Jinja2 Template Essentials

A focused cheat sheet for Python backend developers, specifically designed for quick reference to Jinja2 templating syntax and features. Master variables, control flow, inheritance, and macros with practical examples.

Core Template Syntax

VARIABLES & EXPRESSIONS

`{{ variable }}`	Outputs the value of a variable. Jinja2 automatically escapes HTML by default (autoescape).
`{{ user.name }}`	Accesses attributes of an object. `<p>Hello, {{ user.name }}!</p>`

`{{ items[0] }}`
`{{ data['key'] }}`
	Accesses elements of a list by index or dictionary by key.
	`First item: {{ items[0] }}`
	`Key value: {{ data['my_key'] }}`

`{{ 1 + 2 * 3 }}`	Supports basic arithmetic operations.
`{{ 'Hello ' ~ name }}`	String concatenation using the `~` operator.
`{{ text|upper }}`	Applies a filter to the variable's value. `<h1>{{ title|upper }}</h1>`
`{{ list|length }}`	Filter with an argument (or no arguments). `Total items: {{ items|length }}`
Pro Tip	Keep expressions simple. For complex logic, prepare data in your Python view function before passing to the template.

CONTROL STRUCTURES

`{% if condition %}`	Basic conditional statement.
	`{% if user.is_active %}`
	`  Welcome back!`
	`{% endif %}`

`{% if ... %} {% else %} {% endif %}`	If-else block for alternative content.
	`{% if user %}`
	`  Hello, {{ user.name }}!`
	`{% else %}`
	`  Please log in.`
	`{% endif %}`

`{% if ... %} {% elif ... %} {% else %} {% endif %}`	Multiple conditional branches.

`{% for item in items %}`	Iterates over sequences.
	`<ul>`
	`{% for product in products %}`
	`  <li>{{ product.name }}</li>`
	`{% endfor %}`
	`</ul>`

`{% for ... %} {% else %} {% endfor %}`	An `else` block for `for` loops executes if the sequence is empty.
	`{% for user in users %}`
	`  {{ user.name }}`
	`{% else %}`
	`  No users found.`
	`{% endfor %}`

`loop.index`, `loop.index0`	Current iteration of the loop (1-based, 0-based). `Item {{ loop.index }}: {{ item.name }}`
`loop.first`, `loop.last`	Boolean, true if current item is the first/last in the iteration.
Common Pitfall	Using `break` or `continue` inside Jinja2 loops is not directly supported. Filter or pre-process data in Python.

COMMON FILTERS (DATA MANIPULATION)

`{{ value|default('N/A') }}`	Provides a default value if the original value is undefined or false.

`{{ list|join(', ') }}`	Joins elements of a list with a specified separator.
	`Tags: {{ ['python', 'web', 'dev']|join(', ') }}`
	`{# Output: Tags: python, web, dev #}`

`{{ text|replace('old', 'new') }}`	Replaces all occurrences of a substring.
	`{{ 'Hello World'|replace('World', 'Jinja') }}`
	`{# Output: Hello Jinja #}`

`{{ text|trim }}`	Removes leading and trailing whitespace.
`{{ string|length }}`	Returns the number of items in a sequence or characters in a string.
`{{ number|round(2) }}`	Rounds a number to a specified precision. `round(2, 'floor')`, `round(2, 'ceil')`.
`{{ 'hello'|capitalize }}`	Capitalizes the first letter of the string.
Pro Tip	Chain filters: `{{ '  HELLO  '|trim|lower|capitalize }}`. Filters are applied from left to right.

Template Organization & Reusability

INCLUDES & TEMPLATE INHERITANCE

`{% include 'header.html' %}`	Inserts the content of another template at the current location.
`{% include 'nav.html' with context %}`	Passes the current template's context to the included template. This is often the default behavior.
`{% include 'optional.html' ignore missing %}`	Prevents an error if the included template doesn't exist.
`{% extends 'base.html' %}`	Inherits from a parent template, typically the first statement in a child template.

`{% block content %} ... {% endblock %}`	Defines a block in the parent template that can be overridden by child templates.
	In `base.html`:
	`<body>{% block body %}{% endblock %}</body>`
	In `child.html`:
	`{% extends 'base.html' %}`
	`{% block body %}<h2>Child Content</h2>{% endblock %}`

`{{ super() }}`	Renders the content of the parent block within an overridden block in a child template.
Pro Tip	Use `extends` for overall page layout and `include` for smaller, reusable components like navigation bars or footers that don't change the layout.

MACROS & REUSABLE BLOCKS

`{% macro input(name, type='text', value='') %} ... {% endmacro %}`	Defines a macro (reusable function-like block of HTML) with arguments and default values.
	`{% macro input(name, type='text', value='') %}`
	`  <input type="{{ type }}" name="{{ name }}" value="{{ value }}">`
	`{% endmacro %}`

`{{ input('username', 'text', user.name) }}`	Calls a macro by passing positional arguments.
`{{ input(name='password', type='password') }}`	Calls a macro by passing keyword arguments. Useful for clarity with many arguments.
`{% from 'forms.html' import input %}`	Imports a specific macro from another template file.

`{% import 'forms.html' as forms %} {{ forms.input(...) }}`	Imports all macros from a file into a namespace.
	`{% import 'forms.html' as forms %}`
	`{{ forms.input('email') }}`

`{% call card('Title') %} Content inside card {% endcall %}`	Allows passing a block of content to a macro using the `call` statement. The macro receives this content via the special `caller` variable.
	Macro Definition:
	`{% macro card(title) %}`
	`  <div class="card">`
	`  <h3>{{ title }}</h3>`
	`  {{ caller() }}`
	`  </div>`
	`{% endmacro %}`

Pro Tip	Macros are excellent for creating reusable UI components (e.g., form fields, buttons, cards) that maintain consistent structure and styling across your application.

COMMENTS & WHITESPACE

`{# This is a single-line comment #}`	Comments are ignored by the parser and do not appear in the final output.
`{# This is a multi-line comment #}`	For longer explanations or temporarily disabling blocks of code.

`{{- variable -}}`	Hyphens (`-`) after `{{` or before `}}` remove leading/trailing whitespace around the expression output.
	`<p>`
	`  Hello,`
	`  {{- name -}}!`
	`</p>`
	`{# Output: <p>  Hello,John!</p> (if name='John') #}`

`{%- statement -%}`	Hyphens after `{%` or before `%}` remove leading/trailing whitespace around the statement.
	`<ul>`
	`  {%- for item in items %}`
	`  <li>{{ item }}</li>`
	`  {%- endfor %}`
	`</ul>`
	`{# Output will have no empty lines or extra spaces around <li> tags #}`

`{{ '  text  '|trim }}`	While whitespace control in tags affects the template output, the `trim` filter affects the actual string value itself.
Pro Tip	Use whitespace control (`-`) to prevent empty lines or unwanted spaces in your rendered HTML, especially important for inline elements or when rendering lists without extra newlines.

Advanced Filters & Tests

COMMON FILTERS (OUTPUT & LOGIC)

`{{ html_string|safe }}`

Note: Jinja2 auto-escapes HTML by default. `safe` explicitly marks a string as safe, preventing escaping. Use ONLY with trusted HTML.

`{{ url|urlencode }}`

Encodes a string for use in URLs (e.g., query parameters).

`{{ html_text|striptags }}`

Removes all HTML/XML tags from a string.

`{{ long_text|wordcount }}`

Counts the number of words in a string.

`{{ dictionary|dictsort }}`

Sorts a dictionary by keys or values. Returns a list of (key, value) pairs.

```
{% for key, value in {'b': 2, 'a': 1}|dictsort %}
  {{ key }}: {{ value }}
{% endfor %}
```

Common Pitfall: Over-using `|safe` can introduce XSS vulnerabilities if applied to untrusted user input. Always validate and sanitize input on the server side.

TESTS

`{% if variable is defined %}`

Checks if a variable is defined (exists in the context).

`{% if variable is not none %}`

Checks if a variable's value is not `None`.

`{% if number is divisibleby(3) %}`

Checks if a number is divisible by another number.

`{% if 'hello' is equalto('hello') %}`

Checks for equality. Can also use `==`.

`{% if list is empty %}`

Checks if a sequence or mapping is empty.

`{% if value is number %}`

Checks if a value is a number (integer or float).

`{% if 'item' in my_list %}`

Checks for membership (can also be used in `if` statements, not just `is`).

Pro Tip: Tests are powerful for conditional rendering without writing complex expressions. Use them to check variable states, types, and properties cleanly.