templatelite Documentation

Release 0.2.1

[u'Tony Flury']

Sep 30, 2019

Contents

Note: Every care is taken to try to ensure that this code comes to you bug free. If you do find an error - please report the problem on :

- GitHub Issues
- By email to : Tony Flury

CHAPTER 1

Installation

Installation is very simple :

```
$ pip install templatelite
```

CHAPTER 2

Introduction

Templatelite is a lightweight templating module which is specifically designed to enable code creation as well as creation of other forms of data.

The templates are based on jinja templating language with a the exception that templalite does not support blocks, and does not support template extending; i.e. the ability for one template to override one or more blocks in another template.

templatelite supports a python like language within the template :

- for loops with multiple target variables, and supporting break and continue and else
- if conditional with elif and else

In general the templating process works as follows :

- Create an instance of the Renderer class, using your template either from a string or a file
- Use one of the methods of the class to add data to the template, the data can be created from a dictionary, multiple key word arguments or json data. The process of adding data to the template is termed `rendering`, and the data used to render the template is called the context

2.1 Access to Data

Within the templating language there is support for the display of data from the conex, using `Context Variables` (i.e. variable names surrounded by *{{* & *}}* (as in the example below).

```
{{ name }}
```

When this template is rendered using a dictionary with a key of *name*, the output will be the template with *{{ name }}* replaced with the value in the dictionary for that key. See *Context Variables* for more details.

2.2 Control Structures

As well as accessing data from the context the templating language supports simple looping, and conditional branching through template directives; Python like commands surrounded by *{% & %}* as in the examples below.

2.2.1 Loops

Tempalite supports for loops which allow for iteration around context data :

```
{% for person in people %}
    {{ person }}
{% endfor %}
```

For full details of the for loop see *For Loops*. The templatelite module does not have while loops.

2.2.2 Conditional Branching

Tempalite supports if statements so that your template can implement 'decisions' based on data.

```
{% if person == 'Elizabeth Windsor'  %}
     Her Majest Queen Elizabeth
{% else %}
    A commoner
{% endif %}
```

For full details of the if directive see *If Conditions*.

2.3 Comments

Comments can be added to templates by surrounded your comment by *{# & #}*. In complex templates these comments can be useful in understanding your own templates. They are ignored entirely :

```
This is a {# This is a comment #}message
```

Will be rendered as :

> This is a message

CHAPTER 3

Context Variables

As the name suggests, ContextVariables are the method whereby templates can access data within the 'context'; the context is the dictionary passed to the Renderer class in the 'from_context' or *from_json* methods.

In it's simplest form a Context Variable is a single name, which will be a key within the context data; for instance `name` within a template will look for the key of `name` with the context.

3.1 dotted names

As well as single names, templatelite supports dotted names - which allow deeper access to the data within the context. It is neccessary to distinguish between displayed data access - i.e. those with *{{ }}*, and dotted names with are used in :ref: `ForLoops` and *If Statements*

3.1.1 displayed data

Within displayed data, an example dotted name of `person.name` variable will mean one of three things :

- If the `person` key within the context is a dictionary (or any object which suports a Mapping type interface, then `person.name` in the template is equivalent to python code: `person['name']`
- if `person` key within the context is an object which has a `name` method, then `person.name` in the template will be equivalent to `person.name()` - there is no capability within displayed context variables (i.e. those enclosed in *{{ }}* within dotted names to pass parameters to these methods - but see *Filters*.
- if `person` key within the context is an object which has a `name` attribute, then `person.name` in the template will be equivalent to `person.name`.

These dotted names can be nested as deep as required; there is no reason why a context variable of for instance : `company.client.recent_order.value` shouldn't actually be translated to the equivalent of context.client['recent_order'].value()

3.1.2 Expressions

Within *For Loop* or *If Statement* then as well the tempalite dotted name extensions above, normal python syntax for data access can be used; for instance:

- access by numeric index or slicing using `container[index|slice]`: here container will be a valid *dotted name* as above, and index/slice can either be a literal numeric value, or be another *dotted name*
- access to keys in dictionaries using `container[key]` where container is a dotted name and key can be a literal name, or another dotted name,
- calling methods and using paramaters using `object.method_name(parameters)` so long as the parameters are either positional, and any keyword arguments are passed by unpacking a dictionary (i.e. using the **) operator. All unquoted text is interpreted as a name within the context, and therefore errors are going to happen (either those keywords will be found within the context and replaced with data from the context, or the name won't be found in the context and an error will occurr).

3.2 Filters

Filters can be applied at the end of dotted names (wether in *displayed data*, or *Expressions*) by using a '|' (vertical bar), and there are several builtin filters. Filters must be the last part of a context variable whether that is in a *{{ }}* directive or in an expression.

See *Filters* for a full list of filters.

CHAPTER 4

For Loops

Within templatelite, for loops have the following format:

```
{% for <targets> in <iterable> %}
    <Loop statements>
{% endfor %}
```

`<targets>` a comma separated list of names (without dots) - just as in normal python each of these targets are created as local variables within the template, and can be accessed as if they are context variables.

`<iterable>` any expression which returns an iterable (as per usual python syntax). All names which are not quoted are assumed to be either local names created as the targets of other for loops, or context variables. These are full python syntax expression, including index and slicing, function calls and all mathematical and logical operators.See *Expressions* for more details.

`<Loop statements>` Any combination of text, *Context Variables*, other loops, and *If Conditions*

Note that the `{% endfor %}` directive is mandatory.

Loops Inside a template can contain:

- `{% break %}` which does the same as break in normal python - i.e. it immediately exist the containing loop.
- `{% continue %}` which directives which have the same meanings as in usual for loops within Python - it moves execution to the start of the loop
- `{% else %}` which will also behave in exactly the same way as in Python - the block after the {% else %} is only executed if the loop executes to the end - i.e. no {% break %} is executed.

With the else clause the full syntax is :

```
{% for <targets> in <iterable> %}
    <Loop statements>
{% else %}
    <Else Statements>
{% endfor %}
```

Loops can contain other loops, and there is no practical limit to the level of nesting that can be used.

CHAPTER 5

If Conditions

within tempalite, a template can have conditional sections by using the an `if` directive:

```
{% if <conditional> %}
    <statements>
{% elif <conditional> %}
    <statements>
{% else %}
    <statements>
{% endif %}
```

`<conditional>` Any valid expression which generates a boolean value - just as in usual Python if statements

`<statements>` Any combination of text, *Context Variables*, other loops, and *If Conditions*

The `{% elif <conditional> %}` and `{% else %}` directives are entirely optional (as they are in Python), but the `{% endif %}` statement is mandatory. Unlike normal python code indentation of the directives is not required (but is good practice in order to illustrate the structure of the template.

CHAPTER 6

Filters

6.1 String filters

center Centers the variable into a space, with an optional fill character `{{ var|center 20 }}` : is equivalent to var.center(20) `{{ var|center 20 '#'}}` : is equivalent to var.center(20,'#')

cut Removes all of a given character from the string `{{ var|cut 'x' }}` : is equivalent to var.replace('x',' ')

len Returns the length of the context variable - equivalent to len(<variable>) `{{ var|len }}` : is equivalent to len(var)

split Splits the contex variable into a list. As a default this splits the value at each space character, equivalent to <variable>.split() Takes one optional argument which is the character to split on. `{{var|split 'x' }}` : is equivalent to var.split('x') `{{var|split 'x' 5 }}` : is equivalent to var.split('x', 5)

CHAPTER 7

Templatelite Module

7.1 Exceptions

exception templatelite.**UnknownContextValue**

Raised when a Context Variable does not exist. This is the dot separated version of the variable name

exception templatelite.**UnrecognisedFilter**

Raised when a filter is invoked on a Context Variable, but the filter is not recognised.

exception templatelite.**UnexpectedFilterArguments**

Raised when arguments are provided for a given filter, but where those arguments were not unexpected.

exception templatelite.**TemplateSyntaxError**

Raised when the template does not meet the expected syntax - this will be caused by an missing or unexpected directive

7.2 Renderer Class

class templatelite.**Renderer**(*template_str=None*, *template_fp=None*, *template_file=”*, *errors=False*, *default=None*, *remove_indentation=True*)

A General purpose Template renderer

Parameters

- **template_str** – The Template to render
- **errors** – A Boolean flag - True if errors in the template should cause an exception
- **default** – The default value to insert into the template if an error occurs.
- **remove_indentation** – Whether or not to remove the left margin indentation.

By using the default values from the class, any data access error in a `ContextVariable` will cause that context Variable to be rendered into the template as the unconverted context variable name. Errors in accessing data within for loops or if conditions then the value of None is used.

if `default` is set then this string is used under error conditions (rather than the context variable name)

If `errors` is set then any error within the template will cause a `UnknownContextValue`, `UnrecognisedFilter` or `UnexpectedFilterArguments` exception as appropriate.

The `remove_indentation` flag will strip all left margin indentation from the template as it renders. This setting is suitable for templates where any indentation is inconsequential (e.g. html). If the template is intended to create output where indentation needs to be preserved (Restructured Text (.rst), Python Source Code (.py) then `remove_indentation` needs to set to false).

classmethod execute_filter(*filter_name=”*, *token=”*, *value=None*, *args=()*, *kwargs={}*)

Generic class method to execute a named filter

Run at execute time

from_context(**contexts*)

Public I/f Render the template based on one or more dictionaries

classmethod register_filter(*name*, *filter_callable*)

Register a named modifier - internal use only

Index