

## JINJA TEMPLATES

## variables

```
<p>My name is {{ name }}!</p>
```

## if

```
{% if age > 17 %}
  <p>You may continue.</p>
{% else %}
  <p>Too young.</p>
{% endif %}
```

## for

```
{% for post in blog_posts %}
  <h2>{{ post.title }}</h2>
{% empty %}
  <p>No posts found!</p>
{% endfor %}
```

## filters

```
<p>Hi {{ name|upper }}</p>
```

## include

```
<form>
{% include "form.html" %}
</form>
```

## extends &amp; blocks

```
{% extends "base.html" %}
{% block title %}
  This replaces base.html's
  title block
{% endblock %}

{% block main_content %}
  Body might go here
{% endblock main_content %}
```

## USING JINJA

```
from jinja2 import Template

template = Template("""
<h1>hi {{ name }}!</h1>
""")

result = template.render(
    name="Joaquín",
)
print(result)
```

## PIPVENV

## Creating a new virtualenv

```
pipenv --python 3.6
```

## Enter current virtualenv

```
pipenv shell
```

## Install a new package from PyPI

```
pipenv install jinja2
```

## Install all packages listed in Pipfile

```
pipenv install
```

## MODULES

A module is a file or directory which provides functions, classes, or variables that can be imported and used by other Python files. Syntax and dir structure:

```
# - main.py
# - module_name/
#   - __init__.py
#   - submod_a.py
#   - submodule_b.py
import module_name
from module_name import (
    submod_a,
    submodule_b,
)
from module_name.submod_a import (
    function_name,
    variable_name,
    ClassName,
)
```

## JINJA KEY TERMS

**Context** A dictionary representing a collection of *context variables* to be inserted or otherwise used in various places in a template

**Template** A string or file, often consisting of HTML, containing “placeholder” spots for variable data to be inserted, and sometimes simple logic

**Render** When a template is combined with a context to produce finished results

## OOP TERMINOLOGY

**OOP** Object Oriented Programming - a way of thinking and arranging data in programming that groups types of data (“properties”) with functions (“methods”) and calls the entire thing a “class”

**Class** also known as an object’s “type”, classes form “blueprints” for creating new object instances, defining methods and properties

**Object instance** A specific occurrence of a class. Creating one is called *instantiation* or *construction*.

**Method** A function defined in a class declaration that gets attached (“bound”) to every class instance, and can be accessed with a `.` character

**Property** Data stored by the class, can be accessed with a `.` character

**Constructor** A special method that is run when you instantiate a class

**Extend** Classes (the subclass) can *inherit* or *extend* another class (the base class) which effectively copies over all the methods and property defaults

**Overriding** When a subclass replaces a base class method we say it is overridden

**Super** Super is a keyword to allow subclasses to access an overridden method on a base class

**Software architecture** High-level, executive summary of the design of a piece of software to facilitate collaboration within a team

**Interface** The outwardly facing methods and properties of a class

## PYTHON CLASS SYNTAX

```
class User:
    def __init__(self, name):
        self.name = name
        self.logged_in = False

    def login(self):
        self.logged_in = True

class StudentUser(User):
    def login(self):
        super(self).login()
        self.attended = True
```