



POLITECNICO  
DI TORINO



e-Lite

# Building Web Applications

**Ambient intelligence**

Fulvio Corno

Politecnico di Torino, 2015/2016



# Goal

- Create simple web applications
  - In Python
  - For interactive interfaces
  - For server-side components
- Learn a simple framework
  - Start simple
  - Extensible with modules

# Summary

- Programming the web in Python
- Flask architecture and installation
- First Flask application
- Jinja2 Templates
- User interaction
- Flask extensions
  - Bootstrap

Building Web Applications

# PROGRAMMING THE WEB IN PYTHON

2015/2016

Ambient intelligence



# Python and the Web

- Several libraries & frameworks
- Different features & complexity



<https://www.djangoproject.com/>



Pyramid™

<http://www.pylonsproject.org/>



Flask

web development,  
one drop at a time

<http://flask.pocoo.org/>



CherryPy

<http://www.cherrypy.org/>

SimpleHTTPServer  
(standard library)

And (too) many more...

[https://wiki.python.org/moin/  
WebFrameworks](https://wiki.python.org/moin/WebFrameworks)

Building Web Applications

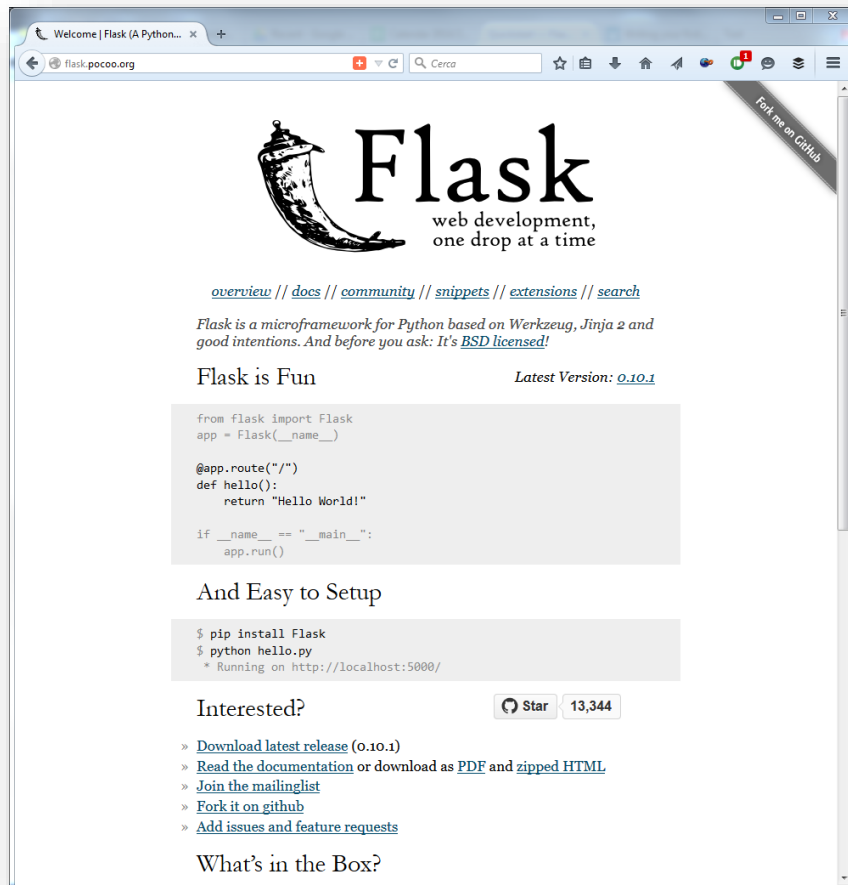
# FLASK ARCHITECTURE AND INSTALLATION

2015/2016

Ambient intelligence



# Resources



The screenshot shows the Flask website homepage in a browser window. The URL is flask.pocoo.org. The page features the Flask logo (a flask) and the text "Flask web development, one drop at a time". Below this, there are links for "overview", "docs", "community", "snippets", "extensions", and "search". A paragraph describes Flask as a microframework for Python based on Werkzeug, Jinja 2, and good intentions. It mentions the latest version is 0.10.1. A code block shows a simple Flask application. Below the code, there are instructions on how to install Flask using pip and run the application. At the bottom, there is a "Star" button with the number 13,344 and a list of links for downloading the latest release, reading the documentation, joining the mailing list, forking on GitHub, and adding issues and feature requests.

Welcome | Flask (A Python... x +

flask.pocoo.org

Flask

web development,  
one drop at a time

[overview](#) // [docs](#) // [community](#) // [snippets](#) // [extensions](#) // [search](#)

Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's [BSD licensed!](#)

Flask is Fun Latest Version: [0.10.1](#)

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

@app.route("/")
def hello():
    return "Hello World!"

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

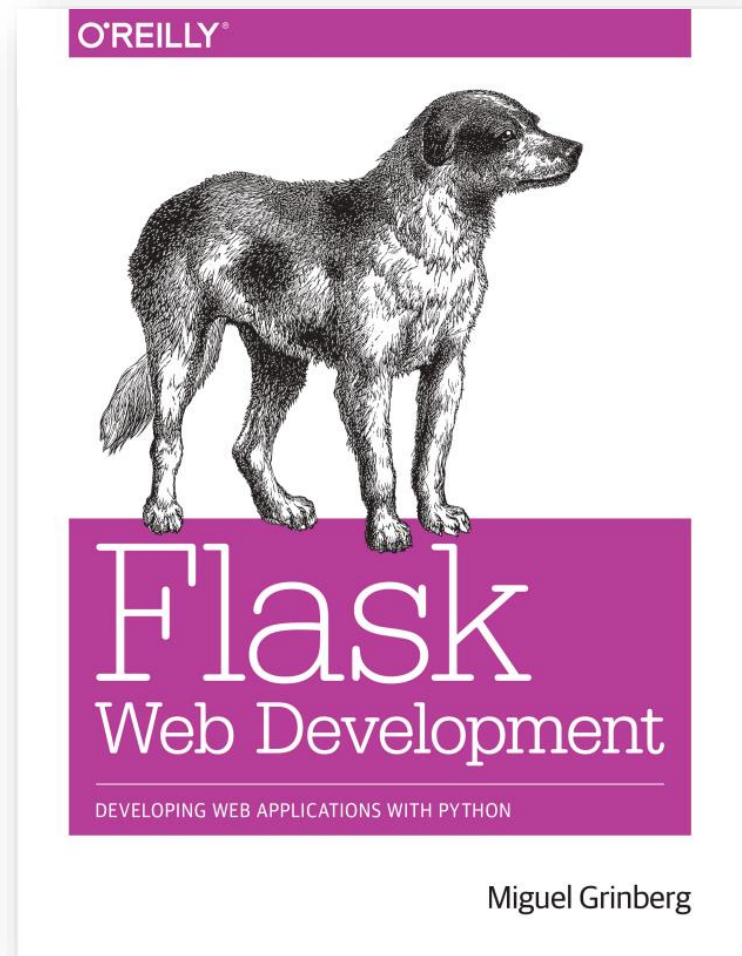
And Easy to Setup

```
$ pip install Flask
$ python hello.py
* Running on http://localhost:5000/
```

Interested? Star 13,344

- » [Download latest release \(0.10.1\)](#)
- » [Read the documentation](#) or download as [PDF](#) and [zipped HTML](#)
- » [Join the mailinglist](#)
- » [Fork it on github](#)
- » [Add issues and feature requests](#)

What's in the Box?



# Basic ingredients

- «Flask is a microframework for Python»
  - Web server
    - Based on Werkzeug (WSGI Utility Library) - <http://werkzeug.pocoo.org/>
  - Application context
  - Default configurations (conventions)
- Templating engine
  - Jinja2 - <http://jinja.pocoo.org/>
  - Easy editing of dynamic HTML pages
  - Powerful: operators and inheritance





# Flask installation

- Install Flask, Werkzeug and Jinja2 in a single step (system-wide installation)

```
$ sudo pip install Flask
```

- Or install them in a virtual environment (see <http://docs.python-guide.org/en/latest/dev/virtualenvs/>)

```
$ mkdir myproject  
$ cd myproject  
$ virtualenv venv
```

```
$ . venv/bin/activate
```

```
$ pip install Flask
```

# Flask applications

- One 'Flask' object represents the whole application

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
## __name__ is the application name
```

- Running the application starts the web server (running until you kill it)

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

# The web server

- By default, Flask runs a web server on:
  - <http://127.0.0.1:5000/>
  - Accessible by localhost, only
  - Running on port 5000
- Can be customized with parameters to the `.run` method:

```
# syntax: app.run(host=None, port=None,
debug=None, **options)
app.run(host='0.0.0.0', port=80) # public
app.run(debug=True) # for development
```

# Running a 'public' web server

- Bind to all IP addresses of your machine
  - `host='0.0.0.0'`
- Use a standard port
  - `port=80` (*must be launched as 'root'*)
  - `port=8080` (*>1024, does not require root*)
- Check the firewall, and open the host/port combination for external access
- **Beware hackers and intruders**

# Web pages

- Each<sup>(\*)</sup> page is implemented by a method:

```
@app.route('/')  
def index():  
    return "Hello, web world!"
```

- Must specify
  - The (local) **URL** at which the page will be visible: '/'
  - The **name** of the page: `index`
  - The (HTML) **content** of the page: `return` statement

<sup>(\*)</sup> not really true... see later

Building Web Applications

# FIRST FLASK APPLICATION



# Exercise 1

## **Ambient Intelligence 2015**

Welcome to the WakeKill project.



© SmartRooster

## **SmartRooster - About us**

This group is composed by the greatest sleepers in the class.

If it wakes us up, you may bet it'll work for you, too.

**Try our WakeKill project**

# Exercise 1

</index.html>

## Ambient Intelligence 2015

Welcome to the WakeKill  
project.



Image

© [SmartRooster](#)

Link

</about.html>

## SmartRooster - About us

This group is composed by  
the greatest sleepers in the  
class.

If it wakes us up, you may  
bet it'll work for you, too.

Try our [WakeKill project](#)

Link



# Solution 1

<https://github.com/Aml-2015/Flask-ex1>

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def index():
```

```
    return """<html><head><title>WakeKill</title></head>
<body><h1>Ambient Intelligence 2015</h1>
<p>Welcome to the WakeKill project.</p>
<p></p>
<p>&copy; <a href="about.html">SmartRooster</a></p>
</body></html>
"""
```

```
@app.route('/about.html')
```

```
def about():
```

```
    return """<html><head><title>WakeKill</title></head>
<body><h1>SmartRooster - About us</h1>
<p>This group is composed by the greatest sleepers in the class.</p>
<p>If it wakes us up, you may bet it'll work for you, too.</p>
<h1>Try our <a href="/">WakeKill</a> project</h2>
</body></html>
"""
```

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

# Generated URLs

- Don't encode destination URL in the HTML string
- Generated URL for function `xyz`

```
url_for('xyz')
```

- Generated URL for static file `abc.jpg` (located in a subfolder that must be called 'static')

```
url_for('static', filename='abc.jpg')
```

# Solution 2

<https://github.com/Aml-2015/Flask-ex1>

```
from flask import Flask
from flask import url_for
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def index():
```

```
    return ('<html><head><title>WakeKill</title></head>' +
           '<body><h1>Ambient Intelligence 2015</h1>' +
           '<p>Welcome to the WakeKill project.</p>' +
           '<p></p>' +
           '<p>&copy; <a href="' + url_for('about') + '">SmartRooster</a></p>' +
           '</body></html>' )
```

```
@app.route('/about.html')
```

```
def about():
```

```
    return ( '<html><head><title>WakeKill</title></head>' +
            '<body><h1>SmartRooster - About us</h1>' +
            '<p>This group is composed by the greatest sleepers in the class.</p>' +
            '<p>If it wakes us up, you may bet it&apos;ll work for you, too.</p>' +
            '<h1>Try our <a href="' + url_for('index') + '">WakeKill</a> project</h2>' +
            '</body></html>' )
```

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



The remaining part of this section (Dynamic and parametric routes is best understood after the HTML forma and Jinja templates)

# Dynamic route rules (1)

- A route rule may be dynamic (includes a `<parameter>`, that is passed as function argument)

```
@app.route('/user/<username>')  
def show_user_profile(username):  
    return 'User %s' % username
```

<http://localhost:5050/user/fulvio>

# Dynamic route rules (2)

- Automatic conversions are available by specifying the parameter type

```
@app.route( '/post/<int:post_id>' )  
def show_post(post_id):  
    return 'Post %d' % post_id # integer value
```

- Parameter type may be:
  - missing (defaults to string), int, float, path (string that may include slashes)

<http://localhost:5050/post/37>

# URLs with parameters

- `url_for` accepts parameters
- Encoded as variable URLs, **if** the route is **dynamic**

```
@app.route('/user/<username>')  
def profile(username):  
    ...
```

```
url_for('profile', username='John Doe') →  
/user/John%20Doe
```

# URLs with parameters

- `url_for` accepts parameters
- Encoded as GET parameters, **if** the route is **static** (or does not contain the named parameter)

```
@app.route('/login')  
def login():  
    ...
```

```
url_for('login') → /login  
url_for('login', next='/') → /login?next=/  
url_for('login', next='/next') → /login?next=/next
```



# HTTP Request methods

- By default, the route applies to the GET method, only
- You may support other methods, e.g., the POST method for submitting HTML forms, by specifying a list of allowed methods:

```
@app.route('/login', methods=['GET', 'POST'])
```

- The actually called method is available in the `request.method` variable

Building Web Applications

# JINJA2 TEMPLATES



# HTML templating

- Embedding HTML in Python strings is
  - Ugly
  - Error prone
  - Complex (i.e., must follow HTML escaping rules and Python quoting rules)
  - Did I say Ugly?
- **Templating** = separating the (fixed) structure of the HTML text (template) from the variable parts (interpolated variables)
- Flask supports the **Jinja2** templating engine

# Jinja2 basics

- Templates should be in the `./templates` subfolder
- Templates are HTML files, with `.html` extension
- Templates can interpolate passed-by values:
  - `{{ parameter }}`
  - `{{ expression }}`
- Templates can include programming statements:
  - `{% statement %}`
- Templates can access some implicit objects
  - `request`, `session`, `g`
- Templates are processed when requested by the Flask page

```
return render_template('hello.html', name=name)
```

# Solution 3 – main.py

<https://github.com/Aml-2015/Flask-ex1>

```
from flask import Flask
from flask import render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about.html')
def about():
    return render_template('about.html')

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

# Solution 3 – templates/index.html

```
<html>
<head>
<title>WakeKill</title>
</head>
<body>
<h1>Ambient Intelligence 2015</h1>
<p>Welcome to the WakeKill project.</p>
<p>

</p>
<p>
&copy; <a href="{{ url_for('about') }}">SmartRooster</a>
</p>
</body>
</html>
```

# Solution 3 – templates/about.html

```
<html>
<head>
<title>WakeKill</title>
</head>
<body>
<h1>SmartRooster - About us</h1>
<p>This group is composed by the greatest sleepers in the
class.</p>
<p>If it wakes us up, you may bet it&apos;ll work for you,
too.</p>
<h1>
Try our <a href="{{ url_for('index') }}">WakeKill</a> project
</h2>
</body>
</html>
```

# Main Jinja2 {% statements %}

- `{% for var in list %} ... {% endfor %}`
- `{% if condition %} ... {% elif cond %} ...  
{% else %} ... {% endif %}`



# Statements vs Expressions

- A `{% statement %}` controls the flow of execution in a template
  - <http://jinja.pocoo.org/docs/dev/templates/#list-of-control-structures>
- An `{{ expression }}` evaluates the variable (or the expression) and «prints» the results in the HTML file
  - <http://jinja.pocoo.org/docs/dev/templates/#expressions>

Building Web Applications

# USER INTERACTION



# Exercise 2

</index.html>

## Ambient Intelligence 2015

Welcome to the WakeKill project.



Enter name:  [Submit]

© SmartRooster

## Ambient Intelligence 2015

Your name: **name**

[Continue](#)

</login.html>

</index.html>

## Ambient Intelligence 2015

Welcome **name** to the WakeKill project.






[Check your alarms](#) | [Logout](#)

© SmartRooster

# HTML Forms

## Forms and Input

Tag	Description
<a href="#"><u>&lt;form&gt;</u></a>	Defines an HTML form for user input
<a href="#"><u>&lt;input&gt;</u></a>	Defines an input control
<a href="#"><u>&lt;textarea&gt;</u></a>	Defines a multiline input control (text area)
<a href="#"><u>&lt;button&gt;</u></a>	Defines a clickable button
<a href="#"><u>&lt;select&gt;</u></a>	Defines a drop-down list
<a href="#"><u>&lt;optgroup&gt;</u></a>	Defines a group of related options in a drop-down list
<a href="#"><u>&lt;option&gt;</u></a>	Defines an option in a drop-down list
<a href="#"><u>&lt;label&gt;</u></a>	Defines a label for an <input> element
<a href="#"><u>&lt;fieldset&gt;</u></a>	Groups related elements in a form
<a href="#"><u>&lt;legend&gt;</u></a>	Defines a caption for a <fieldset> element
<a href="#"><u>&lt;datalist&gt;</u></a>	 Specifies a list of pre-defined options for input controls
<a href="#"><u>&lt;keygen&gt;</u></a>	 Defines a key-pair generator field (for forms)
<a href="#"><u>&lt;output&gt;</u></a>	 Defines the result of a calculation

[http://www.w3schools.com/tags/ref\\_byfunc.asp](http://www.w3schools.com/tags/ref_byfunc.asp)

# Querying request parameters

- All FORM variable are sent with the HTTP request
- Flask packs all FORM variables in the 'request.form' object (a dictionary)
- 'request' is a global implicit object, and must be imported

```
from flask import request  
user = request.form['user']
```

# Using parameters in templates

- Specify name=value of all needed parameters in the `render_template` call
- Within the template, use the `{{ name }}` syntax
- Template parameters need not be the same as FORM parameters (they are independent concepts, independent values)

```
return render_template('welcome.html',  
user=myuser)
```

```
<p>Welcome {{ user }}.</p>
```

# Remembering values

- Values in request.form expire immediately
- We may «remember» values for a longer time
- By storing them in «session» containers
  - Based on HTTP cookies
  - Kept in memory in the web server
  - Valid until browser disconnection or timeout, only
  - <http://flask.pocoo.org/docs/0.10/quickstart/#sessions>
- By storing them in a connected database
  - Persistent storage
  - Kept on disk in the database server
  - Requires explicit DB connection

# Implementing sessions in Flask

- Sessions are automatically initialized and managed by Flask
- Session data is encrypted. Must define a secret key
  - `app.secret_key = 'whoknowsthissecret'`
- The 'session' object is a global shared dictionary that stores attribute-value pairs

```
session['user'] = user
```

```
<p>Welcome {{ session['user'] }} to the  
WakeKill project.</p>
```



# Automatic redirects

- In some cases, a user action doesn't need to generate a response page
  - E.g., the Logout action needs to destroy the session, but will just bring you to the normal 'index' page
- You may use a 'redirect' method to instruct the browser that the current response is empty, and it must load the new page (HTTP 302)

```
return redirect(url_for('index'))
```

# Solution

<https://github.com/Aml-2015/Flask-ex1>

```
from flask import Flask, render_template, request, session, url_for, redirect

app = Flask(__name__)
app.secret_key = 'whoknowsthissecretw'

@app.route('/')
def index():
    return render_template('index2.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/login', methods=['POST'])
def login():
    user = request.form['user']
    session['user'] = user
    return render_template('welcome.html', user=user)

@app.route('/logout')
def logout():
    del session['user']
    return redirect(url_for('index'))

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

# Solution – index2.html

```
<html>
<head>
<title>WakeKill</title>
</head>
<body>
<h1>Ambient Intelligence 2015</h1>
<p>Welcome {{ session['user'] }} to the WakeKill project.</p>
<p>

</p>
<p>
<form action="{{ url_for('Login') }}" method='POST'>
{% if session.user %}
Check your alarms | <a href="{{ url_for('Logout') }}">Logout</a>
{% else %}
Enter name: <input type='text' name='user'> <input type='submit' value='Submit'></form>
{% endif %}
<p>
&copy; <a href="{{ url_for('about') }}">SmartRooster</a>
</p>
</body>
</html>
```

# Solution – welcome.html

```
<html>
<head>
<title>WakeKill</title>
</head>
<body>
<h1>Welcome</h1>
<p>Welcome {{ user }}.</p>

<p><a href="{{ url_for('index') }}">Continue</a>
</p>
</body>
</html>
```

Building Web Applications

# FLASK EXTENSIONS



# Flask extensions

- Web applications share
  - A generally standardized architecture
  - Many common and repetitive actions
  - Many security risks associated with user input and database interactions
- Many extensions are available to automate most of the most boring or most risky tasks
- <http://flask.pocoo.org/extensions/>

# Some Useful Flask Extensions

- **Flask-WTF**: Integration with WTForms (form creation, validation, regeneration). **Mandatory!**
- **Flask-SQLAlchemy**: integration with SQLAlchemy, and object-relational mapping for database storage
- **Flask-Bootstrap**: quick and easy pretty layouts with Twitter's Bootstrap library. **Mandatory!**
- **Flask-Mail**: for sending e-mails through SMTP servers
- **Flask-Login**: Management of user sessions for logged-in users
- **Flask-RESTful**: Tools for building RESTful APIs
- **Flask-OAuth**: Authentication against OAuth providers

Building Web Applications

# FLASK BOOTSTRAP EXTENSION





# Flask-Bootstrap

- «Flask-Bootstrap packages [Bootstrap](#) into an extension that mostly consists of a blueprint named 'bootstrap'. It can also create links to serve Bootstrap from a CDN and works with no boilerplate code in your application.»
- Package available at
  - <https://pypi.python.org/pypi/Flask-Bootstrap>
  - Install with 'pip'
- Documentation available at
  - <http://pythonhosted.org//Flask-Bootstrap/>

# How to use

- Apply Bootstrap Extensions to your Flask application

```
from flask import Flask
from flask_bootstrap import Bootstrap

def create_app():
    app = Flask(__name__)
    Bootstrap(app)
```

- Derive your Jinja2 templates from the “base” bootstrap structure

```
{% extends "bootstrap/base.html" %}
```

# Blocks

- `{% block xxxx %} ... {% endblock %}`
- Includes the specified HTML/template code in a specific part of the Bootstrap template
- Predefined blocks
  - **title**: complete content of the `<title>` tag
  - **navbar**: empty block directly above *content*
  - **content**: Convenience block inside the body. Put stuff here

# Example template

```
{% extends "bootstrap/base.html" %}
{% block title %}This is an example page{% endblock %}

{% block navbar %}
<div class="navbar navbar-fixed-top">
  <!-- ... -->
</div>
{% endblock %}

{% block content %}
  <h1>Hello, Bootstrap</h1>
{% endblock %}
```

# Blocks

Block name	Outer Block	Purpose
doc		Outermost block.
html	doc	Contains the complete content of the <code>&lt;html&gt;</code> tag.
html_attribs	doc	Attributes for the HTML tag.
head	doc	Contains the complete content of the <code>&lt;head&gt;</code> tag.
body	doc	Contains the complete content of the <code>&lt;body&gt;</code> tag.
body_attribs	body	Attributes for the Body Tag.
title	head	Contains the complete content of the <code>&lt;title&gt;</code> tag.
styles	head	Contains all CSS style <code>&lt;link&gt;</code> tags inside head.
metas	head	Contains all <code>&lt;meta&gt;</code> tags inside head.
navbar	body	An empty block directly above <i>content</i> .
content	body	Convenience block inside the body. Put stuff here.
scripts	body	Contains all <code>&lt;script&gt;</code> tags at the end of the body.

# Bootstrap and others

- The Flask-Bootstrap extension works nicely with
  - Flask-WTF for form handling
  - Flask-SQLAlchemy for database access

# License



- These slides are distributed under a Creative Commons license “Attribution – NonCommercial – ShareAlike (CC BY-NC-SA) 3.0”
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - NonCommercial — You may not use the material for commercial purposes.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- <http://creativecommons.org/licenses/by-nc-sa/3.0/>

