NAF's AutoCon3 Prague, Czech Republic

26-May – 30 May, 2025

Templating in Network Automation Workflows

Jinja2 in Action

Real-World Template Patterns

Mini Projects

Repository Directory	Description
00_DataWoodchipper	Demo
01_ Current State Report	Take existing JSON output from SuzieQ and generate a Markdown report with a diagram showing the BGP sessions and their attributes.
02_Procedure	Take a YAML file of configuration details and generate an Installation report suitable to give to a local “Smart Hands” for installation of an appliance.
03_ConfigurationsMonolithic	Examples of Monolithic configurations using Jinja2
04_ConfigurationsModular	Examples of Modular configurations using Jinja2 features including extends and includes.
05_ChangeRequest	Generate Change Request Text
06_ClabTopology	Generate a ContainerLab topology from a SuzieQ topology
07_VerificationReport	Post change verification
08_SoW	Create a Wireless Access Point cabling Statement of Work
09_EmbeddedJ2	Looking at Tools with “embedded” Jinja2
10_Capstone	Capstone Project

Introduction to Jinja2

AC3 Intro to Jinja Repository

Recorded on-line session

Workshop C4

WS:C4 - Jinja2 in Action: 10 Real-World Python Template Patterns
Proctor(s): Claudia De Luna, Mike Lester - EIA, Lucas Boyd - University of Virginia
Level: Intermediate (some knowledge needed) - *Note: For those who are new to Jinja2 but have some Python experience, we'll offer an optional online Jinja2 Introduction session before the conference*
Description: Dive into the power of Jinja2, a Python templating engine, and its role in streamlining network automation workflows. This workshop is designed for participants with a foundational understanding of Python, REST APIs, and Jinja.
Agenda:
+ Jinja Essentials
+ Background on Templating Languages
+ Revision Control
+ Template Format Options
+ Use Case Working Sessions

Course Requirements

- ⌘Basic Python Knowledge
- ⌘UV installed (Installing UV)
- ⌘Clone or Download the Workshop Repository
  - † git clone https://github.com/cldeluna/ac3-ws-c4-jinja2-in-action.git

- ⌘Containerlab
- ⌘Graphviz
- ⌘Ansible

Check Environment

```
% uv run check_repo_env.py
🔍 Checking repository environment...

🐍 Python version in use:
   3.11.12 (main, Apr  9 2025, 03:49:53) [Clang 20.1.0 ]

✅ Virtual environment is currently set to:
   /Users/claudiadeluna/Indigo Wire Networks Dropbox/Claudia de Luna/scripts/python/2025/ac3_templating_workshop/.venv

✅ 'jinja2' module is available.
   Version: 3.1.6

🛠️Checking for Ansible...
✅ 'ansible' module is available. Version: 2.18.5

📊 Checking for 'diagrams' and 'graphviz' modules...
✅ 'diagrams' module is available.
✅ 'graphviz' module is available.
✅ 'dot' command (Graphviz binary) found at: /opt/homebrew/bin/dot

🔧 Testing import from local 'utils' package...
✅ Successfully imported 'utils.utils'
```

Check Environment

```
Windows PowerShell

-a----         5/17/2025   2:24 PM            141 setup.py
-a----         5/17/2025   2:24 PM         263793 uv.lock


PS C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main> uv run .\check_repo_env.py
Using CPython 3.11.12
Creating virtual environment at: .venv
Installed 110 packages in 11.27s
🔍 Checking repository environment...

🐍 Python version in use:
   3.11.12 (main, Apr  9 2025, 04:03:34) [MSC v.1943 64 bit (AMD64)]

✅ Virtual environment is currently set to:
   C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main\.venv

✅ 'jinja2' module is available.
   Version: 3.1.6

🛠️ Checking for Ansible...
✅ 'ansible' module is available. Version: 2.18.5

📊 Checking for 'diagrams' and 'graphviz' modules...
✅ 'diagrams' module is available.
✅ 'graphviz' module is available.
✅ 'dot' command (Graphviz binary) found at: C:\Program Files\Graphviz\bin\dot.EXE

🔧 Testing import from local 'utils' package...
✅ Successfully imported 'utils.utils'
PS C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main>
```

Time Allocation and Skills Summary

⌘Tuesday 27 May 2025 9:00AM – 1:00PM

Time	Topics
9:00 – 10:45	Set up (if needed) Presentation and Mini Project 01
10:45 – 11:15	Break (Working Break Optional)
11:15 – 12:45	Mini Projects
12:45 – 1:00	Q & A and Final Wrap up

Will Show you…

-The many use cases for templates

-How to build artifacts (other than configs) with various data sources

-Examples of multi component solutions (secrets, data, APIs, function reuse, user interface)

-Data models and storing design

Why are we here?

I already have templates!

I already have a phone!

Notepad (Text)

```
interface [interface-id]
 switchport access vlan [data VLAN]
 spanning-tree portfast
 service-policy input PER-PORT-MARKING
 ! attaches per-port marking policy to interface(s)
```

The last device I configured

PowerPoint

Ingress Policy – Untrusted Endpoint

```
interface [interface-id]
 switchport access vlan [data VLAN]
 spanning-tree portfast
 service-policy input PER-PORT-MARKING
 ! attaches per-port marking policy to interface(s)
```

Excel

C8 =CONCATENATE(A8," ",B8)

Ingress Policy – Untrusted Endpoint Interface Model + Service-Policy

The following interface configuration is to be applied to verified untrusted endpoints only (untrusted client, untrusted server, etc.) only – this configuration is not to be applied to switch-to-switch up-links

Statement	Value	Result
interface	interface-id	interface interface-id
switchport access vlan	data Vlan	switchport access vlan data Vlan
spanning-tree portfast		spanning-tree portfast
service-policy input PER-PORT-MARKING		service-policy input PER-PORT-MARKING
! attaches per-port marking policy to interface(s)		! attaches per-port marking policy to interface(s)

AccessSwitch | DistributionSwitch | CoreSwitch

Lets ask a more precise question…

- Are your configuration templates
  - Under formal revision control in a Revision Control System
  - Current
  - Maintained under a rigorous process
  - Used without exception
  - Programmatically consumable

Templates and Templating Languages

⌘Mail Merge!

⌘Typesetting

† TeX/LaTeX for Documents

⌘Web Development

† HTML Manipulation

† We will take shameless advantage of this tooling

```
<html>
<head>
 <title>Hitle>
 <head>
 <p>Heaading</h1>
 aParagrah
 <a href="https:/www.example
.com>italic<em bold
 <strong>
 <ul>
  <li>Item</li>
  <li>Item</li>
 </body>
</html>
```

Templating Landscape (Python)

- ⌘ Mako - Superfast and lets you embed Python code in the template
- ⌘ Mustache/Pystache – No logic support (Logic free)
- ⌘ Python string.Template class
- ⌘ Python f-strings
- ⌘ **Jinja2- Start here and look at other options when you start to outgrow Jinja in terms of speed and/or complex logic**
- ⌘ Others
  - † HAML/PyHAML - Last updated 2019
  - † PyPugJS – Sort of active but very Web Focused
  - † Cheetah – Last updated 2021

DE FACTO STANDARD

Popular in Network Automation Tools

- ⌘ Ansible
- ⌘ Vendor tools like Cisco Catalyst Center (DNAC)
- ⌘ OpsMill InfraHub
- ⌘ Other Automation Environments

Built-In Options

Python string

```
python3
% uv run python
Python 3.11.12 (main, Apr  9 2025, 03:49:53) [Clang 20.1.0 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import string
>>> template = string.Template("I am so happy to be in $location for AutoCon$idx")
>>> location = "Prague"
>>> idx = 3
>>> rendered = template.substitute(location=location, idx=idx)
>>> rendered
'I am so happy to be in Prague for AutoCon3'
>>>
```

Python f-string

```
% uv run python
Python 3.11.12 (main, Apr  9 2025, 03:49:53) [Clang 20.1.0 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> idx = 3
>>> location = "Prague"
>>> rendered = f"I am so happy to be in {location} at AutCon{idx}"
>>> rendered
'I am so happy to be in Prague at AutCon3'
>>> exit()
```

Template Spectrum & Composition

- Where are you in the spectrum
  - How do you want to manage your templates?
  - More monolithic (A template for my access EOS)
  - More granular (Inheritance and Template Blocks)
  - What changes most frequently??

Variable Substitution

Complex Logic and Manipulation

```
hostname {{ cfg['hostname'] }}
```

```
{% if maxstack is defined %}
{% set ms_int=maxstack | int %}
Variable maxstack has been recast to integer {{ ms_int }}
{% if ms_int > 8 %}
    Legacy Stack is too big!
{% elif ms_int > 0 and ms_int < 1 %}
    Stack Size is Good
{% else %}
    ERROR...BAD Stack Size {{ maxstack }}
{% endif %}
{% else %}
    Expected maxstack variable is NOT DEFINED
{% endif %}
```

Revision Control

A Cautionary Tale

⌘ Meaningless
- † "Revision" Numbers
- † Dates
- † Initials

⌘ Inconsistent revision numbering

⌘ Highest revision number was not the most recent file

This is NOT revision control!

Formal (Actual?) Revision Control

- ⌘Storing
- ⌘Updating and Versioning
- ⌘Retrieving
- ⌘Sharing
- ⌘Tracking
  - ✝ Who made each change
  - ✝ When was a change made
  - ✝ Why was a change made
  - ✝ What changes were made
- ⌘50+ Years of evolution
  - ✝ Saving Efficiently
  - ✝ Branching
  - ✝ Cloning
  - ✝ Forking

Abstract-The Source Code Control System (SCCS) is a software tool designed to help programming projects control changes to source code. It provides facilities for storing, updating, and retrieving all versions of modules, for controlling updating privileges, for identifying load modules by version number, and for recording who made each software change, when and where it was made, and why. This paper discusses the SCCS approach to source code control, shows how it is used and explains how it is implemented.

Todays Popular Tools

Local to your System	Remote/Cloud Developer Collaboration & Revision Control
git	**GitHub (Distributed)**
git	GitLab (open source) (Distributed) Bitbucket (Distributed) …many more
Mercurial	(Distributed)
Concurrent Versions System (CVS)	Client/Server
Apache® Subversion® (Successor to CVS)	Client Server

Our repository is on GitHub

Templates & Revision Control

One Authoritative File

With complete history

In a shareable repository

Consumable by automation

Supporting complex data structures and logic

```
% git log --oneline templates/test_list.j2
f8a007f (HEAD -> main) newline at end of file
54af867 Added debug and output var with out of range index
21a8f46 Added RAR variable output
8a1741f Removing dashes
02f26f0 Renamed to show list and added comments after definin
```

Jinja2 Overview

.. And set up

Syntax

The language of Jinja2

Variables by Themselves

⌘ Outside a control structure (jinja2 block)

```
{{ var }}
{{ mylist[0] }}
{{ mydict.key }}  OR {{ mydict[key] }}
```

⌘ Note: Undefined variables render as empty strings by default!

Variables in a Jinja Statement

⌘ Outside of a control structure (jinja2 block)

```
{{ var }}
```

⌘ Inside a control structure

```
{% if var > 1 %}
```

⌘ Notice the absence of the double curly braces when a variable is part of a control structure.

Control Structures (jinja2 block)

⌘ Comparison between Python and Jinja2 {% %}

- In Python, blocks use indentation and generally will not have a close statement
- In Jinja2, blocks use {% ... %} and {% end... %} (they generally have a close statement)

Control Structure	Python Syntax	Jinja2 Syntax	Notes
If	`if condition:` `...`	`{% if condition %}...{% endif %}`	Very similar; `{% %}` encl control.

If-Else	`if condition:`
	`...`
	`else:`
	`...`
		`{% if condition %}...{% else %}...{% endif %}`	Same logic, different blo

Elif (Else if)	`elif condition:`	`{% elif condition %}`	Jinja2 uses `{% elif %}` `%}` .
For loop	`for item in list:` `...`	`{% for item in list %}...{% endfor %}`	Same logic; `{% endfor` loop.

For-Else loop	`for item in list:`
	`...`
	`else:`
	`...`
		`{% for item in list %}...{% else %}...{% endfor %}`	Jinja2 supports `for-els` Python.

While loop	`while condition:` `...`	*Not supported directly*	Must simulate using `{%` conditions.
Break	`break`	`{% break %}`	Available in Jinja2 >= 2.1
Continue	`continue`	`{% continue %}`	Available in Jinja2 >= 2.1
With (scoping)	`with open('file') as f:` `...`	`{% with var=value %}...{% endwith %}`	Different usage; Jinja2 fo variables.
Set Variable	`x = 5`	`{% set x = 5 %}`	Jinja2 requires `{% set %` assignment.

Control Structures

⌘ FOR Loop

```
{% for intf in intf_list %}
   {{ intf }}
{% endfor %}
```

⌘ IF Statement

```
{% if ms_int > 8 %}
   Legacy Stack is too big!
{% elif ms_int > 0 and ms_int < 1 %}
   Stack Size is Good
{% else %}
   ERROR...BAD Stack Size {{ maxstack }}
{% endif %}
```

Assignments

⌘ Assign Value, Type, or Both

```
{% set maxstack=cfg['maxstack'] %}
```

```
{% set ms_int=maxstack | int %}
```

```
{% if ms_int > 8 %}
   Legacy Stack is too big!
{% elif ms_int > 0 and ms_int < 1 %}
   Stack Size is Good
{% else %}
   ERROR...BAD Stack Size {{ maxstack }}
{% endif %}
```

Filters “Out of the Box”

List of Builtin Filters

abs()	forceescape()	map()	select()	unique()
attr()	format()	max()	selectattr()	upper()
batch()	groupby()	min()	slice()	urlencode()
capitalize()	indent()	pprint()	sort()	urlize()
center()	int()	random()	string()	wordcount()
default()	items()	reject()	striptags()	wordwrap()
dictsort()	join()	rejectattr()	sum()	xmlattr()
escape()	last()	replace()	title()	
filesizeformat()	length()	reverse()	tojson()	
first()	list()	round()	trim()	
float()	lower()	safe()	truncate()	

Tests Out of the Box

List of Builtin Tests

boolean()	even()	in()	mapping()	sequence()
callable()	false()	integer()	ne()	string()
defined()	filter()	iterable()	none()	test()
divisibleby()	float()	le()	number()	true()
eq()	ge()	lower()	odd()	undefined()
escaped()	gt()	lt()	sameas()	upper()

Be Careful with Truth

- These are not interchangeable across contexts:
  - In Python code, using true will result in a NameError
  - In Jinja2 templates, using True will not be recognized as a Boolean

- Type Matters!

```
{% if value is true %}
 Using 'is' test works on a boolean
{% else %} Fails: checks for boolean true
{% endif %}

{% if value %}
"Truthy" true
{% endif %}
```

Python vs Jinja2

Concept	Python	Jinja2
True value	True	true
False value	False	false
None/null value	None	none

Example of Truth

```
python3
>>> tmp = " {% if val is True %} TRUE {{ val }} {% endif  %}"
>>> import jinja2 as j2
>>> t = j2.Template(tmp)
>>> r = t.render(val="myvalue")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/claudiadeluna/Indigo Wire Networks Dropbox/Claudia de Luna/scripts/python/2025/ac3_templating_workshop/.v
env/lib/python3.11/site-packages/jinja2/environment.py", line 1295, in render
    self.environment.handle_exception()
  File "/Users/claudiadeluna/Indigo Wire Networks Dropbox/Claudia de Luna/scripts/python/2025/ac3_templating_workshop/.v
env/lib/python3.11/site-packages/jinja2/environment.py", line 942, in handle_exception
    raise rewrite_traceback_stack(source=source)
  File "<template>", line 1, in top-level template code
jinja2.exceptions.TemplateRuntimeError: No test named 'True' found.
>>> tmp = " {% if val is true %} TRUE {{ val }} {% endif  %}"
>>> t = j2.Template(tmp)
>>> r = t.render(val="myvalue")
>>> r
' '
>>> tmp = " {% if val %} TRUE {{ val }} {% endif  %}"
>>> t = j2.Template(tmp)
>>> r = t.render(val="myvalue")
>>> r
'  TRUE myvalue '
>>> 
```

Comments

Multiline {# #}

```
{#
Multi-line comments in a Jinja2
template.
These will not be part of the
rendered text.
#}
```

```
valid_line_comment_prefixes = ["//", "#", ";", "--", "##"]
# In case an invalid or blank line comment is provided to the function
if line_comment not in valid_line_comment_prefixes:
    line_comment = "##"

env = jinja2.Environment(
    loader=jinja2.FileSystemLoader(search_dir, encoding="utf-8"),
    line_comment_prefix=line_comment,  # Only works for single-line comments
```

Single Line {# #} and custom symbol Set in environment

```
{# Single line using the default comment #}
```

If set when establishing the environment

```
## Another single line comment ##
```

Even if “line_comment_prefix” is set, {# #} can be used

Use {#- -#} to strip the white space!

Be careful with Markdown!

Templates - Macros

move_intf_w_macro.j2

```
{#-
This template expects a dictionary where the key is an interface id and the value is a dictionary
The value dictionary contains the following keys:
- desc
- new_vlan
- original_inf
-#}

##- Embedded macro to camel case the description -##
{% macro camel_case(text) -%}
{%- endmacro %}


##- Import the Jinja2 Macro to Expand the interface text (which is just another .j2 file  in the template directory -##
{% import 'intf_expansion_macro.j2' as interface_macros %}

{% for intf, intf_dict in cfg.items() %}
! Original Interface {{ intf_dict['original_intf'] }}
interface {{ interface_macros.expand_interface(intf) }}
{% if intf_dict['desc'] %}
  description {{ camel_case(intf_dict['desc']) }}
{% endif %}
  switchport access vlan {{ intf_dict['new_vlan'] }}
!
{% endfor %}
```

Line 50, Column 1 | main | Spaces: 4 | Plain Text

1. Multiline comment with the standard {# #}
2. Single-line comment with '##' configured in environment
3. Embedded macro to turn text into camel case
4. Importing another Jinja2 template file which contains a macro definition
5. More complex cfg data structure which is now a dictionary with values which are also dictionaries

Macros – Use and Result

```python
# Notice we don't need to import Jinja2 here as we are using the function in utils.py
import utils

def main():

    # Step 1 - Define the environment containing the templates you want to use
    # Here all the templates will be in a directory called "templates" relative to the current directory
    env_obj = utils.jenv_filesystem(line_comment="##", ktn=True, lsb=True, tb=True)

    # Step 2 - Load the specific template to be used
    template_obj = env_obj.get_template("move_intf_w_macro.j2")

    # Template Payload
    # Creating a dictionary for the first 5 GigE interfaces using the GigE shorthand so we can use the
    # interface expansion macro in the template to expand the interface
    cfg_dict = dict()
    desc = "supply kiosk"

    for idx in range(34,39):
        # Cisco UXM switches
        if idx >36:
            intf_prefix = "Te"
        else:
            intf_prefix = "Tw"
        cfg_dict.update({f"{intf_prefix}1/0/{idx}": {"new_vlan": 300, "desc": desc, "original_intf": f"GigE1/0/{idx}"}})

    print(cfg_dict)

    # Step 3 - Render the template with specific payload
    # The template is expecting a dictionary named "cfg"
    desc = ""
    rendered = template_obj.render(cfg=cfg_dict)

    print(rendered)


# Standard call to the main() function.
if __name__ == '__main__':
    main()
```

```
% python move_access_intf_w_macro.py
{'Te1/0/37': {'desc': 'supply kiosk',
              'new_vlan': 300,
              'original_intf': 'GigE1/0/37'},
 'Te1/0/38': {'desc': 'supply kiosk',
              'new_vlan': 300,
              'original_intf': 'Gi
 'Tw1/0/34': {'desc': 'supply kios
              'new_vlan': 300,
              'original_intf': 'Gi
 'Tw1/0/35': {'desc': 'supply kios
              'new_vlan': 300,
              'original_intf': 'Gi
 'Tw1/0/36': {'desc': 'supply kios
              'new_vlan': 300,
              'original_intf': 'Gi
```

```
! Original Interface GigE1/0/34
interface TwoGigabitEthernet1/0/34
  description supplyKiosk
  switchport access vlan 300
!
! Original Interface GigE1/0/35
interface TwoGigabitEthernet1/0/35
  description supplyKiosk
  switchport access vlan 300
!
! Original Interface GigE1/0/36
interface TwoGigabitEthernet1/0/36
  description supplyKiosk
  switchport access vlan 300
!
! Original Interface GigE1/0/37
interface TenGigabitEthernet1/0/37
  description supplyKiosk
  switchport access vlan 300
!
! Original Interface GigE1/0/38
interface TenGigabitEthernet1/0/38
  description supplyKiosk
  switchport access vlan 300
!
```

Macros - Considerations

⌘ Macros are handy but…

- † Its important to think about where you want your logic.

⌘ Do you want to be updating your Templates when a new prefix comes out or is it better to update your Python script which likely has other related logic.

⌘ What if you want to use the interface expansion logic in one of your Python scripts. Will you be better served with a Python function rather than a Jinja2 macro?

⌘ There is no right answer here.

⌘ I keep logic to a minimum in my Templates and do the heavy lifting (more complex data manipulation) in the Python script but that is what works for me and the use cases I am supporting.

Template Environment

The language of Jinja2

Jinja2 Workflow Concept

Python

Jinja2

Template Environment

We must define our templating environment

- Where do we load templates from
- How we are going to load our templates (particularly if using inheritance)
- How we deal with white space
- Do we want debugging
- Do we want to customize

Load Template

- Once our environment is defined, we just have to pick the template we want to use

Templates are loaded into memory as objects

Render Template

- Send our data structure to our template for rendering resulting in a specific artifact we can save or pass on to another part of the workflow

Data Manipulation

Save or Send

Workflow Simple Example

Python

Jinja2

Template Environment

```
# "Environment" is a string
tmp_str = "I am in {{ city }}!"
```

Load Template

```
import jinja2 as j2
# create template object
t = j2.Template(tmp_str)
```

Render Template

```
result = t.render(city="Prague"
print(result)
```

'I am in Prague'

city = "Prague"

Data Manipulation

Save or Send

Interactive Python Interpreter

```
python3
% uv run python
Python 3.11.12 (main, Apr  9 2025, 03:49:53) [Clang 20.1.0 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import jinja2 as j2
>>> tstr = "I am in {{ city }}"
>>> tobj = j2.Template(tstr)
>>> tobj
<Template memory:1007163d0>
>>>
>>> dir(tobj)
['__annotations__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getstate__
', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__re
pr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', '_debug_info', '_from_namespace', '_get_default_module', '_get_defa
ult_module_async', '_module', '_uptodate', 'blocks', 'debug_info', 'environment', 'environment_class', 'filename', 'from_code', 'from_module_dict', '
generate', 'generate_async', 'get_corresponding_lineno', 'globals', 'is_up_to_date', 'make_module', 'make_module_async', 'module', 'name', 'new_conte
xt', 'render', 'render_async', 'root_render_func', 'stream']
>>> tobj.name
>>> tobj.environment
<jinja2.environment.Environment object at 0x1005b7810>
>>> tobj.debug_info
[(1, 13)]
>>>
>>> tobj.render(city="Prague")
'I am in Prague'
>>> city = "Los Angeles"
>>> result = tobj.render(city=city)
>>> result
'I am in Los Angeles'
```

Environment Options

- Load your Templates from
  - A Python Package
  - A Directory
  - A String Variable

```
jinja2.Environment(loader=<>)
```

loader=FileSystemLoader(dir)

dir is a variable with an absolute or relative path to where the templates are stashed. This method is simple but makes you think about directories.

loader=PackageLoader(pkg)

Pkg is a variable with the name of a Python package. This method is highly portable but more complex.

env.from_string(template_string)

template_string is a string variable with template code.

Environment

```
% tree
.
├── move_access_intf_w_macro.py
├── move_access_intf.py
├── one_step_jinja.py
├── payload_named_vs_unpacked.py
├── template_env.py
└── templates
    ├── all_in_one.j2
    ├── control_structures.j2
    ├── layer2_vlan_template.j2
    ├── macro_intf_expansion.j2
    ├── move_intf_w_macro.j2
    ├── move_intf.j2
    ├── test_dict.j2
    ├── test_list.j2
    ├── test_named.j2
    ├── test_tests.j2
    ├── test_unpacked.j2
    ├── vlan_list.j2
    └── vlan_lod.j2
```

additional_examples > move_access_intf.py > ...

```python
import jinja2


def main():

    # Step 1 - Define the environment containing the templates you want to use
    # Here all the templates will be in a directory called "templates" relative to t
    env_obj = jinja2.Environment(
        loader=jinja2.FileSystemLoader("templates", encoding="utf-8")
    )

    # Step 2 - Load the specific template to be used
    template_obj = env_obj.get_template("move_intf.j2")


    # Template Payload
    cfg_dict = {
        "GigabitEthernet1/0/27": 300,
    }

    # Step 3 - Render the template with specific payload
    # The template is expecting a dictionary named "cfg"

    rendered = template_obj.render(cfg=cfg_dict2, desc="Digital Signage")
    print(rendered)


# Standard call to the main() function.
if __name__ == "__main__":
    main()
```

An Environment Function

⌘Example

```python
def main():

    # Step 1 - Define the environment containing the templates you want to use
    # Here all the templates will be in a directory called "templates" relative to

    env_obj = utils.jenv_filesystem(line_comment="", ktn=True, lsb=True, tb=True)

```

utils > utils.py > jenv_filesystem

```python
def jenv_filesystem(
    search_dir="templates", line_comment="##", ktn=False, lsb=False, tb=False
):
    """…

    valid_line_comment_prefixes = ["//", "#", ";", "--", "##", "=", "==", "!"]
    # In case an invalid or blank line comment is provided to the function
    if line_comment not in valid_line_comment_prefixes:
        line_comment = "##"

    env = jinja2.Environment(
        loader=jinja2.FileSystemLoader(search_dir, encoding="utf-8"),
        line_comment_prefix=line_comment,  # Only works for single-line comments
        keep_trailing_newline=ktn,  # Default is False. True preserve trailing newline in templates
        trim_blocks=tb,  # If True remove first newline after a block. Default is False
        lstrip_blocks=lsb,  # IF True remove leading spaces and tabs from block tags. Default is False
        undefined=jinja2.runtime.Undefined,  # Default is Undefined - undefined variables render as empty string
    )

    return env
```

You, 4 days ago • Added Excercise 02 …

Load Template

Turn your template into a "smart" object which can be combined with your data to generate a specific artifact (configuration or otherwise)

additional_examples > move_access_intf.py > ...

```python
import jinja2


def main():

    # Step 1 - Define the environment containing the templates you want to use
    # Here all the templates will be in a directory called "templates" relative to t
    env_obj = jinja2.Environment(
        loader=jinja2.FileSystemLoader("templates", encoding="utf-8")
    )

    # Step 2 - Load the specific template to be used
    template_obj = env_obj.get_template("move_intf.j2")


    # Template Payload
    cfg_dict = {
        "GigabitEthernet1/0/27": 300,
    }

    # Step 3 - Render the template with specific payload
    # The template is expecting a dictionary named "cfg"

    rendered = template_obj.render(cfg=cfg_dict2, desc="Digital Signage")
    print(rendered)


# Standard call to the main() function.
if __name__ == "__main__":
    main()
```

Render Template

⌘Note the variable names

additional_examples > move_access_intf.py > ...

```python
def main():

    # Step 2 - Load the specific template to be used
    template_obj = env_obj.get_template("move_intf.j2")

    # Template Payload
    cfg_dict = { "GigabitEthernet1/0/27": 300,}

    # Step 3 - Render the template with specific payload
    # The template is expecting a dictionary named "cfg"

    rendered = template_obj.render(cfg=cfg_dict, desc="Digital Signage")
    print(rendered)


# Standard call to the main() function.
if __name__ == "__main__":
    main()
```

Script Variable

Script Inline Value

Template Variable

additional_examples > templates > move_intf.j2

```jinja
{# ! This template expects a dictionary where the key is

{% for intf, new_vlan in cfg.items() %}
interface {{ intf }}
{% if desc %}
  description {{ desc }}
{% endif %}
  switchport access vlan {{ new_vlan }}
!
{% endfor %}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
% uv run move_access_intf.py


interface GigabitEthernet1/0/27

  description Digital Signage

  switchport access vlan 300
!
```

Passing Payload to Your Template

`template.render(cfg_data) vs template.render(cfg=cfg_data)`

- Unpacking (**) is the default behavior in Jinja2
  - template.render(**cfg_data) aka template.render(cfg_data)
- If your cfg_data is a dictionary with one key/value pair {"hostname": "my_switch"} if you don't assign your payload to a specific name (like cfg=cfg_data), by default, Jinja2 will make the dictionary keys available as top-level variables in the template.
  - {{ hostname }} in the template will render to "my_switch"
- While using the "unpacking" default behavior can be a bit more streamlined, it inhibits the ability to dump the payload in the template if you are troubleshooting or unsure of the keys or structure.
- {{ cfg }} # Show the entire payload in your template
  - {{ cfg['hostname'] }} # Get the value of hostname
- I find the ability to do this very useful, so I generally always assign my payload (script variable) to a template variable.

Named vs Unpacked

```
rendered =
template_obj_named.render
(cfg=payload_dict)
```

test_named.j2

```
! Passing Named Payload
{{ cfg }}
```

```
==== Passing named variable ====
! Passing Named Payload
{'list': [1, 2, 3, 4, 5], 'maxstack': 9, 'hostname':
'myswitch-01', 'mydict': {'subnet': '192.168.0.0/24',
 'gw': '192.168.0.1', 'mask': '255.255.255.0'}}
```

```
rendered =
template_obj_upacked.render
(payload_dict)
```

test_unpacked.j2

```
! Passing unnamed variable
! In standalone Jinja (outside a framework like Django or Flask) there is not a built in way to
access the
! passed variables without naming them

! Referencing the keys directly but you have to know what they are.
! With a named variable you can dump the entire payload with a single command {{ cfg }}
! In this case I know Im passing a dictionary with the keys list, maxstack, hostname, and mydict
{{ list }}
{{ maxstack }}
{{ hostname }}
{{ mydict }}
```

```
==== Passing Un-named and Unpacked variable ====
! Passing unnamed variable
! In standalone Jinja (outside a framework like Django or
Flask) there is not a built in way to access the
! passed variables without naming them

! Referencing the keys directly but you have to know what
they are.
! With a named variable you can dump the entire payload wi
th a single command
! In this case I know Im passing a dictionary with the key
s list, maxstack, hostname, and mydict
[1, 2, 3, 4, 5]
9
myswitch-01
{'subnet': '192.168.0.0/24', 'gw': '192.168.0.1', 'mask':
'255.255.255.0'}
```

⌘ Price: {{ cfg['list'] }} vs {{ list }}

Strategy for Creating Templates

Modular vs Monolithic

One instance vs N instances (even if N = 1)

Templates In Code

⌘… Doesn't count

⌘Templates in code should be very tactical and targeted.

⌘ Things like a route statement or adding a vlan to a trunk or setting a new access vlan on an access interface.

⌘… or does it? Arista AVD/PyAVD

† Embedded templates without Ansible

Template Strategy

⌘Monolithic

Monolithic templates encompass an entire device configuration including all the specific features enabled for the device.

PRO

- Configuration patterns for a specific model/function are in one template

CON

- Configuration patterns for a specific model/function are in one template.
    - If different teams govern the feature pattern, there may be many making changes to the single template

⌘Modular (Inheritance)

With modular templates, you take on the additional work of putting together a complete configuration for a device, but separate features are in their own ”feature” template and can be managed individually. The WAN team can manage their BGP template, the Security team can manage their NAC and AAA template, etc.

PRO

- Feature templates can be managed by the controlling team

CON

- Added step of building a complete configuration
- Another layer of abstraction

Monolithic

A single template file for a particular type of device.

```
! Access Switch Configuration for location {{ cfg['location'] }}
! Access Switch Hostname is {{ cfg['hostname'] }}
! Using Template: DNAC Sample Jinja Template Base Config
! Configuration Generated {{ cfg['timestamp'] }}
! ----------------------------------------------------------------
!
no service pad
service timestamps debug datetime msec localtime
service timestamps log datetime msec localtime
service password-encryption
!
hostname {{ cfg['hostname'] }}

!
boot-start-marker
boot-end-marker
!
enable secret {{ cfg['enable_sec'] }}
!
!
!
aaa new-model
!
!
aaa authentication login default group tacacs+ enable none
aaa authentication login console enable none
aaa authentication enable default group tacacs+ enable none
aaa authorization exec default group tacacs+ none
aaa authorization exec console none

aaa authorization commands 15 default group tacacs+ none
!
!
!
aaa session-id common
clock timezone {{ cfg['timezone'] }} {{ cfg['timezone_offset'] }}
{% if cfg['summertime_bool'] %}
clock summer-time {{ cfg['timezone'] }} recurring  {{ cfg['summertime_start_stop'] }}
{% endif %}
```

Modular

Includes

Extends

Modular using Jinja2 Includes

modular_include_config_generator.py

```python
def main():

    # Load configuration payload
    cfg_data = load_cfg_data()

    # Create Jinja2 environment
    env = jinja2.Environment(
        loader=jinja2.FileSystemLoader("templates"),
        trim_blocks=True,
        lstrip_blocks=True,
    )

    # Load base template
    template = env.get_template('modular_inc_base.j2')

    # Render configuration
    config = template.render(cfg_data)

    # Check to see if the output directory exists and if it does not, create it
    # This is the directory where we will store the resulting config files
    output_dir = "cfg_output"
    cfg_directory = os.path.join(os.getcwd(), output_dir)
    pathlib.Path(cfg_directory).mkdir(exist_ok=True)

    # Save the configuration to a file
    file_name = f"{cfg_data['hostname']}_router_config_using_includes.txt"
    fp = os.path.join(cfg_directory, file_name)

    with open(fp, "w") as f:
        f.write(config)

    print(f"\nSaved configuration to file {file_name} in current directory.\n")
```

modular_inc_base.j2

```jinja2
{#-
# Directory structure:
# templates/
#   modular_inc_base.j2
#   modular_inc_aaa.j2
#   modular_inc_interfaces.j2
#   modular_inc_ospf.j2
# modular_include_config_generator.py
-#}

! templates/modular_inc_base.j2
hostname {{ hostname }}
!
{% include 'modular_inc_aaa.j2' %}
!
{% include 'modular_inc_interfaces.j2' %}
!
{% include 'modular_inc_ospf.j2' %}
!
end
```

Modular using Extends Blocks

mod_inherit_base.j2

```
{# This is the base template mod_inherit_base.j2 #}
!
! Base Configuration Template
!
hostname {{ hostname }}
!
{# AAA Configuration Block #}
{% block aaa %}
{% endblock %}
!
{# Interface Configuration Block #}
{% block interfaces %}
{% endblock %}
!
{# OSPF Configuration Block #}
{% block ospf %}
{% endblock %}
!
end
```

mod_inherit_aaa.j2

```
{% extends "mod_inherit_base.j2" %}

{% block aaa %}
! AAA Template
aaa new-model
aaa authentication login default group tacacs+ local
aaa authorization exec default group tacacs+ local
aaa accounting exec default start-stop group tacacs+
!
tacacs server TACACS1
 address ipv4 {{ tacacs_server }}
 key {{ tacacs_key }}
{% endblock %}
```

Extends Render Example - Base

```
Windows PowerShell
d-----         5/18/2025   7:22 AM                templates
-a----         5/18/2025   7:22 AM           1940 combo_cfg.py
-a----         5/18/2025   7:22 AM           2608 extend_section.py
-a----         5/18/2025   7:22 AM           3263 modular_inheritance_config_generator.py
-a----         5/18/2025   7:22 AM           2964 modular_sw_cfg_extends.py


PS C:\Users\cldel\Clones\ac3-ws-c4-jinja2-in-action\04_ConfigurationsModular\modular_extends> uv run .\extend_section.py

Using template: mod_inherit_base.j2
Rendering template: mod_inherit_base.j2
!
! Base Configuration Template
!
hostname ROUTER-2
!



!



!



!
end
PS C:\Users\cldel\Clones\ac3-ws-c4-jinja2-in-action\04_ConfigurationsModular\modular
```

```
mod_inherit_base.j2
{# This is the base template mod_inherit_base.j2 #}
!
! Base Configuration Template
!
hostname {{ hostname }}
!
{# AAA Configuration Block #}
{% block aaa %}
{% endblock %}
!
{# Interface Configuration Block #}
{% block interfaces %}
{% endblock %}
!
{# OSPF Configuration Block #}
{% block ospf %}
{% endblock %}
!
end
```

Extends Render Example - AAA

Windows PowerShell

```
PS C:\Users\cldel\Clones\ac3-ws-c4-jinja2-in-action\04_ConfigurationsModular\modular_extends> uv run
.\extend_section.py -s aaa
Using template: mod_inherit_aaa.j2
Rendering template: mod_inherit_aaa.j2
!
! Base Configuration Template
!
hostname ROUTER-2
!


! AAA Template
aaa new-model
aaa authentication login default group tacacs+ local
aaa authorization exec default group tacacs+ local
aaa accounting exec default start-stop group tacacs+
!
tacacs server TACACS1
 address ipv4 192.168.1.10
 key secret123

!



!



!
end
PS C:\Users\cldel\Clones\ac3-ws-c4-jinja2-in-action\04_ConfigurationsModular\mod
```

mod_inherit_base.j2

```
{# This is the base template mod_inherit_base.j2 #}
!
! Base Configuration Template
!
hostname {{ hostname }}
!
{# AAA Configuration Block #}
{% block aaa %}
{% endblock %}
!
{# Interface Configuration Block #}
{% block interfaces %}
{% endblock %}
!
{# OSPF Configuration Block #}
{% block ospf %}
{% endblock %}
!
end
```

mod_inherit_aaa.j2

```
{% extends "mod_inherit_base.j2" %}

{% block aaa %}
! AAA Template
aaa new-model
aaa authentication login default group tacacs+ local
aaa authorization exec default group tacacs+ local
aaa accounting exec default start-stop group tacacs+
!
tacacs server TACACS1
 address ipv4 {{ tacacs_server }}
 key {{ tacacs_key }}
{% endblock %}
```

Modular using Extends and Includes

- ⌘Two strategies
- ⌘Your script renders and assembles
- ⌘You add another template which includes … includes

```python
def main():

    # Load configuration payload
    cfg_data = load_cfg_data()

    # Create Jinja2 environment
    env = jinja2.Environment(
        loader=jinja2.FileSystemLoader('templates'),
        trim_blocks=True,
        lstrip_blocks=True
    )

    # Load templates
    base_template = env.get_template('mod_inherit_base.j2')
    aaa_template = env.get_template('mod_inherit_aaa.j2')
    interfaces_template = env.get_template('mod_inherit_interfaces.j2')
    ospf_template = env.get_template('mod_inherit_ospf.j2')

    # Generate configurations
    config_sections = {
        'base': base_template.render(cfg_data),
        'aaa': aaa_template.render(cfg_data),
        'interfaces': interfaces_template.render(cfg_data),
        'ospf': ospf_template.render(cfg_data)
    }

    print(config_sections['base'])
    print(config_sections['aaa'])
    print(config_sections['interfaces'])
    print(config_sections['ospf'])

    # Write configuration to file
    file_name = f"{cfg_data['hostname']}_router_config.txt"
    with open(file_name, 'w') as f:
        f.write(config_sections['base'])
        f.write(config_sections['aaa'])
        f.write(config_sections['interfaces'])
        f.write(config_sections['ospf'])

    print(f"\nSaved configuration to file {file_name} in current directory.\n")
```

Data Points

⌘ What is out there today (in my experience)?

⌘ No Templates (by our definition)

⌘ Monolithic

⌘ Mostly Monolithic with a little Includes

⌘Keep an eye out on what the professionals are doing

Creating Templates with AI

This is an area where AI is already proving to be a tremendous time saver.

The containerlab mini project was largely generated by AI with some minor tweaking.

Several of the extends and includes examples are also based on AI ”contribution”

This can be quite a time saver however

- don’t be afraid to tell your AI buddy they are wrong!
- double and triple check (I asked for a 3-tier network topology and got the access switches connected to both core and distribution…I'm just saying…)
- this is like “verbal” coding so be precise in your prompts

Whitespace

Space, Tab, Line Feed…What are all those extra lines doing in my configuration…?

Spaces and Line Feeds can be Tricky in Jinja2

- ⌘ Control structures can add unwanted whitespace and empty lines.
- ⌘ Often it will come down to what you can live with and the functional impact of any extraneous whitespace when you push your configs.
- ⌘"One liners" can help
- ⌘ Jinja2 has several white space management options
  - † \+ and \-
  - † Environment knobs (Trim and Lstrip)

What?

Whitespace refers to any space, tab, or newline character that appears in the **template source** and may be preserved, altered, or removed in the rendered output.

This includes:

- Spaces and tabs at the beginning or end of lines

- Blank lines created by newlines

- Indentation used for readability in template code

- Block tags/Control statements

In-Line White Space Control

⌘Whitespace control with "-”

{%- ... %} Removes whitespace before the tag

{% ... -%} Removes whitespace after the tag

{%- ... -%} Removes whitespace both before and after the tag

{{- cfg -}} Removes whitespace both before and after the variable

⌘Whitespace control with "+”

† The plus sign preserves white space which would normally be removed. While it's use is less common it can be useful as an override particularly if you set whitespace controls when defining your environment.

{%+ ... %} Preserve indentation

Environment White Space Control

⌘Whitespace control during Environment definition

```
import jinja2
env = jinja2.Environment(
  trim_blocks=True, # Remove first newline after a block
  lstrip_blocks=True, # Remove leading spaces and tabs from block tags
  keep_trailing_newline=True, # Preserve trailing newline in templates
)
```

⌘ By default, these three settings are False

White Space Recommendations

⌘ Understand what is aesthetics vs functional

⌘ Use the Environment knobs!

⌘ (Otherwise) Expect Trial and Error

⌘ In some cases, a separator in your template can help (!)

Templating Use Cases

Each Use Case or Mini Project will have a README file with additional details.

Course Requirements

- ⌘Basic Python Knowledge
- ⌘UV installed (Installing UV)
- ⌘Clone or Download the Workshop Repository
  - † git clone https://github.com/cldeluna/ac3-ws-c4-jinja2-in-action.git

- ⌘Containerlab
- ⌘Graphviz
- ⌘Ansible

Check Environment

```
⌘ uv run check_repo_env.py
```

```
-zsh
% uv run check_repo_env.py
🔍 Checking repository environment...

🐍 Python version in use:
   3.11.12 (main, Apr  9 2025, 03:49:53) [Clang 20.1.0 ]

✅ Virtual environment is currently set to:
   /Users/claudiadeluna/Indigo Wire Networks Dropbox/Claudia de Luna/scripts/python/2025/

✅ 'jinja2' module is available.
   Version: 3.1.6

🛠Checking for Ansible...
✅ 'ansible' module is available. Version: 2.18.5

📊 Checking for 'diagrams' and 'graphviz' modules...
✅ 'diagrams' module is available.
✅ 'graphviz' module is available.
✅ 'dot' command (Graphviz binary) found at: /opt/homebrew/bin/dot

🔧 Testing import from local 'utils' package...
✅ Successfully imported 'utils.utils'
```

```
Windows PowerShell
-a----        5/17/2025   2:24 PM            141 setup.py
-a----        5/17/2025   2:24 PM         263793 uv.lock


PS C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main> uv run .\check_repo_env.py
Using CPython 3.11.12
Creating virtual environment at: .venv
Installed 110 packages in 11.27s
🔍 Checking repository environment...

🐍 Python version in use:
   3.11.12 (main, Apr  9 2025, 04:03:34) [MSC v.1943 64 bit (AMD64)]

✅ Virtual environment is currently set to:
   C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main\.venv

✅ 'jinja2' module is available.
   Version: 3.1.6

🛠 Checking for Ansible...
✅ 'ansible' module is available. Version: 2.18.5

📊 Checking for 'diagrams' and 'graphviz' modules...
✅ 'diagrams' module is available.
✅ 'graphviz' module is available.
✅ 'dot' command (Graphviz binary) found at: C:\Program Files\Graphviz\bin\dot.EXE

🔧 Testing import from local 'utils' package...
✅ Successfully imported 'utils.utils'
PS C:\Users\cldel\Downloads\ac3-ws-c4-jinja2-in-action-main\ac3-ws-c4-jinja2-in-action-main>
```

Course Infrastructure

In addition to your student laptops and our workshop repository…

00_DataStructure_Woodchipper

⌘ Solara App to illustrate passing variables from your Python script to a Jinja2 template

⌘ **`uv run solara run data_woodchipper.py`**

01_CurrentStateReport

⌘ Illustrates how to turn a JSON data payload into a "current state report"

⌘This script uses the Diagrams module* and JSON output from Suzie to generate a state report on the BGP Sessions for a particular location.

⌘ **`uv run gen_bgp_report.py`**

** Requires Graphviz*

02_Procedure

- Generates installation work instructions for a specific appliance at a location and also introduces some common strategies in templating
  - YAML Payload (unlike the JSON we saw previously which came from an API call, this is “hand made” YAML and so YAML is used because it allows for comments.
  - Artifacts (images in this case) that are folded into the resulting document
  - Utils module

03_ConfigurationsMonolithic

⌘Example of using a single “fully formed” template for each …

⌘View the README.md

⌘ Examine the directory content

04_ConfigurationsModular

⌘Example of additional abstractions in Jinja2 to modularize your templates

⌘View the README.md

⌘ Examine the directory content

⌘Examine Extends

⌘Examine Includes

† Arista AVD

05_ChangeRequest

⌘ We will craft the body of a change request and optionally create a standard change request on a Developer instance of Service Now.

06_ClabTopology

- Example of extracting topology data via an API and creating a Containerlab Topology
  - We pull LLDP based topology information for a namespace (location) from SusieQ via REST and manipulates the data in order to generate a simple Containerlab topology which can then be used for virtual testing and validation
  - Not a Digital Twin (YUK) but a Digital Sibling

07_VerificationReport

- Example of how to use observability tools for interactive functional verification.
  - We will use SuzieQ to verify spanning tree for selected vlans.
  - In addition, we will show how simple it is to create a very user-friendly Web App that can be run locally to mitigate issues with data entry.
    - This is an example of a Streamlit App

08_SoW

- Example of generating a Statement of Work suitable for quote solicitation from a Cabling Vendor for new cabling to support new Access Point locations at a location.
  - A challenge all Network Engineers face is that of storing design or change data in a queriable way.
  - SuzieQ Enterprise supports the integration of external data. In this mini project we have data for new access point cabling runs in an SuzieQ External DB. Once this external data is imported, the familiar methods, syntax, and interfaces used for other SuzieQ interactions can be used on this data. It can also be updated over time, so updates can be tracked!

09_EmbeddedJ2

- This showcases how Jinja2 templates are part of a larger ecosystem.
  - Ansible
  - Arista Validated Design (AVD)
  - PyAVD
  - InfraHub
- Here you will need to focus on the template (probably) and the payload to send to the template. Instantiating the environment will be done by the tool.

10_CapstoneProject

- ⌘ Bringing it all together
  - † Design Data in InfraHub
    - ❖ Diagram
    - ❖ High Level Design Document
    - ❖ Cabling Matrix
    - ❖ Elevations
    - ❖ Testing Topology (Containerlab)
    - ❖ Configurations
    - ❖ Tests
  - † Device Configuration
    - ❖ As Built
  - † Device Monitoring (SuzieQ)
    - ❖ QA

Conclusions

What Now?

Any text is fair game for Jinja2

- Don't limit yourself to configurations
- Once you set up your little "template" factory your focus will be:
- on the data
  - Where to get it
  - How to manipulate it
  - What to add to it
- and its template

- Re-use your environment and rendering steps (turn them into functions!)

Opportunities Abound

- Target Rich Environment
  - Status Emails
  - Activity Summary
  - Prompt construction
  - AI Prompts
- AI can have a significant impact here with good accuracy

Extras

Service Now Developer Instance

Whitespace References

- Good tips for whitespace control if you are using Ansible
  - https://www.redpill-linpro.com/techblog/2023/07/19/jinja_whitespaces.html