How to Easily Use Wildcards for VBA String Searches

Authored by

stats writer

November 20, 2025

In the realm of VBA (Visual Basic for Applications), the ability to manipulate and search text efficiently is paramount for creating robust automation solutions. Wildcards represent a critical feature, acting as placeholders that allow you to match variable characters or sequences of characters within a given text data set. This powerful functionality moves beyond simple exact matching, enabling sophisticated pattern-based searching across large volumes of information.

The core mechanism for leveraging wildcards in VBA is the Like operator. Unlike the standard equality operator (=), the Like operator is specifically designed for flexible string comparison, evaluating whether a given string matches a specified pattern that may include these special placeholder characters. Mastering this operator, combined with an understanding of VBA's built-in wildcards, significantly enhances your capability to perform data validation, filtering, and conditional logic within your automation scripts.

While many programming environments offer complex regular expressions, VBA uses a simpler, highly effective set of four primary wildcards. The two most commonly used are the asterisk (*****) and the question mark (**?**). The asterisk acts as a multi-character placeholder, matching zero or more characters, making it ideal for finding substrings anywhere within a longer string. Conversely, the question mark is used for a single-character match, providing precision when the length of the unknown part of the pattern is fixed. This dual approach offers great flexibility, whether you need to match arbitrary text length or ensure a specific structure is followed.

Essential VBA Wildcard Characters Defined

To effectively utilize pattern matching in VBA, it is essential to understand the four primary wildcards provided by the language, which are used exclusively with the **Like** operator. These characters allow developers to define precise or broad search criteria, facilitating everything from data scrubbing to complex data retrieval operations. The following list details the function of each specialized character:

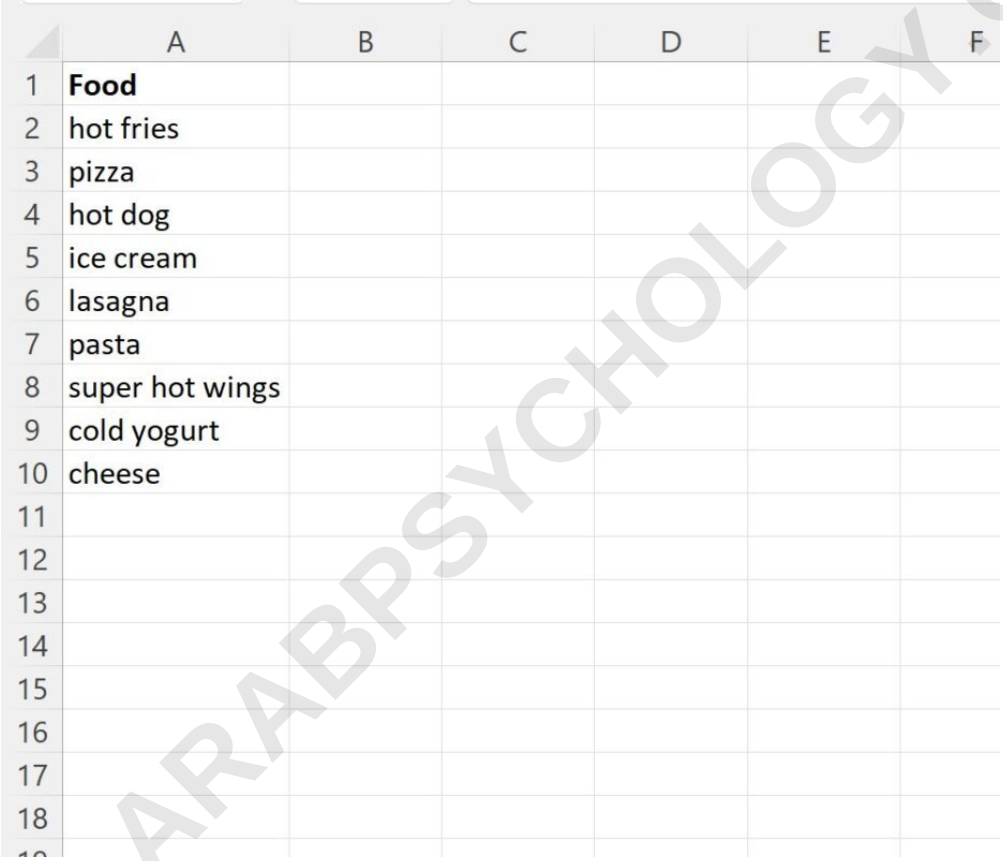
***** : Matches any sequence of characters, including zero characters. This is the most versatile wildcard.
? : Matches exactly one character. This is ideal for fixed-length pattern searching.
: Matches exactly one single numeric digit (0 through 9). Useful for identifying numerical components within alphanumeric strings.
: Matches any single character enclosed within the brackets, or any single character within a specified range. For example, matches A, B, or C.

Understanding the interplay between these characters and the **Like** operator is fundamental to successful VBA programming. The subsequent sections will provide practical, concrete examples demonstrating how to deploy each of these wildcards to solve common data challenges, focusing specifically on their application within an Excel environment using VBA macros.

Detailed Example 1: Utilizing the Asterisk (*) for Substring Matching

The asterisk (*****) wildcard is arguably the most frequently used placeholder in VBA pattern matching. Its function is to represent any sequence of zero or more characters, making it highly effective for checking if a specific sequence of characters, or **substring**, exists anywhere within a longer string. When you bracket a search term with asterisks, such as `*term*`, you are instructing the Like operator to ignore everything before and after that term, focusing solely on its presence.

Consider a scenario where we have a list of food items in Column A of an Excel worksheet. Our goal is to identify which of these items contains the sequence of letters "hot," regardless of whether it is at the beginning, middle, or end of the word. The initial dataset looks like this:



	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 automate this search, we will construct a macro that iterates through the cells A2 through A10. Inside the loop, we use the **Like** operator with the pattern `"*hot*"`. The Like operator returns **True** if the cell value matches the pattern, allowing us to conditionally write the result to the corresponding cell in Column B. This demonstrates a robust method for quick data categorization based on content.

The VBA code snippet below illustrates how this iteration and comparison process is executed. Notice the use of the `For...Next` loop combined with the `If...Then...Else` structure to evaluate

the pattern matching for each row:

```
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
```

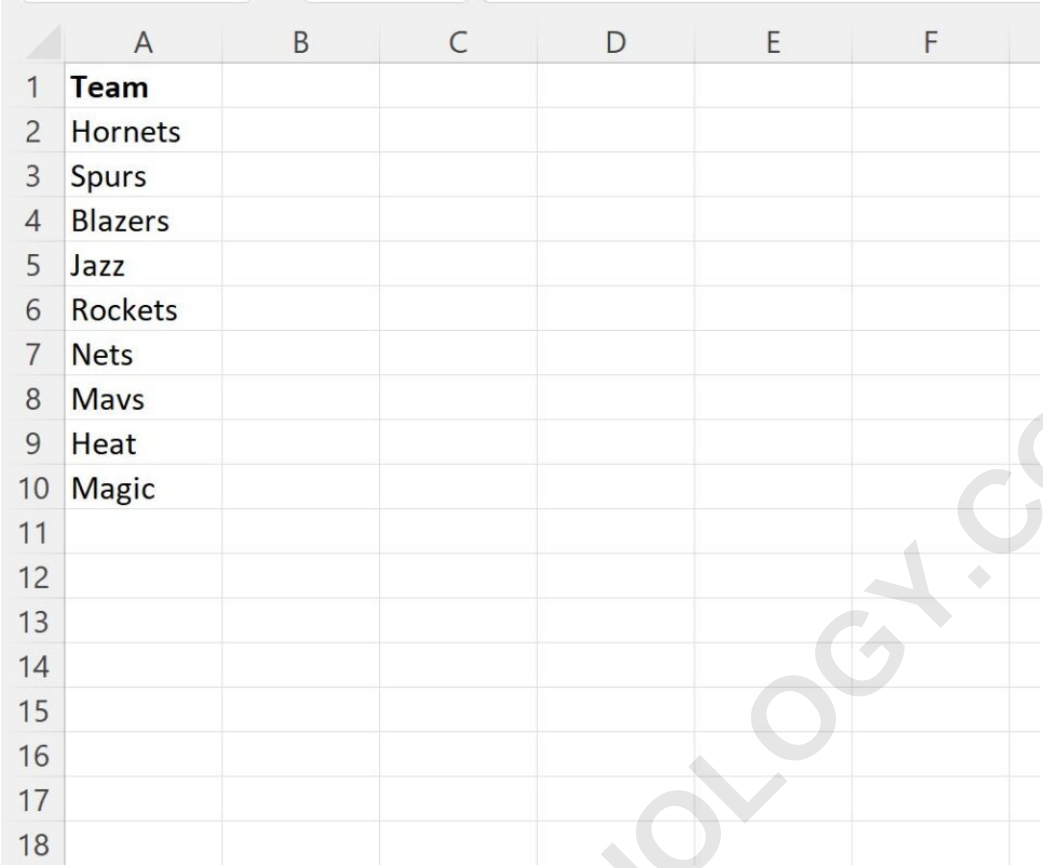
Upon execution of the `FindString` macro, Column B is populated with the validation results. Items like "Hot dog" and "Chili hot pot" return a positive match because the wildcards successfully accommodate the surrounding characters. This confirms the powerful capability of the double asterisk pattern to perform non-positional substring searches within complex datasets, significantly improving data processing efficiency:

	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					

Detailed Example 2: Using the Asterisk (*) to Identify Specific String Endings

While the previous example used the asterisk (*****) to find a substring anywhere, positioning the wildcard character strategically allows us to enforce constraints on the beginning or end of a string. When the asterisk is placed at the beginning of the pattern (e.g., `*suffix`), it matches any characters preceding the specified suffix, effectively searching for strings that terminate with that sequence.

Imagine we are analyzing a list of basketball team names and need to quickly isolate those names that end precisely with the letters "ets." Using the list provided in Column A:



	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						

Our task is to build a macro that evaluates each team name. By utilizing the pattern `"*ets"`, the Like operator ensures that only those strings where "ets" immediately follows zero or more preceding characters will be flagged as a match. This technique is invaluable for parsing standardized naming conventions or identifying records based on common suffixes.

The VBA code for `FindEndingString` implements this pattern search across the specified range. It iterates from row 2 to row 10, applying the conditional logic to check for the terminating pattern. Note how simple yet powerful the expression `Range("A" & i) Like "*ets"` is in achieving this specific positional search:

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
```

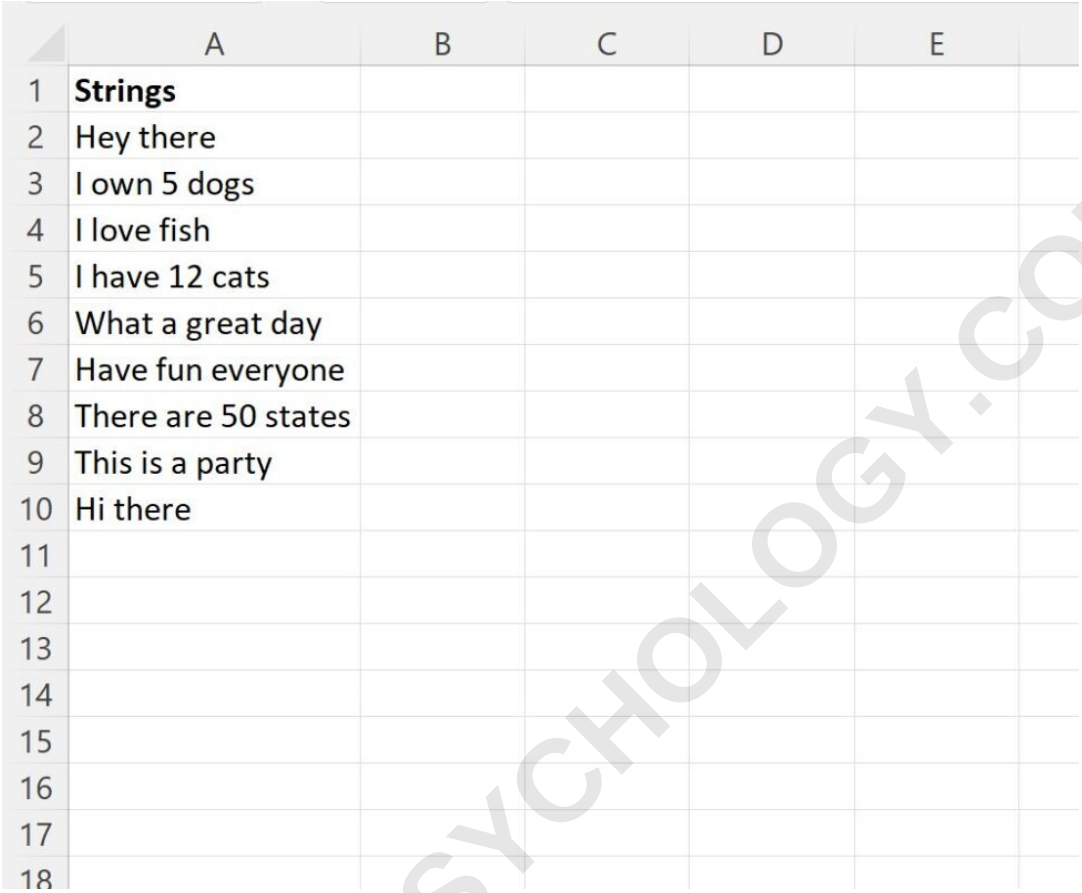
The resulting output clearly shows that only the cells containing "Nets" and "Jets" match the specified ending pattern. This confirms the efficacy of the leading asterisk (*****) in confining the search criteria to the termination point of the string. If we had used `"ets*"` instead, we would be searching for strings that *begin* with "ets," demonstrating the importance of wildcard placement:

	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						

Detailed Example 3: Employing the Pound Sign (#) to Search for Numeric Digits

When dealing with alphanumeric data, the need often arises to specifically identify patterns that contain or consist of numeric digits. The pound sign (#) wildcard in VBA serves this exact purpose: it matches any single numeric character from 0 to 9. Unlike the question mark (**?**), which matches any single character (letter, number, or symbol), the hash sign restricts the match solely to a digit, offering a higher degree of precision when validating structured data formats like serial numbers or identifiers.

To demonstrate this, consider a list in Column A containing various strings, some purely textual, and others containing embedded numbers, such as product codes or version numbers. Our goal is to flag every entry that includes at least one numeric digit:



	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					

By defining the pattern as `"*#*"`, we combine the multi-character wildcard (*****) with the specific numeric digit placeholder (#). The leading and trailing asterisks ensure that we capture the presence of any digit, regardless of its position within the string. This is a crucial technique for isolating data records that do not adhere to purely alphabetical conventions or for segregating quantitative data.

The `FindNumbers` macro structure is similar to previous examples, iterating through the dataset and using the Like operator with the specific numeric pattern. If a cell contains a digit, the condition is met, and the appropriate label is written to Column B:

```
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
```

Running the macro yields the following classification in Column B. Notice how entries like "Item 45B" and "VBA2023" are successfully identified because they contain numeric digits, while purely alphabetical entries like "Apple" and "Banana" are correctly categorized as not containing numbers:

	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					

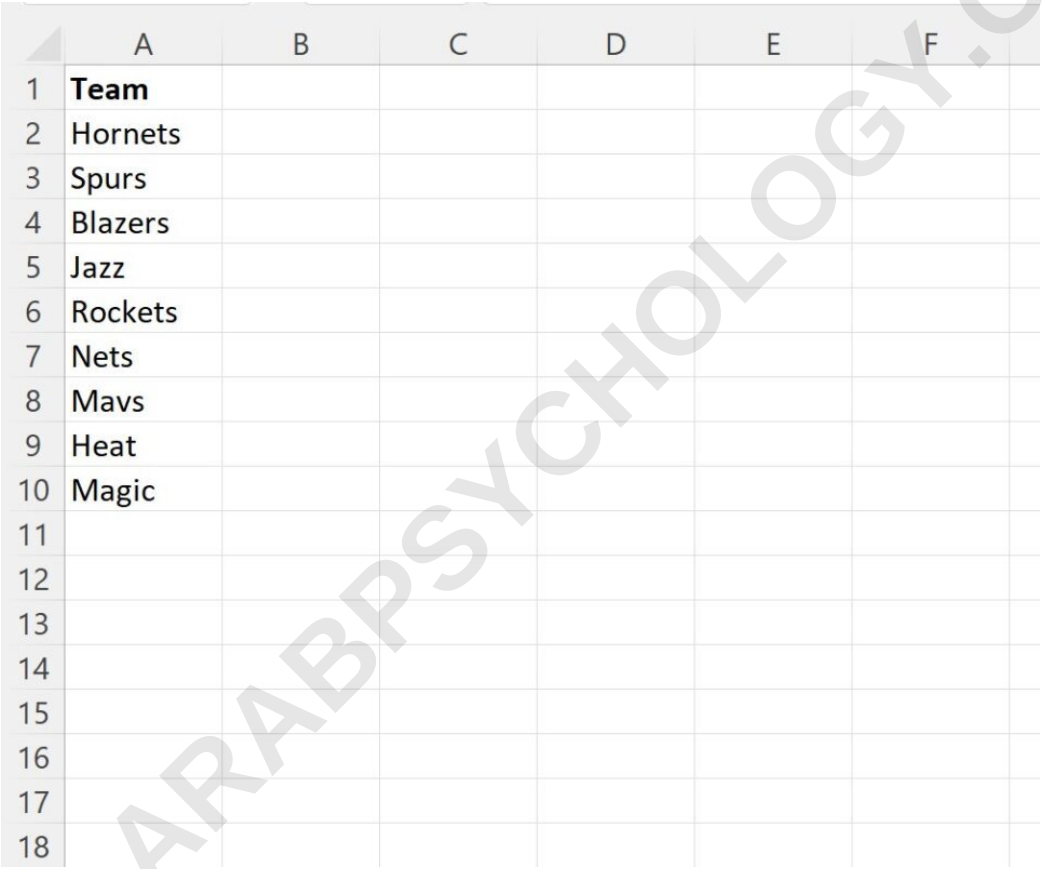
Detailed Example 4: Implementing Bracketed Sets () for Character Range Selection

The bracketed character list () provides a sophisticated way to match a single character that

belongs to a specific set or range. This wildcard is essential when you need precise control over which characters are allowed in a particular position within a pattern, allowing you to validate against groups of letters or numbers simultaneously. You can specify individual characters (e.g.,) or use a hyphen to define a range (e.g., for any uppercase letter).

Let's reuse the list of basketball team names from Example 2. This time, our requirement is to identify all teams whose name contains any of the letters 'r', 's', or 't'. If we were to use traditional VBA string functions, this would require three separate comparisons joined by `Or` statements, leading to complex, less readable code. The bracket wildcard simplifies this dramatically.

Here is the initial 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						

We define the search pattern as `"**"`. The surrounding asterisks (**) ensure the match can occur anywhere in the string, while defines the set of acceptable characters: 'r', 's', or 't'. This ability to specify a range within brackets (rather than listing each character individually) makes the code concise and highly efficient for filtering based on character classes.

The `FindSpecificLetters` macro implements this range search. Note that the character matching in VBA is generally case-insensitive unless the `Option Compare Binary` statement is used at the module level. For standard operations, would match 'r', 's', 't', 'R', 'S', or 'T', providing

comprehensive coverage:

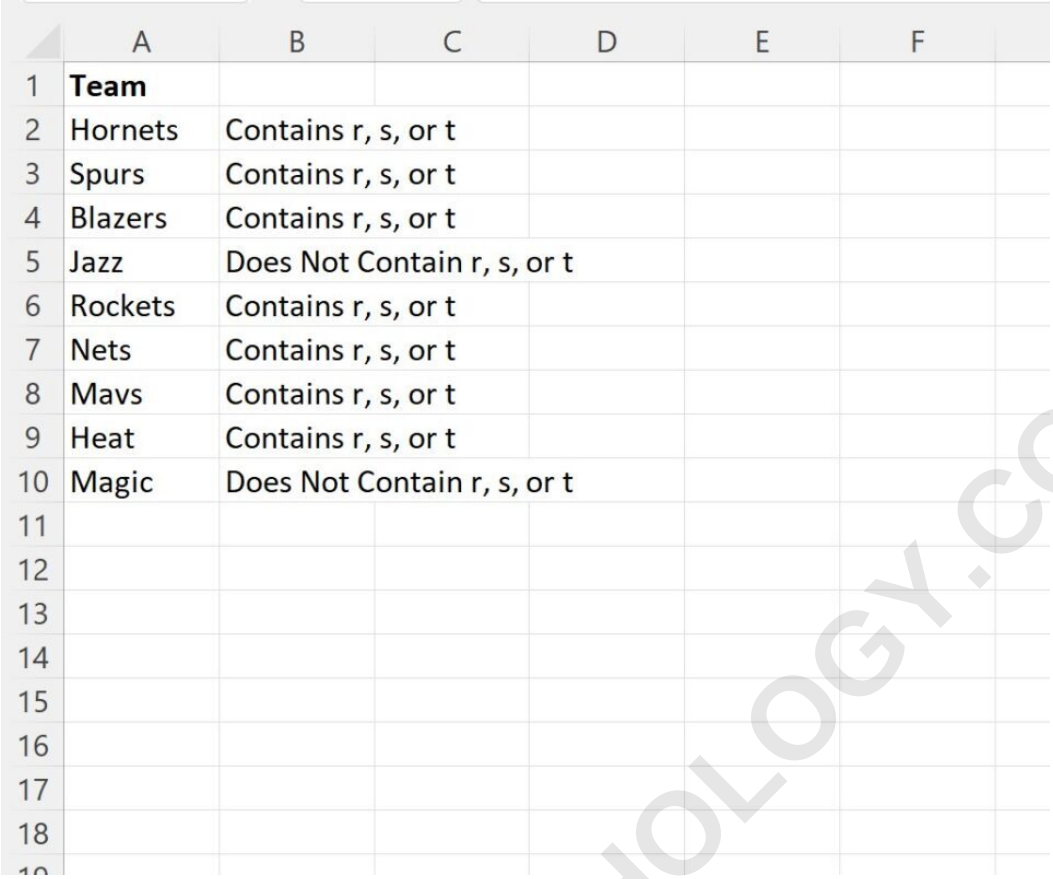
```
Sub FindSpecificLetters()

Dim i As Integer

For i = 2 To 10
If Range("A" & i) Like "**" 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 final output confirms that teams containing 'r', 's', or 't' (such as "Suns," "Clippers," and "Rockets") are correctly identified, while those like "Nets" (which contains an 'e' but not 'r', 's', or 't' after the initial 'N') are excluded. This powerful range matching capability is crucial for advanced data filtering tasks.



	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						

Summary of VBA Pattern Matching Techniques

The four examples provided demonstrate the fundamental flexibility achieved when combining the Like operator with wildcards in VBA. By intelligently positioning the asterisk (*****) to denote arbitrary sequences, the question mark (**?**) for fixed-length gaps, the pound sign (#) for numeric certainty, and brackets () for character sets, developers gain precise control over text validation and search operations. These tools eliminate the need for cumbersome sequential checks (e.g., using `Instr` or multiple `Mid` functions) and make the code cleaner and easier to maintain.

It is important to remember that these wildcard characters are only active within the context of the **Like** operator. If you were to use these characters in a standard equality comparison (=), they would be treated as literal characters. furthermore, when dealing with actual wildcard characters that you wish to match literally (e.g., searching for a string that explicitly contains an asterisk), you must enclose the wildcard in brackets, such as , to escape its special meaning. This crucial distinction ensures that you can always search for the special characters themselves when necessary.

Advanced implementations often involve combining these wildcards. For instance, a pattern like `"ID-??##*"` could be used to validate an identifier that must start with "ID-", followed by two

arbitrary characters, two numeric digits, and then any trailing text. This level of granular control is why the Like operator remains indispensable for data processing within Excel and other Microsoft Office applications powered by VBA.

VBA Resources and Next Steps

The power of VBA lies in its accessibility and specific functionality tailored for Office applications. Utilizing wildcards provides a significant boost to your data manipulation toolkit, allowing you to build highly dynamic and responsive macros capable of handling variable data formats.

Note: For the most comprehensive details and any behavioral nuances concerning the VBA Like operator and its supported wildcard characters, always consult the official Microsoft documentation.

Further VBA Tutorials and Learning

The following resources and tutorials explain how to perform other common tasks using VBA, helping you further master automation and data processing within Excel and related platforms: