How to Round Down Numbers in VBA Using Int and Fix

Authored by

stats writer

November 20, 2025

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2025). *How to Round Down Numbers in VBA Using Int and Fix*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=98270

When working with numerical data in **VBA**, precision control is often paramount. While the **Int function** (Integer) and the **Fix function** are native **VBA** tools used for truncating decimal values, they primarily serve to round a number down to the nearest whole integer. For instance, if you apply the **Int function** to the value 2.8, the result is 2. Similarly, utilizing `Fix(2.8)` yields the same outcome, 2. However, these basic functions lack the flexibility required for precise control over the number of decimal places or rounding to specific significant figures (like hundreds or thousands), which is often necessary in complex financial or statistical computations. For professional applications requiring controlled, directional rounding--specifically rounding exclusively down--the superior choice is the specialized **RoundDown method** available through the **WorksheetFunction** object.

Understanding Basic Rounding Functions: Int vs. Fix

Although this article focuses on the powerful **RoundDown method**, it is beneficial to first understand the foundational functions built directly into the **VBA** environment: **Int** and **Fix**. Both functions effectively truncate the decimal portion of a number, moving towards the lower integer. For positive numbers, their behavior is identical. For example, `Int(4.9)` returns 4, and `Fix(4.9)` also returns 4. This is a simple downward rounding mechanism to the closest integer less than or equal to the input number.

However, a crucial distinction emerges when handling negative numbers. The **Int function** rounds down to the greatest negative integer that is less than or equal to the number. For instance, `Int(-4.3)` returns -5, as -5 is mathematically 'lower' than -4.3. Conversely, the **Fix function** simply truncates the decimal part, moving toward zero. Therefore, `Fix(-4.3)` returns -4. While these differences are important for pure **VBA** mathematical calculations, they do not offer the ability to specify rounding precision beyond the integer level, which is why we must turn to Excel's built-in **WorksheetFunction** library when dealing with cell values and specified precision.

Utilizing the Specialized RoundDown Method

To achieve precise downward rounding that mimics the native Excel function `ROUNDDOWN()`, developers must access this functionality through the **WorksheetFunction** object within **VBA**. This is the preferred method when manipulating values directly within cells or when high precision rounding to a specified digit count is required. The **RoundDown method** ensures that the number is always rounded toward zero, thereby adhering strictly to a downward rounding convention based on the specified precision, regardless of the number's sign.

The primary strength of the **RoundDown method** lies in its second argument, which dictates the rounding scale. Unlike **Int** or **Fix**, which are limited to the integer boundary, **RoundDown** allows you to round to decimal places, tens, hundreds, or even thousands. This versatility makes it

ARABPSYCHOLOGY.COM

indispensable for scenarios like tax calculation, inventory management, or currency conversion where fractional values must be consistently truncated without rounding up.

Syntax and Structure of the RoundDown Method

The syntax for implementing the **RoundDown** method is clean and intuitive, following the structure required by the **WorksheetFunction** object. You typically define the cell range containing the value to be rounded and the cell range where the result will be outputted, all within a standard **macro** procedure.

The basic structure uses the following pattern, where the value is retrieved from a source cell (e.g., A1) and the result is written to a destination cell (e.g., B1). This approach ensures that we utilize the robust calculation engine of Excel directly within our **VBA** code.

You can use the **RoundDown** method in **VBA** to round values down.

This function uses the following basic syntax:

```
Sub RoundDownValue()
Range("B1") = WorksheetFunction.RoundDown(Range("A1"), 0)
End Sub
```

This particular example demonstrates how to call the **RoundDown** method, accessing the value stored in cell **A1**, instructing the function to round it down to the nearest whole number (indicated by the `0` argument), and subsequently writing the calculated result into cell **B1**.

Defining Rounding Precision Arguments (Num_Digits)

A critical component of using the **RoundDown** method is the second argument, known as `Num_Digits`. This numeric value determines the level of precision to which the rounding operation should be performed. Understanding how to correctly set this argument is essential for achieving the desired output, as the number can be positive, negative, or zero, each corresponding to a different rounding scale relative to the decimal point.

When `Num_Digits` is positive, the rounding occurs to the right of the decimal point, controlling the number of decimal places retained. When it is set to zero, the function rounds to the nearest whole number. Crucially, when `Num_Digits` is negative, the function rounds the number to the left of the decimal point, effectively rounding to the nearest ten, hundred, or thousand. This provides immense flexibility for bulk data manipulation where values must be simplified to specific large units.

ARABPSYCHOLOGY.COM

Note that the second argument in the **RoundDown** method specifies the number of digits to round where:

-3 rounds down to the nearest thousand
-2 rounds down to the nearest hundred
-1 rounds down to the nearest ten
0 rounds down to the nearest whole number
1 rounds down to the nearest tenth (one decimal place)
2 rounds down to the nearest hundredth (two decimal places)
3 rounds down to the nearest thousandth (three decimal places)

This system continues for higher positive or negative integers, providing granular control over precision.

Practical Application of RoundDown: Setting up the Macro

To effectively demonstrate the functionality of the **RoundDown method**, we will walk through several practical examples. Each example utilizes a simple **macro** structure designed to read a test value from a source cell (A1) and apply the rounding logic, presenting the calculated result in a target cell (B1). Ensure that you have the Developer tab enabled and can access the **VBA** Editor (Alt + F11) to implement these procedures.

For consistency across all demonstrations, we will use the test value 1,432.78 in cell A1. This number includes both significant digits in the thousands place and a two-digit decimal value, allowing us to clearly observe the effects of various positive, negative, and zero `Num_Digits` arguments.

The following sections provide illustrative examples detailing how to use the **RoundDown** method in practice for different rounding requirements.

Example 1: Round Down to Nearest Whole Number (Zero Decimals)

The most common requirement for downward rounding is to eliminate all fractional parts of a number while ensuring the resultant integer is the floor value of the original number. This is accomplished by setting the `Num_Digits` argument to zero (0). This configuration is similar in effect to the **Fix function** for positive numbers, but it uses the robust **WorksheetFunction** engine, making it ideal for formula integration within Excel-based **macro** operations.

We can create the following **macro** to round down the value in cell **A1** to the nearest whole number and display the result in cell **B1**:

ARABPSYCHOLOGY.COM

```
Sub RoundDownValue()
Range("B1") = WorksheetFunction.RoundDown(Range("A1"), 0)
End Sub
```

When this **macro** is executed, we observe the following transformation of data:

	A	B	C	D	E	F
1	1432.78	1432				
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

ARABPSYCHOLOGY.COM

Upon examination of the output, notice that the original value 1,432.78 in cell **A1** has been precisely rounded down to the nearest whole number, resulting in 1,432, which is displayed in cell **B1**. The decimal portion (.78) is completely truncated due to the `0` argument.

This zero-digit rounding is highly effective for scenarios where only integers are valid outputs, such as counting units, allocating full resources, or calculating elapsed days. It avoids the bias of standard mathematical rounding (which would round 1,432.78 up to 1,433) by strictly adhering to the "round down" instruction.

Example 2: Controlling Scale: Rounding Down to the Nearest Hundred

The true power of the **RoundDown method** becomes apparent when using negative arguments. A negative `Num_Digits` value instructs the function to round to the nearest specified multiple to the left of the decimal point. Setting the argument to `-2`, for instance, rounds the input value down to

the nearest hundred. This technique is invaluable for high-level reporting or financial forecasting where precise unit details are irrelevant, and data needs to be presented in simplified hundreds or thousands.

We can create the following **macro** to round down the value in cell **A1** to the nearest hundred and display the result in cell **B1**:

```
Sub RoundDownValue()
Range("B1") = WorksheetFunction.RoundDown(Range("A1"), -2)
End Sub
```

Upon execution of this procedure, the following result is generated:

	A	B	C	D	E	F
1	1432.78	1400				
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

ARABPSYCHOLOGY.COM

Observe that the value 1,432.78 in cell **A1** has been rounded down to the nearest hundred, yielding the result 1,400 in cell **B1**. The `-2` argument effectively zeroes out the tens and units place (32.78), rounding the entire number down to the closest hundred lower than the original input.

This demonstrates sophisticated control over number scaling. If the argument were `-1`, the number would round down to the nearest ten (1,430). If it were `-3`, the number would round down to the nearest thousand (1,000). Always verify that the negative digit argument corresponds precisely to

the power of ten you intend to round to.

Example 3: Maintaining Decimal Precision: Rounding Down to the Nearest Tenth

When working with currency or measurements, it is often necessary to maintain a specific number of decimal places, but still require a strict downward truncation of any remaining digits. This is achieved by using a positive value for the `Num_Digits` argument. Setting `Num_Digits` to 1, for example, instructs the function to retain the value up to the tenths place (one decimal place) and round down everything that follows.

This ensures that you capture a fixed level of precision while discarding any values that might cause the number to round up mathematically. This precision is vital in regulatory environments where figures must be truncated conservatively.

We can create the following **macro** to round down the value in cell **A1** to the nearest tenth (i.e., one decimal place) and display the result in cell **B1**:

```
Sub RoundDownValue()
Range("B1") = WorksheetFunction.RoundDown(Range("A1"), 1)
End Sub
```

Executing the procedure yields the following result:

ARABPSYCHOLOGY.COM

	A	B	C	D	E	F
1	1432.78	1432.7				
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						

ARABPSYCHOLOGY.COM

The output confirms that the value 1,432.78 in cell **A1** has been rounded down to the nearest tenth, resulting in 1,432.7 in cell **B1**. Although the next digit was 8, which would typically cause a standard round-up to 1,432.8, the **RoundDown** function strictly truncates toward zero, preserving the original tenths value.

Best Practices for Using RoundDown in Production Code

While the **RoundDown** method is exceptionally useful for deterministic rounding, its implementation should follow several best practices, particularly when integrated into large-scale production code. Always handle potential errors gracefully. Since **WorksheetFunction** methods can fail if the input arguments are invalid or if the range object cannot be resolved, incorporating robust error handling (such as `On Error Resume Next` followed by checking the `Err` object, or using the less crash-prone `Application.RoundDown`) is highly recommended.

Furthermore, always ensure data type consistency. The **RoundDown method** typically returns a `Double`, even if the result is an integer. If the calculated value is immediately required for integer-only operations, consider explicitly converting the result using `CLng()` or `CInt()` after the rounding operation is complete. This prevents subtle type mismatch errors later in the code execution flow.

Finally, remember the scope of the rounding. The **RoundDown method** found within the

WorksheetFunction collection is intrinsically linked to Excel's calculation engine. If you are developing pure **VBA** code outside of an Excel context (e.g., in Access), this method will not be available. In such non-Excel environments, you would need to rely solely on the native **Int** or **Fix** functions, or implement a custom function for decimal-place rounding.

Note: You can find the complete official documentation for the **VBA RoundDown method** on the Microsoft website, detailing all possible error states and compatibility notes.

ARABPSYCHOLOGY.COM