Template engine reference

The template engine is the link between your html server and *magic.*

With the template language functional, it is possible to write Python code inside html code and have it recognised and run.

For help implementing the template engine in Tornado, see the next section. Otherwise:

The template engine recognises, interprets and runs Python code, returning the result(s) as html where the Python code was written. The most fundamental feature of the engine is recognizing an expression, the format of which is:

**{{ expr }}**

That is, two opening braces, an expression (denoted by **expr**) and two closing braces. Spaces between the braces and expression are optional, but are recommended for readability. Multi-line expressions are not supported: Expressions must not have new lines inside them, even if it looks neater.

An expression may contain variables – be sure they are in the same context as the expression. As is the case for all recognisable Python code, an expression is evaluated and the result takes its place when the html code is interpreted by the browser.

**For god’s sake *DO NOT:***

1. Guess the template syntax instead of consulting this documentation. **You’ll break the server.**
2. Run your code in an expression without having tested it in IDLE. **You’ll break the server.**
3. Ignore either of the above. **I’ll break your face.**

Implementing the template engine in Tornado:

Be advised that a concise version of this is currently on the whiteboard.  
To use the engine in tornado, its files must first be imported:

**import engine.template as magic**

The files are now accessible to the system. This gives you access to one function, **render\_page**, which passes the page through the engine and runs all python code.

*If the page is not passed through the engine using this function, python code will not run*. The template engine is magic, but only a controlled kind. You must implement the magic yourself. To render a page, use:

**magic.render\_page(‘*Page.html*’, *variables*)**

‘magic’ refers to engine.template, which you imported earlier. Replace *Page.html* with the path to the html page you are rendering (be sure this is a string) and *variables* with variables to be used in expressions that use variable names in the page.  
*Variables* accepts a dictionary; To clarify, keys are variables and values their values. These variables will often be items accessed from the SQLite database, that are dependent on the user.  
If you have no variables, use an empty dictionary {}.

Currently recognisable tags:

All tags perform as they do in Python (with one universal exception), with respect to surrounding html and Python code.  
Tags are written exactly as they are in Python, enclosed in *a pair of percentage signs and braces*, so they are recognisable by the engine. Spacing between the code and percentage signs is optional, but recommended for readability.  
*The engine also requires that*, for tags which manipulate following blocks of code, an ‘end tag’ tag is used. If tags are simple examples of this.

1. Include

The Include tag includes (pun/10) another template in the place of the tag (templates found in the templates directory of our repository), its format being:

**{% include file.html %}**

1. If

The If tag runs Python and html code depending on the truth of the expression included in the tag. Because the expression is part of the tag, it does not need to be enclosed in two pairs of braces:

**{% if expr %}**

That is, an opening brace, percent sign, ‘if’, an expression, a percent sign and closing brace.

An If tag requires that an end If tag is included after the code which depends on the If statement:

**{% end if %}**

1. For

The For tag runs Python and html code in the context of each item in a loop:

**{% for expr %}**

**{% end for %}**

1. Else  
   The Else tag can be contained within an If block and contains of a block of code which is evaluated when the preceding If statement’s condition evaluates to false:

**{% if expr %}**

**{% else %}**

**{% end if %}**

1. Safe

Use the Safe tag to evaluate html code as html code, rather than text, when in a Python expression. An example of this is <br>, which if evaluated as html becomes a newline character, but if no Safe tag is used, is displayed exactly as <br>. The syntax is:

**{% safe expr %}**

The expression is still a Python expression, but any html code inside it is evaluated as html. An example is:

**SAAAAAAFE**!

This

Should

Be

On

Multiple

Lines

**UNSAFE**!

This<br>should<br>not<br>be<br>on<br>multiple<br>lines.

This will not often be needed and may not be easily understood – ask the template team.

1. Comment

This tag makes it possible to add a comment which is not passed to the client and cannot be seen in their source code:

**{% comment text %}**

For example, the code:

**{% comment Kosta what the f@%^ is this s%\*&? %}**

Will temporarily increase Kosta’s bug finding speed in this section of the page by about 48%.

Michael’s expression notes (if you still don’t get them):

Expressions are anything in Python which returns a value. Some examples are 3 + 4, and x \* 2 (where x is a variable).

The **{{ expr }}** syntax does the expression inside and puts it on the page.

If you’d like to put variables on the page, for example, x, you need to put in another parameter which is the “context” of the template. This should be a dictionary with strings as keys and anything as values, like {“x”: 4}.