7th March 2020

Python and Flask Bootcamp: Create a website using Flask. Udemy Jose Portilla

|  |  |
| --- | --- |
| [Flask logo.svg](https://en.wikipedia.org/wiki/File:Flask_logo.svg) | |
| [**Developer(s)**](https://en.wikipedia.org/wiki/Software_developer) | [Armin Ronacher](https://en.wikipedia.org/wiki/Armin_Ronacher) |
| **Initial release** | April 1, 2010; 9 years ago |
| [**Stable release**](https://en.wikipedia.org/wiki/Software_release_life_cycle) | 1.1.1 / July 8, 2019; 7 months ago[[1]](https://en.wikipedia.org/wiki/Flask_(web_framework)#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Flask_(web_framework)#cite_note-2) |
| [**Repository**](https://en.wikipedia.org/wiki/Repository_(version_control)) | [github.com/pallets/flask](https://github.com/pallets/flask) |
| **Written in** | [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) |
| [**Type**](https://en.wikipedia.org/wiki/Software_categories#Categorization_approaches) | [Web framework](https://en.wikipedia.org/wiki/Web_framework) |
| [**License**](https://en.wikipedia.org/wiki/Software_license) | [BSD](https://en.wikipedia.org/wiki/BSD_licenses) |
| **Website** | [palletsprojects.com/p/flask/](https://palletsprojects.com/p/flask/) |

‘myflaskenviro’

Front end design: html, css, bootstrap 4

Templates for displaying webpages in Flask, Forms for user input, databases for storing user input.

Get user input, store it, report it back.

Blueprints for larger applications.

OAuth (outside authorization) like login thru google, facebook, twitter

‘conda’ is the package manager for anaconda. Atom.io

Create virtual environment: conda create –n <enviornName> python=3.7

Activate the virtual enviro: conda activate <enviro name> // deactivate

Download the libraries from file(to the virtual environment): pip install –r requirements.txt

Bootstrap will provide automatic styling and components through CSS and JS

Flask is the web framework that allows to connect python code to web

Flask can do everything that Django can with the use of tonnes of 3rd party libs

Workflow: user > front-end>flask>database.CRUD create, read, update, delete

Forms use WTfoms library. Jinja2 grabs information from python and flask and sends as html

Environment name: myflaskenviro

Html: strong for bold and ‘em’ for emphasis

https://developer.mozilla.org/en-US/ for html tag learning.

Self closing tags in html. <img src=”” alt=””> // label ‘for=’ tag is for ‘id’ of a ‘form input’. // placeholder // dropdown - <select name=”fruits”> <option value=> banana </option> option .. </select> // <textarea rows= cols = >

Stylesheet: define property-value pair for elements. In Html: link to stylesheet using: <link rel=”stylesheet” href=”/file.css”>

CSS file:

h1 {

color: blue;

}

p {

color: red

}

label {

color: rgba(255, 0, 0, 0.5);

}

title {

color: #eab06c

}

HTML file:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="style.css" >

<title></title>

</head>

<body>

Hi this is my fist comment!

<h1> Hello, my comment </h1>

<form>

<label for = 'email\_id'> EMAIL </label>

<input type="text" width="20" >

<input type='email' id='email\_id'>

</form>

</body>

</html>

<div>Border : color, width, style

For linking javascript, <script src=”<file path> “

Every html element can accept a ‘class’ or ‘id’ attribute. It can accessed by ‘.’ And ‘#’ operator.

Font-family: cursive, monospace, font-size, font-weight, font-style, text-align,

Bootstrap has a very large list of easy to use components. [www.getbootstrap.com](http://www.getbootstrap.com)

Bootstrap has components and classes. ‘select multiple’ allows to select multiple options from dropdown list.

Bootstrap:

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<!-- <link rel="stylesheet" href="style.css" > --> #comment

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<title></title>

</head>

<body>

Hi this is my fist comment!

<div class="myClass">

<h1> Hello, my comrades </h1>

<h2 id="myId"> Dear bro's and siters </h2>

</div>

<form>

<label for = 'email\_id'> EMAIL </label>

<input type="text" width="20" >

<input type='email' id='email\_id'>

</form>

<p> Hello dear <span> friends! </span>

<div class="">

<h2> Hi all, come here for a trip! </h1>

<h1 id=""> Hi my brothers and sisters </h1>

</div>

<ul class="nav flex-column">

<li class="nav-item">

<a class="nav-link active" href="#">Active</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Link</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Link</a>

</li>

<li class="nav-item">

<a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>

</li>

</ul>

</body>

</html>

System design technologies: waterfall and agile development

Software Engineering | Classical Waterfall Model

Classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.  
Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:  


Let us now learn about each of these phases in brief details:

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.  
   The feasibility study involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification**: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

* + **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).
  + **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

1. **Design**: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
2. **Coding and Unit testing**: In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.
3. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists three different kinds of testing activities as described below :

* + **Alpha testing:** Alpha testing is the system testing performed by the development team.
  + **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
  + **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

1. **Maintainence:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance :
   * **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   * **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer’s request.
   * **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

**Advantages of Classical Waterfall Model**

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

* This model is very simple and is easy to understand.
* Phases in this model are processed one at a time.
* Each stage in the model is clearly defined.
* This model has very clear and well undestood milestones.
* Process, actions and results are very well documented.
* Reinforces good habits: define-before- design,  
  design-before-code.
* This model works well for smaller projects and projects where requirements are well  
  understood.

**Drawbacks of Classical Waterfall Model**

Classical waterfall model suffers from various shortcomings, basically we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

* **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.
* **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
* **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase the efficiency and reduce the cost, phases may overlap.

Agile: <https://www.guru99.com/agile-scrum-extreme-testing.html>

Python data types:

Strings are ordered sequence of characters.

Lists are ordered sequence of objects

Tuples are ordered sequence of immutable objects

Sets are unordrered collection of unique objects

Dictionaries are collection of key:value pairs

Integers, floats, Booleans

--

What is Full Stack?

Full Stack Web Developer

A full stack web developer is a person who can develop both client and server software.

In addition to mastering HTML and CSS, he/she also knows how to:

• Program a browser (like using JavaScript, jQuery, Angular, or Vue)

• Program a server (like using PHP, ASP, Python, or Node)

• Program a database (like using SQL, SQLite, or MongoDB)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Client Software

(Front End)

• HTML

• CSS

• Bootstrap

• W3.CSS

• JavaScript

• ES5

• HTML DOM

• JSON

• XML

• jQuery

• Angular

• React

• Backbone.js

• Express.js

• Ember.js

• Redux

• Storybook

• GraphQL

• Meteor.js

• Grunt

• Gulp

Server Software

(Back End)

• PHP

• ASP

• C++

• C#

• Java

• Python

• Node.js

• Ruby

• REST

• GO

• SQL

• MongoDB

• Firebase.com

• Sass

• Less

• Parse.com

• PaaS (Azure and Heroku)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Popular Stacks

• LAMP stack: JavaScript - Linux - Apache - MySQL - PHP

• LEMP stack: JavaScript - Linux - Nginx - MySQL - PHP

• MEAN stack: JavaScript - MongoDB - Express - AngularJS - Node.js

• Django stack: JavaScript - Python - Django - MySQL

• Ruby on Rails: JavaScript - Ruby - SQLite - Rails

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Advantages

The advantage of being a full stack web developer is:

• You can master all the techniques involved in a development project

• You can make a prototype very rapidly

• You can provide help to all the team members

• You can reduce the cost of the project

• You can reduce the time used for team communication

• You can switch between front and back end development based on requirements

• You can better understand all aspects of new and upcoming technologies

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Disadvantages

• The solution chosen can be wrong for the project

• The solution chosen can be dependent on developer skills

• The solution can generate a key person risk

• Being a full stack developer is increasingly complex

--

Armin Ronacher, "Flask for Fun and Profit", PyBay2016 // https://www.youtube.com/watch?v=1ByQhAM5c1I

--

Flask is having very fast iteration. Latency in high performance async IO.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return '<h1> Hello Puppies </h1>'

@app.route("/puppy/<name>")

def puppy(name):

return f'<h1> Hellow {name.upper()} puppy! '

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

--

Debugger pin is required to access the debugger console.

Make debug = to False only in the production environment.

‘Templates’ is a directory where Flask will automatically look for ‘html’ pages.

Flask ‘render\_template’ function will return an html page in ‘templates’ folder from main page view function. Jnja template engine.

Unsplash.com is a site for photos

Python logic can be implemented into variable, which can be injected into html using jinja to construct {{}} using Flask render\_template function.

Control flow statements use {%%} for loop and if conditions are used.

url\_for(): to connect to pages in templates. <a href= {{url\_for('puppy',name='Roku')}} > Puppies home </a>

--

@app.errorhandler(404)

def page\_not\_found(e):

return render\_template('error.html'), 404

--

Request.args.get(‘<name>’)

Flask\_wtf library; wtfroms

from flask import Flask, render\_template

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'mysecret\*key'

class MyForm(FlaskForm):

breed = StringField("What's the breed?")

submit = SubmitField('submit')

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

def index():

breed = False

form = MyForm()

if form.validate\_on\_submit():

breed = form.breed.data

form.breed.data=''

return render\_template('index.html', breed=breed, form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--

**Index.html**

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

{% if breed %}

<p> You selected the breeed {{breed}}

{% else %}

Select your breed </p>

{% endif %}

<form method='POST'>

{{form.hidden\_tag()}}

{{form.breed.label}} {{form.breed()}}

{{form.submit()}}

</form>

</body>

</html>

--

Flask’s ‘session’ object to grab the information provided in the form and pass it to another template.

BooleanField is for checkbox. SelectField is for dropdown menu. Choices are a list of tuple items as value and label

--

from flask import Flask, render\_template, session, redirect, url\_for

from flask\_wtf import FlaskForm

from wtforms import StringField, SubmitField, BooleanField, RadioField, TextAreaField, SelectField

from wtforms.validators import DataRequired

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'mysecretfuck\*key'

class MyForm(FlaskForm):

breed = StringField('Whats the breed?', validators=[DataRequired()])

neutered = BooleanField('Neutered?')

mood = RadioField('Whats your mood?', choices=[('Happy', 'Happy'), ('Excited', 'Excited')])

food = SelectField('Which food?', choices=[('chi', 'Chicken'), ('beef', 'Beef'), ('fish', 'Fish')])

feedback = TextAreaField('Any suggestion?')

submit = SubmitField('submit')

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

def index():

form = MyForm()

if form.validate\_on\_submit():

session['breed'] = form.breed.data

session['neutered'] = form.neutered.data

session['mood'] = form.mood.data

session['food'] = form.food.data

session['feedback'] = form.feedback.data

return redirect(url\_for('thankyou'))

return render\_template('nxt\_index.html', form=form)

@app.route('/thankyou')

def thankyou():

return render\_template('thankyou.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--------

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

<h1> You have to fill :)' </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.breed.label}} {{form.breed()}}<br>

{{form.neutered.label}} {{form.neutered()}}<br>

{{form.mood.label}} {{form.mood()}}<br>

{{form.food.label}} {{form.food()}}<br>

{{form.feedback.label}} {{form.feedback()}}<br>

{{form.submit()}}

</form>

</body>

</html>

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

</head>

<body>

<p> You have chosen: </p>

<p> Breed {{session['breed']}} <br>

Neutered {{session['neutered']}} <br>

Mood {{session['mood']}} <br>

Food {{session['food']}} <br>

Suggestion {{session['feedback']}} <br>

</p>

</body>

</html>

Flash() // get\_flashed\_messages()

--

from flask import Flask, render\_template, session, flash, redirect, url\_for

from flask\_wtf import FlaskForm

from wtforms import SubmitField

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY']= 'appkey'

class Alert\_Flash(FlaskForm):

sub = SubmitField('Submit')

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

def index():

form = Alert\_Flash()

if form.validate\_on\_submit():

flash('You just clicked the button')

return redirect(url\_for('index'))

return render\_template('alerts.html', form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

alerts.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

</head>

<body>

{% for mess in get\_flashed\_messages() %}

<div class="alert alert-warning alert-dismissible fade show" role="alert">

<strong>Holy guacamole!</strong> You should check in on some of those fields below.

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">&times;</span>

</button>

{{mess}}

</div>

{% endfor %}

<form method="POST">

{{form.hidden\_tag()}}

{{form.sub()}}

</form>

</body>

</html>

--

ORM- Object relational mapper converts python code into sql commands. The most common Python ORM is SQLAlchemy.

Flask-SQLAlchemy // pip install Flask-SQLAlchemy

1. Setup SQLite in Flask. B) Create a model in Flask. C) Perform basic CRUD in the model

Inherit from db.Model // from flask\_sqlalchemcy import SQLAlchemy

App.config[‘SQLALCHEMY\_DATABASE\_URI’] = ‘sqlite:///’+os.path.join(basedir,’data.sqlite’)

App.config[‘SQLALCHEMY\_TRACK\_MODIFICATIONS’] = False

Db = SQLAlchemy(app) // db.create\_all() –creates database table from the model class // db.session.add\_all([<instance objects>]) // db.session.commit()

Basic.py

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

#create app

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///'+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

#create database

db = SQLAlchemy(app)

#migrate the app with the database

Migrate(app,db)

#create model

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.Text)

age = db.Column(db.Integer)

breed = db.Column(db.Text)

def \_\_init\_\_(self,name,age,breed):

self.name = name

self.age = age

self.breed = breed

def \_\_repr\_\_(self):

return f"Puppy name is {self.name} and is {self.age} year old"setupdatabase.py

from basic import db, Puppy

#converts model class in to database table

db.create\_all()

sam = Puppy()

roku = Puppy()

sam.name = 'Sami'

sam.age = 3

roku.name = 'Roku'

roku.age = 2

#create

janko = Puppy(name='Edan',age=5)

db.session.add\_all([sam,roku])

db.session.commit()

print(sam.id)

print(roku.id)

crud.py

from basic import db, Puppy

#create

dinka = Puppy(name='Dinkan', age=10)

lut = Puppy('Luttappi',7)

db.session.add\_all([dinka,lut])

db.session.commit()

all\_puppies = Puppy.query.all()

print(all\_puppies)

puppy\_one = Puppy.query.get(3)

print(puppy\_one.name, puppy\_one.age)

#filter

puppy\_unni = Puppy.query.filter\_by(name='Unni')

print(puppy\_unni.all())

#update table

puppy\_add = Puppy.query.get(4)

puppy\_add.age = 8

db.session.add(puppy\_add)

print(Puppy.query.get(3).age)

pp = Puppy.query.get(3)

#delete item from table

db.session.delete(pp)

db.session.commit()

--

Flask-Migrate – to update changes to the database model

From flask-migrate import Migrate

Set FLASK\_APP = myapp.py

When the environment variable FLASK\_APP is set, then it can be run using ‘flask run’ and no need to enter below code in the executable python file.

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run()

--

If the FLASK\_APP is not set, then the executable python file is to be run as python <file.py> and should contain below code in the executable file.

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run()

--

flast db init – sets the migration directory

flask db migrate –m ‘some message’ – sets the migration file

flask db upgrade – sets the database migration

(its like git updates)

Flask relationships

A primary key is a column, whose value is unique for each row.

A foreign key in a table is a primary key in another table.

One to many and one to one relationship (uselist=false)

Models.py

import os

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

app = Flask(\_\_name\_\_)

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///'+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

Migrate(app,db)

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.Text)

#set 1 to many relationship (puppy can have many toys)

toys = db.relationship('Toy', backref='puppy', lazy='dynamic')

#set 1 to 1 relationship(puppy can have only one Owner)

owner = db.relationship('Owner', backref='puppy', uselist=False)

def \_\_init\_\_(self, name):

self.name = name

def \_\_repr\_\_(self):

if self.owner:

return f"Puppy name is: {self.name}, owner name is: {self.owner.name}"

return f"Puppy {self.name} has no owner yet!"

def report\_toys(self):

print(f"Here is {self.name}'s toys")

a = 0

for toy in self.toys:

a += 1

print(a,':', toy.item\_name)

return 'finished'

class Toy(db.Model):

\_\_tablename\_\_ = 'toys'

id = db.Column(db.Integer, primary\_key=True)

item\_name = db.Column(db.Text)

puppy\_id = db.Column(db.Integer, db.ForeignKey('puppies.id'))

def \_\_init\_\_(self, item\_name, puppy\_id):

self.item\_name = item\_name

self.puppy\_id = puppy\_id

class Owner(db.Model):

\_\_tablename\_\_ = "owner"

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.Text)

puppy\_id = db.Column(db.Integer, db.ForeignKey('puppies.id'))

def \_\_init\_\_(self, name, puppy\_id):

self.name = name

self.puppy\_id = puppy\_idbasic.py

from models import Puppy, Owner, Toy, db

db.create\_all()

roko = Puppy('Roko')

jno = Puppy('Joono')

db.session.add\_all([roko,jno])

db.session.commit()

ty1 = Toy('Doremon',roko.id) #puppy id of roko

ty2 = Toy('Shinjan',jno.id)

ty3 = Toy('Madumitha',roko.id) # puppy id of roko (1 to many relationship)

own1 = Owner('Jose',roko.id)

own2 = Owner('Maria',jno.id)

db.session.add\_all([ty1,ty2,ty3,own1,own2])

db.session.commit()

all = Puppy.query.all();

print(all)

print(roko.report\_toys())

output: [Puppy name is: Roko, owner name is: Jose, Puppy name is: Joono, owner name is: Maria]

Here is Roko's toys

1 : Doremon

2 : Madumitha

Finished

--

Adoption-site.py

# import flask modules

# create key for the forms

# configure the database

# create the app

# create the model

# create views and add forms

#=========================

import os

from flask import Flask, render\_template, redirect, url\_for

from flask\_migrate import Migrate

from flask\_sqlalchemy import SQLAlchemy

from forms import AddForm, DelForm

#=========================

app = Flask(\_\_name\_\_)

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SECRET\_KEY']='holahoola'

app.config['SQLALCHEMY\_DATABASE\_URI'] = "sqlite:///"+os.path.join(basedir,'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = SQLAlchemy(app)

Migrate(app,db)

#=========================

class Puppy(db.Model):

\_\_tablename\_\_ = 'puppies'

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.Text)

def \_\_init\_\_(self, name):

self.name = name

def \_\_repr\_\_(self):

return f"Puppy name is: {self.name}"

#=======================

@app.route('/')

def index():

return render\_template('home.html')

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

def add\_pup():

form = AddForm()

if form.validate\_on\_submit():

name = form.name.data

new\_pup = Puppy(name)

db.session.add(new\_pup)

db.session.commit()

return redirect(url\_for('list\_pup'))

return render\_template('add.html', form=form)

@app.route('/list')

def list\_pup():

puppies = Puppy.query.all()

return render\_template('list.html',puppies=puppies)

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

def delete\_pup():

form = DelForm()

if form.validate\_on\_submit():

id = form.id.data

pup = Puppy.query.get(id)

db.session.delete(pup)

db.session.commit()

return redirect(url\_for('list\_pup'))

return render\_template('delete.html', form = form)

#======================

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

forms.py

from flask\_wtf import FlaskForm

from wtforms import StringField, IntegerField, SubmitField

class AddForm(FlaskForm):

name = StringField("Name of Puppy")

submit = SubmitField("Add Puppy")

class DelForm(FlaskForm):

id = IntegerField("Puppy id to remove")

submit = SubmitField("Remove puppy")

---

Base.html

<!DOCTYPE html>

<html lang="en" dir="ltr">

<head>

<meta charset="utf-8">

<title></title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

</head>

<body>

<nav class="navbar navbar-light bg-light">

<div class="container-sm">

<a class="navbar-brand" href="{{url\_for('index')}}"> Home </a>

<a class="navbar-brand" href="{{url\_for('add\_pup')}}"> Add a Puppy </a>

<a class="navbar-brand" href="{{url\_for('delete\_pup')}}"> Remove a Puppy </a>

<a class="navbar-brand" href="{{url\_for('list\_pup')}}"> List all Puppy </a>

</div>

</nav>

{% block content %}

{% endblock %}

</body>

</html>

Blocks in Jinja are placeholders for the child templates to fill-in

Home.html

{% extends "base.html" %}

{% block content %}

<h1> Welcome! </h1>

{% endblock %}

--

Add.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Welcome! Add a Puppy </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.name.label}}{{form.name()}}

{{form.submit()}}

</div>

{% endblock %}

--

Delete.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Adopted a puppy? </h1>

<form method='POST'>

{{form.hidden\_tag()}}

{{form.id.label}}{{form.id()}}

{{form.submit()}}

</div>

{% endblock %}

--

List.html

{% extends "base.html" %}

{% block content %}

<div class="jumbotron">

<h1> Here the cute puppies.. </h1>

<ul>

{% for pup in puppies %}

<li> {{pup}} / id:{{pup.id}} </li>

{% endfor %}

</ul>

</div>

{% endblock %}

Larger Applications: Models.py, Views.py, Forms.py

Flask has a built-in **blueprints** capability which will allow us to register modular components for our Flask App

Register blueprints in \_\_init\_\_.py

From flask import Blueprint

Owner\_blueprint = Blueprint(‘owners’,\_\_name\_\_,template\_folder=’templates/owner’)

In \_\_init\_\_.py, register as app.register\_blueprint(owner\_blueprint,url\_prefix=’/owners’)

Use hash function to implement for password security. Libraries for hashing security: Bcrypt, Werkzeug. Pip install flask-bcrypt

Bcrypt.generate\_password\_hash(password) // check\_password\_hash(password, ‘password\_string’)

Pip install Werkzeug // from werkzeug.security import generate\_password\_hash, check\_password\_hash // from flask\_login import LoginManager

From flask-login import user\_login, user\_logout, login\_required

We can use **flask\_dance** library to easily add in OAuth (Open Authorisation) backends for our application.

OAuth 2.0, Flask-OAuth, Flask\_Dance // flask-dance.readthedocs.io

Os.environ[‘AUTHLIB\_INSECURE\_TRANSPORT’] = ‘1’

Os.environ[‘AUTHLIB\_RELAX\_TOKEN\_SCOPE] = ‘1’

From flask-dance.contrib.google import make\_google\_blueprint, google

Blueprint = make\_google\_blueprint(client\_id=’’,client\_secret=’’,offline=True,sope=[‘profile’,’email’])

App.regiser\_blueprint(blueprint,url\_prefix=’/login’)

Social company blog:

Bootstrap allows us to create popups.

If any module has to be imported from a folder, create a \_\_init\_\_.py file under it.

----

Project components: Core, Users, Blogs

Project views: Core – index, info // Users – Register, login, logout, Account, user posts // Blog Posts – create, update, delete, blog posts

Project Models: Users, Blog posts

Users: id, profile image, email, username, password, posts(relationship)

Blog posts: id, user-id, date, title, text

Create CRUD forms for Users and Blogs

Create templates that corresponds with views

Create error\_pages folder to store view file for error messages (when error happens, show our own html pages)

Create static folder to store js, css, image files

----

Components usually associate models with them.

PIL Pillow library allows to handle pictures in python

REST API’s

Flask-Restful library.

REST is a standard way of communication between computers in the web

Systems that support and are compliant with REST are often known as RESTFUL.

Automated systems will communicate with the web application

Postman testing tool, send json objects to rest, wrest commands

A python dictionary maps to an object in Json and a list maps to an array in json

CRUD REST terms (POST, GET, PUT, DELETE)

Postman is a graphical interface for interactive with REST API

Pip install Flask-Resftul. From flask-restful import Resource, API

Resource is the ‘resource’ which is connecting, API is wrapper around the entire application, that allows to connect.

Create a collection in postman. Collection is a folder holding a bunch of requests.

#RESTful API using Postman GUI

#install flask-restful

#create a flask app

# instantiate an API class inheriting Flask app

#create a class inheriting Resource

#define REST method of either post, get, put, delete

#link the class with the api.add\_resource('class','url address')

#run the flask app

#create a collection folder in postman. choose the method to run.

#enter the url to connect wth the app.

#find the result in Postman

#=================================

from flask import Flask

from flask\_restful import Resource, Api

app = Flask(\_\_name\_\_)

api = Api(app)

class HelloWorld(Resource):

def get(self):

return {'hello': 'world'}

api.add\_resource(HelloWorld, '/')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

All the API methods takes the same parameters.

from flask import Flask, request

from flask\_restful import Resource, Api

app = Flask(\_\_name\_\_)

api = Api(app)

# Later on this will be a model call to our database!

# Right now its just a list of dictionaries

# puppies = [{'name':'Rufus'},{name:'Frankie'},......]

# Keep in mind, its in memory, it clears with every restart!

puppies = []

class PuppyNames(Resource):

def get(self,name):

print(puppies)

# Cycle through list for puppies

for pup in puppies:

if pup['name'] == name:

return pup

# If you request a puppy not yet in the puppies list

return {'name':None},404

def post(self, name):

# Add the dictionary to list

pup = {'name':name}

puppies.append(pup)

# Then return it back

print(puppies)

return pup

def delete(self,name):

# Cycle through list for puppies

for ind,pup in enumerate(puppies):

if pup['name'] == name:

# don't really need to save this

delted\_pup = puppies.pop(ind)

return {'note':'delete successful'},404

class AllNames(Resource):

def get(self):

# return all the puppies :)

return {'puppies': puppies}

api.add\_resource(PuppyNames, '/puppy/<string:name>')

api.add\_resource(AllNames,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

--

Flask-JWT (Json Web Token) library to require authorization before creating a REST API call

From flask\_jwt import JWT, jwt\_required

FlaskRestful documentation

from flask import Flask, request

from flask\_restful import Resource, Api

from secure\_check import authenticate,identity

from flask\_jwt import JWT ,jwt\_required

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'mysecretkey'

api = Api(app)

jwt = JWT(app, authenticate, identity)

# Later on this will be a model call to our database!

# Right now its just a list of dictionaries

# puppies = [{'name':'Rufus'},{name:'Frankie'},......]

# Keep in mind, its in memory, it clears with every restart!

puppies = []

class PuppyNames(Resource):

def get(self,name):

print(puppies)

# Cycle through list for puppies

for pup in puppies:

if pup['name'] == name:

return pup

# If you request a puppy not yet in the puppies list

return {'name':None},404

def post(self, name):

# Add the dictionary to list

pup = {'name':name}

puppies.append(pup)

# Then return it back

print(puppies)

return pup

def delete(self,name):

# Cycle through list for puppies

for ind,pup in enumerate(puppies):

if pup['name'] == name:

# don't really need to save this

delted\_pup = puppies.pop(ind)

return {'note':'delete successful'}

class AllNames(Resource):

@jwt\_required()

def get(self):

# return all the puppies :)

return {'puppies': puppies}

api.add\_resource(PuppyNames, '/puppy/<string:name>')

api.add\_resource(AllNames,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Database:

import os

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

from flask\_restful import Api,Resource

from secure\_check import authenticate,identity

from flask\_jwt import JWT ,jwt\_required

from flask\_migrate import Migrate

app = Flask(\_\_name\_\_)

###################################################

################ CONFIGURATIONS ###################

##################################################

# Often people will also separate these into a separate config.py file

app.config['SECRET\_KEY'] = 'mysecretkey'

basedir = os.path.abspath(os.path.dirname(\_\_file\_\_))

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///' + os.path.join(basedir, 'data.sqlite')

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

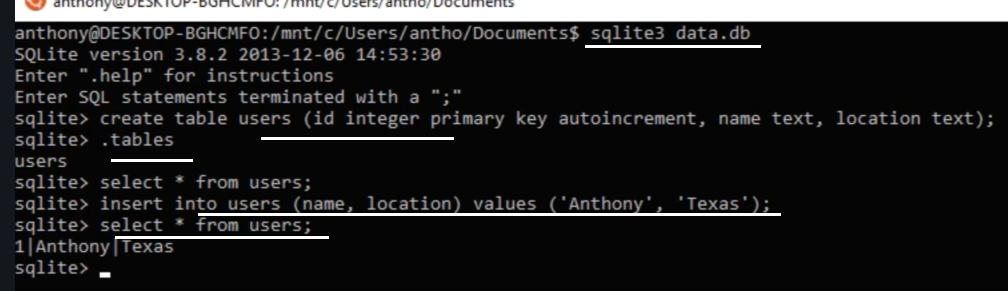
db = SQLAlchemy(app)

Migrate(app,db)

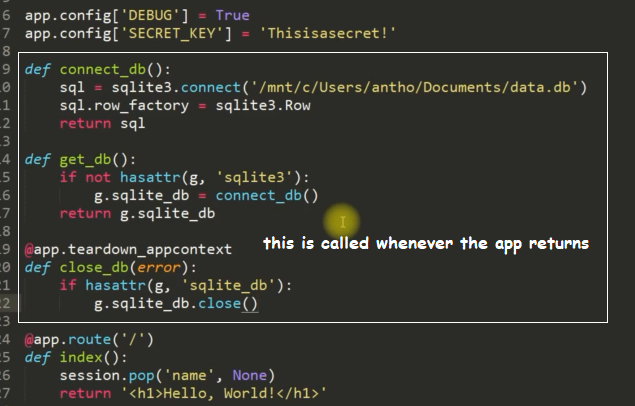
jwt = JWT(app, authenticate, identity)

api = Api(app)

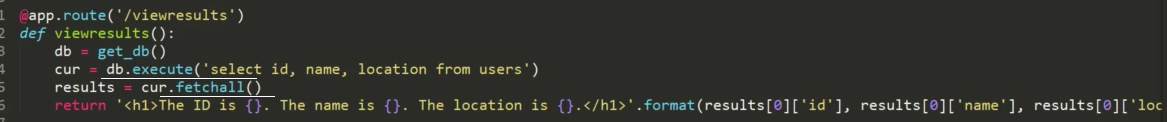
create sqlite3 table:



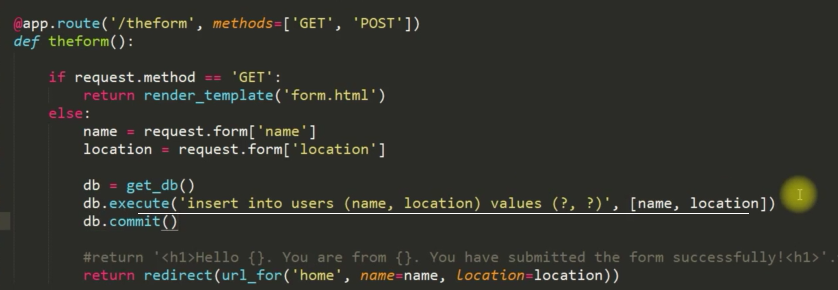
Connecting the database



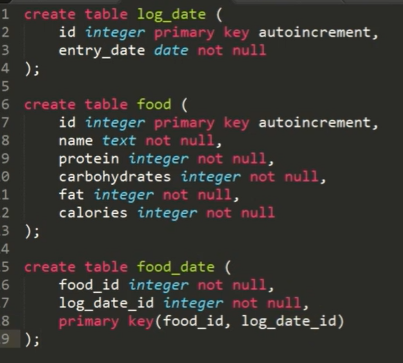
Querying the database:



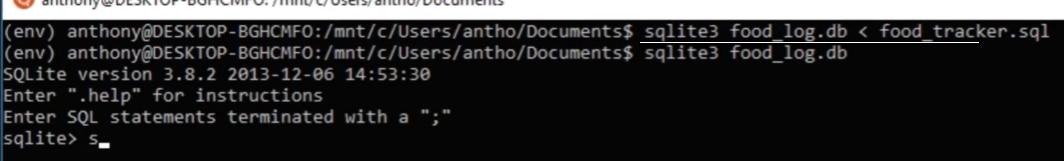
Inserting data into the dictionary:



Creating the sql file:



Linking sql file to the database:

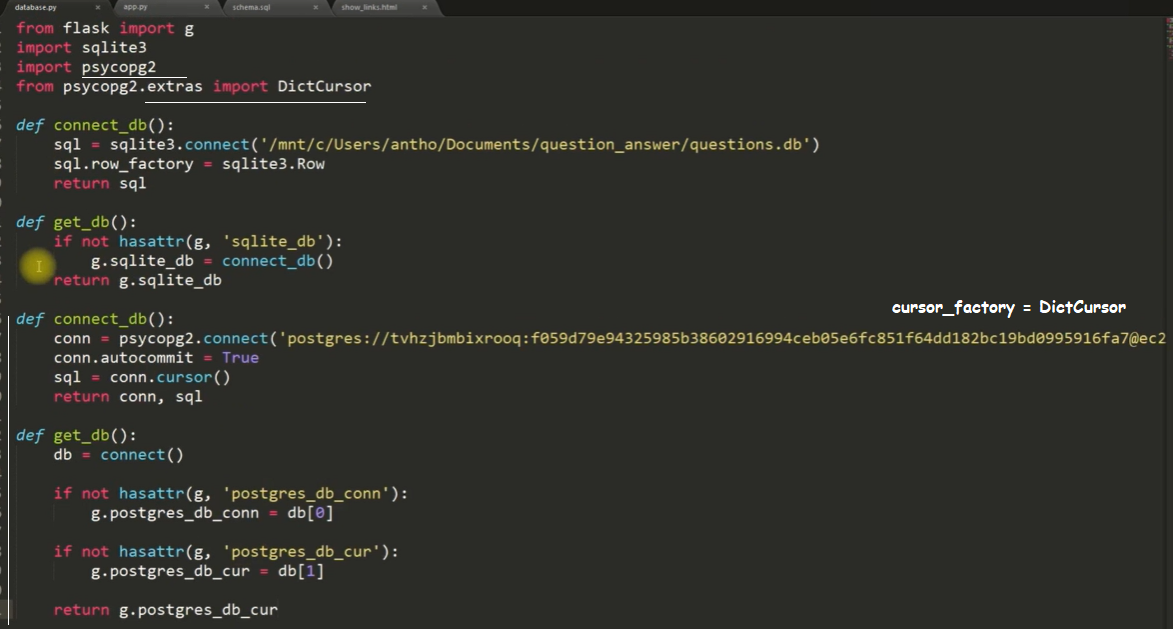


Connecting to postgre database:

First create a uri in Horeku by creating a postgre database. Then install in system by pip install psycong2

Then pip freee > requirements.txt

Then in the application connect to the db as below: (sqlite code here is not needed)



###################################################

################ MODELS ###########################

##################################################

class Puppy(db.Model):

name = db.Column(db.String(80),primary\_key=True)

def \_\_init\_\_(self,name):

self.name=name

def json(self):

return {'name': self.name}

def \_\_str\_\_(self):

return f"{self.name} "

###################################################

################ RESOURCES ###########################

##################################################

class PuppyResource(Resource):

def get(self,name):

pup = Puppy.query.filter\_by(name=name).first()

if pup:

return pup.json()

else:

# If you request a puppy not yet in the puppies list

return {'name':'not found'}, 404

def post(self,name):

pup = Puppy(name=name)

db.session.add(pup)

db.session.commit()

return pup.json()

def delete(self,name):

pup = Puppy.query.filter\_by(name=name).first()

db.session.delete(pup)

db.session.commit()

return {'note':'delete successful'}

class AllPuppies(Resource):

#@jwt\_required()

def get(self):

# return all the puppies :)

puppies = Puppy.query.all()

# return json of (puppies)

return [pup.json() for pup in puppies]

api.add\_resource(PuppyResource, '/puppy/<string:name>')

api.add\_resource(AllPuppies,'/puppies')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

Heroku platform for Flask app deployment.

Steps: install a few libraries and command line tools to connect the local application and push to Heroku platform.

1. Install gunicorn
2. Install Heroku CLI
3. Create Flask app
4. Create requirements.txt file
5. Create a Procfile
6. Create a Heroku account
7. Push and launch the application

Gunicorn is a python wsgi for unix

Env flaskdeploy

Pip freeze > requirements.txt

Create an app in Heroku. From cmd, heroku login, Git init on the folder. Heroku git:remote –a <app name>

Git push heroku master. Copy the http address and run in the browser. <https://jj-my-first-flask-app.herokuapp.com/>

--

Online payments: Stripe is a company that allows to easily integrate their API to your web application to accept payments.

Shopify is one which allows to create online sore app quickly.

Pip install –upgrade stripe // <https://stripe.com/docs/legacy-checkout/flask>

[okstamps@gmail.com](mailto:okstamps@gmail.com) // 1@Puthinthara

The Ultimate Flask Course – Anthony Herbert 20th Apr ‘20

Command to create a virtual python environment: python(or py) –m venv <environment name>

Venv created as Anthony\flask\_app\’env’

Activate venv by running env\scripts\activate. Then install flask by running pip install flask

Create a file as app.py under app\_flask folder and and set environment variable for Flask to look for flask app as [in cmd] set FLASK\_APP=app.py

Pip freeze : to display the contents of the venv

To check if an environment variable is set in cmd, use ‘echo %VARIABLE\_NAME%’

To run the flask app, start the flask server by typing ‘flask run’ in cmd. It will look for the environment variable set for the app file and will provide a url with the id and port where the app is running.

Ctrl + c to quit the server. ‘deactivate’ to end the venv.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/<name>') #<name> is the place holder, in url, we can type anything after the '/'.

def index(name):

return '<h1>Hello {} </h1>'.format(name)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

another method to run flask app is by entering following code in py file and the run the command ‘python app.py’ from cmd

if \_\_name\_\_ == ‘\_\_main\_\_’:

app.run() # app.run(debug=True) is not supported in newer versions of flask.

If the app is changed, export FLASK\_APP=<new app.py>

To run flask in ‘debug’ mode, run export FLASK\_DEBUG=1 in cmd then start the server

Using ‘pipenv’ will create a virtual environment and will install whatever is required. Frist we need to install pipenv by ‘pip install pipenv’. Then ‘pipenv install flask’ then ‘pipenv shell’ will create an environment with the name of the folder we are in.

Flrom flask import jsonify

A dictionary maps to an object in json, a list maps to an array in json

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/<name>') #<name> is the place holder, in url, we can type anything after the '/'.

def index(name):

return '<h1>Hello {} </h1>'.format(names)

@app.route('/home', methods=['GET','POST'])#http methods allowed to access this end point. default is 'GET'

def home():

return **jsonify**({"key1":"Fist","key2":[1,2,3,'a']})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/', **defaults**={"name":"Flask"}) # defaults will replace the place holder value if not given

@app.route('/<**string**: name>') #string is the type of the place holder variable

def index(name):

return '<h1>Hello {} </h1>'.format(name)

#from flask import request # passing a qurey string in the url as /query?name=Jaison&location=Bangalore

@app.route('/query')

def query():

name=request.args.get('name')

location=request.args.get('location')

return '<h1> Hi {}, you are in {}'.format(name,location)

#processing a form

#provides a login form to the user

@app.route('/loginform')

def loginform():

return '''

<form method='POST' action='/process'>

<input type="text" name="firstname">

<input type="text" name="lastname">

<input type="submit" value="Submit"> </form>

'''

#endpoint that receive the form data

@app.route('/process', methods=["POST"])

def process():

fname = request.form["firstname"]

lname = request.form["lastname"]

return '<h1>Hello {} Iam from {} </h1>'.format(fname,lname)

#process a POST request coming from Postman

@app.route('/processjson', methods=["POST"])

def processjson():

data = request.get\_json() #posted as raw data from JSON in POSTMAN

name = data["name"]

location = data["location"]

return jsonify({"result":"success","name":name,"location":location})

#GET/POST requests on the same function

@app.route('/loginform', methods=["GET","POST"])

def loginform():

if request.method == "GET":

return '''

<form method='POST' action='/loginform'>

<input type="text" name="firstname">

<input type="text" name="lastname">

<input type="submit" value="Submit"> </form>

'''

else:

fname = request.form["firstname"]

lname = request.form["lastname"]

return '<h1>Hello {} Iam from {} </h1>'.format(fname,lname)

#provides a login form to the user and then redirects the user to index page

@app.route('/loginform', methods=["GET","POST"])

def loginform():

if request.method == "GET":

return '''

<form method='POST' action='/loginform'>

<input type="text" name="firstname">

<input type="text" name="lastname">

<input type="submit" value="Submit"> </form>

'''

else:

fname = request.form["firstname"]

lname = request.form["lastname"]

return redirect(url\_for("index", fname=fname, lname=lname)) #redirect for\_url

app.config[‘DEBUG’] = True –its one way of setting the configuration keys for Flask.

Use ‘session’ to keep data persistent across pages.

#creating session with flask

from flask import Flask, session

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY']="mykey"

@app.route('/<name>')

def home(name):

if name not in session:

session[name]=name

return '<h1> Hello {} joining..</h1>'.format(name)

else:

session.pop(name,None)

return '<h1> Hello {} already here</h1>'.format(name)

@app.route('/index')

def index():

return '<h1>Index </h1>'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

#templates folder for html files

#creating session with flask

from flask import Flask, session, render\_template, request

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY']="mykey"

@app.route('/home', methods=["POST"])

def home():

name = request.form['fname'] #coming from form.html

return '<h1> Home sweet home {}</h1>'.format(name)

@app.route('/index')

def index():

return render\_template("form.html")# created a templates folder at the same bracnh and placed the form.html inside

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

#variable passed along the html file to be expanded inside the html.

@app.route('/index')

def index():

return render\_template("form.html", cry="Programming")

inside html .. <p> {{ cry }} </p>p>

#conditional

return render\_template("form.html", cry="Programming", rain=False)

<!-- conditional -->

{% if rain %}

<p> Take the umbrella </p>

{% else %}

<p> Enjoy the sunshine </p>

{% endif %}

{% for x in mylist/mylistofdict %}

<h1> {{ x }} </h1>

{% endfor %}

Static files are image, js or css files

Display the static files from html pages. <img src="{{ url\_for('static', filename='images/img1.jpg') }}">

‘blocks’ in jinja are placeholders for the ‘child templates’ to fill in.

{% block title %}

{% endblock %}

{% block content %}

{% endblock %}

--

{% extends ‘base.html’ %}

{% block title %}

{% endblock %}

{% block content %}

{{ super() }} #to display the base template contents. Otherwise child will override.

{% endblock %}

‘include’ taking contents of one template and injecting into another.

**@**app.route('/index', methods=["GET","POST"], defaults={"name":"**defaults**"}) #to include defaults, add another route decorator

@app.route('/index/<name>')

def index(name):

return render\_template("form.html", cry="Programming", rain=False,name=name)# created a templates folder at the same bracnh and placed the form.html inside

#create a database table

(env) C:\Users\jaisojac\anthony\flask\_app>sqlite3 data.db

SQLite version 3.30.0 2019-10-04 15:03:17

Enter ".help" for usage hints.

sqlite> create table users (id integer primary key autoincrement, name text, location text);

sqlite> .tables

users

sqlite> insert into users (name,location) values("Jaison","Bengaluru");

sqlite> select \* from users;

1|Jaison|Bengaluru

From flask import sqlite3, g # g is a flask global object to store data throughout the application

Bydefault, sqlite3 return tuples as query result.

def db\_connect():

connect = sqlite3.connect('C:/Users/jaisojac/anthony/flask\_app/data.db')

connect.row\_factory = sqlite3.Row # all the attribute names should be exact. sqlite3.Row returns data in dictionary format

return connect

def get\_db():

if not hasattr(g, 'sqlite3'):

g.sqlite\_db = db\_connect()

return g.sqlite\_db

@app.teardown\_appcontext

def db\_close(error):

if hasattr(g, 'sqlite\_db'):

g.sqlite\_db.close()

@app.route('/viewdb')

def viewdb():

db = get\_db()

cur = db.execute("select \* from users")

data = cur.fetchall()

return '<h1> id {} name {} location {} </h1>'.format(data[0]['id'],data[0]['name'],data[0]['location'])

# output: id 1 name Jaison location Bengaluru

fname = request.form["firstname"]

lname = request.form["lastname"]

db = test.get\_db()

db.execute(**'insert** into users (name,location) values(?,?)', [fname, lname])

db.commit()

{% for item in data %} #displaying items from a database

'<h1> {{ item.id }} {{ item.name }} {{ item.location }} </h1>'

{% endfor %}

#sql commands to create table.

create table log\_date (

id integer primary key autoincrement,

entry\_date date not null

);

create table food (

id integer primary key autoincrement,

name text not null,

protein integer not null,

carbohydrates integer not null,

fat integer not null,

calories integer not null

);

create table food\_date (

food\_id integer not null,

log\_date\_id integer not null,

primary key(food\_id, log\_date\_id)

);

Save the file with .sql format.

In cmd, run sqlite3 food\_log.db < ‘sql script filename>

To exit from sqlite3 in cmd, enter .exit

JADE available in chrome app store for sqlite3 data

Amazon lightsail – to deploy an app. Nginx connects outside world, gunicorn process the requests from and back to nginx.

Sql joins: A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table
* A self JOIN is a regular join, but the table is joined with itself.

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

--

Werkzeug module is for security. From werkzeug.security import generate\_password\_hash, check\_password\_hash

generate\_password\_hash(“password”, method=’sha256’)

check\_password\_hash(“password”, “password”)

secret key is necessary for sessions to work.

App.config[‘SECRET\_KEY] = os.urandom(24)

When the session[‘user’] is True, then user is logged in.

Macro in jinja is like a function in python

{% macro show\_links(user) %}

<html>

{% endmacro %}

{% from “sho\_links.html” import show\_links %}

{{ show\_links(user) }}

--

Heroku does not support sqlite3.

Pip install gunicorn inside the venv

Pip install git // pip freeze >requirements.txt

Pip install psycopg2 –to install Postgre.

Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection).

In postgre db, integer is called serial. Postgre uses %s in place of ? and (tuple, ) in place of [list]. Query does not return to a variable. Its in memory of db.

Prettyprinted.com for sqlalchemy

5 request methods used in rest api: GET,PUT,PUT,PATCH,DELETE

To get data from JSON object, like from Postman, use request.get\_json()

Use jsonify({“one”:”first”,”two”:”second”}) to convert a python dictionary to JSON object.

Database row update command: db.execute(“update table set name = ?, email =?, level = ? where id = ?”, [name,email,level,member\_id])

Delete database command: (“delete from table where id = ?”, [id])

Authentication: request.authourization.username

request.authourization.password

match against the locally assigned variables to give authorization. Do this in a decorator.

Status code like 200, 401 can be returned to the api caller putting a coma after the josnify message.

Deploy API to Python Anywhere.

In python anywhere, in bash, use $ mkvirtualenv --python=/usr/bin/python3 test

Freemysqlhosting.net

To setup sqlalchemy, a configuration file .cfg has to be created.

Configuration entries are:

SQLALCHEMY\_DATABASE\_URI= uri, SLQALCHEMY\_TRACK\_MODIFICATIONS=False

DEBUG=True

From flask\_sqlalchemy import SQLALchemy

App.config.from\_pyfile(.cfg)

Db = SQLAlchemy(app)

Class Member(db.Model):

Sqlalchemy uses the name of the class to create the table. Every table in sqlalchemy needs a primary key.

Sqlalchemy datatypes: integer, string, text(Unicode), Boolean, pickletype, longbinary

Return ‘<member %r> % self.username // note the ‘usage’ of %r.

In cmd python , from application import db // db.create\_all() – creates the database.

Create an object of class/table in python, then db.session.add(obj) // db.session.commit()

To update a row in table, use: object.columname = ‘dsfsdf’

To delete a row from a table in sqlalchemy, use: db.session.delete(object)

To query everything from a table, use: Member.query.all() // query.first()

To query a single row, use: Member.query.filter\_by(username = ‘anthony’).first() – first returns first row.

Use:filter : Member.query.filter(Member.username == ‘Anthony’).first()

Filter query ‘in’ is ‘in\_([‘a…’,’b…’]) – note the use of ‘in’.

A tilde ‘~’ represents ‘not’ in query. It is used with in\_. and uses and\_. Or uses or\_

All these are called generative queries in sqlalchemy.

Member.query.order\_by(Member.id).all()

member.query.limit(5).all() – returns a limit of 5 records

member.query.offset(2).all() – offset skips the first 2 records and return the rest.

Member.query.count() – counts all the records in the table.

Foreign key creates a relationship in database. Backref creates a relationship in sqlalchemy.

Many-to-many relationship requires a mapping table to connect the two tables.

A db table can be created in the application without using a class by using. Db.Table(‘tablename’, db.Column(columnname type, …))

Pip install flask-wtf // from flask\_wtf import FlaskForm // from wtforms import StringField, PasswordField

FlaskForm does cross site request token reference.

Cross-site-request-forgery (csrf) token. // {{ csrf\_token }}

Form.validate() – validates on either GET/POST requests.

Form.validate\_on\_submit() – validates form only on POST.

.format(form.username.data)

For using validators use: from wtforms.validators import InputRequired, Length(min=, max=, message=) AnyOf(values=[“secret”,”password”]

Validators = [InputRequired()] // form.username.errors

IntegerField has a built-in validation.

To prepopulate a FlaskForm, create and initiate a class with attributes. Create an instance of that class and pass it as form = FlaskForm(obj=inst). Then the corresponding attributes of the FlaskForm will be populated with the attribute values from the corresponding class.

FlaskForm(csrf\_enabled=True)

App.config[WTF\_CSRF\_ENABLED]=True

App.config[WTF\_CSRF\_SECRET\_KEY]=True App.config[WTF\_CSRF\_TIME\_LIMIT]=True

field(\*\*kwargs)|safe – can be passed inside a macro

in an application file, only one class inherit a FlaskForm. Other classes can inherit from wtforms import Form.

For ‘field enclosures’ use a class that inherits Form and declare fields in that class.

Assign variable = FormField(classname) – in FlaskForm class. Access data via form.variablename.fieldname

Variable = FieldList(FormField(classname))

A dynamic form is declared inside a route.

Recaptcha field is to prevent robots from filling up the form.

Google reCAPTCHA. App.config[‘RECAPTCHA\_PUBLIC\_KEY’] =

App.config[‘RECAPTCHA\_PRIVATE\_KEY’] =

From flask\_wtf import RecaptchaField

App.config[‘TESTING’] = True – tells flask that this is conducting a testing, so recaptcha can be neglected.

Pip install flask-bootsrap

App.py

from flask import Flask, render\_template

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField

from flask\_bootstrap import Bootstrap

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = "mysecretkey"

bootstrap = Bootstrap(app)

class LoginForm(FlaskForm):

username = StringField("username")

password = StringField("password")

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

def index():

form = LoginForm()

if form.validate\_on\_submit():

return "Form successfully submitted"

return render\_template("boot-form.html", form=form)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

boot-form.html

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

{% import "bootstrap/wtf.html" as wtf %}

{% block title %}

WTForms example

{% endblock %}

{% block body %}

<div class="container">

<form method="POST" action="/">

<d1>

{{ form.csrf\_token }}

{{ wtf.form\_field(form.username) }}

{{ wtf.form\_field(form.password) }}

<input class="btn btn-primary" type="submit" value="Login">

</d1>

</form>

</div>

{% endblock %}

form.populate\_obj(obj) – will populate the fields with the attribute values of the object.

Inline validator is a function that validates a field.

def validate\_username(form, field): #username is the fieldname

if field.data != [okstamps@gmail.com](mailto:okstamps@gmail.com):

raise ValidationError(“wrong username”)

--

DateField(“birthdate”, format=”%Y/%m/%d”) – always use format

--

Flask mail: pip install flask-mail

From flask\_mail import Mail

from flask import Flask

from flask\_mail import Mail

app = Flask(\_\_name\_\_)

app.config['DEBUG'] = True

app.config['TESTING'] = False

app.config['MAIL\_SERVER'] = 'localhost'

app.config['MAIL\_PORT'] = 25

app.config['MAIL\_USE\_TLS'] = False

app.config['MAIL\_USE\_SSL'] = False

app.config['MAIL\_DEBUG'] = True

app.config['MAIL\_USERNAME'] = None

app.config['MAIL\_PASSWORD'] = None

app.config['MAIL\_DEFAULT\_SENDER'] = None

app.config['MAIL\_MAX\_EMAILS'] = None

app.config['MAIL\_SUPPRESS\_SEND'] = False

app.config['MAIL\_ASCII\_ATTACHMENTS'] = False

open\_resource(‘file) – is the method to open file folders

pip install flask-migrate

in cmd, flask db init // flask db migrate

flask migrate only works with sqlalchemy

pip install flask-sqlalchemy

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

db = app.config[‘SQLALCHEMY\_DATABASE\_URI’] = ‘database uri’

migrate = Migrage(db,app)

module ‘alembic’ is related to migration in migrations file

flask db upgrade – will apply all the updations in the migration file to the database

**SQLAlchemy** is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

Pip install flask-script

From flask-script import Manager

From flask-migrate import MigrateCommand

manager = Manager(app)

manager.add\_command(‘db’, MigrateCommand)

under if \_\_name\_\_ == ‘\_\_main\_\_’:

manager.run()

in mcd, python app.py runserver

pip install flask-uploads. From flask\_uploads import Manager

pip install flask-login . from flask\_login import LoginManager. Flask\_login uses sessions. Sessions need ‘SECRET\_KEY’ to set.

From urllib.parse import urlparse, urljoin

User\_login(user, remember=True) – remember creates a cookie for the user.

From itsdangerous import SignatureExpire, UrlSafeSerializer, UrlSafeTimesSerializer – it embed a timestamp token inside the cookies. SignatureExpired is an exception

Pymongo – a mongo database with flask login.

From flask-pymongo import PyMongo

Mongo = PyMongo(app)

Members = mongo.db.members

Mlab for mongodb

Pip install flask-admin

From flask\_admin import Admin

From flask\_admin.contrib.sqla import ModelView

A class inherited from (db.Model) is creating the database and a view inheriting from (ModelView) is the view added to the admin.

Admin.add\_view(UserView(User, db.session))

To manage files, from flask\_admin.contrib.fileadmin import FileAdmin

From os.path import dirname, join

Auth has 2 methods: is\_accessable, inaccessible\_callback

Pip install flask-login

Pip install flask-user

UserMixin has is\_authenticated, get\_id, is\_active, is\_anonymous methods defined

App.config.from\_pyfile(‘config.cfg’)

Pip install flask-babel

One of the important thing in flask bebel is lacale. It’s a combination of language and region.

from flask import Flask, request

from flask\_babel import Babel, get\_locale

app = Flask(\_\_name\_\_)

babel = Babel(app)

@babel.localeselector

def localeselector():

return request.accept\_languages.best\_match(['en','es','de'])

@app.route('/')

def index():

return '<h1> Locale is {} </h1>'.format(get\_locale())

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

pip install flask-restless

for wrestles flask api, put and patch does the same.

Pip install-socketio // pip install eventlet – how the messages are sent to the lowest level

Socket.io has many libraries. Funcitons: send, emit, user\_join, user\_leave …

Pip install flask-security

Blog on flask: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

<https://github.com/realpython/discover-flask>

<https://pythonspot.com/python-flask-tutorials/>

<https://flask.palletsprojects.com/en/1.1.x/>

db migrate stages: db init, db migrate, db upgrade.

In windows, the databse uri sqlite ‘///’. In linux its ‘////’ 4

Workaround on: werkzeug ‘secure\_filename’ for uploads

.schema <tablenme> to describe table in sqlite3

csrf\_token() is mandatory for submitting form.

Can’t create two table having foreign key relation at a time using migrate and upgrade?

Basic flask application refraction: app.py, models.py, views.py, forms.py

To call random function in query, use order\_by(db.func.random())

SQLAlchemy

Setup free dabase in freemysqlhosting.net

Uri = mysql://username:password@hostname/database

Pip install flask-sqlalchemy

Pip install mysql-connector-python (database driver – for some databases to be used)

Pip install mysqlclient

Python2: Info on installing MySQL-Python: https://stackoverflow.com/questions/25865270/how-to-install-python-mysqldb-module-using-pip

Python3: Info on PyMySQL: <https://github.com/PyMySQL/PyMySQL>

Enter configuration key=values in .cfg file

Remotemysql.com // Username: BvzMUw2JYJ

Database name: BvzMUw2JYJ

Password: Jsa0m9ljvB

Server: remotemysql.com

Port: 3306

Phpmyadmin.co

SQLALCHEMY\_DATABASE\_URI='mysql://BvzMUw2JYJ:Jsa0m9ljvB@remotemysql.com:3306/BvzMUw2JYJ' ( mysql://username:password@server/db)

https://flask-sqlalchemy.palletsprojects.com

adding data into database is by creating class/table object and adding to it

db.session.add(object) // db.session.delete(object) then db.session.commit()

results = Table.query.all()

for r in results:

print(r.username)

results = Member.query.filter\_by(username=’jaison).first() – returns the first row from the query results. All() return all the rows.

results = Member.query.filter(Member.username==’jaison’).first() – use of ‘filter’

in\_([“ “,” “]) // db.and\_(“ “,” “)

Member.query.order\_by(Member.id).all()

12many – member-order analogy

The foreign key is for the database tables to have relationship. The ‘relationship’ is for the sqlalchemy. Backref column is a virtual column in the 2nd table where the foreignkey is defined. The value of that column should only be the object of the first column, not the field.

Many2many – students-courses analogy

Many to many relationship requires a mapping table to connect the two tables.

--

App.py

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config.from\_pyfile('config.cfg')

db = SQLAlchemy(app)

class Test(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(30))

class Member(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(30), unique=True)

password = db.Column(db.String(20))

email = db.Column(db.String(25), unique=True)

join\_date = db.Column(db.DateTime)

orders = db.relationship('Order', backref='member', lazy='dynamic')

courses = db.relationship('Course', secondary='user\_courses', backref='member', lazy='dynamic')

def \_\_repr\_\_(self):

return 'Member: %r' %self.username

class Order(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(30))

member\_id = db.Column(db.Integer, db.ForeignKey('member.id'))

class Course(db.Model):

id = db.Column(db.Integer, primary\_key=True)

course\_name = db.Column(db.String(30))

db.Table('user\_courses', db.Column('member\_id',db.Integer, db.ForeignKey('member.id')),\

db.Column('course\_id',db.Integer, db.ForeignKey('course.id')))

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

print("hello")

config.cfg

SQLALCHEMY\_DATABASE\_URI='mysql://BvzMUw2JYJ:Jsa0m9ljvB@remotemysql.com:3306/BvzMUw2JYJ'

SQLALCHEMY\_TRACK\_MODIFICATIONS=False

DEBUG=True

>>> from app import db, Member, Course

>>> mq1 = Member.query.filter(Member.username=='Roopa').first()

>>> mq2 = Member.query.filter(Member.username=='Savin').first()

>>>

>>> cq1 = Course.query.filter(Course.course\_name=='Python').first()

>>> cq2 = Course.query.filter(Course.course\_name=='Flask').first()

>>>

>>> cq1

<Course 1>

>>> mq1

Member: 'Roopa'

>>>

>>> cq1.member.append(mq1)

>>> cq2.member.append(mq1)

>>> cq2.member.append(mq2)

>>> db.session.commit()

>>>

>>> q1 = Member.query.filter(Member.username=='Roopa').first()

>>> q1

Member: 'Roopa'

>>> q1.courses.all()

[<Course 1>, <Course 2>]

>>> q2 = Member.query.filter(Member.username=='Savin').first()

>>> q2

Member: 'Savin'

>>> q2.courses.all()

[<Course 2>]

>>>

>>> q1 = Course.query.filter(Course.course\_name=='Python').first()

>>> q1

<Course 1>

>>> q1.member #member is a list. So no .all()

[Member: 'Roopa']

>>> q1 = Course.query.filter(Course.course\_name=='Flask').first()

>>> q1.member

[Member: 'Roopa', Member: 'Savin']

https://www.tutorialspoint.com/sqlalchemy/index.htm

Username: BvzMUw2JYJ

Database name: BvzMUw2JYJ

Password: Jsa0m9ljvB

Server: remotemysql.com

Port: 3306

---

db4free.net

Database: jjtestdb port:3306

Username: creativeuser

Email: okstamps@gmail.com

pwd: 1@Myshirt

--

Db.drop\_all() – drops all the table

https://www.pythonsheets.com/notes/python-sqlalchemy.html

Your account number is: 410961

**Server:** [sql12.freemysqlhosting.net](http://sql12.freemysqlhosting.net/)  
**Name:** sql12336666  
**Username:** sql12336666  
**Password:** UIgHmlpKXX  
**Port number:** 3306

---

File upload:

import os

from flask import Flask, flash, request, redirect, url\_for,render\_template, send\_from\_directory

from werkzeug.utils import secure\_filename

UPLOAD\_FOLDER = 'static'

ALLOWED\_EXTENSIONS = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

app = Flask(\_\_name\_\_)

app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER

app.config['DEBUG']=True

def allowed\_file(filename):

return '.' in filename and \

filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

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

def upload\_file():

if request.method == 'POST':

# check if the post request has the file part

if 'file' not in request.files:

flash('No file part')

return redirect(request.url)

file = request.files['file']

# if user does not select file, browser also

# submit an empty part without filename

if file.filename == '':

flash('No selected file')

return redirect(request.url)

if file and allowed\_file(file.filename):

filename = secure\_filename(file.filename)

file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

return redirect(url\_for('uploaded\_file',

filename=filename))

return render\_template('file-load.html')

@app.route('/uploads/<filename>')

def uploaded\_file(filename):

return send\_from\_directory(app.config['UPLOAD\_FOLDER'],

filename)

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

\*\* refer in system ‘refactor’ folder ‘run.py’ for refactor configs

Random.choice(‘ABCD’)

Pip install requests – for python import

Openweathermap.org

R = requests.get(url.format(city)).json()

How the db.session.commit(record) when there are morethan 1 table in the database. Data goes to which table?

Pip install flask-security // from flask\_security import Security, SQLALchemyUserDatastore, UserMixin, RoleMixin

Pip install bcrypt – for password encryption.

Enter all the models and tables as parameters to data\_store = SQLAlchemyUserDatastore(..)

Give Security(app, data\_store, register\_form=)

App.config[‘SECURITY\_REGISTRABLE’) = True

App.config[‘SECURITY\_PASSWORD\_SALT’) = True

App.config[‘SECURITY\_SEND\_REGISTER\_EMAIL’) = True

From flask\_security.forms import RegisterForm

--

For upload error in flask:

open your [flask\_uploads.py](https://flask_uploads.py/) from venv

and paste:

from werkzeug.datastructures import FileStorage

from werkzeug.utils import secure\_filename

instead of:

from werkzeug import secure\_filename, FileStorage

--

Flask login: From flask\_login import LoginManager, UserMixin, login\_user, login\_required, logout\_user, current\_user. It requires SECRET\_KEY configuration to app because it uses session.

Loginmanager = LoginManager(app)

@Loginmanager.load\_user

Def load\_user(user\_id):

Return User.query.get(int(user\_id)) #this function specifies which user is login from the sqlalchemy object database.

Login\_user(user) – login the user, adds the user:id to the session and returns the id to flask eachtime the user visits a page in the session. User\_loader checks this id.

{% with messages = get\_flashed\_messages() %}

Loginmanager.login\_view = ‘login’

Loginmanager.message = ‘message’ – used to flash message when user tries to access protected page

App.config[‘USE\_SESSION\_FOR\_NEXT’]=True # it has a bug now.

From urllib.parse import urlparse, urljoin

Def safe\_url(target):

Ref\_rul = urlparse(request.host\_url)

Test\_url = urlparse(urljoin(request.host\_url, target)

Return test\_url in (‘http’, ‘https’) and ref\_url.net\_loc == test\_url.net\_loc

Safe\_url(next)

--

Login\_user(user, remember=True) – remember keeps the cookie information.

App.config[‘REMEMBER\_COOKIE\_DURATION’] = timedelta(seconds=20)

--

Fresh\_login\_required

Loginmanager.refresh\_view = ‘login’

Loginmanager.needs\_refresh\_message = ‘you need to login again’

--

Session token: from itsdangerous import URLSafeSerializer

Roles allow a user to access a particular page only if that user has role assigned. @roles\_accepted(‘admin’)

From flask\_mail import Mail

--

Flask\_admin.contrib.sqla import ModelView – helps to combine SQLAlchemy with admin

Admin.add\_view(ModelView(User, db.session)

Db.drop\_all() – will drop the tables

Pythonhosted.org for docs

Pythonanywhere: username: jaisonjacob/1@678582

Database host address: jaisonjacob.mysql.pythonanywhere-services.com

Username: jaisonjacob

Start a console on: [jaisonjacob$default](https://www.pythonanywhere.com/user/jaisonjacob/databases/database_console/mysql/jaisonjacob$default/)

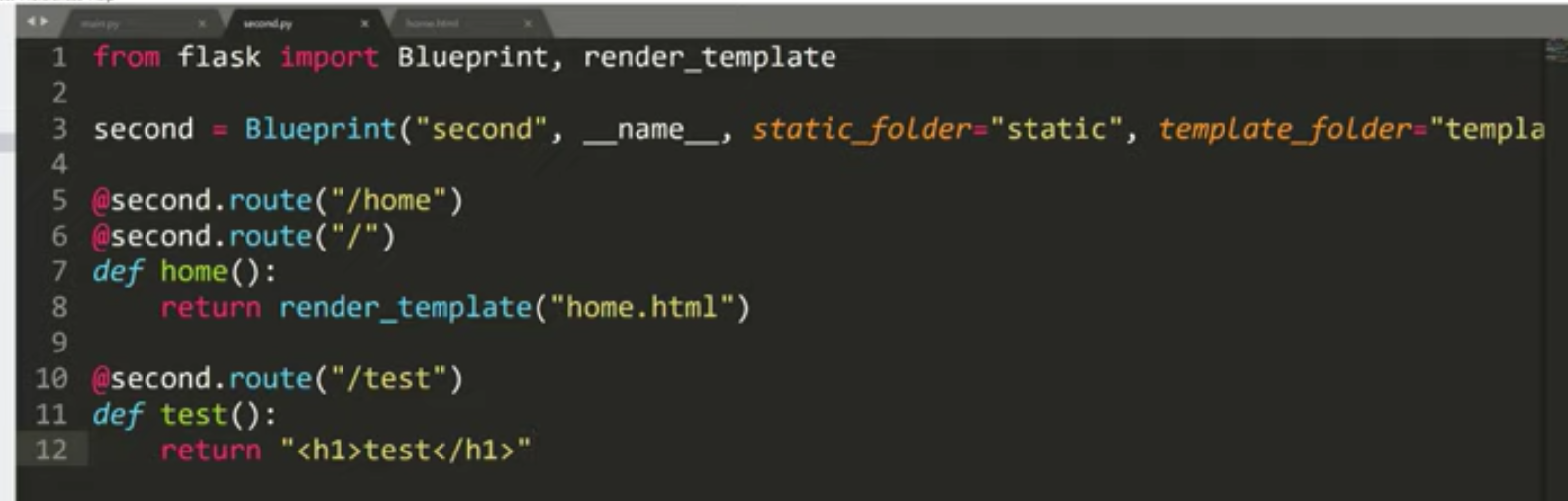
Start a console on: [jaisonjacob$warehouse](https://www.pythonanywhere.com/user/jaisonjacob/databases/database_console/mysql/jaisonjacob$warehouse/)

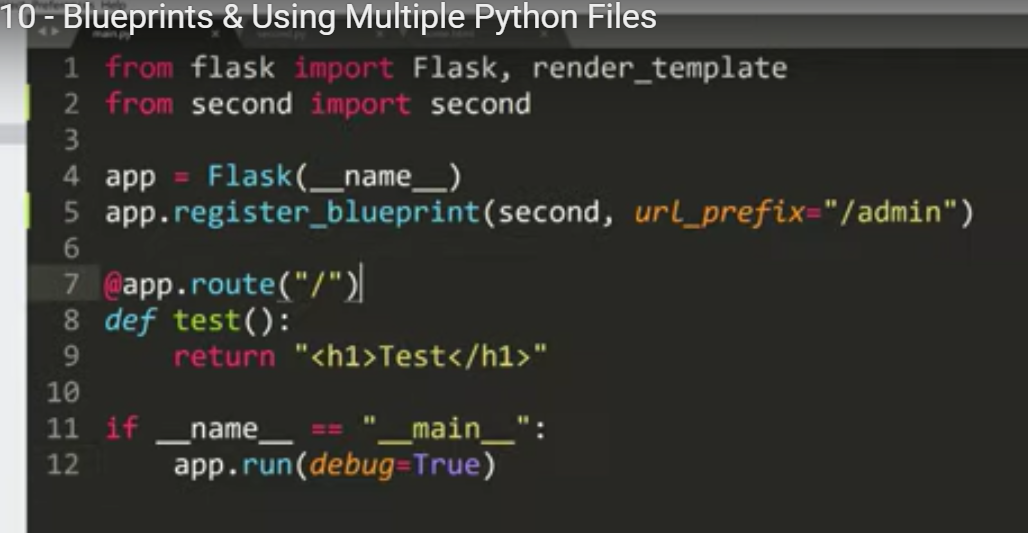
MySQL password: 1@Myshirt

Bash console: mkvirtualenv <pythonpath> <environment name>

To install from requirements file: pip install –r requirements.txt

Blueprint;







<https://www.youtube.com/watch?v=WteIH6J9v64> – for flask blueprint

<https://www.youtube.com/watch?v=dam0GPOAvVI> - for full tutorial from Tim

<https://www.youtube.com/watch?v=YW8VG_U-m48> (1) <https://www.youtube.com/watch?v=Q2eafQYgglM> (2) <https://www.youtube.com/watch?v=qEfduVAQ8FQ> (3)

<https://www.youtube.com/watch?v=-tBdmfov0hM> (4) - use flask with react

xcross