ARRAYLEN Function

Computes the number of elements in the arrays in the specified column, array literal, or function that returns an array.

- This function calculates the number of elements in the outer layer of an array. If your array is nested, the count of inner elements is not factored.
- If a row contains a missing array, the returned value is `0`. If it contains a value that is not recognized as an array, the returned value is blank.

Basic Usage

Array literal reference example:

```
derive type:single value:ARRAYLEN([A,B,C,D])
```

Output: Generates the count of elements in the array, which is `4` .

Column reference example:

```
derive type:single value:ARRAYLEN([myValues]) as:'length_myValues'
```

Output: Generates the new `length_myValues` column containing the count of elements in the `myValues` column.

Array function example:

```
derive type:single value:ARRAYLEN(concat([colA,colB])) as:'length_myValues'
```

Output: Generates the new `length_myValues` column containing the count of elements in the array returned from concatenating `colA` and `colB` .

Syntax and Arguments

```
derive type:single value:ARRAYLEN(array_ref)
```

Argument	Required?	Data Type	Description
array_ref	Y	string	Name of Array column, Array literal, or function returning an Array to apply to the function

For more information on syntax standards, see *Language Documentation Syntax Notes*.

array_ref

Name of the array column, array literal, or function returning an array whose elements you want to count.

- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value

Yes	String (column reference or function) or array literal	`myArray1`

Examples

> **Tip:** For additional examples, see *Common Tasks*.

Example - Unnest an array

Source:

You have the following data on student test scores. Scores on individual scores are stored in the `Scores` array, and you need to be able to track each test on a uniquely identifiable row. This example has two goals:

1. One row for each student test
2. Unique identifier for each student-score combination

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Charles	[88,81,85,78]

Transform:

When the data is imported from CSV format, you must add a `header` transform and remove the quotes from the `Scores` column:

```
header
```

```
replace col:Scores with:'' on:`"` global:true
```

Validate test date: To begin, you might want to check to see if you have the proper number of test scores for each student. You can use the following transform to calculate the difference between the expected number of elements in the `Scores` array (4) and the actual number:

```
derive type:single value: (4 - ARRAYLEN(Scores)) as: 'numMissingTests'
```

When the transform is previewed, you can see in the sample dataset that all tests are included. You might or might not want to include this column in the final dataset, as you might identify missing tests when the recipe is run at scale.

Unique row identifier: The `Scores` array must be broken out into individual rows for each test. However, there is no unique identifier for the row to track individual tests. In theory, you could use the combination of `LastName-FirstName-Scores` values to do so, but if a student recorded the same score twice, your dataset has duplicate rows. In the following transform, you create a parallel array called `Tests`, which contains an index array for the number of values in the `Scores` column. Index values start at `0`:

```
derive type:single value:RANGE(0,ARRAYLEN(Scores)) as:'Tests'
```

Also, we will want to create an identifier for the source row using the `SOURCEROWNUMBER` function:

```
derive type:single value:SOURCEROWNUMBER() as:'orderIndex'
```

One row for each student test: Your data should look like the following:

LastName	FirstName	Scores	Tests	orderIndex
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2
Burns	Bonnie	[98,94,92,85]	[0,1,2,3]	3
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4

Now, you want to bring together the `Tests` and `Scores` arrays into a single nested array using the `ARRAYZIP` function:

```
derive type:single value:ARRAYZIP([Tests,Scores])
```

Your dataset has been changed:

LastName	FirstName	Scores	Tests	orderIndex	column1
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2	[[0,81],[1,87],[2,83],[3,79]]
Adams	Bonnie	[98,94,92,85]	[0,1,2,3]	3	[[0,98],[1,94],[2,92],[3,85]]
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4	[[0,88],[1,81],[2,85],[3,78]]

With the `flatten` transform, you can unpack the nested array:

```
flatten col: column1
```

Each test-score combination is now broken out into a separate row. The nested Test-Score combinations must be broken out into separate columns using `unnest`:

```
unnest col:column1 keys:'[0]','[1]'
```

After you delete `column1`, which is no longer needed you should rename the two generated columns:

```
rename mapping:[column_0,'TestNum']
```

```
rename mapping:[column_1,'TestScore']
```

Unique row identifier: You can do one more step to create unique test identifiers, which identify the specific test for each student. The following uses the original row identifier `OrderIndex` as an identifier for the student and the `TestNumber` value to create the `TestId` column value:

```
derive type:single value: (orderIndex * 10) + TestNum as: 'TestId'
```

The above are integer values. To make your identifiers look prettier, you might add the following:

```
merge col:'TestId00','TestId'
```

Extending: You might want to generate some summary statistical information on this dataset. For example, you might be interested in calculating each student's average test score. This step requires figuring out how to properly group the test values. In this case, you cannot group by the `LastName` value, and when executed at scale, there might be collisions between first names when this recipe is run at scale. So, you might need to create a kind of primary key using the following:

```
merge col:'LastName','FirstName' with:'-' as:'studentId'
```

You can now use this as a grouping parameter for your calculation:

```
derive type:single value:AVERAGE(TestScore) group:studentId as:'avg_TestScore'
```

Results:

After you delete unnecessary columns and move your columns around, the dataset should look like the following:

TestId	LastName	FirstName	TestNum	TestScore	studentId	avg_TestScore
TestId0021	Adams	Allen	0	81	Adams-Allen	82.5
TestId0022	Adams	Allen	1	87	Adams-Allen	82.5
TestId0023	Adams	Allen	2	83	Adams-Allen	82.5
TestId0024	Adams	Allen	3	79	Adams-Allen	82.5
TestId0031	Adams	Bonnie	0	98	Adams-Bonnie	92.25
TestId0032	Adams	Bonnie	1	94	Adams-Bonnie	92.25
TestId0033	Adams	Bonnie	2	92	Adams-Bonnie	92.25
TestId0034	Adams	Bonnie	3	85	Adams-Bonnie	92.25
TestId0041	Cannon	Chris	0	88	Cannon-Chris	83
TestId0042	Cannon	Chris	1	81	Cannon-Chris	83
TestId0043	Cannon	Chris	2	85	Cannon-Chris	83
TestId0044	Cannon	Chris	3	78	Cannon-Chris	83