Comparison and Conditional functions

The following list contains the functions that you can use to compare values or specify conditional statements.

For information about using string and numeric fields in functions, and nesting functions, see Evaluation functions.

For information about Boolean operators, such as AND and OR, see Boolean operators.

case(X,"Y",...)

Description

Accepts alternating conditions and values. Returns the first value for which the condition evaluates to TRUE.

This function takes pairs of arguments X and Y. The X arguments are Boolean expressions that are evaluated from first to last. When the first X expression is encountered that evaluates to TRUE, the corresponding Y argument is returned. The function defaults to NULL if none are true.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic example

This example uses the sample data from the Search Tutorial, but should work with any format of Apache Web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

The following example returns descriptions for the corresponding http status code.

```
sourcetype=access_* | eval description=case(status == 200, "OK", status ==404, "Not found", status == 500, "Internal Server Error") | table status description
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>408</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>404</td>
<td>Not found</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>406</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
</tbody>
</table>
For an example of how to display a default value when that status does not match one of the values specified, see the True function.

Extended example

This example shows you how to use the case function in two different ways, to create categories and to create a custom sort order.

This example uses recent earthquake data downloaded from the USGS Earthquakes website. The data is a comma separated ASCII text file that contains magnitude (mag), coordinates (latitude, longitude), region (place), and so forth, for each earthquake recorded.

You can download a current CSV file from the USGS Earthquake Feeds and upload the file to your Splunk instance if you want follow along with this example.

You want classify earthquakes based on depth. Shallow-focus earthquakes occur at depths less than 70 km. Mid-focus earthquakes occur at depths between 70 and 300 km. Deep-focus earthquakes occur at depths greater than 300 km. We'll use Low, Mid, and Deep for the category names.

source=all_month.csv | eval Description=case(depth<=70, "Low", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description

The eval command is used to create a field called Description, which takes the value of "Low", "Mid", or "Deep" based on the Depth of the earthquake. The case() function is used to specify which ranges of the depth fits each description. For example, if the depth is less than 70 km, the earthquake is characterized as a shallow-focus quake; and the resulting Description is Low.

The search also pipes the results of the eval Command into the stats Command to count the number of earthquakes and display the minimum and maximum magnitudes for each Description.

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Description</th>
<th>count</th>
<th>min(Mag)</th>
<th>max(Mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep</td>
<td>35</td>
<td>4.1</td>
<td>6.7</td>
</tr>
<tr>
<td>Low</td>
<td>6236</td>
<td>-0.60</td>
<td>7.70</td>
</tr>
<tr>
<td>Mid</td>
<td>635</td>
<td>0.8</td>
<td>6.3</td>
</tr>
</tbody>
</table>

You can sort the results in the Description column by clicking the sort icon in Splunk Web. However in this example the order would be alphabetical returning results in Deep, Low, Mid or Mid, Low, Deep order.

You can also use the case function to sort the results in a custom order, such as Low, Mid, Deep. You create the custom sort order by giving the values a numerical ranking and then sorting based on that ranking.

source=all_month.csv | eval Description=case(depth<=70, "Low", depth>70 AND depth<=300, "Mid", depth>300, "Deep") | stats count min(mag) max(mag) by Description | eval sort_field=case(Description="Low", 1, Description="Mid", 2, Description="Deep",3) | sort sort_field

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>Description</th>
<th>count</th>
<th>min(Mag)</th>
<th>max(Mag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>6236</td>
<td>-0.60</td>
<td>7.70</td>
</tr>
</tbody>
</table>

2
cidrmatch("X", Y)

**Description**

Returns TRUE or FALSE based on whether an IP address matches a CIDR notation.

Use this function to determine if an IP address belongs to a particular subnet. This function returns TRUE, when IP address Y belongs to a particular subnet X. Both X and Y are string arguments. X is the CIDR subnet. Y is the IP address to match with the subnet. This function is compatible with IPv6.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example uses the `cidrmatch` and `if` functions to set a field, `isLocal`, to "local" if the field `ip` matches the subnet. If the `ip` field does not match the subnet, the `isLocal` field is set to "not local".

```plaintext
... | eval isLocal=if(cidrmatch("123.132.32.0/25", ip), "local", "not local")
```

The following example uses the `cidrmatch` function as a filter to remove events that do not match the `ip` address:

```plaintext
... | where cidrmatch("123.132.32.0/25", ip)
```

coalesce(X,...)

**Description**

This function takes an arbitrary number of arguments and returns the first value that is not NULL.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

You have a set of events where the IP address is extracted to either `clientip` or `ipaddress`. This example defines a new field called `ip`, that takes the value of either the `clientip` field or `ipaddress` field, depending on which field is not NULL (does not exist in that event). If both the `clientip` and `ipaddress` field exist in the event, this function returns the first argument, the `clientip` field.

```plaintext
... | eval ip=coalesce(clientip,ipaddress)
```
false()

Description

Use this function to return FALSE.

This function enables you to specify a conditional that is obviously false, for example $1==0$. You do not specify a field with this function.

Usage

This function is often used as an argument with other functions.

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

Basic examples

if(X,Y,Z)

Description

If the condition X evaluates to TRUE, returns Y, otherwise returns Z.

This function takes three arguments. The first argument X must be a Boolean expression. If X evaluates to TRUE, the result is the second argument Y. If X evaluates to FALSE, the result evaluates to the third argument Z.

Usage

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

The if function is frequently used with other functions. See Basic examples.

Basic examples

The following example looks at the values of the field error. If error=200, the function returns err=OK. Otherwise the function returns err=Error.

... | eval err=if(error == 200, "OK", "Error")

The following example uses the cidrmatch and if functions to set a field, isLocal, to "local" if the field ip matches the subnet. If the ip field does not match the subnet, the isLocal field is set to "not local".

... | eval isLocal=if(cidrmatch("123.132.32.0/25",ip), "local", "not local")

in(FIELD, VALUE-LIST)

Description

The function returns TRUE if one of the values in the list matches a value in the field you specify.
This function takes a list of comma-separated values.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions with other commands.

The following syntax is supported:

```sh
...| where in(field,"value1","value2", ...)
...| where field in("value1","value2", ...)
...| eval new_field=if(in(field,"value1","value2", ...), "value-if_true","value-if-false")
```

The `eval` command cannot accept a Boolean value. You must specify the IN function inside the IF function, which can accept a Boolean value as input.

The string values must be enclosed in quotation marks. You cannot specify wildcard characters with the values to specify a group of similar values, such as HTTP error codes or CIDR IP address ranges. Use the IN operator instead.

The `IN` operator is similar to the `in` function. You can use the `IN` operator with the `search` and `tstats` commands. You can use wildcard characters in the VALUE-LIST with these commands.

**Basic examples**

The following example uses the `where` command to return `in=TRUE` if one of the values in the `status` field matches one of the values in the list.

```sh
... | where status in("400", "401", "403", "404")
```

The following example uses the `in` function as the first parameter for the `if` function. The evaluation expression returns TRUE if the value in the `status` field matches one of the values in the list.

```sh
... | eval error=if(in(status, "error", "failure", "severe"),"true","false")
```

**Extended example**

The following example combines the `in` function with the `if` function to evaluate the `status` field. The value of `true` is placed in the new field `error` if the `status` field contains one of the values 404, 500, or 503. Then a count is performed of the values in the `error` field.

```sh
... | eval error=if(in(status, "404","500","503"),"true","false") | stats count by error
```

**See also**

Blogs

Smooth operator | Searching for multiple field values

**like(TEXT, PATTERN)**
**Description**

This function returns TRUE if TEXT matches PATTERN.

This function takes two arguments, a string to match TEXT and a string expression to match PATTERN. It returns TRUE if, and only if, TEXT matches PATTERN. The pattern matching supports an exact text match, as well as single and multiple character matches.

- Use the percent ( % ) symbol as a wildcard for multiple characters.
- Use the underscore ( _ ) character for a single character match.

**Usage**

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

**Basic examples**

The following example returns like-TRUE if the field value starts with foo:

```plaintext
... | eval is_a_foo=if(like(field, "foo\%"), "yes a foo", "not a foo")
```

The following example uses the where command to return like-TRUE if the ipaddress field starts with the value 198.. The percent ( % ) symbol is a wildcard with the like function:

```plaintext
... | where like(ipaddress, "198.\")
```

**match(SUBJECT, "REGEX")**

**Description**

This function returns TRUE or FALSE based on whether REGEX matches SUBJECT.

This function compares the regex string REGEX to the value of SUBJECT and returns a Boolean value. It returns TRUE if the REGEX can find a match against any substring of SUBJECT.

**Usage**

The match function is regex based. For example use the backslash ( \ ) character to escape a special character, such as a quotation mark. Use the pipe ( | ) character to specify an OR condition.

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

**Basic examples**

The following example returns TRUE if, and only if, field matches the basic pattern of an IP address. This examples uses the caret ( ^ ) character and the dollar ( $ ) symbol to perform a full match.

```plaintext
... | eval n=if(match(field, "^d{1,3}\./d{1,3}\./d{1,3}\./d{1,3}\.$"), 1, 0)
```
The following example uses the `match` function in an `<eval-expression>`. The SUBJECT is a calculated field called `test`. The "REGEX" is the string `yes`.

This example uses the `match` function in an `<eval-expression>`. The SUBJECT is a calculated field called `test`. The "REGEX" is the string `yes`.

```
... | eval matches = if(match(test,"yes"), 1, 0)
```

If the value is stored with quotation marks, you must use the backslash (\) character to escape the embedded quotation marks. For example:

```
| makeresults | eval test="\"yes\"" | eval matches = if(match(test, "\"yes\""), 1, 0)
```

`null()`

**Description**

This function takes no arguments and returns NULL. The evaluation engine uses NULL to represent "no value". Setting a field value to NULL clears the field value.

**Usage**

NULL values are field values that are missing in a some results but present in another results.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

Suppose you want to calculate the average of the values in a field, but several of the values are zero. If the zeros are placeholders for no value, the zeros will interfere with creating an accurate average. You can use the `null` function to remove the zeros.

**See also**

- You can use the `fillnull` command to replace NULL values with a specified value.
- You can use the `nullif(X,Y)` function to compare two fields and return NULL if X = Y.

`nullif(X,Y)`

**Description**

This function is used to compare fields. The function takes two arguments, X and Y, and returns NULL if X = Y. Otherwise it returns X.

**Usage**

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.
Basic examples

The following example returns NULL if fieldA=fieldB. Otherwise the function returns fieldA.

```plaintext
... | eval n=nullif(fieldA,fieldB)
```

searchmatch(X)

**Description**

Use this function to return TRUE if the search string (X) matches the event.

This function takes one argument X, which is a search string. The function returns TRUE if, and only if, the event matches the search string.

**Usage**

You must use the `searchmatch` function inside an `if` function.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**

The following example uses the `makeresults` command to create some simple results. The `searchmatch` function is used to determine if any of the results match the search string "x=hi y=*".

```plaintext
| makeresults 1 | eval _raw = "x=hi y=bye" | eval x="hi" | eval y="bye" | eval test=if(searchmatch("x=hi y=*"), "yes", "no") | table _raw test x y
```

The result of the `if` function is `yes`; the results match the search string specified with the `searchmatch` function.

true()

**Description**

Use this function to return TRUE.

This function enables you to specify a condition that is obviously true, for example 1==1. You do not specify a field with this function.

**Usage**

This function is often used as an argument with other functions.

You can use this function with the `eval`, `fieldformat`, and `where` commands, and as part of eval expressions.

**Basic examples**
This example uses the sample data from the Search Tutorial, but should work with any format of Apache Web access log. To try this example on your own Splunk instance, you must download the sample data and follow the instructions to get the tutorial data into Splunk. Use the time range Yesterday when you run the search.

The following example shows how to use the true() function to provide a default value to the case function. If the values in the status field are not 200, or 404, the value used is Other.

```
sourcetype=access_* | eval description=case(status==200,"OK", status==404, "Not found", true(), "Other") | table status description
```

The results appear on the Statistics tab and look something like this:

<table>
<thead>
<tr>
<th>status</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>408</td>
<td>Other</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>404</td>
<td>Not found</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>406</td>
<td>Other</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
</tbody>
</table>

**validate(X,Y,...)**

**Description**

Use this function to return the string Y corresponding to the first expression X that evaluates to FALSE. This function is the opposite of the case function.

This function takes pairs of arguments, Boolean expressions X and strings Y. The function returns the string Y corresponding to the first expression X that evaluates to FALSE. This function defaults to NULL if all evaluate to TRUE.

**Usage**

You can use this function with the eval, fieldformat, and where commands, and as part of eval expressions.

**Basic examples**

The following example runs a simple check for valid ports.

```
... | eval n=validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")
```