Liquid Guidelines

Introduction

Liquid is an open-source template language created by Shopify and written in Ruby. It is the backbone of Shopify themes and is used to load dynamic content on storefronts.

In MailUp Liquid is used as markup language for the email messages that use the Advanced content personalization.

The following guide describes how to use Liquid within MailUp and which statements are supported.

When talking about *dynamics fields*, we refer to the MailUp Recipient fields.

To retrieve a specific dynamic field value, you have to use the following syntax: **dynamicfields.<dynamic field name>**.

In the liquid syntax, not all characters are supported in a variable name. To be sure to have a correct syntax when using MailUp dynamic field, we recommend to use only:

- **digits** (0...9)
- **letters** (a...z and A...Z)
- **dash** (-) and **underscore** (_) signs

For MailUp Recipient fields that contain these characters:

- **accented characters** should be substitute with normalized character (e.g. 'CrèmeBrûlée' -> 'CremeBrulee')
- **special characters**, such as full stop (.), space (' '), dollar('$'), should be removed from the field name (e.g. 'Is a colleague' -> 'Isacolleague').

Basics

Operators

Operator	Meaning	Work	Syntax

==	equals	✓	**This operator can be used with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name == 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name == 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'equalTo', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'equalTo', '8' %} {% if c1 == true %} Text {% endif %}`

!=	does not equal	✓	**It is possible use this operator with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name != 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name != 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'otherThan', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'otherThan', '8' %} {% if c1 == true %} Text {% endif %}`

>	greater than	✓	**It is possible use this operator with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name > 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name > 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'moreThan', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'moreThan', '8' %} {% if c1 == true %} Text {% endif %}`

<	less than	✓	**It is possible use this operator with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name < 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name < 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'lessThan', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'lessThan', '8' %} {% if c1 == true %} Text {% endif %}`

>=	greater than or equal to	✓	**It is possible use this operator with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name >= 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name >= 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'equalOrMoreThan', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'equalOrMoreThan', '8' %} {% if c1 == true %} Text {% endif %}`

<=	less than or equal to	✓	**It is possible use this operator with dynamic fields 'as is' only to compare strings.**
			`{% if dynamicfields.name <= 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name <= 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			In this case the check is case sensitive.
			To compare numbers or data type different from string it is possible use the built in functions:
			`{% assign c1 = dynamicfields.name| evaltext: 'equalOrLessThan', 'Jennifer' %} {% if c1 == true %} Text {% endif %}`
			`{% assign c1 = dynamicfields.age | evalinteger: 'equalOrLessThan', '8' %} {% if c1 == true %} Text {% endif %}`

or	logical or	✓	`{% if dynamicfields.name == 'alan' or dynamicfields.age == 'child' %}`
			`is alan or a child`
			`{% else %}`
			`It is an adult alan or another adult`
			`{% endif %}`

and	logical and	✓	`{% if dynamicfields.name == 'alan' and dynamicfields.age == 'child' %}`
			`alan is a child`
			`{% else %}`
			`it is an adult`
			`{% endif %}`

contains	checks for the presence of a substring inside a string	✓	`{% if dynamicfields.name contains 'oof' %}`
			`could be alan?`
			`{% else %}`
			`surely isn't alan`
			`{% endif %}`

Types

Type	Meaning	Work	Syntax

String	Declare a string by wrapping a variable's value in single quotes:	✅	**Input**
			`{% assign thisIsAString = 'this is a string' %}`
			`{{ thisIsAString | upcase }}`
			
			Output
			`THIS IS A STRING`

Number	Numbers include floats and integers:	✅	**Input**
			`{% assign thisIsAnInteger = 10 %}`
			`{{ thisIsAnInteger | minus: 2 }}`
			
			Output
			`8`
			
			Input
			`{% assign thisIsAFloat = 10.234 %}`
			`{{ thisIsAFloat  | minus: 2 }}`
			
			Output
			`8.23400020599365`

Boolean		✅	**Input**
			`{% assign thisIsTrue = true %}`
			`{{ thisIsTrue }}`
			
			Output
			`true`
			
			Input
			`{% assign thisIsFalse = true %}`
			`{{ thisIsFalse }}`
			
			Output
			`false`

Nil	Nil is a special empty value that is returned when Liquid code has no results. It is **not** a string with the characters “nil”.
	
	Nil is treated as false in the conditions of `if` blocks and other Liquid tags that check the truthfulness of a statement.
		✅	**Input**
			`{% if user %}`
			`  Hello user!`
			`{% endif %}`
			
			Output
			`Hello user!`
			
			Input
			`{% unless user %}`
			`  Hello user!`
			`{% endunless %}`
			
			Output
			`Hello user!`

Array	Arrays hold lists of variables of any type.	✓	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{{ my_array.first }}`
			`{{ my_array.last }}`
			Output
			`alan`
			`mark`
			Input
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array %}`
			`{{ user }}`
			`{% endfor %}`
			Output
			`alan catherine tom erin mark`
			Input
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{{ my_array[2] }}`
			Output
			`tom`

TAGS

Comment

Operator	Meaning	Work	Syntax

comment	Any text within the opening and closing `comment` blocks will not be output, and any Liquid code within will not be executed	✓	**Input**
			`Text 1 {% comment %} Text 2 {% endcomment %}`
			
			Output
			`Text 1`

Control flow

Operator	Meaning	Work	Syntax

if	Executes a block of code only if a certain condition is **`true`**	✓	**Input**
			`{% assign c1 = dynamicfields.age| evalinteger: 'equalTo', '10' %}`
			`{% if c1 == true %}`
			`10 years old`
			`{% endif %}`
			
			Output
			`10 years old`
			
			Input
			`{% if dynamicfields.name == 'alan' %}`
			`alan`
			`{% endif %}`
			
			Output
			`alan`

unless	Executes a block of code only if a certain condition is **not** match	✅	**Input**
			`{% assign c1 = dynamicfields.age| evalinteger: 'equalTo', '10' %}`
			`{% unless c1 == true %}`
			`Not 10 years old`
			`{% endunless %}`
			
			Output
			`Not 10 years old`
			
			Input
			`{% unless dynamicfields.name == 'alan' %}`
			`alan`
			`{% endunless %}`
			
			Output
			`alan`

elsif/else	Adds more conditions within an `if` or `unless` block.	✅	**Input**
			`{% if dynamicfields.name == 'alan' %}`
			`alan`
			`{% elsif dynamicfields.name == 'tom' %}`
			`tom`
			`{% else %}`
			`catherine`
			`{% endif %}`
			
			output if name is goofy
			`alan`
			
			output if name is tom
			`tom`
			
			output for each other name
			`catherine`

case/when	Creates a switch statement to compare a variable with different values	✓	**Input**
			`{% case dynamicfields.name%}`
			`  {% when 'alan' %}`
			`  This is a alan`
			`  {% when 'tom' %}`
			`  This is a tom`
			`  {% else %}`
			`  This is not a alan nor a tom. can it be a catherine?`
			`{% endcase %}`
			output if name is goofy
			`This is a alan`
			output if name is mickey
			`This is a tom`
			output for each other name
			`This is not a alan nor a tom. can it be a catherine?`

Iteration

For

Operator	Meaning	Work	Syntax

for	Repeatedly executes a block of code.	✓	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array %}`
			`  {{ user }}`
			`{% endfor %}`
			Output
			`alan catherine tom erin mark`

break	Causes the loop to stop iterating when it encounters the `break` tag.	✓	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array %}`
			` {% if user == 'erin' %}`
			`  {% break %}`
			`  {% else %}`
			`  {{ user }}`
			`  {% endif %}`
			`{% endfor %}`
			Output
			`alan catherine tom`

continue	Causes the loop to skip the current iteration when it encounters the `continue` tag.	✓	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array %}`
			` {% if user == 'erin' %}`
			`  {% continue %}`
			`  {% else %}`
			`  {{ user }}`
			`  {% endif %}`
			`{% endfor %}`
			Output
			`alan catherine tom mark`

limit	Limits the loop to the specified number of iterations.	✓	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array limit:2 %}`
			` {{ user }}`
			`{% endfor %}`
			Output
			`alan catherine`

offset	Begins the loop at the specified index.	✅	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array offset:2 %}`
			` {{ user }}`
			`{% endfor %}`
			
			Output
			`tom erin mark`

range	Defines a range of numbers to loop through. The range can be defined by both literal and variable numbers.	✅	**Input**
			`{% for i in (3..5) %}`
			`  {{ i }}`
			`{% endfor %}`
			
			Output
			`3 4 5`
			
			Input
			`{% assign num = 4 %}`
			`{% for i in (1..num) %}`
			`  {{ i }}`
			`{% endfor %}`
			
			Output
			`1 2 3 4`

reversed	Reverses the order of the loop.	✅	**Input**
			`{% assign my_array = 'alan,catherine, tom,erin,mark' | split: ',' %}`
			`{% for user in my_array reversed %}`
			` {{ user }}`
			`{% endfor %}`
			
			Output
			`mark erin tom catherine alan`

Cycle

Operator	Meaning	Work	Syntax

Cycle	Repeatedly executes a block of code.	✓	**Input**
			`{% assign my_array = 'alan,catherine,tom,erin,mark' | split: ',' %}`
			`{% for user in my_array %}`
			`{% cycle 'first one', 'second one', 'third one' %}`
			`{% endfor %}`
			Output
			`first one second one third one first one second one`

Variable

Operator	Meaning	Work	Syntax

assign	Creates a new variable	✓	**Input**
			`{% assign c1 = 10 %}`
			`{% if c1 == 10 %}`
			`It is ten!`
			`{% endif %}`
			Output
			`It is ten!`
			Input
			`{% assign c1 = 'alan' %}`
			`{% if c1 == 'alan' %}`
			`It is alan!`
			`{% endif %}`
			Output
			`It is alan!`

capture	Captures the string inside of the opening and closing tags and assigns it to a variable. Variables created through `{% capture %}` are strings.	✓	**Input**
			`{% assign name = dynamicfields.name %}`
			`{% assign surname = dynamicfields.surname %}`
			
			`{% capture about_me %}`
			`I am {{ name }} {{ surname }} and my favorite food is pizza.`
			`{% endcapture %}`
			
			`{{ about_me }}`
			
			output
			`I am Catherine Martin and my favorite food is pizza.`

increment	Creates a new number variable, and increases its value by one every time it is called. The initial value is 0.	✗	
decrement	Creates a new number variable, and decreases its value by one every time it is called. The initial value is -1.	✗	

Filters

Operator	Meaning	Work	Syntax
abs	Returns the absolute value of a number.	✗	

append	Concatenates two strings and returns the concatenated value.	✓	**Input**
			`{{ "Hello " | append: "Carl" }}`
			Output
			`Hello Carl`
			Input
			`{{ "Welcome" | append: dynamicfields.name }}`
			Output
			`Welcome Luke`
			Input
			`{{ dynamicfields.name | append: " welcome!"}}`
			Output
			`Luke welcome!`

at_least	Limits a number to a minimum value.	✗	
at_most	Limits a number to a maximum value.	✗	

capitalize	Makes the first character of a string capitalized.	✅	**Input**
			`{{ "my great title" | capitalize }}`
			Output
			`My great title`
			Input
			`{{ dynamicfields.nome | capitalize }}`
			Output
			`Luke`

ceil	Rounds the input up to the nearest whole number. Liquid tries to convert the input to a number before the filter is applied.	❌	
compact	Removes any `nil` values from an array.		
concat	Concatenates (joins together) multiple arrays. The resulting array contains all the items from the input arrays.	❌	
date	Converts a timestamp into another date format. The format for this syntax is the same as `strftime`. The input uses the same format as Ruby's `Time.parse`.		
default	Allows you to specify a fallback in case a value doesn't exist. `default`will show its value if the left side is `nil`, `false`, or empty.	❌	

divided_by	Divides a number by the specified number.	⚠️	Does not work on *dynamicfields*.
			
			Input
			`{{ 16 | divided_by: 4 }}`
			
			Output
			`4`
			
			Input
			`{{ subTotal| divided_by: 4 }}`
			
			Output
			`100`

downcase	Makes each character in a string lowercase. It has no effect on strings which are already all lowercase.	✅	**Input**
			`{{ "Parker Moore" | downcase }}`
			
			Output
			`parker moore`
			
			Input
			`{{ dynamicfields.nome | downcase}}`
			
			Output
			`luke`

escape	Escapes a string by replacing characters with escape sequences (so that the string can be used in a URL, for example). It doesn't change strings that don't have anything to escape.	❌	
escape_once	Escapes a string without changing existing escaped entities. It doesn't change strings that don't have anything to escape.	❌	

first	Returns the first item of an array.		Assuming the *userid* dynamic field contains "1,2,3,4,5".
			Input
			`{% assign my_array = dynamicfields.userid | split: "," %}`
			`{{ my_array.first }}`
			Output
			`1`
			Input
			`{% assign my_array = "apples, oranges, peaches, plums" | split: ", " %}`
			`{{ my_array.first }}`
			Output
			`apples`

floor	Rounds a number down to the nearest whole number. Liquid tries to convert the input to a number before the filter is applied.	✗	

join	Combines the items in an array into a single string using the argument as a separator.	✅	Assuming the *userid* dynamic field contains "1,2,3,4,5".
			
			Input
			`{% assign my_array = dynamicfields.userid | split: "," %}`
			`{{ my_array | join: " and " }}`
			
			Output
			`1 and 2 and 3 and 4 and 5`
			
			Input
			`{% assign beatles = "John, Paul, George, Ringo" | split: ", " %}`
			`{{ beatles | join: " and " }}`
			
			Output
			`John and Paul and George and Ringo`

last	Returns the last item of an array.	✅	Assuming the *userid* dynamic field contains "1,2,3,4,5".
			Input
			`{% assign my_array = dynamicfields.userid | split: "," %}`
			`{{ my_array.last}}`
			Output
			`5`
			Input
			`{% assign my_array = "apples, oranges, peaches, plums" | split: ", " %}`
			`{{ my_array.last }}`
			Output
			`plums`

lstrip	Removes all whitespaces (tabs, spaces, and newlines) from the beginning of a string. The filter does not affect spaces between words.	❌	

map	Creates an array of values by extracting the values of a named property from another object.	✅	Assuming the *products* contains:
			
			`{ "products": [{ "title": "Product 1", "price": 100 }, { "title": "Product 2", "price": 200 }, { "title": "Product 3", "price": 300 }] }`
			
			Input
			`{% assign all_titles = products | map: "title" %}`
			
			`{% for item in all_titles %} {{ item }} {% endfor %}`
			
			Output
			`Product 1 Product 2 Product 3`

minus	Subtracts a number from another number.	⚠️	Does not work on *dynamicfields*.
			Input
			`{{ 4 | minus: 2 }}`
			Output
			`2`
			Input
			`{{ SubTotal | minus: 100 }}`
			Output
			`900`

modulo	Returns the remainder of a division operation.	⚠️	Does not work on *dynamicfields*.
			Input
			`{{ 3 | modulo: 2 }}`
			Output
			`1`
			Input
			`{{ SubTotal | modulo: 6}}`
			Output
			`4`

newline_to_br	Replaces every newline (`\n`) with an HTML line break (` `).	❌	

plus	Adds a number to another number.	✓	Assuming the *TotalIncome* dynamic field contains 500.
			Input
			`{{ dynamicfields.TotalIncome| plus: 2 }}`
			Output
			`502`
			Input
			`{{ 5| plus: 2 }}`
			`7`

prepend	Adds the specified string to the beginning of another string.	✓	**Input**
			`{{ "apples, oranges, and bananas" | prepend: "Some fruit: " }}`
			Output
			`Some fruit: apples, oranges, and bananas`
			Input
			`{{ dynamicfields.Name| prepend: "Hello " }}`
			Output
			`Hello Luke`

Input

```
{{ dynamicfields.Name|
prepend: dynamicfields.
Surname}}
```

Output

```
Luke Skywalker
```

Input

```
{{ "Welcome" |
prepend: dynamicfields.
Name }}
```

Output

```
Welcome Luke
```

remove	Removes every occurrence of the specified substring from a string.	✓	**Input**
			`{{ "I strained to see the train through the rain" | remove: "rain" }}`
			
			Output
			`I sted to see the t through the`
			
			Input
			`{{ dynamicfields.Name| remove: "uk"}}`
			
			Output
			`Le`
			
			Input
			`{{"Luke Skywalker" | remove: dynamicfields.Name}}`
			
			Output
			`Skywalker`

remove_first	Removes only the first occurrence of the specified substring from a string.	✅	**Input**
			`{{ "I strained to see the train through the rain" | remove_first: "rain" }}`
			Output
			`I sted to see the train through the rain`

replace	Replaces every occurrence of an argument in a string with the second argument.	✅	**Input**
			`{{ "Take my protein pills and put my helmet on" | replace: "my", "your" }}`
			Output
			`Take your protein pills and put your helmet on`

replace_first	Replaces only the first occurrence of the first argument in a string with the second argument.	✅	**Input**
			`{% assign my_string = "Take my protein pills and put my helmet on" %}`
			`{{ my_string | replace_first: "my", "your" }}`
			Output
			`Take your protein pills and put my helmet on`

reverse	Reverses the order of the items in an array. `reverse` cannot reverse a string.	❌	

round	Rounds an input number to the nearest integer or, if a number is specified as an argument, to that number of decimal places.	✅	Does not work on *dynamicfields*.
			Input
			`{{ 1.2 | round }}`
			Output
			`1`

rstrip	Removes all whitespace (tabs, spaces, and newlines) from the right side of a string.	❌	

size	Returns the number of characters in a string or the number of items in an array. `size` can also be used with dot notation (for example, `{{ my_string.size }}`). This allows you to use `size` inside tags such as conditionals.	✅	**Input**
			`{{ "Ground control to Major Tom." | size }}`
			Output
			`28`
			Input
			`{% assign my_array = "apples, oranges, peaches, plums" | split: ", " %}`
			`{{ my_array | size }}`
			Output
			`4`

slice	Returns a substring of 1 character beginning at the index specified by the argument passed in. An optional second argument specifies the length of the substring to be returned.
	
	String indices are numbered starting from 0.
		✅	**Input**
			`{{ "Liquid" | slice: 0 }}`
			
			Output
			`L`
			
			Assuming *title* contains "Return to the future"
			
			Input
			`{{title | slice: 0}}`
			
			Output
			`R`
			
			Input
			`{{ dynamicfields.title| slice: 0 }}`
			
			Outuput
			`R`

sort	Sorts items in an array by a property of an item in the array. The order of the sorted array is case-sensitive.	⚠️	Same behaviour of the *sort_natural* filter.
			
			Input
			`{% assign my_array = "zebra, octopus, giraffe, Sally Snake" | split: ", " %}`
			
			`{{ my_array | sort_natural | join: ", " }}`
			
			Output
			`giraffe, octopus, Sally Snake, zebra`

sort_natural	Sorts items in an array by a property of an item in the array.	❌	

split	Divides an input string into an array using the argument as a separator. `split` is commonly used to convert comma-separated items from a string to an array.	✓	**Input**
			`{% assign beatles = "John, Paul, George, Ringo" | split: ", " %}`
			`{% for member in beatles %}`
			`{{ member }}`
			`{% endfor %}`
			Output
			`John`
			`Paul`
			`George`
			`Ringo`
			Assuming the dynamic field *albumid* contains "1,2,3,4,5"
			Input
			`{% assign ids= dynamicfields.albumid | split: "," %}`
			`{% for id in ids%}`
			`{{ id }}`
			`{% endfor %}`
			Output
			`1`
			`2`
			`3`
			`4`
			`5`

strip	Removes all whitespace (tabs, spaces, and newlines) from both the left and right side of a string. It does not affect spaces between words.	✗	

strip_html	Removes any HTML tags from a string.	✓	**Input**
			`{{ "Have <em>you</em> read <strong>Ulysses</strong>?" | strip_html }}`
			Output
			`Have you read Ulysses?`

strip_newlines	Removes any newline characters (line breaks) from a string.	❌	

times	Multiplies a number by another number.	⚠️	Does not work on *dynamicfields*.
			Input
			`{{ 24 | times: 7 }}`
			Output
			`168`

truncate	`truncate` shortens a string down to the number of characters passed as a parameter. If the number of characters specified is less than the length of the string, an ellipsis (…) is appended to the string and is included in the character count.	✅	**Input**
			`{{ "Ground control to Major Tom." | truncate: 20 }}`
			Output
			`Ground control to...`
			Input
			`{{ "Ground control to Major Tom." | truncate: 25, ", and so on" }}`
			Output
			`Ground control, and so on`

truncatewords	Shortens a string down to the number of words passed as the argument. If the specified number of words is less than the number of words in the string, an ellipsis (…) is appended to the string.	❌	
uniq	Removes any duplicate elements in an array.	❌	

upcase	Makes each character in a string uppercase. It has no effect on strings which are already all uppercase.	✅	**Input** `{{ dynamicfields.name | upcase }}` **Output** LUKE
url_decode	Decodes a string that has been encoded as a URL or by `url_encode`.	❌	
url_encode	Converts any URL-unsafe characters in a string into percent-encoded characters.	❌	