Learning VBA: Using Wildcards with the Like Operator for Pattern Matching

Authored by

Mohammed looti

November 15, 2025

In the realm of data processing and automation within Microsoft Office environments, **VBA** (Visual Basic for Applications) stands as a crucial tool for developers seeking to manipulate and analyze large datasets efficiently. When dealing with text data--often referred to as strings--one of the most powerful and flexible mechanisms for pattern recognition is the **Like operator**. This operator enables programmers to compare an input string against a defined pattern, dramatically simplifying complex searching, filtering, and validation tasks that would otherwise require extensive, cumbersome code.

The true strength of the **Like operator** lies in its integration with special **wildcard characters**. These characters are not matched literally; instead, they represent one or more variable characters, allowing for highly flexible and efficient comparisons. Mastering these built-in wildcards is fundamental for writing robust search routines and ensuring the integrity of data workflows orchestrated through **VBA**. Ignoring these specialized tools means resorting to slower, less scalable methods like nested loops or complex conditional statements.

This guide delves into the specifics of each available **wildcard character** in **VBA**, providing detailed explanations and practical, real-world examples demonstrating how they revolutionize text pattern matching, particularly within Microsoft Excel automation tasks. We will explore how to leverage these symbols to move beyond simple exact matches and tackle sophisticated data filtering challenges across various datasets.

Understanding the Core VBA Wildcards for Pattern Matching

The set of primary **wildcard characters** available for use with the **Like operator** in **VBA** offers distinct capabilities, each designed to handle a different type of ambiguity in pattern matching. Developers must understand the exact function of each character to construct effective search patterns. It is important to note that these wildcards are distinct from those used in standard file system searches; they are tailored specifically for string comparison logic within the programming environment.

The standard **VBA** wildcards are:

*** (Asterisk)**: The most versatile wildcard, matching **any sequence of zero or more characters**. It is commonly used for identifying substrings or when the length of the string segment is unknown. Using the asterisk allows the pattern to be inclusive of any characters surrounding the specific target text.
? (Question Mark): This precise wildcard matches **exactly one single character** at its position. It is ideal for searching for patterns where the length is fixed but one specific character is unknown or variable (e.g., searching for "C?t" to match "Cat," "Cot," or "Cut").
 # (Hash/Pound Sign): This specialized character matches **exactly one single numeric digit** (0 through 9). It is indispensable for validating specific numerical formats or ensuring that a part of a

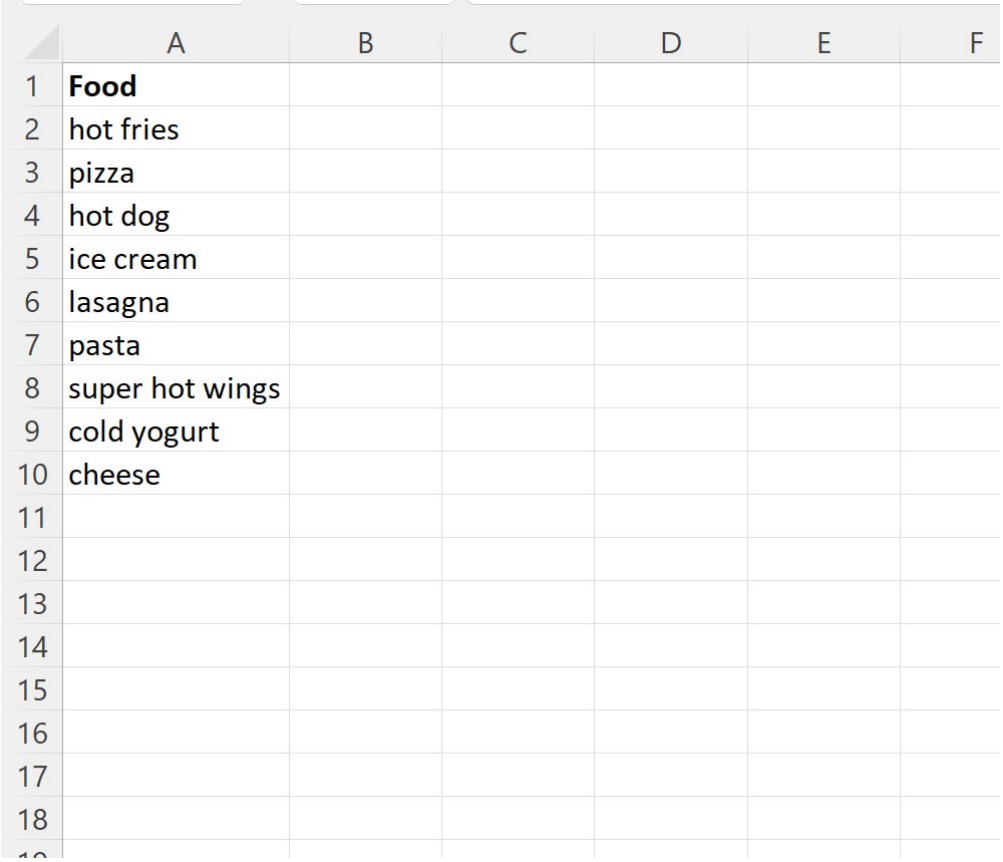
string contains only digits.
(Square Brackets): These brackets are used to match **any single character specified within the list or range** defined inside them. For instance, matches any single uppercase letter, while matches only the digits 1, 3, or 5. This provides powerful control over character inclusion or exclusion.

The following practical examples illustrate how these **wildcard characters** can be implemented to solve common data manipulation tasks, ranging from basic substring identification to complex pattern validation across cell ranges in a spreadsheet. We will use various scenarios to demonstrate the necessary syntax and logical application for each wildcard type.

Example 1: Leveraging the * Wildcard for Inclusive Substring Identification

The asterisk (*****) wildcard is arguably the most frequently used tool in **VBA** pattern matching due to its ability to represent any number of characters, including none at all. This makes it invaluable when searching for a specific substring that might appear anywhere within a larger data string. By surrounding the target search term with asterisks, we instruct the **Like operator** to look for the term regardless of what characters precede or follow it.

Consider a practical scenario where we maintain an inventory list of food items recorded in Column A of an Excel worksheet. Our objective is to quickly flag all items that contain the specific sequence of letters "hot." The initial dataset is shown below, demonstrating varied formats and item names.



	A	B	C	D	E	F
1	**Food**					
2	hot fries					
3	pizza					
4	hot dog					
5	ice cream					
6	lasagna					
7	pasta					
8	super hot wings					
9	cold yogurt					
10	cheese					
11						
12						
13						
14						
15						
16						
17						
18						

To achieve this inclusive search, we will develop a simple **macro**. This routine will iterate through the cells in the specified range (A2 to A10), apply the **Like operator** using the pattern `"*hot*"`, and then output the result of the determination into the corresponding cell in Column B. This demonstrates the immense power of the ***** wildcard in performing broad, inclusive, and highly adaptable substring searches, saving significant time over manual data review and complex logical structures.

Sub FindString()

Dim i As Integer

For i = 2 To 10
If Range("A" & i) Like "*hot*" Then
Range("B" & i) = "Contains hot"
Else
Range("B" & i) = "Does Not Contain hot"
End If
Next i

End Sub

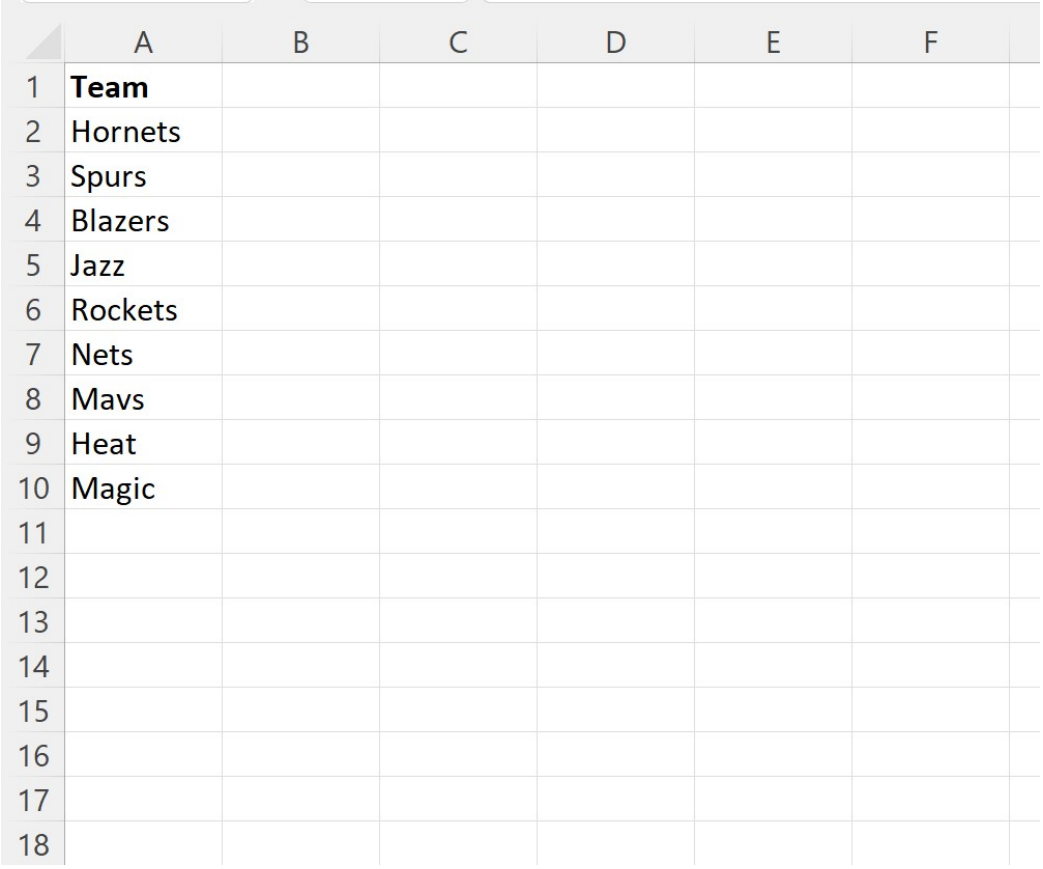
As anticipated, executing this **macro** successfully populates Column B. The crucial observation here is how the pattern `"*hot*"` successfully identifies entries such as "Hotdog" and "Hottub" because the asterisks flanking the search term allow for any combination of characters before and after the central sequence. This validates the effectiveness of the ***** wildcard for generalized substring matching within any given data entry.

	A	B	C	D	E
1	**Food**				
2	hot fries	Contains hot			
3	pizza	Does Not Contain hot			
4	hot dog	Contains hot			
5	ice cream	Does Not Contain hot			
6	lasagna	Does Not Contain hot			
7	pasta	Does Not Contain hot			
8	super hot wings	Contains hot			
9	cold yogurt	Does Not Contain hot			
10	cheese	Does Not Contain hot			
11					
12					
13					
14					
15					
16					
17					
18					

Example 2: Defining Boundary Conditions for Strings with the * Wildcard

While the previous example demonstrated an inclusive search, the position of the ***** wildcard can be strategically altered to define strict boundary conditions for the pattern match. This grants the developer precise control, allowing them to isolate strings that only begin with or only end with a specific sequence of characters. This is a common requirement in data cleansing and validation routines where file extensions or specific departmental codes need to be checked for accurate formatting.

If our objective is solely to identify data strings that conclude with a certain sequence, we must place the asterisk only at the beginning of the pattern. Let us apply this logic to a list of basketball team names in Column A, which serve as our sample dataset:



	A	B	C	D	E	F
1	**Team**					
2	Hornets					
3	Spurs					
4	Blazers					
5	Jazz					
6	Rockets					
7	Nets					
8	Mavs					
9	Heat					
10	Magic					
11						
12						
13						
14						
15						
16						
17						
18						

To specifically isolate team names that end exactly with the sequence "ets," we define the search pattern as `"*ets"`. The asterisk at the front signals that any number of characters are acceptable leading up to the final sequence. However, the lack of a wildcard following "ets" imposes a mandatory termination requirement--the match must conclude precisely with those three letters. This targeted boundary check is implemented efficiently using the following **macro**, leveraging the **Like operator**:

Sub FindEndingString()

Dim i As Integer

For i = 2 To 10
If Range("A" & i) Like "*ets" Then
Range("B" & i) = "Ends in ets"
Else
Range("B" & i) = "Does Not End in ets"
End If
Next i

End Sub

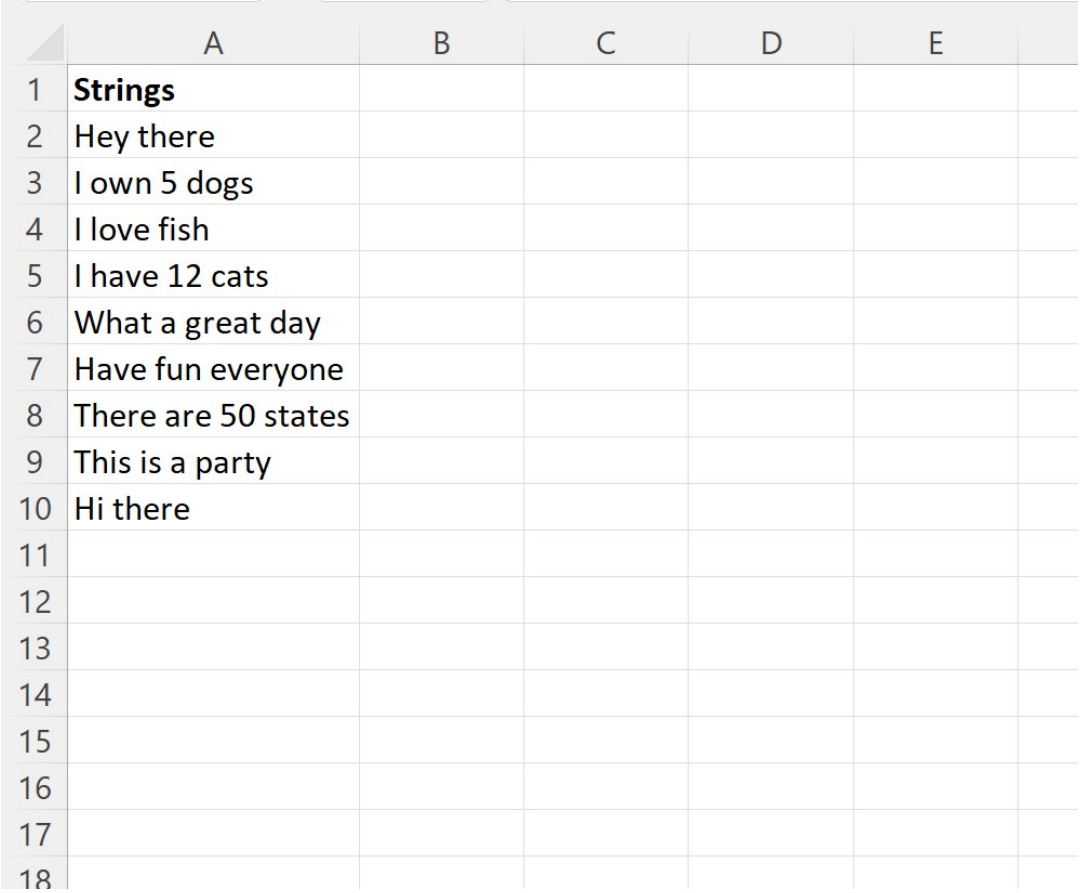
Upon execution, the **VBA** code successfully filters the list. Only the "New Jersey Nets" and the "Charlotte Hornets" are identified, as they are the only entries that strictly satisfy the ending pattern defined by the `"*ets"` search term. This demonstrates how minor adjustments to wildcard placement can drastically alter the scope and precision of pattern matching, resulting in highly effective data segregation.

	A	B	C	D	E	F
1	**Team**					
2	Hornets	Ends in ets				
3	Spurs	Does Not End in ets				
4	Blazers	Does Not End in ets				
5	Jazz	Does Not End in ets				
6	Rockets	Ends in ets				
7	Nets	Ends in ets				
8	Mavs	Does Not End in ets				
9	Heat	Does Not End in ets				
10	Magic	Does Not End in ets				
11						
12						
13						
14						
15						
16						
17						
18						

Example 3: Validating Numerical Content Using the # Wildcard

When dealing with alphanumeric identifiers, product codes, or addresses, it is often necessary to confirm the presence and position of numerical data. The hash symbol (#) in **VBA** is tailored precisely for this purpose, as it matches any single numeric digit (0-9). This makes the # wildcard exceptionally useful for validating specific numerical structures or confirming that records contain at least one digit embedded within the text.

Let's consider a dataset in Column A containing various alphanumeric identifiers, potentially representing product IDs or location codes. Our goal is simply to flag every item that contains at least one digit. The dataset is presented below:



	A	B	C	D	E
1	**Strings**				
2	Hey there				
3	I own 5 dogs				
4	I love fish				
5	I have 12 cats				
6	What a great day				
7	Have fun everyone				
8	There are 50 states				
9	This is a party				
10	Hi there				
11					
12					
13					
14					
15					
16					
17					
18					

To quickly determine which of these items contain numbers, we use the pattern `"*#*"`. This pattern is structured logically: the first ***** allows for any preceding characters; the # enforces the match of a single digit; and the second ***** allows for any subsequent characters. This combination instructs **VBA** to perform a search for the presence of any digit, regardless of its position. The following **macro** implements this highly efficient validation logic across the dataset:

```
Sub FindNumbers()

Dim i As Integer

For i = 2 To 10
If Range("A" & i) Like "*#*" Then
Range("B" & i) = "Contains Numbers"
Else
Range("B" & i) = "Does Not Contain Numbers"
End If
Next i

End Sub
```

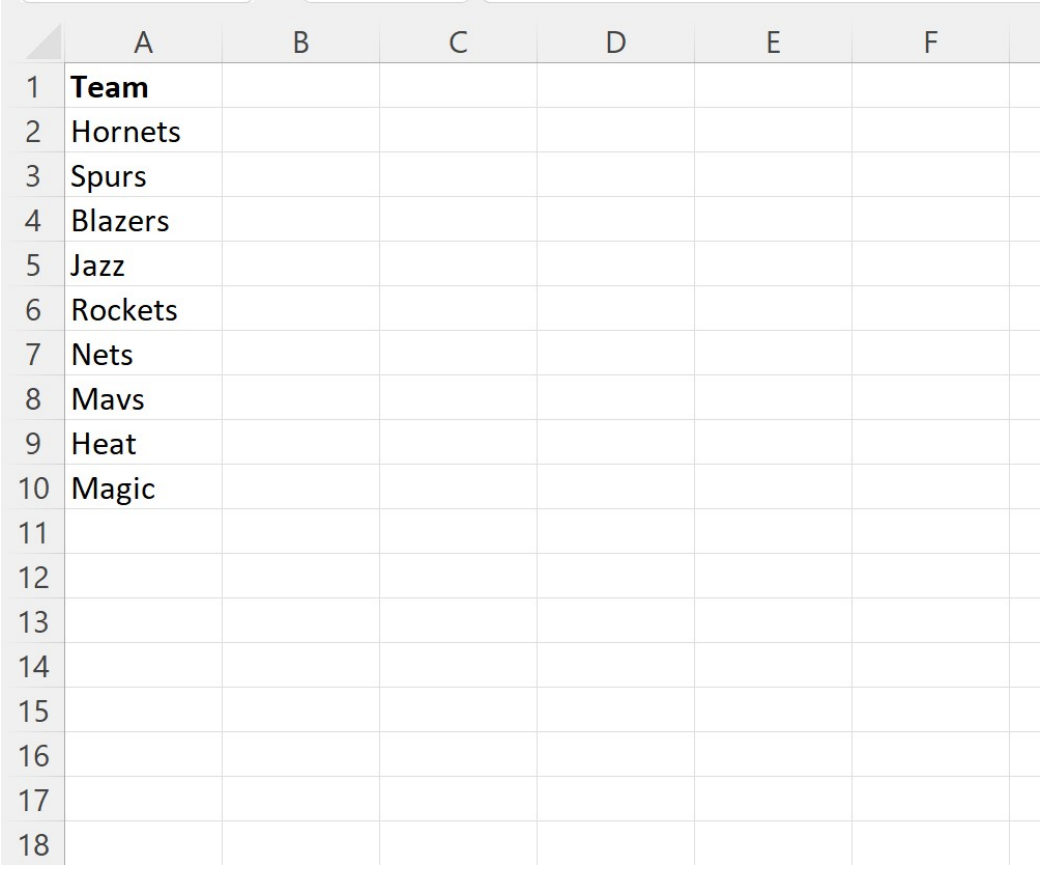
The resulting output confirms that the # wildcard correctly identified all entries containing numerical digits, such as "Item 302," "4th Street," and "ID 9001." This technique is significantly more robust and scalable than trying to manually parse characters or using string functions like `IsNumeric` on the entire cell content, especially when dealing with complex, mixed-format data entries common in large datasets.

	A	B	C	D	E
1	**Strings**				
2	Hey there	Does Not Contain Numbers			
3	I own 5 dogs	Contains Numbers			
4	I love fish	Does Not Contain Numbers			
5	I have 12 cats	Contains Numbers			
6	What a great day	Does Not Contain Numbers			
7	Have fun everyone	Does Not Contain Numbers			
8	There are 50 states	Contains Numbers			
9	This is a party	Does Not Contain Numbers			
10	Hi there	Does Not Contain Numbers			
11					
12					
13					
14					
15					
16					
17					
18					

Example 4: Searching for Character Sets and Ranges using Wildcards

For highly controlled and nuanced pattern matching, the square bracket characters () provide the mechanism to define a specific set or range of characters that must match a single position within a string. This is particularly useful when you need to match one character out of a small, known group, offering a concise alternative to complex branching logic (e.g., numerous `If...Or` statements).

Let's reuse our list of basketball team names from Column A. Suppose we are tasked with identifying any team name that contains the letter 'r', 's', or 't'.



	A	B	C	D	E	F
1	**Team**					
2	Hornets					
3	Spurs					
4	Blazers					
5	Jazz					
6	Rockets					
7	Nets					
8	Mavs					
9	Heat					
10	Magic					
11						
12						
13						
14						
15						
16						
17						
18						

Instead of listing each letter separately, we define a character range within the square brackets: `"[r-t]"`. This range notation specifies that the character in that position must be an 'r', 's', or 't'. By combining this precise range check with the ***** wildcards on either side, the resulting pattern `"*[r-t]*"` searches for the presence of any character in that defined range anywhere within the string. This single pattern encapsulates a complex logical search. The subsequent **macro** executes this targeted and efficient search:

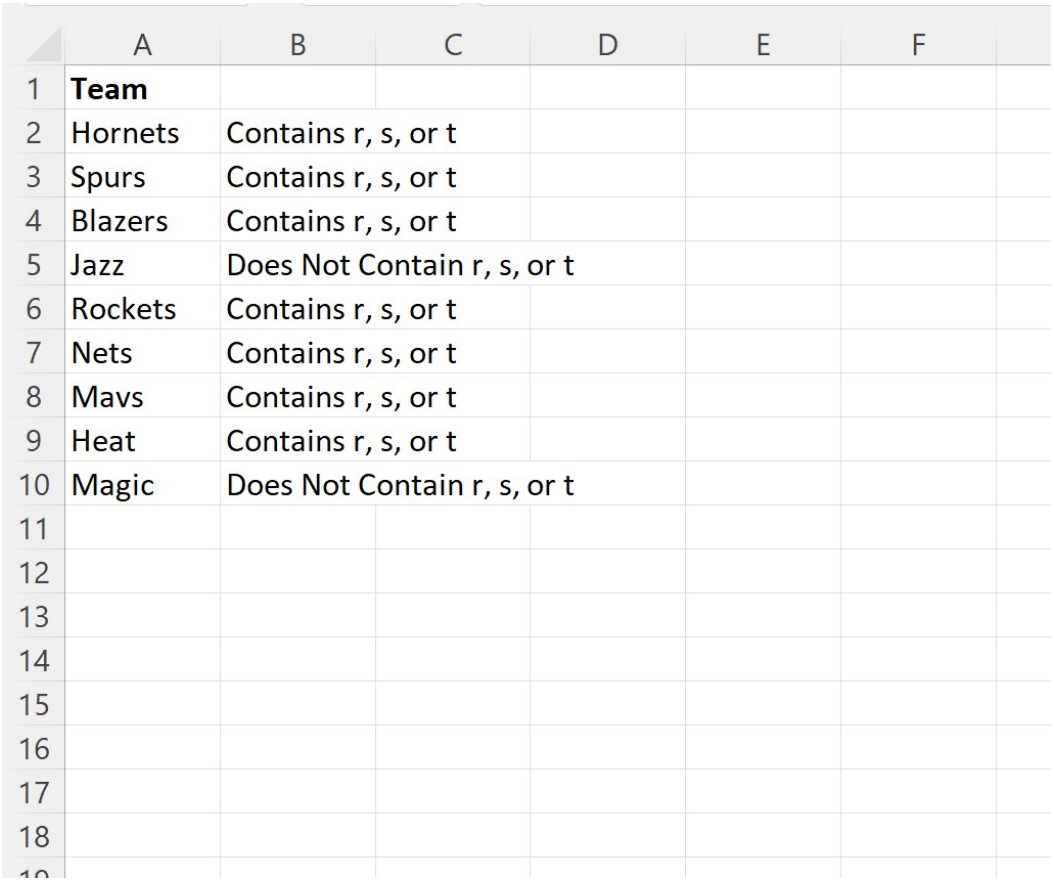
Sub FindSpecificLetters()

Dim i As Integer

For i = 2 To 10
If Range("A" & i) Like "*[r-t]*" Then
Range("B" & i) = "Contains r, s, or t"
Else
Range("B" & i) = "Does Not Contain r, s or t"
End If
Next i

End Sub

The execution results confirm the efficiency of using character ranges. Teams like the "Rockets," "Warriors," "Timberwolves," and "Hornets" are correctly identified because they contain at least one instance of 'r', 's', or 't'. The bracket wildcard approach significantly streamlines complex logical checks, making **VBA** pattern matching adaptable to sophisticated categorization and validation tasks requiring high precision.



	A	B	C	D	E	F
1	**Team**					
2	Hornets	Contains r, s, or t				
3	Spurs	Contains r, s, or t				
4	Blazers	Contains r, s, or t				
5	Jazz	Does Not Contain r, s, or t				
6	Rockets	Contains r, s, or t				
7	Nets	Contains r, s, or t				
8	Mavs	Contains r, s, or t				
9	Heat	Contains r, s, or t				
10	Magic	Does Not Contain r, s, or t				
11						
12						
13						
14						
15						
16						
17						
18						

Conclusion: Mastering Advanced String Manipulation in VBA

The ability to effectively use **wildcard characters** in conjunction with the **Like operator** is not just an optional feature; it is a fundamental skill for any serious **VBA** developer. These specialized symbols--the inclusive *****, the single-character **?**, the numeric-only #, and the precise range definition --allow for pattern matching that is both highly precise and incredibly flexible. By integrating these tools, programmers can drastically reduce the complexity and length of code required for typical string manipulation and data filtering challenges, leading to faster execution and more maintainable solutions.

While the examples provided cover the most common use cases, the **VBA** pattern matching engine also handles specialized scenarios, such as matching literal wildcard characters (by enclosing them in brackets) or dealing with international character sets. For those tackling

advanced text processing, or seeking clarification on edge cases, consulting the official documentation is highly recommended to ensure compliance and optimal performance.

Note: The complete reference guide for the **VBA** wildcard characters and the **Like operator** is available on the Microsoft documentation website, providing definitive information on behavior and syntax rules.

Additional VBA Resources

To further enhance your automation skills, the following tutorials cover other common and essential tasks achievable through **VBA** programming: