DATA WEAVE FUNCTIONS

(1) ++ : The string concatenation is performed using "++" (double plus) operator. The string concatenation is performed only for same data types like :

{ } ++ { }

[ ] ++ [ ]

string ++ stirng

{ } ++ [ ] - not performed

[ ] ++ { } - not performed

Syntax : source ++ with

Here all is concatenated into same string in with.

Example : {aa: "a", bb: "b"} ++ {cc: "c"}

Output :

{"aa": "a", "bb": "b", "cc": "c"}

A screenshot of a computer

Description automatically generated

(2) isEven : This command is used to check if a number is even or not. Returns true if the number or numeric result of a mathematical operation is even, false if not. This command dosen't work for strings.

Syntax : isEven(number)

Example : [ isEven(0), isEven(2.0), isEven(1+1) ]

Output :

[true, true, true]

A white background with black lines

Description automatically generated

(3) isOdd : This command is used to check if a number is odd or not. Returns true if the number or numeric result of a mathematical operation is odd, false if not. This command dosen't work for strings.

Syntax : isOdd(number)

Example : [ isOdd(5), isOdd(2.0), isOdd(1+1) ]

Output :

[true, false, false]

A white background with black lines

Description automatically generated

(4) isDecimal : This command is used to check if a number is decimal or not. Returns true if the given number contains a decimal, false if not. This command dosen't work for strings.

Syntax : isDecimal(number)

Example : [ isDecimal(1.1), isDecimal(1), isDecimal("1.1") ]

Output :

[true, false, true]

A screen shot of a computer

Description automatically generated

(5) isInteger : This command is used to check if a number is integer or not. Returns true if the given number contains a integer, false if not. This command dosen't work for strings.

Syntax : isInteger(number)

Example : [isInteger(10), isInteger(10.6), isInteger(6353)]

Output :

[true, false, true]

A computer screen shot of a computer

Description automatically generated

(6) isBlank : This command returns true if the given string is empty (""), completely composed of whitespaces, or null. Otherwise, the function returns false. Here spaces are treated as empty.

Syntax : isBlank(payload)

Example 1 :

Payload : " "

Script : isBlank(payload)

Output : true

Example 2 :

Payload : null

Script : isBlank(payload)

Output : true

Example 3 :

Payload : "hi"

Script : isBlank(payload)

Output : false

A screenshot of a computer script

Description automatically generated

(7) isEmpty : This command returns true if the given input value is empty, false if not. This version of isEmpty acts on an array. Other versions act on a string or object, and handle null values. Here spaces are also treated as not empty.

Syntax : isEmpty([]/{}/""/payload)

Example : [ isEmpty([]), isEmpty([1]) ]

Output :

[true, false]

A screenshot of a computer script

Description automatically generated

(8) isLeapYear : This command is used to check if the given DataTime a leap year or not. This command returns true if it receives a date for a leap year, false if not. This version of leapYear acts on a DateTime type.

Syntax : isLeapYear("DateTime\_according\_to\_now\_command")

Example : [isLeapYear("2024-08-23T09:49:21.477158Z"), isLeapYear("2023-08-23T09:49:21.477158Z")]

Output :

[true, false]

A screenshot of a computer

Description automatically generated

(9) now : This command returns a DateTime value for the current date and time.

Syntax : now()

Example : now()

Output : "2023-08-23T10:30:29.224755Z"

A white background with black lines

Description automatically generated

(10) ceil : This command rounds a number up to the nearest whole number. This command works for only single number. This command dosen't work for strings.

Syntax : ceil(number)

Example : [ ceil(1.5), ceil(2.1), ceil(3) ]

Output :

[2, 3, 3]

A screenshot of a computer

Description automatically generated

(11) floor : This command rounds a number down to the nearest whole number. This command dosen't work for strings.

Syntax : floor(number)

Example : [floor(8.2), floor(8.9)]

Output :

[8, 8]

A white rectangular object with black lines

Description automatically generated

(12) round : This command rounds a number up or down to the nearest whole number. This round command is used for accurate results. This command dosen't work for strings.

Syntax : round(number)

Example : [round(7.6), round(8.2), round(9.5)]

Output :

[8, 8, 10]

A screenshot of a computer

Description automatically generated

(13) avg : This command returns the average of numbers listed in an array. An array that is empty or that contains a non-numeric value results in an error. This command dosen't work for strings.

Syntax : avg(value1,value2,value3)

Example : { a: avg([1, 1000]), b: avg([1, 2, 3]) }

Output :

{ "a": 500.5, "b": 2.0 }

A screenshot of a computer

Description automatically generated

(14) flatten : This command turns a set of subarrays into a single, flattened array. Note that it flattens only the first level of subarrays and omits empty subarrays.

Syntax : flatten(payload)

Example : %dw 2.0

output application/json

var array1 = [1,2,3]

var array2 = [4,5,6]

var array3 = [7,8,9]

var arrayOfArrays = [array1, array2, array3]

---

flatten(arrayOfArrays)

Output : [ 1,2,3,4,5,6,7,8,9 ]

A screenshot of a computer program

Description automatically generated

(15) flatmap : This command iterates over each item in an array and flattens the results. Instead of returning an array of arrays, flatMap returns a flattened array. flatMap is similar to flatten, but flatten only acts on the values of the arrays, while flatMap can act on values and indices of items in the array.

Syntax : flatMap ((item, index) -> )

Payload : [[1,2],[3,4]]

Example 1 :

Script : payload flatMap ((data, index) -> "data" : data)

Output :

[{"data": [1, 2]}, {"data": [3, 4]}]

Example 2 :

Script : payload flatMap ((item, index) -> index)

Output :

[0, 1]

A screenshot of a computer

Description automatically generated

(16) map : This command iterates over items in an array and outputs the results into a new array. This command works only for arrays.

$$ - index

$ - item

Syntax : map ((item, index) -> )

Example :

Payload :

[{

"fname": "Gayathri",

"age": 12

}]

Script :

payload map ((item, index) -> "data":

"fname" : upper(item.fname) )

Output :

[{"data": {"fname": "GAYATHRI"}}]

A screenshot of a computer

Description automatically generated

(17) mapObject : This commandobject input object output This command works only for single object.

$ - value

$$ - keys

$$$ - index

Syntax : mapObject ((value, key, index) -> )

Example :

Payload :

[{

"fname": "Gayathri",

"age": 12

}]

Script : payload mapObject ((value, key, index) -> "value" : index )

Output :

{"value": 0, "value": 1}

A screenshot of a computer

Description automatically generated

(18) upper : This command returns the provided string in uppercase characters. This command dosen't work for integers.

Syntax : upper(text)

Example : { "name" : upper("mulesoft") }

Output : { "name": "MULESOFT" }

A screenshot of a computer

Description automatically generated

(19) distinctBy : This command is used to iterate over an array and returns the unique elements in it. This command is used to remove the duplicates.

Syntax : distinctBy ((item, index) -> )

Example : [0, 1, 2, 3, 3, 2, 1, 4] distinctBy ((item, index) -> item )

Output : [0, 1, 2, 3, 4]

A computer screen shot of a computer code

Description automatically generated

(20) filter : This command iterates over an array and applies an expression that returns matching values. The expression must return true or false. If the expression returns true for a value or index in the array, the value gets captured in the output array. If it returns false for a value or index in the array, that item gets filtered out of the output. If there are no matches, the output array will be empty.

Syntax : filter ((item, index) -> )

Example :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : payload filter ((item, index) -> index>-1 )

Output :

[

{"name": "Bar", "language": "Scala"},

{"name": "Foo", "language": "Java"},

{"name": "FooBar", "language": "Java"}

]

A screenshot of a computer

Description automatically generated

(21) filterObject : This command iterates a list of key-value pairs in an object and applies an expression that returns only matching objects, filtering out the rest from the output. The expression must return true or false. If the expression returns true for a key, value, or index of an object, the object gets captured in the output. If it returns false for any of them, the object gets filtered out of the output. If there are no matches, the output object will be empty.

Syntax : filterObject ((value, key, index) -> )

Example : { "1": "a", "2": "b", "3": "c"} filterObject ((value, key, index) -> index < 1)

Output : { "1": "a" }

A screen shot of a computer

Description automatically generated

(22) daysBetween : This command returns the number of days between two dates.

Syntax : daysBetween(from\_date, to\_date)

Example : daysBetween("2023-08-24T09:09:39.519393Z","2023-11-30T09:09:39.519393Z")

Output : 98

A blue and red text with numbers

Description automatically generated with medium confidence

(23) joinBy : This command merges an array into a single string value and uses the provided string as a separator between each item in the list. joinBy performs the opposite task of splitBy.

Syntax : joinBy " value/ symbol "

Example : [1, 2, 3, 4, 5, 6, 7, 8] joinBy "-"

Output : "1-2-3-4-5-6-7-8"

A screenshot of a computer

Description automatically generated

(24) splitBy : This command splits a string into a string array based on a value that matches part of that string. It filters out the matching part from the returned array. This version of splitBy accepts a Java regular expression (regex) to match the input string. The regex can match any character in the input string. splitBy performs the opposite operation of joinBy.

Syntax : splitBy " value/ symbol "

Example : [1, 2, 3, 4, 5, 6, 7, 8] joinBy ",,,,," splitBy ",,,,,"

Output : ["1", "2", "3", "4", "5", "6", "7", "8"]

A screenshot of a computer

Description automatically generated

(25) typeOf : This command is used to return the type of a value.

Syntax : typeOf(value)

Example 1 :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : typeOf(payload)

Output : "Array"

Example 2 : [ typeOf("A b"), typeOf([1,2]), typeOf(34), typeOf(true), typeOf({ a : 5 }) ]

Output : [ "String", "Array", "Number", "Boolean", "Object" ]

A screenshot of a computer script

Description automatically generated

(26) sizeOf : This command returns the number of elements in an array. It returns 0 if the array is empty.

Syntax : sizeOf(value)

Example 1 :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : sizeOf(payload)

Output : 3

Example 2 : sizeOf({a: 1, b: 2})

Output : 2

A screenshot of a computer script

Description automatically generated

(27) contains : This command returns true if an input contains a given value, false if not. Here objects are not taken as input values.

Syntax : contains(value)

Example 1 :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : payload.name contains("Foo")

Output : true

Example 2 : "mulesoft" contains("mule")

Output : true

A screenshot of a computer script

Description automatically generated

(28) pluck : This command is useful for mapping an object into an array, pluck iterates over an object and returns an array of keys, values, or indices from the object. If there are 2 functions, then 1st function should be in (). Otherwise it is considered as string. It is an alternative to mapObject, which is similar but returns an object, instead of an array.

Syntax : pluck ((value, key, index) -> )

Example : {"a":"b","c":"d"} pluck (value,key,index) -> { (index) : { (value):key} }

Output :

[

{"0": { "b": "a"}}, {"1": { "d": "c"} }

]

A screenshot of a computer

Description automatically generated

(29) orderBy : This command is used to reorder the elements in a object based on some criteria. It takes both array and object as input.

Syntax : orderBy ((item, index) -> )

Example :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : payload orderBy ((item, index) -> item.name)

Output :

[

{"name": "Bar", "language": "Scala"},

{"name": "Foo", "language": "Java"},

{"name": "FooBar", "language": "Java"}

]

A screenshot of a computer

Description automatically generated

(30) groupBy : This command groups the items in the array based on some criteria and returns a object as an output. Here array is given as input and object is given as output.

Syntax : groupBy ((item, index) -> )

Example :

Payload :

[

{ "name": "Foo", "language": "Java" },

{ "name": "Bar", "language": "Scala" },

{ "name": "FooBar", "language": "Java" }

]

Script : payload groupBy ((item, index) -> item.language)

Output :

{

"Java": [

{"name": "Foo", "language": "Java"}, {"name": "FooBar", "language": "Java"}

],

"Scala": [

{"name": "Bar", "language": "Scala"}

]

}

A screenshot of a computer

Description automatically generated

(31) lower : This command returns the provided string in lowercase characters. This command dosen't work for integers.

Syntax : lower(text)

Example : { "name" : lower("MULESOFT" )}

Output : { "name": "mulesoft" }

A screenshot of a computer

Description automatically generated