**Flask/Python, Jinja/HTML, CSS**

(Learning from RealPython’s *Discover Flask* tutorial series - <https://github.com/realpython/discover-flask>, Flask Documentation Tutorials, Jinja Documentation, RealPython’s *Primer on Templating*)

- Environment: Ubuntu 16.04 LTS, Python 3.6.10, Flask 1.0.2, Jinja 2.10

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Flask** | | | | | | | | | | | | | |
| - Micro web framework powered by Python  - Small API, easy to learn and simple to use  - Can support enterprise-level applications handling large amounts of traffic  - Offers suggestions but doesn’t enforce any dependencies or project layout  - Up to developer to choose the tools, libraries to use | | | | | | | | |  | | | | |
| Note:  - Put *jquery* javascript first among all javascripts when referencing in the HTML file | | | | | | | | |  | | | | |
| Project structure | | | flask-intro  | app.py  | static  | bootstap.min.css  | bootstrap.min.js  | templates  | welcome.html | | | | | **File/Folder** | | |  | | | |
| app.py | | | - Utilise Model-View-Controller (MVC) design pattern in the back-end  - Controller decides how to handle requests, returns responses to end user  - Don’t name file flask.py, avoid conflicting with Flask itself | | | |
| static | | | Houses **stylesheets** (i.e. css), Javascript files, images | | | |
| templates | | | Holds **HTML** files | | | |
| pip3 freeze | | List installed packages and their versions  - Can write packages to a file which can be used later to set up a new environment | | | | | | |  | | | | |
| Bootstrap | | - Front-end open source toolkit | | | | | | | Add **bootstrap.min.css, bootstrap.min.js** to static folder | | | | |
| Debug mode | | - In debug mode, server will reload itself on code changes  - Comes with a helpful interactive debugger if things go wrong  - Allows the execution of arbitrary code  - Does not work in forking environments, near impossible to use on production servers  - Major security risk, should **never** be used on production machiens | | | | | | |  | | | | |
| Routing | | - Use route() decorator to bind a function to a URL  - Can also make parts of URL dynamic and attach multiple rules to a function | | | | | | | @app.route(‘/hello’)  def hello():  return ‘Hello, World’ | | | | |
| Variable Rules | | - Add variable sections to a URL by marking sections with <variable\_name>  - Function receives <variable\_name> as keyword argument  - (Optional) Use converter to specify the type of argument like <converter:variable\_name> | | | | | | | from **markupsafe** import **escape**  @app.route(‘/user**/<username>**’)  def show\_user\_profile(username):  # show the user profile for that user  return ‘User %s’ **% escape(username)**  @app.route(‘/post**/<int:post\_id>**’)  def show\_post(post\_id):  # show the post with the given id, id is integer  return ‘Post %d’ **% post\_id**  @app.route(‘/path**/<path:subpath>**’)  def show\_subpath(subpath):  # show the subpath after /path/  return ’Subpath %s’ **% escape(subpath)** | | | | |
| **Converter Type** | |  | | | | |
| string | | (Default) Accepts any test without a slash | | | | |
| int | | Accepts positive integers | | | | |
| float | | Accepts positive floating point values | | | | |
| path | | Similar to string but also accepts slashes | | | | |
| uuid | | Accepts UUID strings | | | | |
| Unique URLs/  Redirection Behaviour | | - Canonical URL has a trailing slash  - Similar to a folder in a file system  - If access URL without trailing slash, Flask redirects to canonical URL with trailing slash  - Canonical URL without trailing slash  - Similar to a path-name of a file  - Accessing this URL with an added trailing slash produces a 404 ‘Not Found’ error  - Helps keeps URLs unique for these resources  - Helps search engines avoid indexing the same page twice | | | | | | | # Access URL with trailing slash  @app.route(‘/projects/’)  def projects():  return ‘’The project page’  # Access URL without trailing slash  @app.route(‘/about’)  def about():  return ‘The about page’ | | | | |
| URL Building | | - Usee the url\_for() function  - Accepts name of function as its first argument, any number of keyword arguments  - Each keyword argument corresponds to a variable part of the URL rule  - Unknown variable parts are appended to the URL as query parameters  - Benefits of building URLS using URL reversing function url\_for()  - Avoid hardcoding URLS into templates  - Reversing is often more descriptive than hardcoding URLS  - Can change URLS in one go instead of manually changing hard-coded URLS  - Handles escaping of special characters and Unicode data transparently  - If application placed outside URL root (e.g. not /), function handles it | | | | | | | from flask import Flask, url\_for  from markupsafe import escape  # tell Flask to behave as though handling a request even while using Python shell  app = Flask(\_\_name\_\_)  …  @app.route(‘/user/<username>’)  def profile(username):  return ‘{}\’s profile’.format(escape(username)  with app.test\_request\_context():  print(url\_for(‘login’, next = ‘’/)) # returns ‘/login?next=/’  print(url\_for(‘profile’, username=’John Doe’)) # returns /user/John%20Doe | | | | |
| HTTP Methods | | - (Default) Route only answers to GET requests  - Can use methods argument of route() decorator to handle different HTTP methods  - If GET present, Flask automatically  - Adds support for the HEAD method  - Handles HEAD requests according to HTTP RFC  - Implements OPTIONS | | | | | | | from flask import **request**  @app.route(‘/login’, **methods=[‘GET’, ‘POST’])**  def login():  if request.method == ‘POST’:  return do\_the\_login()  else:  return show\_the\_login\_form() | | | | |
| Static Files | | - Files such as CSS and JavaScript  - Ideally web server configured to serve static files, but can be done by Flask during development as well  - Place files in folder called static in package or next to module  - Generate URLS for static files using special ‘static’ endpoint name | | | | | | | # To generate URLS for static files  # File stored on filesystem at static/style.css  url\_for(‘static’, filename=’style.css’) | | | | |
| Rendering Templates | | - Use render\_template() method  - Provide name of template and variables to pass to template engine as keyword arguments  - Place templates in folder called templates  - If application is a module, place next to module  /app.py  /templates  /hello.html  - If application is a package, place inside package  /application  /\_\_init\_\_.py  /templates  /hello.html  - fsf | | | | | | | from flask import render\_template  @app.route(‘/hello’)  @app.route(‘/hello/<name>’)  def hello(name=None):  return render\_template(‘hello.html’, name = name); | | | | |
| Jinja2 Template | | - Within template, access to request, session, g objects and get\_flashed\_messages() function  - Useful if inheritance is used  - Makes it possible to keep certain elements on each page  - E.g. header, navigation, footer  - Enables automatic escaping  - E.g. if name contains HTML, it will be escaped automatically  - Can mark a variable as safe HTML using the Markup class or using the |safe filter in the template | | | | | | | <!doctype html>  <title>Hello from Flask</title>  {% if name %}  <h1>Hello {{ name }}!</h1>  {% else %}  <h1>Hello, World!</h1>  {% endif %} | | | | |
| Accessing Request Data | | - Data client sends to server is provided by the global request object  - **method** attribute  - Current request method available  - **form** attribute  - Access format data  - Data transmitted in a POST or PUT request  - Use request.form[‘key’]  - If key does not exist in form attribute, special KeyError is raised  - Can catch like a standard error, else shows a HTTP 400 Bad Request error page | | | | | | | from flask import request  @app.route(‘/login’, methods=[‘POST’, ‘GET’])  def login():  error = None  if request.method == ‘POST’:  if valid\_login(**request.form**[‘username’], request.form[‘password’]):  return log\_the\_user\_in(request.form[‘username’])  else:  error = ‘Invalid username/password’  # execute if GET request method or credentials invalid  return render\_template(‘login.html’, error = error) | | | | |
| Access parameters submitted in URL  - Recommended to access them with get or by catching KeyError  - More user friendly to avoid giving users a 400 bad request page when they change the URL | | | | | | | searchword = request.args.get(‘key’, ‘’) | | | | |
| Upload File | | - Uploaded files are stored in memory or temporary location on filesystem  - **files** attribute (of request object)  - Access uploaded files  - Behaves like a standard Python file object  - Has a save() method to allow storage of file on filesystem of server  - Remember to set enctype=”multipart/form-data” attribute on HTML form for browser to transmit files | | | | | | | from flask import request  @app.route(‘/upload’, methods=[‘GET’, ‘POST’])  def upload\_file():  if request.method == ‘POST’:  f = **request.files**[‘the\_file’]  f**.save**(‘path/to/file.txt’)  …  # find what file was named on client before upload  … secure\_filename(f.**filename**) | | | | |
| Cookies | | - **cookies** attribute (of request object)  - A dictionary with all the cookies the client transmits  - Set cookies using set\_cookie method of response objects  - Do not use cookies directly but use Sessions in Flask for added security on cookies  - Cookies are set on response objects  - Use *Deferred Request Callbacks* pattern when setting a cookie where response objects does not exit yet | | | | | | | # read cookies  from flask import **request**  @app.route(‘/’)  def index(): username = request**.cookies.get**(‘username’)  # don’t use indexing to avoid KeyError  # store cookies  from flask import make\_response  @app.route(‘/’)  def index():  resp = **make\_response**(render\_template(…))  resp**.set\_cookie**(‘username’, ‘the username’)  return resp | | | | |
| Redirects and Errors | | - Use **redirect()** function  - Redirect user to another endpoint  - Use **abort()** function  - Abort a request early with an error code | | | | | | | from flask import **abort**, **redirect**, url\_for  @app.route(‘/’)  def index():  return **redirect**(url\_for(‘login’))  @app.route(‘/login’)  def login():  **abort**(401)  this\_is\_never\_executed() | | | | |
| - By default, black and white error page is shown for each error code  - Customisation of error page can be done using errorhandler() decorator  - After render\_template() call is error code  - For example, tell Flask the status code of the page should be 404 (i.e. not found)  - By default, assume error code 200 (i.e. all went well) | | | | | | | from flask import render\_template  #app.**errorhandler**(404)  def page\_not\_found(error):  return render\_template(‘page\_not\_found.html’), **404** | | | | |
| Responses | | - Return values from view function is converted into response objects  - If return value is  - Has a response object of correct type, it’s directly returned from the view  - A string, it is converted into a response object with string as response body, a 200 OK status code and a text/html mimetype (i.e. created with data and default parameters)  - A dictionary, jsonify() is called to produce a response  - A tuple, items in tuple can provide extra information  - If none works, Flask will assume the return value is a valid WSGI application and convert that into a response object  - Use make\_response() function to get resulting object inside view | | | | | | | @app.errorhandler(404)  def not\_found(error):  resp = make\_response(render\_template('error.html'), 404)  resp.headers['X-Something'] = 'A value'  return resp | | | | |
| APIs with JSON | | - Returning a dictionary from a view will be converted to a JSON response | | | | | | | @app.route("/me")  def me\_api():  user = get\_current\_user()  return {  "username": user.username,  "theme": user.theme,  "image": url\_for("user\_image", filename=user.image),  } | | | | |
| - Creating JSON responses for types other than dictionaries  - Use jsonify() function  - Serialises any supported JSON data type | | | | | | | @app.route("/users")  def users\_api():  users = get\_all\_users()  return **jsonify**([user**.to\_json()** for user in users]) | | | | |
| Sessions | | - Object that helps to store information specific to a user from one request to the next  - Implemented on top of cookies  - Signs cookies cryptographically  - User can look at contents of cookie but not modify, unless they have the secret key used for signing  - Requires a secret key to use | | | | | | | from flask import Flask, session, redirect, url\_for, request  from markupsafe import escape  app = Flask(\_\_name\_\_)  # set secret key to some random bytes (SECRET)  **app.secret\_key = b'\_5#y2L"F4Q8z\n\xec]/'**  @app.route(‘/’)  def index():  if ‘username’ in **session**:  return ‘Logged in as %s’ % escape(**session**[‘username’])  return ‘You are not logged in’  @app.route('/login', methods=['GET', 'POST'])  def login():  if request.method == 'POST':  **session**['username'] = request.form['username']  return redirect(url\_for('index'))  return '''  <form method="post">  <p><input type=text name=username>  <p><input type=submit value=Login>  </form>  '''  @app.route('/logout')  def logout():  # remove the username from the session if it's there  **session.pop**('username', None)  return redirect(url\_for('index')) | | | | |
| - Generate a secret key | | | | | | | # run on terminal  python -c 'import os; print(os.urandom(16))' | | | | |
| Message Flashing | | - Use **flash()** method  - A simple way to give feedback to a user  - Makes it possible to record a message at the end of a request and access it on the next (only the next) request  - Usually combined with a layout template to expose the message | | | | | | | @app.route('/login', methods=['GET', 'POST'])  def login():  error = None  if request.method == 'POST':  if request.form['username'] != 'admin' or \  request.form['password'] != 'secret':  error = 'Invalid credentials'  else:  **flash**('You were successfully logged in')  return redirect(url\_for('index'))  return render\_template('login.html', error=error) | | | | |
| - Use **get\_flashed\_messages()** to get hold of messages to use | | | | | | | <!doctype html>  <title>My Application</title>  {% with messages = get\_flashed\_messages() %}  {% if messages %}  <ul class=flashes>  {% for message in messages %}  <li>{{ message }}</li>  {% endfor %}  </ul>  {% endif %}  {% endwith %}  {% block body %}{% endblock %} | | | | |
| Logging | | - Log when something fishy happened  - For example, when user tamper with data or client code failing but don’t want to reply with a 400 Bad Request | | | | | | | # examples  app.**logger.debug**(‘A value for debugging’)  app.logger.**warning**(‘A warning occurred (%d apples)’, 42)  app.logger.**error**(‘An error occurred’) | | | | |
|  | |  | | | | | | |  | | | | |
|  | |  | | | | | | |  | | | | |
| **Routes** | | | | | | | | | | | | | |
| # file app.py  # import the Flask class from the flask module  **from** flask **import** Flask, render\_template  # create application object  **app = Flask(\_\_name\_\_)**  # use decorators to link the function to a url  **@app.route(‘/’)**  def home():  return “Hello, World!” # return string  **@app.route(“/welcome”)**  def welcome():  return **render\_template**(‘welcome.html’)  # start the server with the ‘run()’ method  if \_\_name\_\_ == ‘\_\_main\_\_’:  **app.run**(debug=True) | | | | | | **Code** | | | | | | **Explanation** | |
| **from** flask **import** Flask | | | | | | Import flask class | |
| **app = Flask(\_\_name\_\_)** | | | | | | Instantiate application object  - Use \_\_name\_\_ if using a single module  - The name will be different (‘\_\_main\_\_’ versus the actual import name) depends on if it’s started as an application or imported as a module  - Needed to direct Flask where to look for templates, static files etc | |
| def home():  return “Hello, World!” | | | | | | Define views to respond to requests | |
| if \_\_name\_\_ == ‘\_\_main\_\_’:  **app.run**(debug=True) | | | | | | When server is running, Flask is in **debug mode**  - Auto-reload mechanism  - Just need to refresh browser to see code changes  - **Must be at the end of file** | |
| **@app.route**(‘/welcome’) | | | | | | Route decorator  - Maps an URL to a function defined below it  - When end user requests URL, view will call the function | |
| **render\_template**(‘welcome.html’) | | | | | | Render the HTML template | |
| **Output (Run on Terminal)** | | | | | | | |
| python3 app.py | | | | | | - Server is only accessible on local computer | |
| export FLASK\_ENV=development  flask run | | | | | | - Enable all development features (including debug mode)  - Activates debugger, automatic reloader, debug mode on Flask application | |
| export FLASK\_APP=app.py  flask run | | | | | | - Sets environment variable to name of module to import when running flask  - Not in debug mode | |
| flaks run --host=0.0.0.0 | | | | | | Make server publicly available  - Tells OS to listen on all public IPs | |
| - On <http://localhost:5000/>, page has just the words “Hello, World!”  - This file can be called the “controller” | | | | | | | |
|  | **HTML Template for welcome.html** | | | | | | | | | | | | |
| - Add bootstrap.min.css, bootstrap.min.js to static folder | | | | | | | | | | | | |
| <!DOCTYPE html>  **<html>**  <head>  <!--Tab title-->  <title>Flask Intro</title>  <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>  **<link href=”static/bootstrap.min.css” rel=”stylesheet” media=”screen”>**  </head>  <body>  <div class=”container”>  <h1>Welcome to Flask!</h1>  <br>  <p>Click <a href=”/”>here</a> to go home.</p>  </body>  **</html>** | | | | | **HTML Element** | | | **Contains** | | | | |  |
| <html></html> | | |  | | | | | Highest element |
| <head></head> | | | <title></title> | | | | | Tab title |
| <meta name=”…”, content=”…”> | | | | |  |
| <link href=”…css” rel= “stylesheet” media=”…”> | | | | | Link to css stylesheet |
| <body></body> | | | <h1></h1> | | | | |  |
| <p></p> | | | | |  |
| <div class=”…”> | | | | |  |
| **<a href=**”/”**>** here **</a>** | | | | | Links  - Example: goes to root page |
| <br> | | | | | Line break |
|  | | | | | | | | |
| **Login Page** | | | | | | | | | | | | | |
| from flask import Flask, render\_template, **redirect, url\_for, request**  app = Flask(\_\_name\_\_)  …  @app.route(‘/login’, **methods=[‘GET’, ‘POST’])**  def login():  error = **None**  if **request.method** == ‘POST’:  if **request.form**[‘username’] != ‘admin’ or request.form[‘password’] != ‘admin’:  error = ‘Invalid Credentials. Please try again.’  else:  return **redirect**(**url\_for**(‘home’))  return render\_template(‘login.html’, **error=error**)  if \_\_name\_\_ = ‘main’:  app.run(debug=True) | | | | | | **Code** | | | | | **Explanation** | | |
| redirect, url\_for, request | | | | | Relevant imports | | |
| **@app.route**(‘/login’, methods=[‘GET’, ‘POST’]) | | | | | Arguments in route decorator  1. **GET**  - Default method even if not specified, always available  2. **POST**  - Allows end users to send a POST request with their login credentials to that /login endpoint | | |
| **request.method** | | | | |  | | |
| **request.form**[‘username’]  request.form[‘password’] | | | | |  | | |
| **redirect**(url\_for(‘home’)) | | | | | Redirect user to another endpoint | | |
| redirect(**url\_for**(‘home’)) | | | | | Function generates an endpoint for the provided method | | |
| **render\_template**(‘login.html’, error=error) | | | | | - Error string will be passed to the HTML file to use | | |
| **Output** | | | | | | | |
| - Credentials for username and password  - If correct, redirect user to main route (i.e. / aka home)  - If incorrect, populate error  - Type of request  - If GET request: login page rendered  - If POST request: gets credentials  - Explanation:  - When form is submitted, POST request is sent along with form data  value=”{{ request.form.username }}” and value=”{{ request.form.password }}”  to controller app.py  - Controller handles requests and either responds with an error message or redirects user to root URL | | | | | | | |
|  | | | | | | | |
|  | **HTML Template for login.html** | | | | | | | | | | | | |
| - Add bootstrap.min.css, bootstrap.min.js to static folder | | | | | | | | | | | | |
| <html>  <head>  <title>Flask Intro - login page</title>  <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>  <link href=”static/bootstrap.min.css” rel=””stylesheet” media=”screen”>  </head>  <body>  <div class=”container”>  <h1>Please login</h1>  <br>  **<form action=”” method=”post”>**  <input type=”text” placeholder=”Username” name=”username” value=”{{**request.form.username**}}”>  <input type=”password” placeholder=”Password” name=”password” value=”{{**request.form.password**}}”>  **</form>**    **{% if error %}**  **<p class=”error”>Error:<strong></strong> {{ error }}**  **{% endif %}**  </div>  </body>  </html> | | | | | | A screenshot of a cell phone  Description automatically generated  - If error is not None, then display the actual error message (which is passed from the template from the views) | | | | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |
|  | | |  | | | | | |  | | | | |

|  |  |  |
| --- | --- | --- |
| **Jinja** | | |
| Macros | - Abstract out commonly used code  snippets  - How:  1. Add macros.html file to templates directory | <!--macros.html-->  {% **macro** nav\_link(endpoint, name) %}  {% **endmacro**%} |
| Set a variable |  | {% set x = obj.attr %} |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |