Templaty

Release 1.0.0

Dec 06, 2019

Contents:

1 Templaty Language Reference **3**
- 1.1 Sample Templaty File . . . 3
- 1.2 Regular Text . . . 4
- 1.3 Expressions . . . 4
- 1.4 Regular Code . . . 4
- 1.5 Control Flow . . . 4
- 1.6 Indentation Control . . . 5
- 1.7 Built-in Variables and Functions . . . 7

2 Command-Line Interface **11**
- 2.1 Examples . . . 11

3 Indices and tables **13**

Important: This software is undergoing rapid development. Be aware that some things might not yet work as advertised. This message will be removed once a stable release is reached.

Templaty is a code generator written in Python that focuses on generating code for programming languages other than HTML. It features fine-grained control over whitespacing and indentation and a rich meta-language that allows full control over the code to be generated.

CHAPTER 1

Templaty Language Reference

This page explains in detail how the Templaty language is structured.

1.1 Sample Templaty File

The following example demonstrates most of Templaty's capabilities. It generates a C program that prints a predefined array of random numbers.

```
{% if 'author' in globals() %}
  // Copyright {{year}} {{author}}
  // All rights reserved
{% endif %}

{!
  from random import randint
  MAX_RANDOM = 10
!}

int randomData[] = [
  {% join i in range(0, random_data_length) with ',' %}
    {{randint(0, MAX_RANDOM)}}
  {% endjoin %}
]

int main() {
    {% for i in range(0, random_data_length) %}
      {% noindent %}
        #if defined(SAMPLE_DEBUG)
      {% endnoindent %}
      fprintf(stderr, "Going to write random number #{{i}} ...");
      {% noindent %}
        #endif // SAMPLE_DEBUG
      {% endnoindent %}
```

(continues on next page)

(continued from previous page)

```
      printf("%i\n", randomData[{{i}}])
    {% endfor %}
}
```

1.2 Regular Text

Regular text is just passed along as-is, without any preprocessing done to it.

1.3 Expressions

Expressions are used in the control-flow statements and can also appear anywhere in the template to generate specific text. If you want to add an expression to a piece of text, you have to wrap it between `{{` and `}}`, like so:

```
// Did you know that 25 + 17 equals {{25 + 17}}?
```

1.3.1 Variable References

Any identifier that is not a keyword may be used to reference one of the built-in expressions or an expression that was defined elsewhere in the template.

```
/* This file was generated on {{now}} by {{author}}. */
```

1.4 Regular Code

At any time, you can write regular Python code using `{!` and `!}`.

The text surrounding the code block will be automatically stripped so that no excessive newlines are generated.

```
{!
  my_var = 42
!}
The answer is {{my_var}}.
```

1.5 Control Flow

The following statements are used in Templaty to change what text is written when the template is run. Most of them should be very familiar, as they resemble the constructs found in most regular programming languages.

1.5.1 Conditional Code Generation

Conditionals are used to generate a piece of text depending on whether a given predicate is met. Just like in regular programming languages, you can have conditionals with multiple alternatives or none at all.

```
{% if header %}
  // This header is only generated if 'header' is set to true
  // in the environment. You can create a JSON file that contains
  // this variable, or define it somewhere in the template itself.
{% endif %}
```

The `else`-directive is used to provide some text when no predicate matched, like so:

```
{% if long_header %}
  // This file does stuff. It is really cool because it first does
  // stuff and then some more stuff. Once the stuff is finished, it calls
  // a 'thing' to do other stuff.
{% else %}
  // This file does stuff.
{% endif %}
```

1.5.2 Generating Repititions

Templates for code generation wouldn't be particularly useful if we couldn't use them to auto-generate repetitive code. The `for`-statement is one of the simplest methods for generating (possibly huge) amounts of code.

```
IDENTITY_MATRIX = [
    {% join i in range(0, 10) with ',' %}
      [{% join j in range(0, 10) with ',' %}{% if j == i %}1{% else %}0{% endjoin %}]
    {% endjoin %}
]
```

Generates the following code:

```
IDENTITY_MATRIX = [
    [1,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,0,0],
    [0,0,1,0,0,0,0,0,0,0],
    [0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,1,0,0,0,0],
    [0,0,0,0,0,0,1,0,0,0],
    [0,0,0,0,0,0,0,1,0,0],
    [0,0,0,0,0,0,0,0,1,0],
    [0,0,0,0,0,0,0,0,0,1]
]
```

1.6 Indentation Control

A feature of Templaty that stands out is how it handles indentation and whitespaces. Because the code generated by Templaty might be read by other developers, special care has been taken that spaces and newlines are correctly generated.

Consider the following template code for a Python program:

```
def main():
    {% if enable_print_foo %}
      foo = get_foo();
```

(continues on next page)

(continued from previous page)

```
      if foo == 2:
          print("Foo is two!")
      else:
          print("Foo is not two :(")
    {% endif %}
```

Some users might be surpised to learn that this template generates the following code:

```
def main():
    foo = get_foo()
    if foo == 2:
        print("Foo is two!")
    else:
        print("Foo is not two :(")
```

However, the rules are quite natural. Templaty takes the indentation of the leading `{%` and applies it to each line that is generated within the block. In order to make sure there isn't too much indentation, Templaty removes any indentation that is shared by all the lines inside the statement block.

This rule also works when nesting multiple statements inside each other. For example:

```
POINTS = [
    {% join i in range(0, 10) with ',' %}
        {% if use_vector %}
          Vec({{i}}, {{i}})
        {% else %}
          ({{i}}, {{i}})
        {% endif %}
    {% endjoin %}
  ]
```

A call to this program with `use_vector` set to `True` could result in the following code:

```
POINTS = [
    Vec(7, 3),
    Vec(4, 9),
    Vec(9, 1),
    Vec(3, 2),
    Vec(4, 5),
    Vec(8, 3),
    Vec(5, 8),
    Vec(1, 8),
    Vec(1, 6),
    Vec(2, 1)
]
```

1.6.1 The `setindent`-block

The special statement `{% setindent indent_level %}` can be used to override the auto-inferred indentation level.

```
int main() {
  {% noindent %}
    #ifndef FOO
  {% endnoindent %}
```

(continues on next page)

(continued from previous page)

```
  fprintf(stderr, "Warning: FOO was not defined at compile-time.");
  {% noindent %}
    #endif // #ifndef FOO
  {% endnoindent %}
}
```

Output:

```
int main() {
#ifndef FOO
  fprintf(stderr, "Warning: FOO was not defined at compile-time.");
#endif // #ifndef FOO
}
```

If you need even more control over the indentation level, you can make use of the special `indent()` function. When called with no arguments, it increases the indentation with one level for the rest of the file. When called with an integer, it will set the indentation level to that number.

```
if not prompt("Attempt no 1"):
{% for i in range(2, 3) %}
  {! indent() !}
  if not prompt("Attempt no {{i}}"):
{% endfor %}
error("I gave up.");
{! clearindent() !}
```

The above snippet will generate the following code:

```
if not prompt("Attempt no 1"):
      if not prompt("Attempt no 2"):
          if not prompt("Attempt no 3"):
              error("I gave up.")
```

1.7 Built-in Variables and Functions

Templaty contains a growing number of built-in functions and variables to make it easy for programmers to write their templates without much hassle. The folllowing is an incomplete list of functions and variables that are supported out-of-the-box.

`v |> f`

A special operator that applies a given function `f` to `v`.

This operator allows you to write code such as:

```
'FooBarBaz' |> snake |> upper
```

Which is equivalent to the following code:

```
upper(snake('FooBarBaz'))
```

Note the similarity with Jinja2's *filter* concept, with the difference that Templaty implements it as a regular operator rather than a syntactic extension.

`upper(text)`

Simply converts the given text to uppercase, using Python's standard behaviour.

`lower(text)`

Simply converts the given text to lowercase, using Python's standard behaviour.

`indent(level)`

> **Warning:** This feature is currently under development.

When called with no arguments, it increases the indentation with one level for the rest of the file. When called with an integer, it will set the indentation level to that number.

```
if not prompt("Attempt no 1"):
{% for i in range(2, 3) %}
  {! indent() !}
  if not prompt("Attempt no {{i}}"):
{% endfor %}
{! indent(0) !}
error("I gave up.");
```

The above snippet will generate the following code:

```
if not prompt("Attempt no 1"):
      if not prompt("Attempt no 2"):
          if not prompt("Attempt no 3"):
              error("I gave up.")
```

`snake(name)`

Converts an identifier to snake-case.

This function should work on most common use-cases. For more complex ones, you probably should write your own logic.

```
snake('FooBarBaz')
```

Output:

```
FOO_BAR_BAZ
```

`now`

A variable holding the time the generator started, formatted using some default rules.

`a + b`

Add two expressions to each other.

`a - b`

Subtract two expressions from one another.

`a * b`

Multiply two expressions with each other.

`a / b`

Divide `a` by `b`, returning the result.

`a % b`

Find the remainder after the division of the two given numbers.

CHAPTER 2

Command-Line Interface

Templaty comes with a simple command-line interface (CLI) so that you don't have to write a single line of Python code. The command is called `templaty` and is available by default when installing using `pip install`.

2.1 Examples

Writing the resulting text of a Templaty file to *stdout*:

```
templaty mytemplate.tply
```

Passing data to the template in JSON format:

```
echo '{"author":"Sam Vervaeck","copyright":"2019"}' | templaty mytemplate.cc.tply --
↪stdin
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search