EXCEL – STRING COMPARISON FUNCTION IN VBA

Types of string comparisons in VBA

- **Binary String Comparison (Case sensitive) in VBA**

For any formula **If** you want to compare two string in such a manner that each individual characters is compared with its counterpart in a case sensitive manner (Ex. "**This**" is not equal to "**this**" because **'T'** is not equal to **'t'**), you can do either of the two things:

1. Declare the statement **Option Compare Binary** at the very onset of **VBA** code or
2. Specify the comparison to be made as **Binary** in the formula (***as an argument***)

The character byte order that would be followed in such case would be:

A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø

In case you were wondering why **A** has a lesser value than **a**, it is so because **'A'** is represented by a lower byte than **'a'** and occurs earlier in the list when starting from **0**. The ***binary comparison*** is the one that **VBA** would use if nothing else has been explicitly specified.

- **Text Comparison in VBA (Case insensitive comparison)**

For any ***formula*** if you want to compare two string in such a manner that each individual characters is compared with its counterpart in a case insensitive manner (Ex. "**THis**" is would be equal to "**this**" even though some of the characters don't have the same case as their counterparts in the other string), you can do either of the two things:

1. Declare the statement **Option Compare Text** at the very onset of **VBA** code or
2. Specify the comparison to be made as **Text** in the formula (***as an argument***)

The order that would be followed in this case would be:

(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)

As you can see, **'A'** is equal to **'a'** in this case. This helps in comparing strings when they have the same sequence of characters but each may have a case different from its counterpart.

VBA String Operators

- **Wildcards * ? and #**

There are three types of wildcards that can be used in **VBA**. These include:

- **?** – Indicates any single character. This has to be repeated as many times as the number of characters that you would like to enter as wildcard. (Ex. "**?ip**" can be any of the following – "**Tip**", "**Hip**" or "**Lip**")
- ***** – Indicates zero or more characters in a sequence. You can use this is the number of wildcard characters is not certain. (Ex – "***ike**" can equate to any of the following "**Like**", "**Mike**", "**Hike**" or "**Strike**")
- **#** – Indicates a single digit from 0 to 9. Again, has to be repeated as many times as there are digits that you would like to use as wildcards.

As we have seen above, the wildcards allows partial sub-strings to matched to strings while providing some amount of flexibility. However there is a constraint – when you specify *, # or ?, you can not specify which of the characters you would like to exclude from the list of characters that generate a positive match.

For example "**?one**" will generate a match for both "**Bone**" and "**Tone**". However if you wanted it to generate a match only for alphabets A, B and C and not for any other character, this method will not suffice. However, there is a way – string patterns. Read on.

String Patterns

VBA provides the option to specify and narrow down the a list of values that can generate a positive match when a compared with a wildcard. The same mechanism can also be used to exclude some other characters from being matched. In the previous example, suppose we wanted to use only A,B and C as the matching characters and not others we could have written "***[A-C]one**" (which essentially means "(A or B or C)+one").

By enclosing the list in brackets [], we ensure that only those characters are used to return a positive match. You hyphen (-) specifies that all the characters lying between the staring and the end character can be used. You can also specify a custom list within the parenthesis such as **[a,b,c,t,r]** etc.

For example, both the below expressions would both evaluate to **TRUE**:

"this" Like "*[a-z]his"
"this" Like "*[s,t,u]his"

However the one shown below would evaluate to **FALSE**:

"this" Like "*[x,y,x]his"

Now suppose rather than including a list of characters, what **If** you wanted to exclude a certain set? Well you can specify an exclusion set by using the ! symbol within the bracket [!].

For example, you can write ***[!a-c]** to exclude the characters a,b and c from being used as replacements in the wildcard.

So for example the following would evaluate to **FALSE**.

"this" Like "*[!s-u]his"

String comparison and search functions in VBA

- **Comparing strings in VBA using =, > and < operators**

The simplest way to compare two strings is use =, > and < operators. By default VBA will use the binary comparison (read about binary and text comparisons above). However, if you want to do a case insensitive comparison, you will need to explicitly set the comparison option to Text as shown in the example below.

If the option is not set (or is set to **Option Compare Binary**), the following code will show **FALSE** because the binary equivalent of "**Text**" is not the same as "**this**".

```
Sub match_strings()
MsgBox "This" = "this"
End Sub
```

However then the binary comparison has been explicitly set to **Text**, the code shows **TRUE**

```
Option Compare Text
Sub match_strings()
MsgBox "This" = "this"
End Sub
```

- **String Comparison in VBA using INSTR**

INSTR VBA function can be used when you would like to find the position of a ***sub-string*** within another string. The format of the **INSTR** function is:

INSTR("position_to_start", "string_to_search", "string_to_find","comparison_type")

Where:

- ***position_to_start*** = The position where you would like to function to begin the comparison (optional)
- ***string_to_search*** = The string which is being searched
- ***string_to_find*** = The string or ***sub-string*** which needs to be located in the above string
- ***comparison_type*** = Specifies whether the comparison should be carried out as a binary (**byte level**, case sensitive) or text (byte level, case insensitive) (optional)

The function returns the position of the ***string_to_find*** within the ***string_to_search*** starting from the first position. If it is not able to locate the ***sub-string***, it will return **0**. If any of the strings is **NULL**, it returns a **NULL**.

If the string_to_find is a zero length (say “”), the function would return the ***position_to_start*** numerical value.

Example:

INSTR(5, “This is a string”, “string”)
would give the result as 11 (the given sub-string starts at position 11).

INSTR(5, “This is a string”, “String”)
would give the result as 0 (the given substring could not be located).

INSTR(5, “This is a string”, “”)
would give the result as 5 (the given substring is of 0 length and hence return the number specified as the position_to_start).

INSTR(5, “This is a string”, “”)
would give the result as 5 (the given substring is of 0 length and hence return the number specified as the position_to_start).

- **String Comparison in VBA using LIKE**

Jon Walkenbach highlighted this function in one of his blog posts that find the existence of a word within another. The code was originally written by **Rick Rothstein** (MVP – VB). The beauty of this function is that’s its just one line of code (oftentimes you will see coders (including me) write code that runs into several lines to achieve similar results. You may want to take a minute to study the function (to be honest, it took me a while to get the entire thing right).

You may also want to make a few changes to see how the **LIKE** function, when used with operators and string patterns, actually works.

```
Function ExactWordInString(Text As String, Word As String) As Boolean

ExactWordInString = " " & UCase(Text) & " " Like "*[!A-Z]" & UCase(Word) & "[!A-Z]*"

End Function
```

The spaces before and after the Text string are provided so that the searched for word can be identified when it is the first or last word in the text string. The first word would not have a character in front of it that the “***[!A-Z]**” part of the Like pattern string could match.

- **Comparing strings in VBA using strComp**

StrComp VBA function can be used when you would like to compare two strings and return a value indicating whether both are identical or which one is *greater/smaller* than the other. The syntax of the **VBA** StrComp function is:

StrComp(“first_string_to_compare”, “second_string_to_compare”,*“comparison_type”***)**

Where:

- ***first_string_to_compare*** = The first string that you would like to compare.
- ***second_string_to_compare*** = The second string that you would like to compare.
- ***string_to_find*** = The string or sub-string which needs to be located in the above string
- ***comparison_type*** = Specifies whether the comparison should be carried out as a binary (byte level, case sensitive) or text (byte level, case insensitive) (optional)

Please bear in mind that if the comparison type is not explicitly specified within the function, the value specified in the Option Compare Statement at the module level will be used.

The function returns:

- -1 when ***second_string_to_compare*** is greater than ***first_string_to_compare***
- 0 when ***second_string_to_compare*** is equal to the ***first_string_to_compare***
- 1 when ***second_string_to_compare*** is less than ***first_string_to_compare***
- Null when either of the strings is **NULL**

For example the following function would show -1 if no comparison_type is explicitly specified.

```
Sub match_strings()
MsgBox StrComp("This", "this")
End Sub
```

However when we explicitly specify the comparison type to be text, the function would show 0.

```
Sub match_strings()
MsgBox StrComp("This", "this", 1) 'Compare as text while ignoring case
End Sub
```

A Few Generic String operations in VBA

Define a string in VBA
Dim myStr as String

Assign value to a string in VBA
myStr = “This is a string”

Define an Array of strings in VBA
Dim myStr() as String

Convert to string
Cstr(Expresion) where expression is a string, a string literal, a string constant, a string variable or a string Variant

Converting strings to uppercase
UCase(myStr)

Converting strings to lowercase
LCase(myStr)