representation must be accurately converted into a compatible data type, such as the Integer data type.

Fortunately, VBA provides several built-in functions designed specifically for this purpose. The two primary methods involve the CInt() function, which stands for Convert to Integer, and the less strict Val() function. Understanding the nuances of each is essential for robust code development, especially concerning how they handle non-numeric characters and potential overflow errors. We will examine both functions in detail and provide practical examples demonstrating effective implementation, including critical error handling techniques.

The Primary Conversion Tool: Understanding the CInt() Function

The CInt() function is the standard method recommended in VBA for converting an expression into an Integer. It attempts to cast the input value--whether it is a String, Double, or another numeric type--into a whole number that fits within the range of the standard 16-bit Integer type (typically from -32,768 to 32,767). If the input value contains characters that cannot be interpreted as part of a number, or if the resulting number exceeds the maximum allowed range, the CInt() function will generate a runtime error, necessitating proper error trapping in production code.

When converting floating-point numbers, CInt() employs banking rounding (or "round to even"). This means that if the fractional part is exactly 0.5, the function rounds the number to the nearest even whole number. For instance, CInt(2.5) evaluates to 2, while CInt(3.5) evaluates to 4. This behavior is distinct from standard arithmetic rounding (round half up) and is important to consider when dealing with financial or statistical data where rounding precision is critical. For most basic String conversions, however, its behavior is straightforward: CInt("450") yields the numeric value 450.

Utilizing the CInt() function is highly recommended when you are certain that the input String represents a valid integer and falls within the acceptable range. The following sections illustrate its usage in common macro structures, demonstrating how to iterate through a range of cells and apply the conversion systematically. This approach is highly efficient for bulk data processing within Microsoft Excel environments integrated with VBA.

You can use the **CInt** function in VBA to convert a text string to an integer.

Here are two common ways to use this function in practice:

Method 1: Direct Conversion of String to Integer in VBA

This first method involves a direct conversion using the CInt() function applied to every cell within a specified range. This is the simplest approach, suitable when the source data is guaranteed to be clean and purely numeric, thereby minimizing the risk of runtime errors. We use a standard `For...Next` loop to iterate through the rows, retrieve the text value, convert it, and output the resulting Integer into the destination column.

The code snippet below demonstrates the fundamental structure for this operation. Notice the declaration of the iteration variable `i` as an Integer, which is standard practice for loop counters. The core logic resides in the assignment statement, where `Range("A" & i)` retrieves the String input, CInt() performs the conversion, and the result is placed into the corresponding cell in column B.

It is important to remember that if any cell within the range `A2:A11` contains non-numeric text (e.g., "N/A" or "Fifty") or a number too large for the Integer type, this macro will halt execution with an error. Therefore, while simple, this method should be reserved for highly controlled datasets or accompanied by advanced error handling mechanisms.

Sub ConvertStringToInteger()

Dim i As Integer

For i = 2 To 11
Range("B" & i) = CInt(Range("A" & i))
Next i

End Sub

This particular macro will convert each string in the range **A2:A11** to an integer and display the resulting integers in the range **B2:B11**.

Method 2: Controlled Conversion with Error Prevention using IsNumeric()

When dealing with heterogeneous data where some cells might contain non-numeric characters, relying solely on CInt() is risky. A much more robust solution involves utilizing the built-in IsNumeric() function prior to attempting conversion. This function evaluates the contents of the cell and returns `True` if the value can be successfully interpreted as a number, thereby preempting runtime errors caused by invalid input.

By wrapping the CInt() call within an `If...Then...Else` block that checks `IsNumeric()` first, we

ensure that the conversion only happens when it is safe. This drastically improves the stability and reliability of the macro when processing real-world data feeds that often contain headers, footnotes, or blank entries mixed with numerical values. This conditional logic is critical for maintaining data integrity and ensuring the macro completes its task without interruption.

In the scenario presented below, if the value in column A is not numeric, the macro assigns a default value--in this case, zero (`0`)--to the output cell in column B. This mechanism allows the programmer to define how invalid data should be treated, rather than letting the program crash. Using `0` is a common placeholder, but depending on requirements, a text value like "Error" or leaving the cell empty might also be appropriate alternatives. This methodology represents a best practice for data cleansing and transformation in VBA.

```
Sub ConvertStringToInteger()

Dim i As Integer

For i = 2 To 11
If IsNumeric(Range("A" & i)) Then
Range("B" & i) = CInt(Range("A" & i))
Else
Range("B" & i) = 0
End If
Next i

End Sub
```

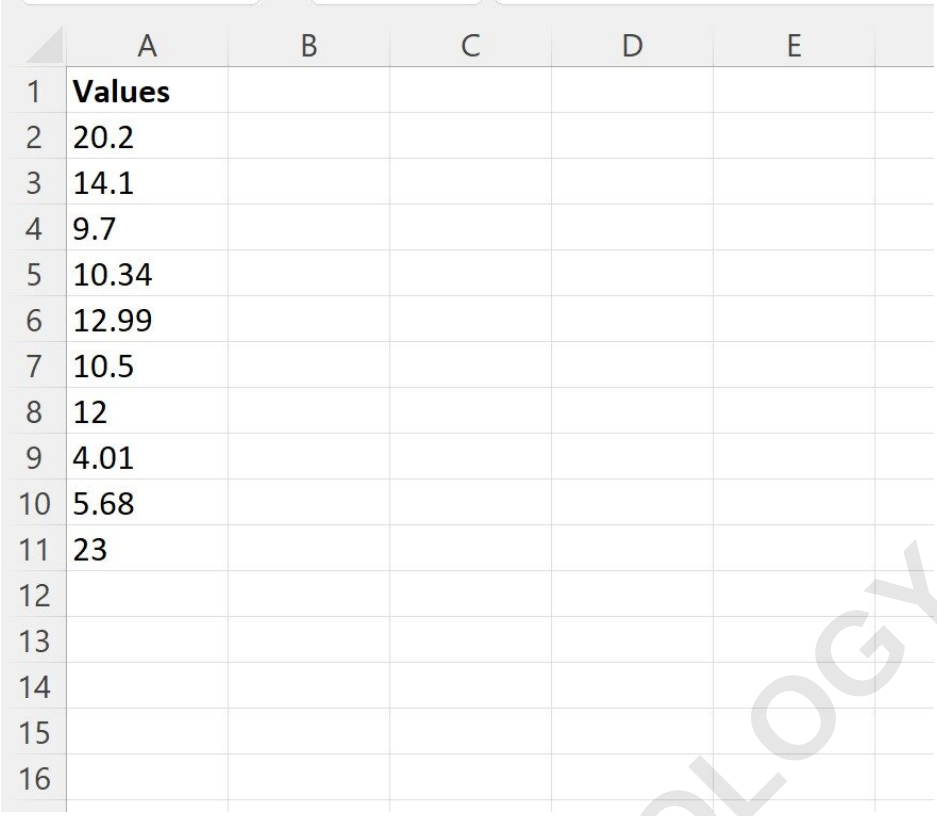
This particular macro will convert each string in the range **A2:A11** to an integer only if the string is a number. Otherwise, the string will be converted to a value of zero.

The following examples show how to use each method in practice.

Example 1: Direct String to Integer Conversion Implementation

Scenario Setup and Execution of Direct CInt()

Imagine we have collected data where values intended to be numeric have been inadvertently formatted as text strings within an Excel column. This often occurs during CSV imports or when data entry defaults to text format. Suppose we have the following column of values in Excel, contained in range A2:A11, which are currently formatted as text strings:



	A	B	C	D	E
1	**Values**				
2	20.2				
3	14.1				
4	9.7				
5	10.34				
6	12.99				
7	10.5				
8	12				
9	4.01				
10	5.68				
11	23				
12					
13					
14					
15					
16					

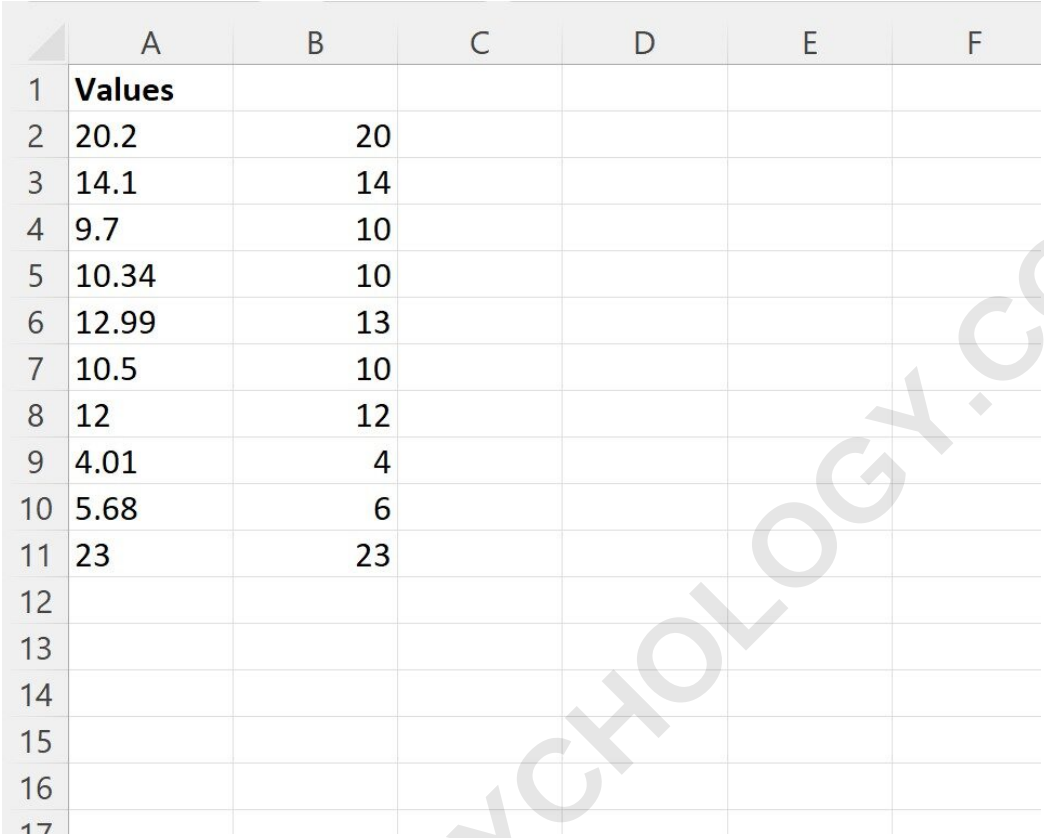
Our objective is to convert each of these text strings into genuine integer values and then display these converted numerical outputs in the adjacent column B. Since we can visually confirm that all input values are valid numbers within the Integer limits, the direct application of the CInt() function is the fastest and most efficient choice.

To achieve this transformation, we create the following straightforward macro, which leverages a loop to process the entire range iteratively. The macro requires minimal setup, focusing purely on the conversion logic. The use of the `Range()` object concatenated with the loop counter `i` allows dynamic access to each cell in the input column.

```
Sub ConvertStringToInteger()

Dim i As Integer

For i = 2 To 11
Range("B" & i) = CInt(Range("A" & i))
Next i

End Sub
```

Upon successfully running this macro, the values in column B are populated with the numerical

equivalents of the text strings from column A. The visual confirmation in Excel typically includes the numbers aligning to the right (default for numeric formats) and being available for mathematical functions. We receive the following output, confirming successful type conversion:

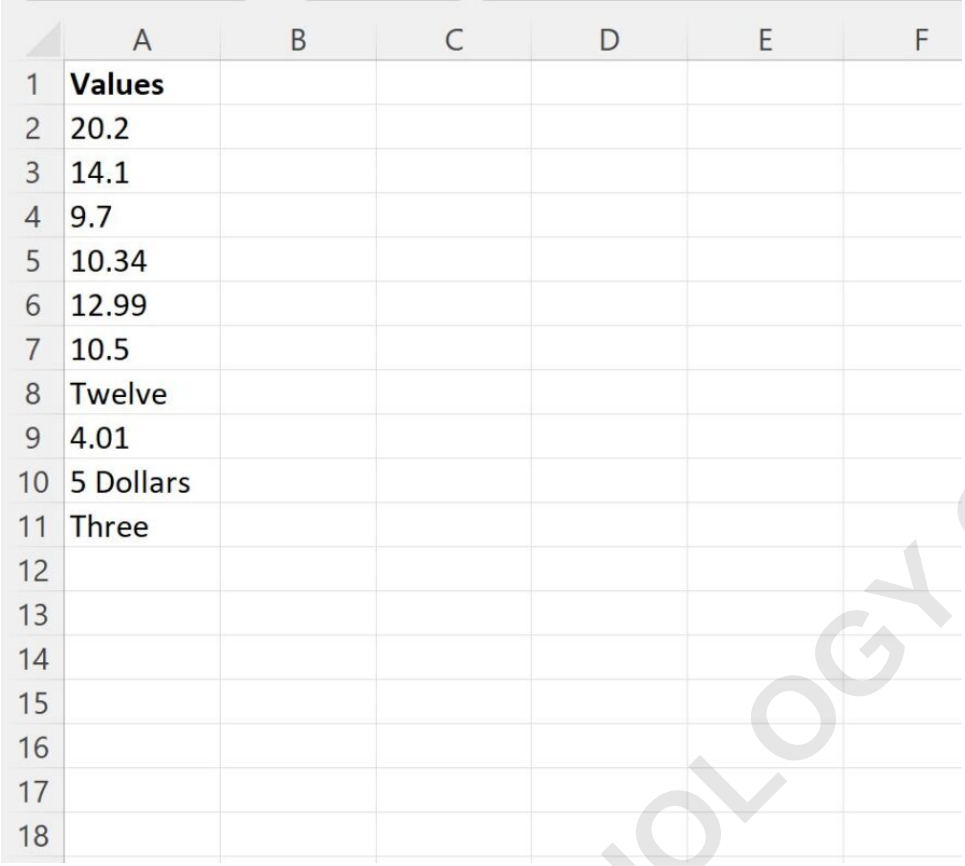


	A	B	C	D	E	F
1	**Values**					
2	20.2	20				
3	14.1	14				
4	9.7	10				
5	10.34	10				
6	12.99	13				
7	10.5	10				
8	12	12				
9	4.01	4				
10	5.68	6				
11	23	23				
12						
13						
14						
15						
16						

Example 2: Conditional Conversion and Error Mitigation

Handling Mixed Data Types Using IsNumeric()

A far more common scenario in data manipulation is dealing with mixed input. Suppose our input column contains a mixture of valid numeric strings and non-numeric text entries (e.g., identifiers, error codes, or descriptive text). For instance, consider the following column of values in Excel that are currently formatted as text strings, which includes "N/A" and "Missing" entries:



	A	B	C	D	E	F
1	**Values**					
2	20.2					
3	14.1					
4	9.7					
5	10.34					
6	12.99					
7	10.5					
8	Twelve					
9	4.01					
10	5 Dollars					
11	Three					
12						
13						
14						
15						
16						
17						
18						

In this situation, directly applying the CInt() function (as in Example 1) would cause the macro to crash upon reaching rows containing "N/A" or "Missing." Therefore, we need a refined approach: we want to convert each string to an integer *only if the string is inherently numeric*, and assign a default value otherwise.

To implement this robust logic, we utilize the `IsNumeric()` validation function within the loop structure. This preemptive check ensures that the CInt() conversion is only attempted on cells proven to hold a numerical value. If the check fails (i.e., the cell contains non-numeric text), the `Else` block is executed, handling the exception gracefully by assigning the value `0`, thus preventing a runtime error and ensuring continuous execution of the macro across the entire dataset.

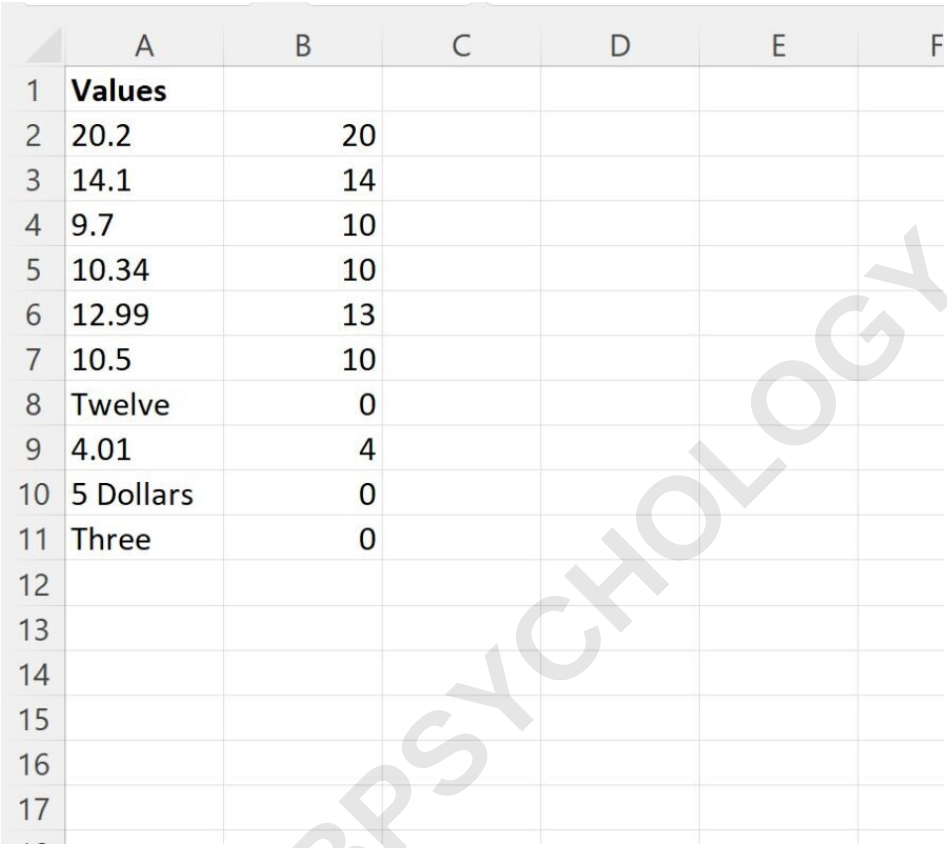
Sub ConvertStringToInteger()

Dim i As Integer

For i = 2 To 11
If IsNumeric(Range("A" & i)) Then
Range("B" & i) = CInt(Range("A" & i))
Else
Range("B" & i) = 0

```
End If
Next i

End Sub
```

When we run this macro, we receive the following output:



	A	B	C	D	E	F
1	**Values**					
2	20.2	20				
3	14.1	14				
4	9.7	10				
5	10.34	10				
6	12.99	13				
7	10.5	10				
8	Twelve	0				
9	4.01	4				
10	5 Dollars	0				
11	Three	0				
12						
13						
14						
15						
16						
17						

Notice that only the text strings in column A that are numbers are converted to integers in column B.

Otherwise, the text strings are simply converted to a value of zero.

Alternative Conversion Method: Utilizing the Val() Function

While CInt() is the preferred function for strict type casting, VBA also offers the Val() function. The behavior of Val() is fundamentally different from that of CInt(), making it suitable for specific scenarios, although it is generally less favored for controlled conversions. The Val() function reads a string sequentially from the first character and stops reading as soon as it encounters a character it does not recognize as part of a number (including spaces, commas, or letters).

For example, `Val("123ABC")` returns 123, while `Val("ABC123")` returns 0. If the string starts with numeric characters, Val() extracts those leading digits and returns them as a numerical Double type, which can then be coerced into an Integer if necessary. Crucially, Val() does not generate a runtime error when encountering non-numeric characters; it simply stops processing, which might mask errors if the entire string was expected to be numeric.

Due to its behavior of returning a Double and its peculiar stopping criteria, Val() is less suited for direct integer conversion unless strict input validation confirms the leading portion of the string is the desired numeric part. For reliable, type-safe conversion, particularly in data processing environments integrated with Excel, CInt() combined with `IsNumeric()` remains the superior and professional choice for converting a full String value to an Integer.

Summary of Conversion Considerations

Choosing the correct conversion function in VBA depends entirely on the nature and cleanliness of the source data. When speed and direct conversion are paramount and data quality is guaranteed (i.e., every input is a valid number within the range), the standalone `CInt(value)` function offers the most direct solution, as demonstrated in Example 1. This method is preferred for homogeneous datasets.

However, when dealing with external imports, user input, or legacy data where text and numeric values might be intermingled, employing conditional logic using `If IsNumeric(value) Then CInt(value)` is indispensable. This approach prevents macro failure, ensures data continuity, and allows the developer to define precise fallback behavior (like assigning 0 or null) for invalid entries, maximizing code resilience.

Finally, while conversion functions like Val() exist, they often introduce ambiguity due to their partial parsing behavior and return type, making them less suitable for the explicit conversion of an entire text field into an Integer. Adherence to the standard type conversion functions (like `CInt`, `CDbl`, `CLng`, etc.) ensures that the conversion respects the defined constraints of the target data type and provides more predictable error handling.

Note: You can find the complete documentation for the VBA **Cint** function here.