DotLiquid Scripting Basics

05/20/2025 3:54 pm CDT

Overview

service**minder** uses DotLiquid as a multi-purpose scripting language for dynamically generating print and email template content. It is an adaptation of the Liquid scripting language used by Shopify, with a few notable differences which we attempt to document below.

For a full list of properties accessible via DotLiquid, please visit: https://serviceminder.com/support/dotliquid

This article will review:

Introduction

Liquid code can be categorized into **objects**, **tags**, and **filters**.

Objects

Objects tell Liquid where to show content on a page. Objects and variable names are denoted by double curly braces: `{{` and `}}`.

Input

```
{{ page.title }}
```

Output

```
Introduction
```

In this case, Liquid is rendering the content of an object called `page.title`, and that object contains the text `Introduction`.

Tags

Tags create the logic and control flow for templates. They are denoted by curly braces and percent signs: `{%` and `%}`.

The markup used in tags does not produce any visible text. This means that you can assign variables and create conditions and loops without showing any of the Liquid logic on the page.

Input

```
{% if user %}
 Hello {{ user.name }}!
{% endif %}
```

Output

```
Hello Adam!
```

Tags can be categorized into three types:

- Control flow
- Iteration
- Variable assignments

You can read more about each type of tag in their respective sections.

Filters

Filters change the output of a Liquid object. They are used within an output and are separated by a `|`.

Input

```
{{ "/my/fancy/url" | append: ".html" }}
```

Output

```
/my/fancy/url.html
```

Multiple filters can be used on one output. They are applied from left to right.

Input

```
{{ "adam!" | capitalize | prepend: "Hello " }}
```

Output

```
Hello Adam!
```

Date Format Strings

When using the `date` filter, DotLiquid will use .NET date format strings as opposed to Python "strftime".

.NET Format strings can be found here: http://msdn.microsoft.com/en-us/library/8kb3ddd4(v=vs.110).aspx

Using a .NET date format string within a template: `{{ somedatefield | date: "MMMM dd, yyyy" }}`

Operators

Liquid includes many logical and comparison operators.

Basic Operators

Operator	Description
`==`	Equals
`!=`	Does not equal
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to
`or`	Logical OR
`and`	Logical AND

For example:

```
{% if product.title == "Awesome Shoes" %}
  These shoes are awesome!
{% endif %}
```

You can use multiple operators in a tag:

```
{% if product.type == "Shirt" or product.type == "Shoes" %}
  This is a shirt or a pair of shoes.
{% endif %}
```

contains

`contains` checks for the presence of a substring inside a string.

```
{% if product.title contains "Pack" %}
  This product's title contains the word Pack.
{% endif %}
```

`contains` can also check for the presence of a string in an array of strings.

```
{% if product.tags contains "Hello" %}
  This product has been tagged with "Hello".
{% endif %}
```

`contains` can only search strings. You cannot use it to check for an object in an array of objects.

Order of operations

In tags with more than one `and` or `or` operator, operators are checked in order *from right to left*. You cannot change the order of operations using parentheses — parentheses are invalid characters in Liquid and will prevent your tags from working.

```
{% if true or false and false %}
  This evaluates to true, since the `and` condition is checked first.
{% endif %}
```

```
{% if true and false and false or true %}
  This evaluates to false, since the tags are checked like this:

  true and (false and (false or true))
  true and (false and true)
  true and false
  false
{% endif %}
```

Truthy and Falsy

In programming, anything that returns `true` in a conditional is called **truthy**. Anything that returns `false` in a conditional is called **falsy**. All object types can be described as either truthy or falsy.

- Truthy
- Falsy
- Summary

Truthy

All values in Liquid are truthy except `nil` and `false`.

In the example below, the string “Tobi” is not a boolean, but it is truthy in a conditional:

```
{% assign tobi = "Tobi" %}

{% if tobi %}
  This condition will always be true.
{% endif %}
```

Strings, even when empty, are truthy. The example below will result in empty HTML tags if `settings.fp_heading` is

empty:

Input

```
{% if settings.fp_heading %}
{{ settings.fp_heading }}
{% endif %}
```

Output

```

```

Falsy

The falsy values in Liquid are `nil` and `false`.

Summary

The table below summarizes what is truthy or falsy in Liquid.

Input	Truthy	Falsy
true	•	
false		•
nil		•
string	•	
empty string	•	
0	•	
integer	•	
float	•	
array	•	
empty array	•	
page	•	
EmptyDrop	•	

Types

Liquid objects can have one of five types:

- String
- Number
- Boolean
- Nil

- Array

You can initialize Liquid variables with the assign or capture tags.

String

Declare a string by wrapping a variable's value in single or double quotes:

```
{% assign my_string = "Hello World!" %}
```

Number

Numbers include floats and integers:

```
{% assign my_int = 25 %}
{% assign my_float = 39.756 %}
```

Boolean

Booleans are either `true` or `false`. No quotations are necessary when declaring a boolean:

```
{% assign foo = true %}
{% assign bar = 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.

In the following example, if the user does not exist (that is, `user` returns `nil`), Liquid will not print the greeting:

```
{% if user %}
  Hello {{ user.name }}!
{% endif %}
```

Tags or outputs that return `nil` will not print anything to the page.

Input

```
The current user is {{ user.name }}
```

Output

```
The current user is
```

Array

Arrays hold lists of variables of any type.

Accessing items in arrays

To access all the items in an array, you can loop through each item in the array using an iteration tag.

Input

```
{% for user in site.users %}
  {{ user }}
{% endfor %}
```

Output

```
Tobi Laura Tetsuro Adam
```

Accessing specific items in arrays

You can use square bracket `[` `]` notation to access a specific item in an array. Array indexing starts at zero.

Input

```
{{ site.users[0] }}
{{ site.users[1] }}
{{ site.users[3] }}
```

Output

```
Tobi
Laura
Adam
```

Initializing arrays

You cannot initialize arrays using only Liquid.

You can, however, use the split filter to break a string into an array of substrings.

Whitespace Control

In Liquid, you can include a hyphen in your tag syntax `{{-` , `-}}` , `{%-` , and `-%}` to strip whitespace from the left or right side of a rendered tag.

Normally, even if it doesn't print text, any line of Liquid in your template will still print a blank line in your rendered

HTML:

Input

```
{% assign my_variable = "tomato" %}
{{ my_variable }}
```

Notice the blank line before “tomato” in the rendered template:

Output

```
tomato
```

By including hyphens in your `assign` tag, you can strip the generated whitespace from the rendered template:

Input

```
{%- assign my_variable = "tomato" -%}
{{ my_variable }}
```

Output

```
tomato
```

If you don’t want any of your tags to print whitespace, as a general rule you can add hyphens to both sides of all your tags (`{%-` and `-%}`):

Input

```
{% assign username = "John G. Chalmers-Smith" %}
{% if username and username.size > 10 %}
  Wow, {{ username }}, you have a long name!
{% else %}
  Hello there!
{% endif %}
```

Output without whitespace control

```

  Wow, John G. Chalmers-Smith, you have a long name!


```

Input

```
{%- assign username = "John G. Chalmers-Smith" -%}
{%- if username and username.size > 10 -%}
  Wow, {{ username }}, you have a long name!
{%- else -%}
  Hello there!
{%- endif -%}
```

Output with whitespace control

Wow, John G. Chalmers-Smith, you have a long name!