Drupal Book ☰

TWIG

TABLE OF CONTENTS

`views` `602`

Drupal 10 uses **Twig 3**. Drupal 9 uses Twig 2. Drupal 8 used Twig 1.

Overview

Theme System Overview

Drupal's theme system allows a theme to have nearly complete control over the appearance of the site, which includes both the markup and the CSS used to style the markup. For this system to work, instead of writing HTML markup directly, modules return "render arrays", which are structured hierarchical arrays that include the data to be rendered into HTML, and options that affect the markup. Render arrays are ultimately rendered into HTML or other output formats by recursive calls to \Drupal\Core\Render\RendererInterface::render(), traversing the depth of the render array hierarchy. At each level, the theme system is invoked to do the actual rendering. See the documentation of \Drupal\Core\Render\RendererInterface::render() and the Theme system and Render API topic for more information about render arrays and rendering.

Twig Templating Engine

Drupal uses the templating engine Twig. Twig offers developers a fast, secure, and flexible method for building templates for Drupal 8 sites. Twig does not require front-end developers to know PHP to build and manipulate Drupal themes.

For more on theming in Drupal see https://www.drupal.org/docs/theming-drupal .

For further Twig documentation see https://twig.symfony.com/doc/2.x and https://twig.symfony.com/doc/3.x

Note. Drupal 10 uses **Twig 3**, Drupal 9 uses Twig 2 and Drupal 8 used Twig 1.

Displaying Data

Fields or Logic

Twig can do things that PHP can't such as whitespacing control, sandboxing, automatic HTML escaping, manual contextual output escaping, inclusion of custom functions and filters that only affect templates.

Double curly braces are used to output a variable. E.g.

```
{{ content.title }}
```

Brace and percent are used to put logic into Twig templates e.g. if, then, else or for loops. E.g.

```
{% if content.price is defined %}
  <h2>Price: {{ content.price }} </h2>
{% endif %}
```

Use brace and pound symbol (hash) for comments e.g.

```
{# this section displays the voting details #}
```

Here are some of the Twig functions that you can use in twig templates: https://www.drupal.org/docs/8/theming/twig/functions-in-twig-templates There are lots of them e.g.

- file_url($uri)
- link($text, $uri, $attributes)
- path($name, $parameters, $options)
- url($name, $parameters, $options)

And even more Twig fun at https://twig.symfony.com/doc/3.x/functions/index.html

Which template, which variables?

There is usually one page.tpl.php and *multiple* node templates. One node template per content type. Eg. node-news-story.html.twig, node-event.html.twig. There can also be field specific templates e.g. web/themes/custom/txg/templates/field/field--field-3-column-links.html.twig

In the page.html.twig, you can refer to variables as page.content or node.label

whereas node templates expect content.field_image or node.field_myfield

Note. If you don't see a field output for a node, try specifying the preface node. instead of content..

Field specific template are usually very simple and refer to

```
{{ items }}
```

and

```
{{ item.content }}
```

e.g. from txg/web/themes/contrib/zurb_foundation/templates/page.html.twig .

```
<section>
  {{ page.content }}
</section>
```

And from txg/web/themes/custom/txg/templates/content/page--node--event.html.twig I accidentally started implementing this in the page template. See below for the node template.

```twig
{{ drupal_field('field_image', 'node') }}

<h1>{{ node.label }}</h1>
<div>For: {{ node.field_for.0.value }}</div>
<div>DATE: {{ node.field_event_date.0.value|date('n/j/Y') }}</div>
<div>Time: {{ node.field_event_date.0.value|date('h:ia') }} - {{ node.field_event_date.0.end_value|date('h:ia') }}</div>

<div>
  Location:
    {% if node.field_event_location_link.0.url %}
    <a href="{{ node.field_event_location_link.0.url }}">{{ node.field_event_location.0.value }}</a>
    {% else %}
    {{ node.field_event_location.0.value }}
    {% endif %}
</div>

{% if node.field_event_cta_link.0.url %}
  CTA:<div class="button"><a href="{{ node.field_event_cta_link.0.url }}">{{ node.field_event_cta_link.0.title }}</a></div>
{% endif %}
```

Here is the same basic stuff (as above) but implemented in the node template at txg/web/themes/custom/txg/templates/content/node--event.html.twig:

> Note. That node.label becomes label and node.field_for becomes content.field_for.

```twig
<h1>{{ label }}</h1>
{{ content.field_image }}
<div>Node: {{ node.id }}</div>
<div>For: {{ content.field_for }}</div>
<div>DATE: {{ node.field_event_date.0.value|date('n/j/Y') }}</div>
<div>Time: {{ node.field_event_date.0.value|date('h:ia') }} - {{ node.field_event_date.0.end_value|date('h:ia') }}</div>
<div>
  Location:
  {% if node.field_event_location_link.0.url %}
    <a href="{{ node.field_event_location_link.0.url }}">{{ node.field_event_location.0.value }}</a>
  {% else %}
    {{ node.field_event_location.0.value }}
  {% endif %}
</div>

{% if node.field_event_cta_link.0.url %}
  CTA:<div class="button"> <a href="{{ node.field_event_cta_link.0.url }}">{{ node.field_event_cta_link.0.title }}</a></div>
{% endif %}
```

Display fields or variables

Using node.field_myfield will bypass the rendering and display any markup in the field. Using content.field_myfield uses the rendering system and is the preferred way to display your content.

This will display all the content rendered

```twig
{{ content }}
```

Node Title with and without a link

Render node title (or label) (with markup – so it may include <span> tags)

```twig
{{ label }}
```

Render node label (without markup – no html in this version)

```
{{ node.label }}
```

Render link to node

```
<a href="{{ url }}">{{ label }}</a>
```

// Or a little more complex..

```
<div class="title"><a href="{{ url }}">{{ label }}</a> | <span>{{ content.field_vendor_ref }}</span></div>
```

Fields

There are many ways to limit things and only show some of the content. Mostly often you will need to show specific fields. Note. This will include rendered info such as labels etc.

```
{{ content.field_yomama }}
```

or

```
{{ content.field_ref_topic }}
```

Any field – just jam content. in front of it

```
{{ content.field_intl_students_and_scholars }}
```

You can also grab node specific fields if content. type fields don't do the trick.

In a node template, you can display specific node fields by prefacing them with node e.g.:

```
{{ node.id }}
{{ node.label }}
{{ node.field_date.value }}
{{ node.field_date.end_value }}
```

Paragraph field

These still work fine: content.field_abc or node.field_ref_topic but instead of node, you preface fields with paragraph like this:

```
termid0: {{ paragraph.field_ref_tax.0.target_id }}
termid1: {{ paragraph.field_ref_tax.1.target_id }}
```

and we get this result if we have selected two terms 13 and 16.

```
termid0: 13
termid1: 16
```

To dump a taxonomy reference field for debugging purposes use the code below. The pre tags format it a little nicer than if we don't have them.

```
<pre>
{{ dump(paragraph.field_ref_tax.value) }}
</pre>
```

Loop thru paragraph reference fields

Here we go looping thru all the values in a multi-value reference field.

```twig
{% for tax in paragraph.field_ref_tax %}
  <div>target_id: {{ tax.target_id }}</div>
{% endfor %}
```

It's the same as outputting these guys:

```twig
termid0: {{ paragraph.field_ref_tax.0.target_id }}
termid1: {{ paragraph.field_ref_tax.1.target_id }}
```

and to make this more useful, here we build a string of them to pass to a view.

From: dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--news-preview.html.twig

```twig
{# Figure out parameters to pass to view for news items #}
{% set params = '' %}
{% for item in paragraph.field_ref_tax_two %}
  {% set params = params ~ item.target_id %}
  {% if not loop.last %}
    {% set params = params ~ '+' %}
  {% endif %}
{% endfor %}
params: {{ params }}
```

This will output something like: 5+6+19

Body

```twig
{{ content.body }}
```

Or

```twig
{{ node.body.value }}
```

And for summary

```twig
{{ node.body.summary | raw }}
```

Multi-value fields

Fields that you preface with `node.` can also handle an index (the `0` below) i.e. to indicate the first value in a multi-value field, 1 to indicate the second etc.

```twig
{{ node.field_iso_n3_country_code.0.value }}
```

Fields with HTML

If a field has html that you want rendered, use the keyword raw. Be aware this has security considerations which you can mitigate using [striptags](#) filters:

```twig
<div>How to order: {{ how_to_order|raw }}</div>
```

And maybe you want to only allow <b> tags

```twig
{{ word|striptags('<b>')|raw }}
```

Or several tags. In this case <b><a><pre>

```twig
{{ word|striptags('<b>,<a>,<pre>')|raw }}
```

The date/time a node is published, updated or created

Each of these calls return a datetime value in string form which can be massaged by the twig date() function for formatting.

```
<pre>
Created:  {{ node.created.value }}
Created:  {{ node.createdtime }}
Created:  {{ node.created.value|date('Y-m-d') }}

Modified:  {{ node.changed.value }}
Modified:  {{ node.changedtime }}
Modified:  {{ node.changed.value|date('Y-m-d') }}

Published: {{ node.published_at.value }}
Published: {{ node.published_at.value|date('Y-m-d') }}
</pre>
```

Here is the output you might see. Note. The first published is apparently blank because I didn't use the drupal scheduling to publish the node (maybe?) and the second one seems to have defaulted to today's date.

```
Created: 1604034000
Created: 1604034000
Created: 2020-10-30
Modified: 1604528207
Modified: 1604528207
Modified: 2020-11-04
Published:
Published: 2020-11-20
```

Updated/changed

```
{% set post_date = node.changedtime %}
```

Created (same as authored on date on node edit form):

```
{{ node.createdtime }}
```

And pretty formatted like Sep 2, 2023

```
{{ node.createdtime\|date('M d, Y') }}
```

Also

```
<div class="date">Date posted: {{ node.getCreatedTime|date('m/d/Y') }}</div>
<div class="date">Date posted: {{ node.getChangedTime|date('m/d/Y') }}</div>
```

Node published date:

```
Date published: {{ _context.node.published_at.value }}
Date published: {{ node.published_at.value }}
```

Format a date field

Use the field's format settings; include wrappers. This example includes wrappers.

```
{{ content.field_blog_date }}
```

The examples below do not include wrappers.

Use the field's format settings. This will use the format defined in Content type » Manage Displays »Your View Mode.

```
{{ content.field_blog_date.0 }}
```

Using Twig date filter and a defined Drupal date format

```
{{ node.field_blog_date.value|date('U')|format_date('short_mdyyyy') }}
```

Use Twig date filter

```
{{ node.field_blog_date.value|date('n/j/Y') }}
```

Smart date field formatting

When using the smart date module, dates are stored as timestamps so you have to use the twig date function to format them. If you just put this in your template:

```
{{ content.field_when }}
```

The output will include whichever formatting you specify in Drupal. While I assume there is a way to pass a smart date formatting string to twig, I haven't discovered it yet. Here are ways to format a smart date.

Specify the index (the 0 indicating the first value, or 1 for the second) e.g. node.field.0.value and pipe the twig date function for formatting:

Date as in July 18, 2023

```
{{ node.field_when.0.value|date('F j, Y') }}
```

End date

```
{{ node.field_when.0.end_value|date('F j, Y') }}
```

Timezone as in America/Chicago

```
{{ node.field_when.0.value|date('e') }}
```

Timezone as in CDT

```
{{ node.field_when.0.value|date('T') }}
```

Day of the week

```
{{ node.field_when.0.value|date('l') }} {# day of week #}
```

Hide the end date if it is the same as the start date

```
{% set start = node.field_when.0.value|date('l F j, Y') %}
{% set end = node.field_when.0.end_value|date('l F j, Y') %}
  <p class="date"> {{ start }}</p>
{% if not start is same as(end) %}
  <p class="date"> {{ end }}</p>
{% endif %}
```

Entity Reference field

If you have an entity reference field such as field_ref_topic (entity reference to topic content) you have to specify the target_id like this. If you have only 1 reference, use the .0, for the second one use .1 and so on.

```
{{ node.field_ref_topic.0.target_id }}
```

Note. This will show the node id of the entity reference field. See below to see the content that the entity reference field points to.

Entity reference destination content

If you have an entity reference and you want to display the content from the node that is referenced i.e. if you have a contract with a reference to the vendor node and you want to display information from the vendor node on the contract you can dereference fields in the entity destination:

From dirt/web/themes/custom/dirt_bootstrap/templates/content/node--contract--vendor-list.html.twig:

```
{{ node.field_sf_contract_ref.entity.field_contract_overview.value }}
```

Or

```
{{ content.field_sf_contract_ref.entity.field_contract_overview }}
```

The field in the contract node is called field_sf_contract_ref. The field in the referenced entity is called field_contract_overview. Notice how with the node. style, you must specify .value at the end.

Here is an example of a taxonomy term where the title of the term will be displayed.

```
<pre>
Dump category:
{{ dump(node.field_ref_tax.entity.label) }}
</pre>
```

Taxonomy term

Here is an example of displaying a taxonomy term.

```
<pre>
Dump category: {{ dump(node.field_ref_tax.entity.label) }}
</pre>
```

Render a block

Example block with a machine name of block---system-powered-by-block.html.twig from a custom theme

```
{%
  set classes = [
    'block',
    'block-' ~ configuration.provider|clean_class,
    'block-' ~ plugin_id|clean_class,
  ]
%}
<div{{ attributes.addClass(classes) }}>
  {{ title_prefix }}
  {% if label %}
    <h2{{ title_attributes }}>{{ label }}</h2>
  {% endif %}
  {{ title_suffix }}
  {% block content %}
    {{ content }}
  {% endblock %}
  also powered by <a href="http://austinprogressivecalendar.com">Austin Progressive Calendar</a>
</div>
```

Render a list created in the template_preprocess_node()

Here we create a list in the function:

```php
function burger_theme_preprocess_node(&$variables) {

  $burger_list = [
    ['name' => 'Cheesburger'],
    ['name' => 'Mushroomburger'],
    ['name' => 'Chickenburger'],
  ];
  $variables['burgers'] = $burger_list;
}
```

and render it in the node--article--full.html.twig

```twig
<ol>
  {% for burger in burgers %}
  <li>{{ burger['name'] }}</li>
  {% endfor %}
</ol>
```

Links

There are a bajillion kertrillion or more ways to render a link

Link field (URL)

This is the simplest way. Just set the display mode to link

And output the link without a label.

```twig
{{ content.field_suggest_button }}
```

If you need a little more control you might use this version which allows classes etc. We are adding several classes onto the anchor to make it look like a button. In this case with an internal link, it shows up using the alias of the link i.e. it shows /contracts instead of node/7 when you hover over the link.

```twig
<p><a class="btn secondary navy centered" href="{{ node.field_suggest_button.0.url }}">{{ node.field_suggest_button.0.title }}</a></p>
```

Using .uri causes the link (internal only. External links are fine) to show up as node/7 when you hover over the link.

```twig
<p><a class="btn secondary navy centered" href="{{ node.field_suggest_button.uri }}">{{ node.field_suggest_button.0.title }}</a></p>
```

Don't try this as it won't work:

```twig
//bad
{{ node.field_suggest_button.url }}.
//bad
```

Want to use the text from a different field? No problem.

```twig
<div class="title"><a href="{{ node.field_link.uri }}">{{ node.field_contract_number.value }}</a></div>
```

Links to other pages on site

Absolute link:

```
<a href="{{ url('entity.node.canonical', {node: 3223}) }}">Link to Weather Balloon node 3223 </a>
```

Relative link

See path vs url:

```
<a href="{{ path('entity.node.canonical', {node: 3223}) }}">Link to Weather Balloon node 3223 </a>
```

Link to a user using user id

You can link to users using the following:

```
<a href="{{ url('entity.user.canonical', {user: 1}) }}">Link to user 1 </a>
```

External link in a field via an entity reference

Here we have a node with an entity reference field (`field_sf_contract_ref`) to another entity.

In a preprocess function, you can grab the link. Note, you can just grab the `first()` one. Later on you can see that in the twig template, you can specify the first one with `.0`

From dirt/web/themes/custom/dirt_bootstrap/dirt_bootstrap.theme

```
$vendor_url = $node->field_sf_contract_ref->entity->field_vendor_url->first();
if ($vendor_url) {
  $vendor_url = $vendor_url->getUrl();
  if ($vendor_url) {
    $variables['vendor_url'] = $vendor_url->getUri();
  }
}
```

And in the template we retrieve the URI with `.uri`:

```
<p><a class="styled-link ext" href="{{ node.field_sf_contract_ref.entity.field_vendor_url.uri }}">Vendor Website</a></p>
```

Here we check if there is a target value and output that also. E.g. `target="_blank"` and also display the title – this is the anchor title as in the words "Vendor Website" below

```
<a href="https://www.duckduckgo.com">Vendor Website</a></p>
```

From `inside-marthe/themes/custom/dp/templates/paragraph/paragraph--sidebar-product-card.html.twig` we wrap some stuff in a link:

```
<a href="{{content.field_link.0['#url']}}" {% if content.field_link.0['#options']['attributes']['target'] %} target="{{content.field_link.0['#options']['attributes']['target']}}" {% endif %} class="b
  {{ content.field_image }}
  <h2 class="module-header">{{content.field_text}}</h2>
  {{content.field_text2}}
</a>
```

And from `txg/web/themes/custom/txg/templates/content/node--event--card.html.twig` if there is a url, display the link with the url, otherwise just display the title for the link. I'm not 100% sure this is really valid. Can you put in a title and no link?

```twig
{% if node.field_event_location_link.0.url %}
    <a href="{{ node.field_event_location_link.0.url }}">{{ node.field_event_location.0.value }}</a>
{% else %}
    {{ node.field_event_location.0.value }}
{% endif %}
```

Render an internal link programatically

Here we want to render an internal link to a page on our Drupal site (as opposed to a link to another site.) We grab the link in a preprocess function. Extract out the title and the URI.

```php
$instructions_node = Node::load($order_type_instructions_nid);
if ($instructions_node) {
  $order_link = $instructions_node->field_link->first();
  if ($order_link) {
    $uri = $order_link->uri;
    $variables['order_link_title'] = $order_link->title;
    $order_url = $order_link->getUrl();
    if ($order_url) {
      $variables['order_type_link'] = $order_url;
    }
  }
}
```

We can put the pieces in the twig template like this

```twig
<a href="{{ order_type_link }}">{{ order_link_title }}</a>
```

Render an image with an image style

From inside-marthe/themes/custom/dp/templates/paragraph/paragraph--sidebar-resource.html.twig

Here we use sidebar_standard image style

```twig
<aside class="module module--featured" data-interchange="[{{ content.field_image.0['#item'].entity.uri.value | image_style('sidebar_standard') }}, small]">
```

Or for a media field, set the image style on the display mode and use this:

```twig
{{ content.field_banner_image.0 }}
```

Hide if there is no content in a field or image

From inside-marthe/themes/custom/dp/templates/content/node--video-detail.html.twig I check to see if there are any values in this array related_lessons_nids and display the view.

```
{% if related_lessons_nids|length %}
  <div class="section section--featured">
    <div class="grid-container">
      <h2 class="section-header text-center large-text-left">Related Lessons</h2>
      <div class="grid-x grid-margin-x" data-equalizer data-equalize-on="large">
        {{ drupal_view('video', 'embed_collection_related_lessons', related_lessons_nids|join(', ')) }}
      </div>
    </div>
  </div>
{% endif %}
```

Not empty:

```
{% if content.field_myfield is not empty %}
  {# Do something here #}
{% endif %}
```

Hide if there is no image present

If there is an image (and it is renderable) display the image

```
{% if content.field_teacher_commentary_image|render %}
  <img src="{{file_url( content.field_teacher_commentary_image['#items'].entity.uri.value ) }}" width="420" height="255" alt="" class="left">
{% endif %}
```

Attributes

From https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes:

Elements in HTML have **attributes**; these are additional values that configure the elements or adjust their behavior in various ways to meet the criteria the users want.

Read more about using attributes in templates https://www.drupal.org/docs/8/theming-drupal-8/using-attributes-in-templates

To add a data attribute use:

```
{{ attributes.setAttribute('data-myname','tommy') }}
```

e.g.

```
<article{{ attributes.addClass(classes).setAttribute('my-name', 'Selwyn') }}>
```

Produces:

```
<article
  data-history-node-id="3224"
  data-quickedit-entity-id="node/3224"
  role="article"
  class="contextual-region node node--type-article node--promoted node--view-mode-full"
  about="/burger1"
  typeof="schema:Article"
  my-name="Selwyn"
  data-quickedit-entity-instance-id="0"
></article>
```

More useful examples at https://www.drupal.org/docs/8/theming-drupal-8/using-attributes-in-templates such as:

```
{% set classes = ['red', 'green', 'blue'] %}
{% set my_id = 'specific-id' %}
{% set image_src = 'https://www.drupal.org/files/powered-blue-135x42.png' %}

<img{{ attributes.addClass(classes).removeClass('green').setAttribute('id', my_id).setAttribute('src', image_src) }}>
```

Which outputs the following:

```
<img
  id="specific-id"
  class="red blue"
  src="https://www.drupal.org/files/powered-blue-135x42.png"
/>
```

Check if an attribute has a class

```
{{ attributes.hasClass($class) }}
```

Remove an attribute

```
{{ attributes.removeAttribute() }}
```

Convert attributes to array

```
{{ attributes.toArray () }}
```

Output the content but leave off the field_image

From very/web/themes/very/templates/node--teaser.html.twig:

```
<div{{ content_attributes.addClass('content') }}>
  {{ content|without('field_image')|render|striptags }}
</div>
```

Add a class

```
<div{{ content_attributes.addClass('node__content') }}>
```

Add a class conditionally

From very/web/themes/very/templates/node--teaser.html.twig

For an unpublished node, wrap this class around the word unpublished

```
{% if not node.published %}
  <p class="node--unpublished">{{ 'Unpublished'|t }}</p>
{% endif %}
```

Links to other pages on site

Absolute:

```
<a href="{{ url('entity.node.canonical', {node: 3223}) }}">Link to WEA node 3223 </a>
```

Relative (see path vs url):

```
<a href="{{ path('entity.node.canonical', {node: 3223}) }}">Link to WEA node 3223 </a>
```

Could also link to users using

```
<a href="{{ url('entity.user.canonical', {user: 1}) }}">Link to user 1 </a>
```

Loop.index in a paragraph twig template

From: web/themes/custom/dprime/templates/field/field--paragraph--field-links--sidebar-cta.html.twig

Notice the use of loop.index to only output this for the first item

```
{% for item in items %}
  {% if loop.index == 1 %}
    <div class="cell medium-6">
      <a href="{{item.content['#url']}}" class="button {% if loop.index == 2 %}hollow {% endif %}button--light m-b-0"{% if item.content['#options']['attributes']['target'] %} target="{{item.
    </div>
  {% endif %}
{% endfor %}
```

Loop thru an array of items with a separator

This loads all the authors and adds and between them except for the last one:

```
<div>
  {%- if content.author -%}
    by
  {%- for author in content.author -%}
    {% if loop.last %}
      {% set separator = '' %}
    {% else %}
      {% set separator = ' and ' %}
    {% endif %}
    {{ author }} {{ separator }}
  {%- endfor -%}
  {%- endif -%}
</div>
```

This version inserts commas:

From org/docroot/themes/custom/org/templates/field/field--node--field-categories--opinion.html.twig

```
{% if label_hidden %}
  {% if multiple %}
    {% for item in items %}
      {%if loop.index > 1 %}, {% endif %}{{ item.content }}
    {% endfor %}
  {% else %}
    {% for item in items %}
      {%if loop.index > 1 %}, {% endif %}{{ item.content }}
    {% endfor %}
  {% endif %}
{% else %}
  <div{{ title_attributes }}>{{ label }}</div>
  {% for item in items %}
    {%if loop.index > 1 %}, {% endif %}{{ item.content }}
  {% endfor %}
{% endif %}
```

Add Javascript into a twig template

```
<script>
 function hidePRSnippet(config, data) {
  if (data.average_rating < 3.5 && data.average_rating !== 0) {
   document.getElementById('pr-reviewsnippet').remove();
  }
 }

 var powerReviewsConfig = {{ accPowerreviews | raw}};
 for (var i = 0; i < powerReviewsConfig.length; i++) {
  if (powerReviewsConfig[i].hasOwnProperty('components') && powerReviewsConfig[i].components.hasOwnProperty('ReviewSnippet')) {
   powerReviewsConfig[i]['on_render'] = hidePRSnippet;
  }
 }
 POWERREVIEWS.display.render(powerReviewsConfig);
</script>
```

Control/Logic

Concatenate values into a string with join

This would typically be used when passing a series of node id's to a view to filter its output.

```
{% set blah = [node.field_ref_unit.0.target_id,node.field_ref_unit.1.target_id,node.field_ref_unit.2.target_id,node.field_ref_unit.3.target_id]|join('+') %}
```

This produces 1+2+3+4

Include partial templates

```
{% include '@txg/partials/searchfilterform.html.twig' %}
```

Loop through entity reference items

In txg/web/themes/custom/txg/templates/content/node--news-story.html.twig I need to loop through a bunch of entity reference values and build a string of id+id+id… (with an undefined number) so

```
{% set blah = '' %}
{% for item in node.field_ref_unit %}
 {% set blah = blah ~ item.target_id %}
 {% if not loop.last %}
  {% set blah = blah ~ '+' %}
 {% endif %}
{% endfor %}

<div>blah:{{ blah }}</div>
<div>node id: {{ node.id }}</div>
{{ drupal_view('related_news_for_news_story', 'block_unit', node.id, blah) }}
```

IF OR

If there is a value in field_event_date or field_display_date, then display it/them.

```
<div{{ content_attributes.addClass('teaser__content') }}>
 {% if content.field_event_date or content.field_display_date %}
  <div class="teaser__date">
   {{ content.field_event_date|render|striptags }}
   {{ content.field_display_date|render|striptags }}
  </div>
 {% endif %}
```

Test if a formatted text field is empty

To check a body field or other formatted text field, use |render to render it first.

```twig
{% if content.body|render %}
  <li><a class="scroll" href="#section-overview">Overview</a></li>
{% endif %}
```

Test empty variable

This code checks if a variable is empty using empty test if the attributes variable is not set. From
https://www.drupal.org/project/drupal/issues/2558079:

```twig
{% if attributes is empty %}
  {{ link(item.title, item.url) }}
{% else %}
  {{ link(item.title, item.url, attributes) }}
{% endif %}
```

You can also use:

```twig
{% if blah is not empty %}
  {{content.name}}
{% endif %}
```

Conditionals (empty, defined, even)

```twig
{% if rows %}
  {{rows}}
{% elseif empty %}
  {{ empty }}
{% endif %}

{% if var is defined %}
  {{ content.name}}
{% endif %}

{%if var is even %}
  {{ content.name}}
{% endif %}
```

e.g. from inside-marthe/themes/custom/dp/templates/paragraph/paragraph--highlight-card.html.twig

```
{% set showCat = TRUE %}
{% if view_mode == 'overview' or view_mode == 'home' %}
  {% set showCat = FALSE %}
{% endif %}
<div class="cell medium-4 home-highlight-card m-b-3" data-equalizer-watch>
  <a href="{% spaceless %}{{ content.field_link.0 }}{% endspaceless %}" class="card h-100" data-interchange="[{{ content.field_image.0['#item'].entity.uri.value | image_style('high
    <div class="card-content">
      {%  if showCat %}
        <span class="card-label">{{content.field_text2}}</span>
      {% endif %}
      <h3 class="card-header">{{content.field_text}}</h3>
      {{content.field_description}}
      {% if content.field_button_text|render %}
        <span class="button full-width show-on-hover">{{ content.field_button_text }}</span>
      {% else %}
        <span class="button full-width show-on-hover">{{ content.field_text|render }}</span>
      {% endif %}
    </div>
  </a>
</div>
```
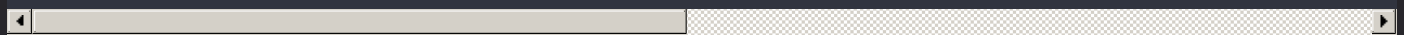
Test if a paragraph is empty using striptags

From /inside-marthe/themes/custom/dp/templates/content/node--video-collection.html.twig:

Normally you wouldn't need the striptags, but when twig debugging is enabled, the render information includes debug tags. See https://www.drupal.org/project/drupal/issues/2547559#comment-12103048

```
{% if content.field_related_lessons|render|striptags|trim is not empty %}
  {{ content.field_related_lessons}}
{% endif %}
```

Or this much simpler version which also comes from the same issue page above (it doesn't seem to work as well as the version above):

```
{% if content.field_related_lessons.value %}
  {{ content.field_related_lessons}}
{% endif %}
```

Comparing strings

For complicated strings, you have to use the Twig [same as](#) function because using if x == y doesn't work. See the commented out part where I tried ==:

```
{% set start = node.field_when.0.value|date('l F j, Y') %}
{% set end = node.field_when.0.end_value|date('l F j, Y') %}
<p class="date"> {{ start }}</p>
{#{% if not start == end %}#}
{% if not start is same as(end) %}
  <p class="date"> {{ end }}</p>
{% endif %}
```

Include other templates as partials

In very/web/themes/very/templates/node--featured.html.twig

You can re-use templates. Just put them in the partials directory (you don't have to but it is a good convention) and include them.

```
{{ include('node--teaser.html.twig') }}
```

Check if an attribute has a class

```
{{ attributes.hasClass($class) }}
```

Remove an attribute

```
{{ attributes.removeAttribute() }}
```

Convert attributes to array

```
{{ attributes.toArray () }}
```

Views

Render a view with contextual filter

> Pro tip: Create `embed` displays (rather than blocks or pages) so users don't see these blocks appearing in the block management page. see https://drupal.stackexchange.com/questions/287209/what-does-the-embed-display-type-do

To use a field value in a view as an argument, using twig_tweak, you can render the view and its arguments/parameters. In the example below, these are the contextual filters defined in the view.

```
{{ drupal_view('map_data_for_a_country', 'block_stats', node.field_iso_n3_country_code.0.value) }}
```

> Note. Using content.field as a parameter doesn't work because content.fields get rendered so they are usually filled with HTML or labels or both. Parameters need to simply be numbers or strings.

Other examples. Here an entity reference field is passed as a parameter. This works for taxonomy terms like this also.

```
{{ drupal_view('news_stories_for_a_topic','block_1', node.field_ref_topic.0.target_id) }}
```

Or

```
{{ drupal_view('resellers_for_this_vendor', 'embed_1', node.field_vendor_id.value ) }}
```

Note. If you ever see a 502 bad gateway error when embedding a drupal_view, delete the display and create a new one and it may just work fine.

Count how many rows returned from a view

https://www.drupal.org/docs/8/modules/twig-tweak/twig-tweak-and-views

Check if View has Results

```
{% set view = drupal_view_result('related', 'block_1')|length %}
{% if view > 0 %}
  {{ drupal_view('related', 'block_1') }}
{% endif %}
```

If view results empty, show a different view

In `txg/web/themes/custom/txg/templates/content/node--news-story.html.twig` we show units (the first view) but if there aren't any, show `aofs` (the second view.)

```
{% if drupal_view_result('related_news_for_news_story', 'block_unit', node.id, unit_ids) %}
  {{ drupal_view('related_news_for_news_story', 'block_unit', node.id, unit_ids) }}
{% elseif drupal_view_result('related_news_aof', 'block_aof', node.id, aof_ids) %}
  {{ drupal_view('related_news_aof', 'block_aof', node.id, aof_ids) }}
{% endif %}
```

Selectively pass 1 termid or 2 to a view as the contextual filter

In the view, you can allow multiple terms for a contextual filter

From: https://drupal.stackexchange.com/questions/78701/views-multiple-contextual-filters-taxonomy:

Instead of Content: The name of Taxonomy (taxonomy_vocabulary_#) you need to select Content: Has taxonomy term ID *Contextual filter* and enable *Allow multiple values* to able to use multiple values in the form of 1+2+3 (for OR) or 1,2,3 (for AND).

Then in the template, check if there is a second value and build the arguments in the form id+id (e.g. "13+16") In this example, I have to assume the setup allows only 2 taxonomy terms to be entered. See below for an unlimited amount of terms.

From /Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--upcoming-events.html.twig:

```twig
{# if there is a second category, pass it separated by + #}
{% if paragraph.field_ref_tax.1.target_id %}
  {% set args = paragraph.field_ref_tax.1.target_id~'+'~paragraph.field_ref_tax.1.target_id %}
  args: {{ args }}
  {{ drupal_view('events', 'embed_2', paragraph.field_ref_tax.0.target_id~'+'~paragraph.field_ref_tax.1.target_id) }}
{% else %}
  {{ drupal_view('events', 'embed_2', paragraph.field_ref_tax.0.target_id) }}
{% endif %}
```

Or even nicer, we could loop thru an unlimited number of terms, build a string of them to pass to a view.

From: /Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/paragraphs/paragraph--news-preview.html.twig

```twig
{# Figure out parameters to pass to view for news items #}
{% set params = '' %}
{% for item in paragraph.field_ref_tax_two %}
  {% set params = params ~ item.target_id %}
  {% if not loop.last %}
    {% set params = params ~ '+' %}
  {% endif %}
{% endfor %}
params: {{ params }}
```

This will output something like: 5+6+19

And pass the output to a view like this:

```twig
{{ drupal_view('news', 'embed_2', params) }}
```

Views templates

Using machine names for the view, you can copy the base view templates (just like in Drupal 7) to make specific templates.

From https://www.drupal.org/docs/theming-drupal/twig-in-drupal/twig-template-naming-conventions:

Using a View named foobar with its style: unformatted and row style: Fields, and using display:Page.

```
views-view--foobar--page.html.twig
views-view--page.html.twig
views-view--foobar.html.twig
views-view.html.twig


views-view-unformatted--foobar--page.html.twig
views-view-unformatted--page.html.twig
views-view-unformatted--foobar.html.twig
views-view-unformatted.html.twig


views-view-fields--foobar--page.html.twig
views-view-fields--page.html.twig
views-view-fields--foobar.html.twig
views-view-fields.html.twig
```

or for views-view-list.html.twig, you could use views-view-list---foobar.html.twig

e.g. /Users/selwyn/Sites/dirt/web/themes/custom/dirt_bootstrap/templates/views/views-view-list--resource-library.html.twig

Inject variables

You can inject variables into a view using hook_preprocess_views_view() eg. from txg/web/themes/custom/txg/txg.theme. The code below used to was load up various items to populate the select dropdown controls in the view:

```
function txg_preprocess_views_view(&$variables) {
  $view = $variables['view'];
  $id = $view->storage->id();
  $display_id = $view->current_display;


  // Build /newsroom/search
  if ($id == 'news_events_search' && $display_id == 'page_news') {
    $variables['filter_data'] = generate_search_filter_data();
  }
}
```

Here is the code that builds the data for the select controls:

```
/**
 * Populate dropdowns.
 *
 * Builds the search filter data to populate select dropdowns on
 * /newsroom, /newsroom-search etc. pages.
 *
 * @return array
 *   Array of values for dropdown.
 *
 * @throws \Drupal\Component\Plugin\Exception\InvalidPluginDefinitionException
 * @throws \Drupal\Component\Plugin\Exception\PluginNotFoundException
 */
function generate_search_filter_data() {

  // Grab the Get parameters.
  $country_nid = \Drupal::request()->query->get('country');
  $aof_nid = \Drupal::request()->query->get('aof');
  $unit_nid = \Drupal::request()->query->get('unit');
  $continent_arg = \Drupal::request()->query->get('continent');
  $topic_tid = \Drupal::request()->query->get('topic');

  // Office: AOF and child units from main menu.
  $office_nid = 0;
```

```php
    if (!empty($aof_nid)) {
      if (is_numeric($aof_nid)) {
        $office_nid = $aof_nid;
      }
    }
    if (!empty($unit_nid)) {
      if (is_numeric($unit_nid)) {
        $office_nid = $unit_nid;
      }
    }

    $storage = get_offices($office_nid);
    $data[] = [
      'type' => 'office',
      'info' => $storage,
    ];

    // Topic taxonomy terms.
    $vid = 'topic';
    $terms = \Drupal::entityTypeManager()->getStorage('taxonomy_term')->loadTree($vid);
    $storage = [];
    foreach ($terms as $term) {
      $storage[] = [
        "value" => $term->tid,
        "title" => $term->name,
      ];
    }
    // Update the select to show the current value from the url.
    foreach ($storage as &$item) {
      if ($item['value'] == $topic_tid) {
        $item['selected'] = 'selected';
        break;
      }
    }
    $data[] = [
      'type' => 'topic',
      'info' => $storage,
    ];

    // Continent.
    $continents = [
      ['title' => 'Africa', 'value' => 'Africa'],
      ['title' => 'Antarctica', 'value' => 'Antarctica'],
      ['title' => 'Asia', 'value' => 'Asia'],
      ['title' => 'Europe', 'value' => 'Europe'],
      ['title' => 'North America', 'value' => 'north%20america'],
      ['title' => 'Oceania', 'value' => 'Oceania'],
      ['title' => 'South America', 'value' => 'south%20america'],
    ];
    // Update the select to show the current value from the url.
    foreach ($continents as &$continent) {
      if (!empty($continent_arg)) {
        if (strtolower($continent['title']) == strtolower($continent_arg)) {
          $continent['selected'] = 'selected';
        }
      }
    }
    $data[] = [
      'type' => 'continent',
```

```
    'info' => $continents,
  ];


  // Country.
  $data_storage = Drupal::entityTypeManager()->getStorage('node');
  $query = Drupal::entityQuery('node')
    ->condition('status', 1)
    ->condition('type', 'country')
    ->sort('title', 'ASC');
  $nids = $query->execute();
  $nodes = $data_storage->loadMultiple($nids);
  $storage = [];
  foreach ($nodes as $node) {
    $storage[] = [
      'title' => $node->getTitle(),
      'value' => $node->id(),
    ];
  }
  // Set the default country so it appears in the select dropdown.
  foreach ($storage as &$item) {
    if ($item['value'] == $country_nid) {
      $item['selected'] = 'selected';
      break;
    }
  }
  $data[] = [
    'type' => 'country',
    'info' => $storage,
  ];


  return $data;
}
```

Here the output for all views uses the views-view.html.twig template

```
<!-- BEGIN OUTPUT from 'core/themes/classy/templates/views/views-view.html.twig' -->
```

If we want to override the frontpage view we can copy the template from above to our theme and rename it views-view--frontpage.html.twig

Notice that it will override all displays (in this case the page and the feed displays – "page_1" and "feed_1" respectively) so we can be more specific

Rename it to views-view--frontpage--page_1.html.twig

From https://api.drupal.org/api/drupal/core%21modules%21views%21views.theme.inc/group/views_templates/8.5.x

All views templates can be overridden with a variety of names, using the view, the display ID of the view, the display type of the view, or some combination thereof.

For each view, there will be a minimum of two templates used. The first is used for all views: views-view.html.twig.

The second template is determined by the style selected for the view. Note that certain aspects of the view can also change which style is used; for example, arguments which provide a summary view might change the style to one of the special summary styles.

The default style for all views is views-view-unformatted.html.twig.

Many styles will then farm out the actual display of each row to a row style; the default row style is views-view-fields.html.twig.

Here is an example of all the templates that will be tried in the following case:

View: foobar Style: unformatted Row style: Fields. Display:Page.

- views-view--foobar--page.html.twig

- views-view--page.html.twig

- views-view--foobar.html.twig

- views-view.html.twig

- views-view-unformatted--foobar--page.html.twig

- views-view-unformatted--page.html.twig

- views-view-unformatted--foobar.html.twig

- views-view-unformatted.html.twig

- views-view-fields--foobar--page.html.twig

- views-view-fields--page.html.twig

- views-view-fields--foobar.html.twig

- views-view-fields.html.twig

> **IMPORTANT**
>
> When adding a new template to your theme, be sure to flush the theme registry cache!

From
https://api.drupal.org/api/drupal/core%21modules%21views%21views.theme.inc/function/template_preprocess_views_view_field/8.2.x:

When changing the value of a field in a view, use `preprocess_views_view_field()`.

```
function mytheme_preprocess_views_view_field(&$variables) {
  $view = $variables['view'];
  if($view->id() == 'my_view') {
    if($variables['field']->field == 'field_my_field') {
      $my_field_value = $variables['field']->getValue($variables['row']);
      $my_altered_value = 'xx';
      $variables['output'] = $my_altered_value; // Default variable in Twig file is "output"
      // OR
      $variables['my_output'] = $my_altered_value; // New variable in Twig file (views-view-field--my-view--field-my-field.html.twig)
    }
  }
}
```

SAME FIELD USED TWICE

Note. If you use the same field twice in a view i.e. if you need to display different parts of the same field in different places, views names them something like this: field_library_media and field_library_media_1. In that circumstance, you have to refer to them in the function like this:

// if($variables['field']->field == 'field_library_media') { if($variables['field']->options['id'] == 'field_library_media_1') {

Here is a real example where a media field id is being displayed and I switch it out with the formatted size of the media file.

```
/**
 * Implements hook_preprocess_views_view_field().
 */
function dirt_bootstrap_preprocess_views_view_field(&$variables) {
  $view = $variables['view'];
  if($view->id() == 'resource_library') {
//    if($variables['field']->field == 'field_library_media') {
    if($variables['field']->options['id'] == 'field_library_media_1') {
      $target_id = $variables['field']->getValue($variables['row']);
      $file_size = 0;
      if ($target_id) {
        $media_item = Media::load($target_id);
        // Get the file.
        if ($media_item->hasField('field_media_document')) {
          $file_id = $media_item->field_media_document->getValue()[0]['target_id'];
        }
        elseif ($media_item->hasField('field_media_image')) {
          $file_id = $media_item->field_media_image->getValue()[0]['target_id'];
        }
        elseif ($media_item->hasField('field_media_audio_file')) {
          $file_id = $media_item->field_media_audio_file->getValue()[0]['target_id'];
        }
        elseif ($media_item->hasField('field_media_video_file')) {
          $file_id = $media_item->field_media_video_file->getValue()[0]['target_id'];
        }
        if (isset($file_id)) {
          $file = File::load($file_id);
          if ($file) {
            // Get file size.
            $file_size = format_size($file->getSize());
          }
        }
      }
      $variables['output'] = $file_size; // Default variable in Twig file is "output"
    }
  }
}
```

Concatenate values into a string with join

This would typically be used when passing a series of node id's to a view to filter its output.

```
{% set blah = [node.field_ref_unit.0.target_id,node.field_ref_unit.1.target_id,node.field_ref_unit.2.target_id,node.field_ref_unit.3.target_id]|join('+') %}
```

This produces 1+2+3+4

Loop through entity reference items

In txg/web/themes/custom/txg/templates/content/node--news-story.html.twig I need to loop through a bunch of entity reference values and build a string of id+id+id… (with an undefined number) so:

```
{% set blah = '' %}
{% for item in node.field_ref_unit %}
  {% set blah = blah ~ item.target_id %}
  {% if not loop.last %}
    {% set blah = blah ~ '+' %}
  {% endif %}
{% endfor %}


<div>blah:{{ blah }}</div>
<div>node id: {{ node.id }}</div>
{{ drupal_view('related_news_for_news_story', 'block_unit', node.id, blah) }}
```

Twig filters and functions

Use these to do almost anything with variables in your twig templates.

See cheat sheet at https://www.drupal.org/docs/contributed-modules/twig-tweak/cheat-sheet#s-view-filter also https://twig.symfony.com/doc/3.x/ for filters and functions

Here are some examples. A complete list is included below:

```
{{ <span>Hello I am an html twig, but my html will be stripped</span> | striptags }}

{{'welcome' | upper }}

{{ data | json_encode() }}

{%filter upper %}
  This test becomes uppercase
{% endfilter %}
```

Filters

- abs
- batch
- capitalize
- column
- convert_encoding
- country_name
- currency_name
- currency_symbol
- data_uri
- date
- date_modify
- default
- escape
- filter
- first
- format
- format_currency
- format_date
- format_datetime
- format_number
- format_time
- html_to_markdown
- inky_to_html
- inline_css

- [join](#)
- [json_encode](#)
- [keys](#)
- [language_name](#)
- [last](#)
- [length](#)
- [locale_name](#)
- [lower](#)
- [map](#)
- [markdown_to_html](#)
- [merge](#)
- [nl2br](#)
- [number_format](#)
- [raw](#)
- [reduce](#)
- [replace](#)
- [reverse](#)
- [round](#)
- [slice](#)
- [slug](#)
- [sort](#)
- [spaceless](#)
- [split](#)
- [striptags](#)
- [timezone_name](#)
- [title](#)
- [trim](#)
- [u](#)
- [upper](#)
- [url_encode](#)

Functions

- [attribute](#)
- [block](#)
- [constant](#)
- [country_names](#)
- [country_timezones](#)
- [currency_names](#)
- [cycle](#)
- [date](#)
- [dump](#)
- [html_classes](#)
- [include](#)
- [language_names](#)
- [locale_names](#)
- [max](#)
- [min](#)
- [parent](#)
- [random](#)
- [range](#)
- [script_names](#)
- [source](#)

- [template_from_string](#)
- [timezone_names](#)

Twig Tweak

This is an **essential** module to add to all projects.

Documentation

There are some docs at https://www.drupal.org/docs/contributed-modules/twig-tweak/rendering-blocks-with-twig-tweak

and a cheat sheet at https://git.drupalcode.org/project/twig_tweak/-/blob/3.x/docs/cheat-sheet.md

Display a block with twig_tweak

Here is a simple example:

```
{{ drupal_block('plugin_id') }}
```

It looks like the best source of information is really in the [source file](#) or at: web/modules/contrib/twig_tweak/src/TwigExtension.php

All the variations are listed there including the instructions to get drush to list all the blocks on your site.

```
drush ev "print_r(array_keys(\Drupal::service('plugin.manager.block')->getDefinitions()));"
```

It outputs something like:

```
    [37] => system_menu_block:tools
    [38] => system_messages_block
    [39] => system_powered_by_block
    [40] => user_login_block
    [41] => views_block:comments_recent-block_1
    [42] => views_block:content_recent-block_1
    [43] => views_block:events-block_1
    [44] => views_block:related_products_and_services-block_1
    [45] => views_block:who_s_new-block_1
    [46] => views_block:who_s_online-who_s_online_block
    [47] => views_exposed_filter_block:news_listing_for_news_landing-page_1
    [48] => local_actions_block
    [49] => local_tasks_block
    [50] => page_title_block
    [51] => broken
)
```

Display filter form block

You can then use this to display your ajax exposed filter form block

```
{{ drupal_block('views_exposed_filter_block:news_listing_for_news_landing-page_1') }}
```

Embed view in twig template

In inside-marthe/themes/custom/dprime/templates/content/node-overview.html.twig there is a view rendered in the twig template. This requires the [twig tweak](#) module:

```
<div class="l-sidebar-content">
  {% include '@danaprime/partials/subnav.html.twig' %}
  {{content.field_ref_sidebars}}
  {{ drupal_view('news', 'embed_page_sidebar', content.field_news_categories|render|trim) }}
</div>
```

You can, also specify additional parameters which map to contextual filters you have configured in your view.

```
{{ drupal_view('who_s_new', 'block_1', arg_1, arg_2, arg_3) }}
```

Some tricky quotes magic

Here I am trying to create a string type="aof" so I had to escape at least one of the quotes like this \" (backslash and double quote)

```
{% set office_type = 'type=\"' ~ item.type ~ '"' %}
```

The entire piece of debug code is reproduced below:

```
<div>
  {% for filter in filter_data %}
    {% if filter.type == 'office' %}
      {% for item in filter.info %}
        type={{ filter.type }}, title = {{ item.title }}, value = {{ item.value }}, selected= {{ item.selected }}, item.type = {{ item.type }}<br>
        {% set officetype = '' %}
        {% if item.type is defined and item.type %}
          {% set office_type = 'type=\"' ~ item.type ~ '"' %}
          Office_type:{{ office_type }} <br>
        {% endif %}
      {% endfor %}
    {% endif %}
  {% endfor %}
</div>
```

The real implementation is shown below:

From txg/web/themes/custom/txg/templates/partials/searchfilterform.html.twig

See the line below that sets office_type = …

```
{% for item in filter.info %}
  {% set selected = '' %}
  {% if item.selected is defined and item.selected %}
    {% set selected = 'selected' %}
  {% endif %}
  {% set officetype = '' %}
  {% if item.type is defined and item.type %}
    {% set office_type = 'type=\"' ~ item.type ~ '"' %}
  {% endif %}
  <option value="/search-news?{{ filter.type }}={{ item.value }} {{ office_type }}" {{ selected }}>{{ item.title }}</option>
{% endfor %}
```

Troubleshooting

Enable Twig debugging and disable caches

This will cause twig debugging information to be displayed in the HTML code like the following:

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'toolbar' -->
<!-- BEGIN OUTPUT from 'core/themes/stable/templates/navigation/toolbar.html.twig' -->
```

and

```
<!-- THEME DEBUG -->
<!-- THEME HOOK: 'page' -->
<!-- FILE NAME SUGGESTIONS:
   * page--teks--admin--srp--program--expectation--correlation--vote-all.html.twig
   * page--teks--admin--srp--program--expectation--correlation--852136.html.twig
   * page--teks--admin--srp--program--expectation--correlation--%.html.twig
   * page--teks--admin--srp--program--expectation--correlation.html.twig
   * page--teks--admin--srp--program--expectation--852131.html.twig
   * page--teks--admin--srp--program--expectation--%.html.twig
   * page--teks--admin--srp--program--expectation.html.twig
   * page--teks--admin--srp--program--852061.html.twig
   * page--teks--admin--srp--program--%.html.twig
   * page--teks--admin--srp--program.html.twig
   * page--teks--admin--srp.html.twig
 x page--teks--admin.html.twig
   * page--teks.html.twig
   * page.html.twig
-->
```

In `sites/default/development.services.yml` in the `parameters`, `twig.config`, set `debug:true`. See `core.services.yml` for lots of other items to change for development.

```yaml
# Local development services.
#
parameters:
  http.response.debug_cacheability_headers: true
  twig.config:
    debug: true
    auto_reload: true
    cache: false

# To disable caching, you need this and a few other items
services:
  cache.backend.null:
    class: Drupal\Core\Cache\NullBackendFactory
```

You also need this in `settings.local.php`:

```php
/**
 * Enable local development services.
 */
$settings['container_yamls'][] = DRUPAL_ROOT . '/sites/development.services.yml';
```

You also need to disable the render cache in `settings.local.php`. Here all caching is disabled with:

```php
$config['system.performance']['css']['preprocess'] = FALSE;
$config['system.performance']['js']['preprocess'] = FALSE;
$settings['cache']['bins']['render'] = 'cache.backend.null';
$settings['cache']['bins']['page'] = 'cache.backend.null';
$settings['cache']['bins']['dynamic_page_cache'] = 'cache.backend.null';
```

Debugging - Dump a variable

When troubleshooting or trying to make sense of what is being output, use dump.

```
<pre>
Dump node.created.value:
{{ dump(node.created.value) }}
Dump node.changed.value:
{{ dump(node.changed.value) }}
Dump node.published_at.value:
{{ dump(node.published_at.value) }}
</pre>
```

The output might look like this. Note the published value may be null as I didn't use Drupal scheduling to publish the node:

```
Dump node.created.value:
string(10) \"1604034000\"


Dump node.changed.value:
string(10) \"1604528207\"


Dump node.published_at.value:
NULL
```

Dump taxonomy reference field

Here we dump a taxonomy reference field which is useful for debugging purposes. The pre tags format it a little nicer than if we don't have them.

```
<pre>
  {{ dump(paragraph.field_ref_tax.value) }}
</pre>
```

And get ouput:

```
array(2) {
 [0]=>
 array(4) {
  ["target_id"]=>
  string(2) "13"
  ["_attributes"]=>
  array(0) {
  }
  ["_loaded"]=>
  bool(true)
  ["_accessCacheability"]=>
  object(Drupal\Core\Cache\CacheableMetadata)#7683 (3) {
   ["cacheContexts":protected]=>
   array(1) {
    [0]=>
    string(16) "user.permissions"
   }
   ["cacheTags":protected]=>
   array(0) {
   }
   ["cacheMaxAge":protected]=>
   int(-1)
  }
 }
 [1]=>
 array(4) {
  ["target_id"]=>
  string(2) "16"
  ["_attributes"]=>
  array(0) {
  }
  ["_loaded"]=>
  bool(true)
  ["_accessCacheability"]=>
  object(Drupal\Core\Cache\CacheableMetadata)#7705 (3) {
   ["cacheContexts":protected]=>
   array(1) {
    [0]=>
    string(16) "user.permissions"
   }
   ["cacheTags":protected]=>
   array(0) {
   }
   ["cacheMaxAge":protected]=>
   int(-1)
  }
 }
}
```

Using kint or dump to display variable in a template

With devel and devel: kint enabled, you can display variables in templates. Here we show the content variable from the above block template. Note. There is also a built in dump() function which is super useful.

```
{{ kint(content) }}
```

You can also

```
{{ dump(content) }}
```

And dump a value from a paragraph field. The pre tags will format the output a little more sanely.

```
<pre>
{{ dump(paragraph.field_ref_tax.value) }}
</pre>
```

Or the body field:

```
{{ kint(content['body']) }}
```

Or the tags field content['field_tags']

```
{{ kint(content['field_tags']) }}
```

502 bad gateway error
While working on Twig changes, if you ever see a 502 bad gateway error, try commenting out the twig template code you just added and see if it displays. I know, it's not a friendly error at all!

Views error
If you ever see a 502 bad gateway error when embedding a drupal_view, delete the display and create a new one and it may just work fine.

Striptags (when twig debug info causes if to fail)
When you care about the output being affected by twig debugging, you need to use striptags. In this case, because I enabled twig debugging, the content.field_landing_opinion_page_type was not ever 'ORD'

So here I compare a field value so I have to use striptags to remove all html. I ended up using the combination of render|striptags|trim:

```
{% if content.field_landing_opinion_page_type|render|striptags|trim == 'ORD' %}
  {{ drupal_block('opinion_landing', wrapper=false) }}
{% endif %}
```

Reference
- Drupal 10 uses **Twig 3**. Drupal 9 uses Twig 2. Drupal 8 used Twig 1.

- [Theme system overview on api.drupal.org](#)

- [Twig 3 documentation](#)

- [Drupal.org Theming documentation](#)

- [Handy Twig functions you can use directly in templates - Updated Jan 2023](#)

- [Twig Tweak 3 cheat sheet Updated October 2022](#)

- [Twig Tweak 2 cheat sheet Updated May 2022](#)

- [Using attributes in templates updated March 2023](#)

- [Twig tweaks and Views has some useful notes on using twig tweak with views - Updated November 2020](#)

Page last modified: Apr 28 2023.

This site uses Just the Docs, a documentation theme for Jekyll.