DotLiquid Scripting Tags

05/20/2025 3:57 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:

Comment

Allows you to leave un-rendered code inside a Liquid template. Any text within the opening and closing `comment` blocks will not be printed, and any Liquid code within will not be executed.

Input

```
Anything you put between {% comment %} and {% endcomment %} tags
is turned into a comment.
```

Output

```
Anything you put between  tags
is turned into a comment.
```

Control Flow

Control flow tags can change the information Liquid shows using programming logic.

if

Executes a block of code only if a certain condition is `true` .

Input

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

Output

```
These shoes are awesome!
```

unless

The opposite of `if` – executes a block of code only if a certain condition is**not** met.

Input

```
{% unless product.title == "Awesome Shoes" %}
  These shoes are not awesome.
{% endunless %}
```

Output

```
These shoes are not awesome.
```

This would be the equivalent of doing the following:

```
{% if product.title != "Awesome Shoes" %}
  These shoes are not awesome.
{% endif %}
```

elsif / else

Adds more conditions within an `if` or `unless` block.

Input

```
{% if customer.name == "kevin" %}
  Hey Kevin!
{% elsif customer.name == "anonymous" %}
  Hey Anonymous!
{% else %}
  Hi Stranger!
{% endif %}
```

Output

```
Hey Anonymous!
```

case/when

Creates a switch statement to compare a variable with different values. `case` initializes the switch statement, and `when` compares its values.

Input

```
{% assign handle = "cake" %}
{% case handle %}
 {% when "cake" %}
    This is a cake
 {% when "cookie" %}
    This is a cookie
 {% else %}
    This is not a cake nor a cookie
{% endcase %}
```

Output

```
This is a cake
```

Iteration

Iteration tags run blocks of code repeatedly.

for

Repeatedly executes a block of code. For a full list of attributes available within a `for` loop, see forloop (object).

Input

```
{% for product in collection.products %}
 {{ product.title }}
{% endfor %}
```

Output

```
hat shirt pants
```

else

Specifies a fallback case for a `for` loop which will run if the loop has zero length.

Input

```
{% for product in collection.products %}
  {{ product.title }}
{% else %}
  The collection is empty.
{% endfor %}
```

Output

```
The collection is empty.
```

break

Causes the loop to stop iterating when it encounters the `break` tag.

Input

```
{% for i in (1..5) %}
  {% if i == 4 %}
    {% break %}
  {% else %}
    {{ i }}
  {% endif %}
{% endfor %}
```

Output

```
1 2 3
```

continue

Causes the loop to skip the current iteration when it encounters the `continue` tag.

Input

```
{% for i in (1..5) %}
  {% if i == 4 %}
    {% continue %}
  {% else %}
    {{ i }}
  {% endif %}
{% endfor %}
```

Output

```
1 2 3  5
```

---

for (parameters)

limit

Limits the loop to the specified number of iterations.

Input

```
{% for item in array limit:2 %}
  {{ item }}
{% endfor %}
```

Output

```
1 2
```

offset

Begins the loop at the specified index.

Input

```
{% for item in array offset:2 %}
  {{ item }}
{% endfor %}
```

Output

```
3 4 5 6
```

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 %}

{% assign num = 4 %}
{% for i in (1..num) %}
  {{ i }}
{% endfor %}
```

Output

```
3 4 5
1 2 3 4
```

reversed

Reverses the order of the loop. Note that this flag's spelling is different from the filter `reverse`.

Input

```
{% for item in array reversed %}
  {{ item }}
{% endfor %}
```

Output

```
6 5 4 3 2 1
```

---

cycle

Loops through a group of strings and prints them in the order that they were passed as arguments. Each time `cycle` is called, the next string argument is printed.

`cycle` must be used within a for loop block.

Input

```
{% cycle "one", "two", "three" %}
{% cycle "one", "two", "three" %}
{% cycle "one", "two", "three" %}
{% cycle "one", "two", "three" %}
```

Output

```
one
two
three
one
```

Uses for `cycle` include:

- applying odd/even classes to rows in a table
- applying a unique class to the last product thumbnail in a row

---

cycle (parameters)

`cycle` accepts a “cycle group” parameter in cases where you need multiple `cycle` blocks in one template. If no name is supplied for the cycle group, then it is assumed that multiple calls with the same parameters are one group.

Input

```
{% cycle "first": "one", "two", "three" %}
{% cycle "second": "one", "two", "three" %}
{% cycle "second": "one", "two", "three" %}
{% cycle "first": "one", "two", "three" %}
```

Output

```
one
one
two
two
```

---

tablerow

Generates an HTML table. Must be wrapped in opening `<table>` and closing `</table>` HTML tags.

Input

```
{% tablerow product in collection.products %}
  {{ product.title }}
{% endtablerow %}
```

Output

```
```

Cool ShirtAlien PosterBatman PosterBullseye ShirtAnother Classic VinylAwesome Jeans

---

tablerow (parameters)

cols

Defines how many columns the tables should have.

Input

```
{% tablerow product in collection.products cols:2 %}
  {{ product.title }}
{% endtablerow %}
```

Output

```
```

Cool Shirt Alien Poster
Batman Poster Bullseye Shirt
Another Classic Vinyl Awesome Jeans

limit

Exits the tablerow after a specific index.

```
{% tablerow product in collection.products cols:2 limit:3 %}
  {{ product.title }}
{% endtablerow %}
```

offset

Starts the tablerow after a specific index.

```
{% tablerow product in collection.products cols:2 offset:3 %}
  {{ product.title }}
{% endtablerow %}
```

range

Defines a range of numbers to loop through. The range can be defined by both literal and variable numbers.

```
{% assign num = 4 %}

{% tablerow i in (1..num) %}
  {{ i }}
{% endtablerow %}
```

{% tablerow i in (3..5) %}{{ i }}{% endtablerow %}

Raw

Raw temporarily disables tag processing. This is useful for generating content (eg, Mustache, Handlebars) which

uses conflicting syntax.

Input

```
{% raw %}
 In Handlebars, {{ this }} will be HTML-escaped, but
 {{{ that }}} will not.
{% endraw %}
```

Output

```
In Handlebars, {{ this }} will be HTML-escaped, but {{{ that }}} will not.
```

Variable

Variable tags create new Liquid variables.

assign

Creates a new variable.

Input

```
{% assign my_variable = false %}
{% if my_variable != true %}
 This statement is valid.
{% endif %}
```

Output

```
This statement is valid.
```

Wrap a variable value in quotations `"` to save it as a string.

Input

```
{% assign foo = "bar" %}
{{ foo }}
```

Output

```
bar
```

capture

Captures the string inside of the opening and closing tags and assigns it to a variable. Variables created through `capture` are strings.

Input

```
{% capture my_variable %}I am being captured.{% endcapture %}
{{ my_variable }}
```

Output

```
I am being captured.
```

Using `capture`, you can create complex strings using other variables created with `assign`:

Input

```
{% assign favorite_food = "pizza" %}
{% assign age = 35 %}

{% capture about_me %}
I am {{ age }} and my favorite food is {{ favorite_food }}.
{% endcapture %}

{{ about_me }}
```

Output

```
I am 35 and my favourite 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.

Input

```
{% increment my_counter %}
{% increment my_counter %}
{% increment my_counter %}
```

Output

```
0
1
2
```

Variables created through the `increment` tag are independent from variables created through `assign` or `capture`.

In the example below, a variable named “var” is created through `assign`. The `increment` tag is then used several times on a variable with the same name. Note that the `increment` tag does not affect the value of “var” that was created through `assign`.

Input

```
{% assign var = 10 %}
{% increment var %}
{% increment var %}
{% increment var %}
{{ var }}
```

Output

```
0
1
2
10
```

decrement

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

Input

```
{% decrement variable %}
{% decrement variable %}
{% decrement variable %}
```

Output

```
-1
-2
-3
```

Like increment, variables declared inside `decrement` are independent from variables created through `assign` or `capture`.