Jinja 2

Python template engine brief overview

Stepan S
ssin@tuta.io

Agenda

1. Templating engines
2. Jinja: features and basics
3. Flask examples
4. Rendering to files

Templating engines

What is template engine?

- Framework built mainly to separate the code (logic) from the layout.
- Template engines take in tokenized strings and produce rendered strings with values in place of the tokens as output.

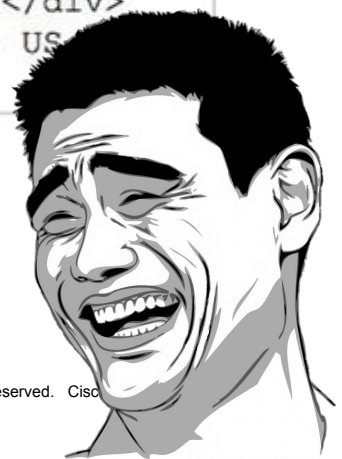
Combining template+data produces the final output

```
{
  Name: "Joe Marini",
  Phone: "555-1234",
  Address: "Anytown, US"
}
```

```
<div>
  <div>{{Name}}</div>
  <div>{{Phone}}</div>
  <div>{{Address}}</div>
</div>
```

Template Engine

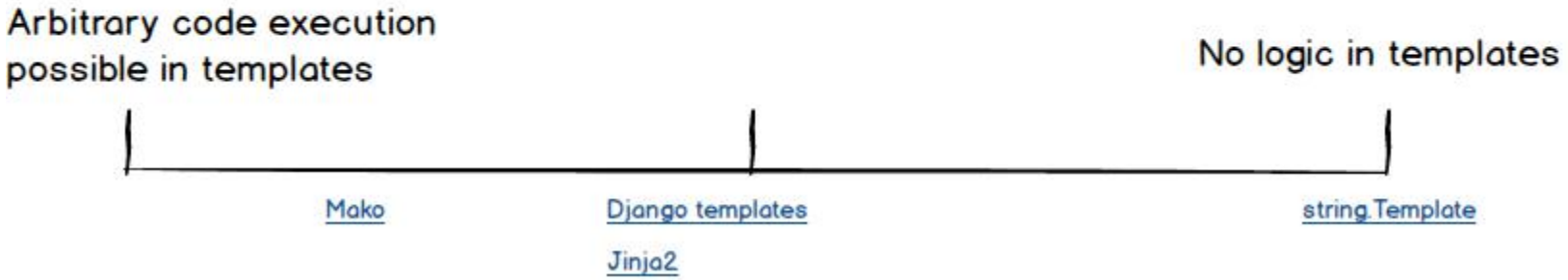
```
<div>
  <div>Joe Marini</div>
  <div>555-1234</div>
  <div>Anytown, US
</div>
```



Python template engines

- Django template engine
- Mako
- Chameleon
- Cheetah
- Jinja

Template engine embedded logic spectrum



Jinja

Is the name of a Japanese temple and temple and template share a similar pronunciation. It is not named after the city in Uganda.

Features:

- Sandboxed execution
- Powerful automatic HTML escaping
- Template inheritance
- Compiles to the optimal python code
- High performance
- Easy to debug

```
>>> from jinja2 import Template
>>>
>>> t = Template("Hello {{ something }}!")
>>> t.render(something="World")
u'Hello World!'


>>> t = Template("My favorite numbers: {% for n in range(1,10) %}{{n}} " "{% endfor %}")
>>> t.render()
u'My favorite numbers: 1 2 3 4 5 6 7 8 9 '
```

```
from jinja2 import Environment

HTML="""<html><head><title>{{title}} Title </title></head> <body>Hello.</body>
</html> """

def print_html():
        print Environment().from_string(HTML).render(
                title='Hello'

print_html()
```

```
<html><head><title>Hello Title</title></head> <body>Hello.</body> </html>
```

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">
                    {{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

{% ... %} for Statements
{{ ... }} for Expressions to print to the template output
{# ... #} for Comments not included in the template output

{% **extends** "layout.html" %} – *inheritance*
{% **block** body %} – *super blocks*
...
{% **for** user **in** users %} – *control structure (if/elif/else)*
{{ user.username | capitalize }} – *filters*
{% **endfor** %} – *end of statement*
...
{% **endblock** %} – *end of block*

And powerful

Base Template

```
<head>
   {% block head %}
   <link rel="stylesheet" href="style.css" />
   <title>{% block title %}{% endblock %} - Webpage</title>
   {% endblock %}
</head>
<body>
   <div id="content">
      {% block content %}
            {% endblock %}
   </div>
   <div id="footer">
      {% block footer %}
      Copyright 2017
      {% endblock footer %}
   </div>
</body>
</html>
```

Child Template

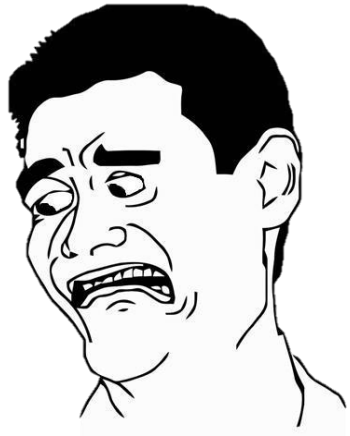
```
{% extends "base.html" %}
{% block title %}Index{% endblock %}
{% block head %}
   {{ super() }}
   <style type="text/css">
      .important { color: #336699; }
   </style>
{% endblock %}
{% block content %}
   <h1>Index</h1>
   <p class="important">
     Welcome to my awesome homepage.
   </p>
{% endblock %}
```

Macros

```
{% macro input(name, value='', type='text', size=20) -%}
    <input type="{{ type }}" name="{{ name }}"
        value="{{ value|e }}" size="{{ size }}">
{%- endmacro %}
```

```
{% from 'macros.html' import input %}
```

```
<p>{{ input('username') }}</p>
<p>{{ input('password', type='password') }}</p>
```



Imports:

```
{% import 'forms.html' as forms %}
...
<p>{{ forms.input('username') }}</p>
<p>{{ forms.textarea('comment') }}</p>
```

```
{% from 'forms.html' import input as input_field, textarea %}
...
<p>{{ input_field('username') }}</p>
<p>{{ textarea('comment') }}</p>
```

```
{% from 'forms.html' import input with context %}
```


Filters

Variables can be modified by filters:

{{ name|striptags|title }}

' first name ' >> 'First Name'

Some of builtin filters:

default(value, default_value=u'', boolean=False)

{{ ''|default('the string was empty', true) }}

escape(s), e(s)

join(value, d=u'', attribute=None)

length(object)

reverse(value)

Simple Custom Filter

```
@app.template_filter('full_name')
def full_name_filter(product):
    return '%s / %s' % (product['category'], product['name'])

...

app.jinja_env.filters['full_name'] = full_name_filter


...

{{ product|full_name }}
```

Environment

class jinja2.**Environment**([options])

autoescape – makes XML/HTML autoescaping feature enabled by default
Loader – the template loader for environment
trim_blocks

block_start_string - Defaults to '{%'.
block_end_string - Defaults to '%}'.
variable_start_string - Defaults to '{{'.
variable_end_string - Defaults to '}}'.
comment_start_string - Defaults to '{#'.
comment_end_string - Defaults to '#}'.

Loaders

jinja2.FileSystemLoader(searchpath, encoding='utf-8', followlinks=False)

jinja2.PackageLoader(package_name, package_path='templates', encoding='utf-8')

jinja2.DictLoader(mapping)

get_template(*name, parent=None, globals=None*)

Load a template from the loader.

Custom Loader

jinja2.BaseLoader – baseclass for all loaders

```
from jinja2 import BaseLoader, TemplateNotFound
from os.path import join, exists, getmtime

class MyLoader(BaseLoader):
    def __init__(self, path):
        self.path = path
    def get_source(self, environment, template):
        path = join(self.path, template)
        if not exists(path):
            raise TemplateNotFound(template)
        mtime = getmtime(path)
        with file(path) as f:
            source = f.read().decode('utf-8')
        return source, path, lambda: mtime == getmtime(path)
```

jinja2.Template

The central template object.

```
>>> template = Template('Hello {{ name }}!')
>>> template.render(name='John Doe') == u'Hello John Doe!'
True
```

globals - *dict with the globals of that template.*

name - *loading name of the template*

render([context]) - *method accepts the same arguments as the dict constructor: A dict, a dict subclass or some keyword arguments:*

template.render(knights='that say nih') template.render({'knights': 'that say nih'})

will return the rendered template as unicode string.

```
from jinja2 import Environment, PackageLoader, select_autoescape
env = Environment(
    loader=PackageLoader('yourapplication', 'templates'),
    autoescape=select_autoescape(['html', 'xml'])
)


template = env.get_template('mytemplate.html')


print template.render(the='variables', go='here')
```

Global Functions

range([start,]stop[, step])

*dict(**items)*
A convenient alternative to dict literals. {'foo': 'bar'} is the same as dict(foo='bar').

*class cycler(*items)*

class joiner(sep=', ')

Flask examples

```
from flask import Flask, render_template
app = Flask(__name__)


@app.route("/")
def template_test():
    return render_template('template.html',
                           my_string="Wheeeee!",
                           my_list=[0,1,2,3,4,5])

if __name__ == '__main__':
    app.run()
```

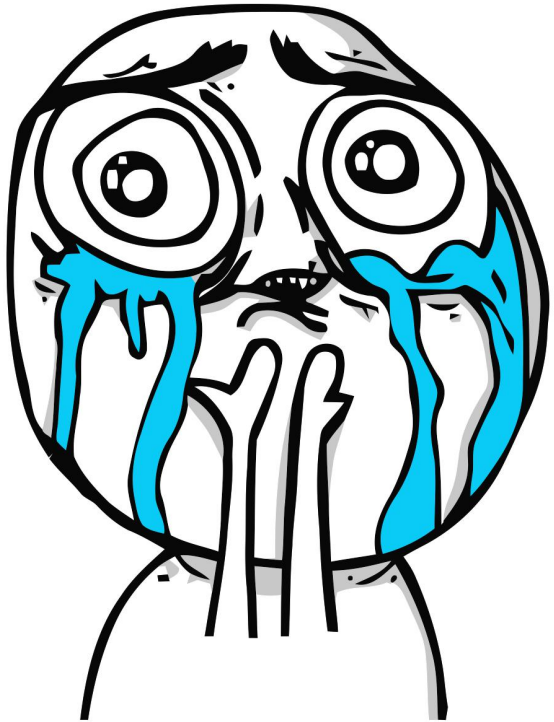
Rendering to files

```
TEMPLATE_ENV = Environment(
    autoescape=False,
    loader=FileSystemLoader(os.path.join(PATH, 'templates')),
    trim_blocks=False)
...

def render_template(template_filename, context):
    return TEMPLATE_ENV.get_template(
        template_filename).render(context)

def create_file(fname, context):
    with open(fname, 'w') as f:
        html = render_template('template.html', context)
        f.write(html)

create_file(output_file, config_file_context)
```

```
...
<div class="container">
  {% block heading %}
    <h1>{% block page %}
        {{ my_title }}
        {% endblock %} - File Generation Example</h1>
  {% endblock %}
  <h2>Actually looks the same as Flask example</h2>
  <br>
  <p>Some string: {{ my_string }}</p>
  {% if 'kitten' in my_list %}
    <p>We have kitten in list!!!</p>
  {% endif %}
</div>
...
```

Questions?

Thank you!



Resources

1. Jinja docs
2. Template Engines - Full Stack Python
3. Jinja2 - Full Stack Python
4. Primer on Jinja Templating - Real Python