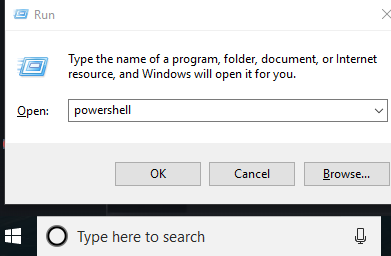
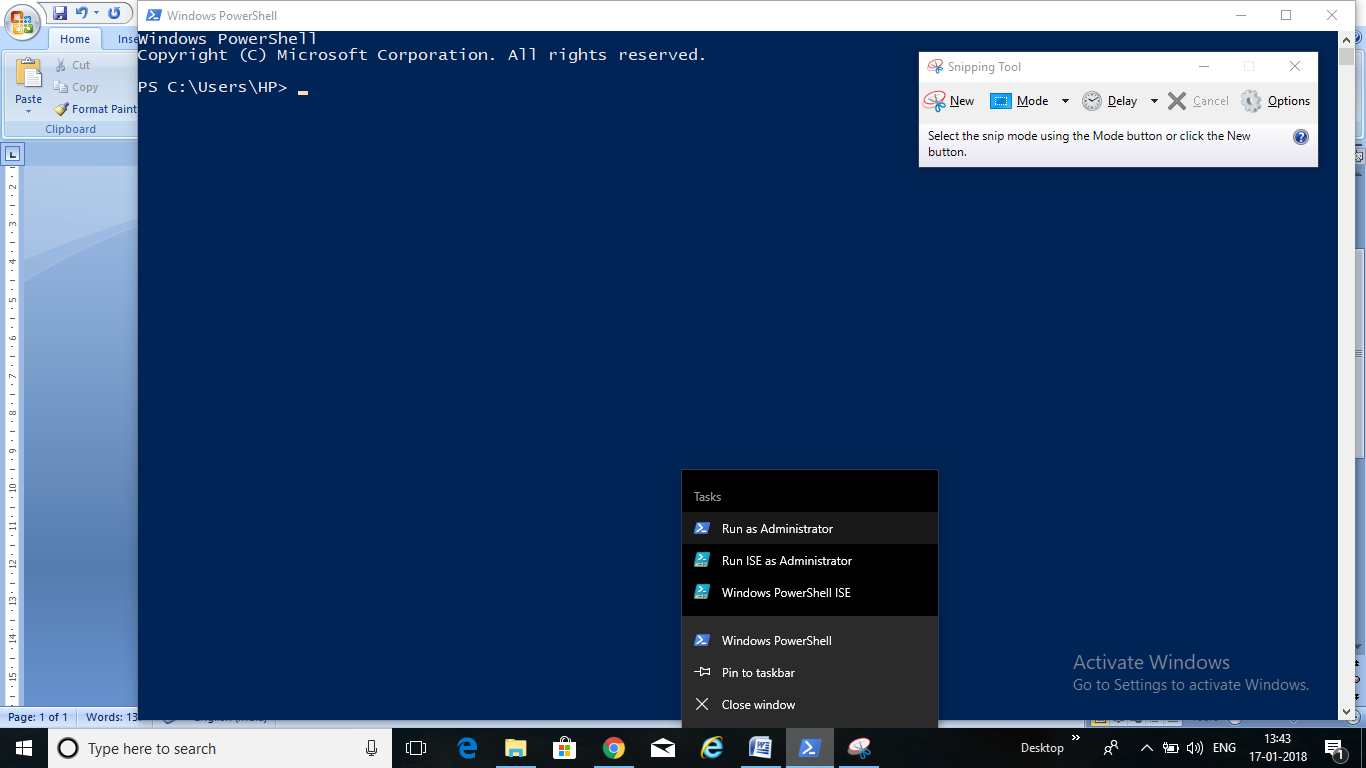
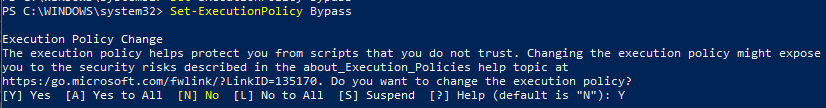
1. Backend development:
   * You have a browser like, internet explorer/ google/ Mozilla, you have a server. When you enter a url in your web browser, server goes to the database and get back all the info in the form of a webpage
2. Why python?
   * It’s easy, easy to read without comments/ going to its documentation.
   * It is object oriented, meaning; we can use other people’s code without having to re-write everything.
   * It has a lot of libraries and community
3. Setting up local dev environment, python 3 in windows with Chocolatey
   * The package manager we will use for windows is chocolatey
   * Chocolatey installs itself in the windows system, hence, we need to relax the execution policy first
   * windows + r
   * type in powershell



* + Right–click on the icon in taskbar and click on - Run as administrator



* + Type in –
  + Set-ExecutionPolicy Bypass and then press Y afterwards



* + Go to 
    - and install chocolatey

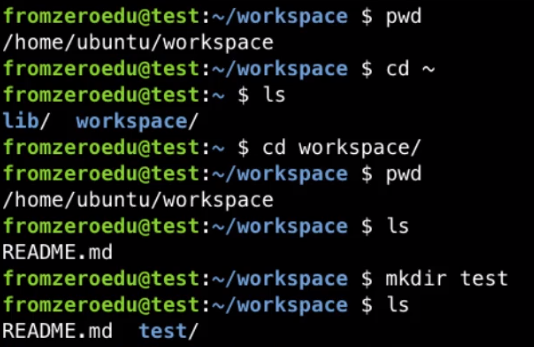


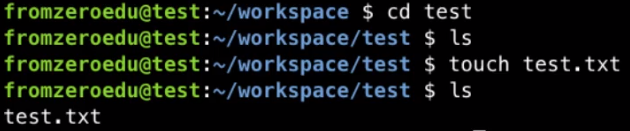
* + - And copy the code in there and paste in the powershell window and press enter. Then, open the new powershell and run as administrator :

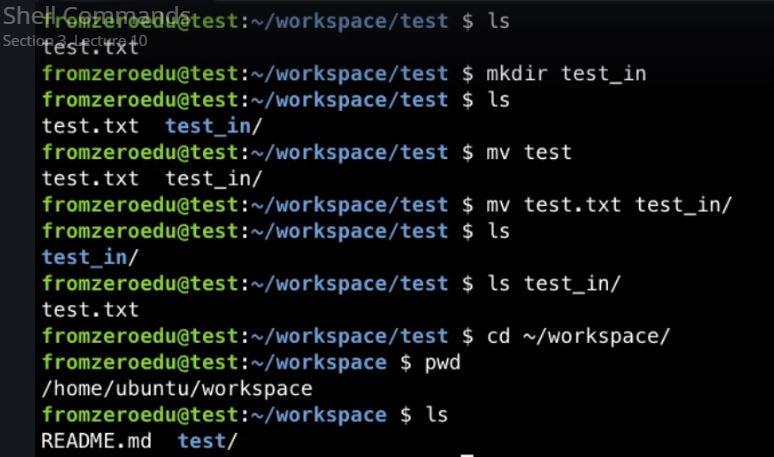


* + In new poweshell, as administrator, type in this to know if chocolatey was installed successfully or not
  + 
  + Now we **install git** with chocolatey
  + 
  + And then press Y
  + Close and open a new powershell as admin and check if git was installed or not
  + 
  + Next, we **install python3**
  + 
  + And press Y
  + Close and open a new powershell as admin and check for python3
  + 
  + Python3 has also installed pip. **Pip is a python package manager**
  + 

1. Setting up the Cloud9 environment
   * We can use a local dev environment (python3) or web service (cloud9) to develop our website.
   * Shell commands:
   * Pwd – present working directory
   * Cd ~ – change directory to home
   * **Ls** – list (of items)
   * Mkdir – make new directory – **mkdir** test\_in/
   * How to make a new file instead? **Touch** test.txt
   * How to move a file to a directory? **mv** test.txt test\_in/
   * If you have many files like, test1.txt, test2.txt, test3.txt and you want to move them to the directory test\_in/ --- **mv test\*.txt test\_in/**
   * To view the content of a file? **cat** test1.txt
   * To view more? **more** test1.txt

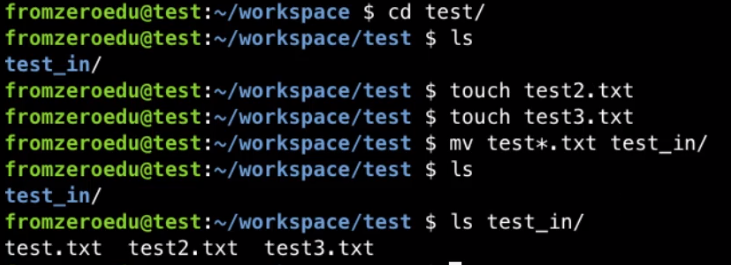


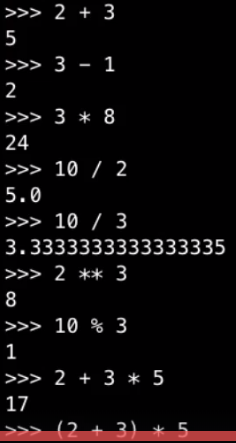
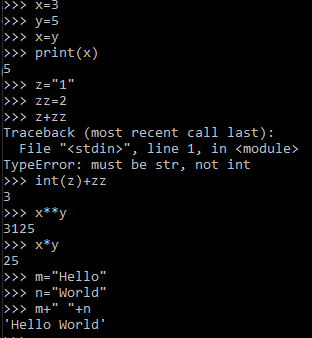
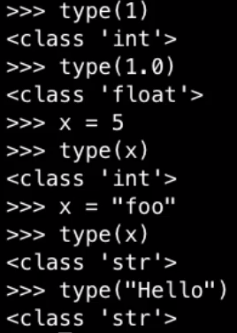




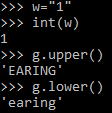
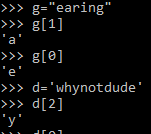






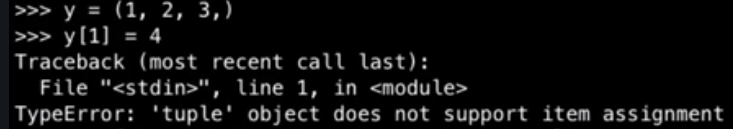
1. Python Basics
   * Search for python3 on your desktop and you can start working
   * Arithmetic
   * 
   * Variables – are boxes which store values which can be referenced later
   * 
   * Checking type-
   * 
   * Pep8 : can read it in free time <https://www.python.org/dev/peps/pep-0008/>
   * Variable format:
     + Use underscore (my\_var) not myVar,
     + should be self explanatory,
     + all caps when variable is to remain constant



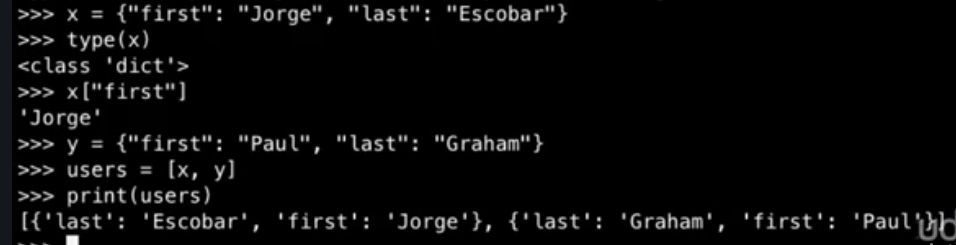
* + Strings:
  + 
  + 
  + Lists, Tuples and Dictionaries:
    - Lists are mutable (not fixed, can be modified)

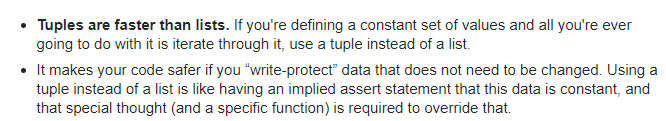


* + - Tuples can’t be modified. We use a comma to end a tuple



* + - Dictionaries: key,value pairs might not remain in the same order you had put them in. That is why we refer a value via it’s key. Not by their serial order

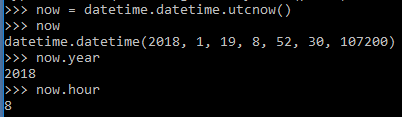




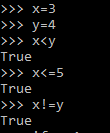


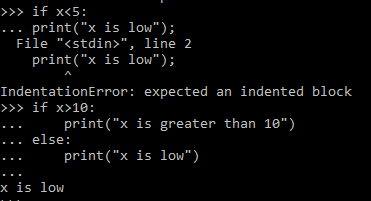
* + - E.g. here we are defining a dictionary
    - 
  + Date and Time:



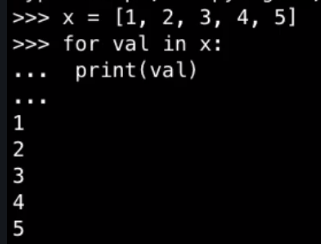


* + Conditional and Control Flow:

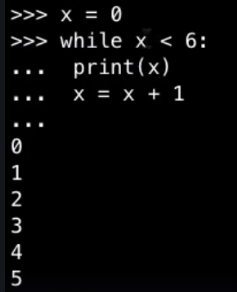


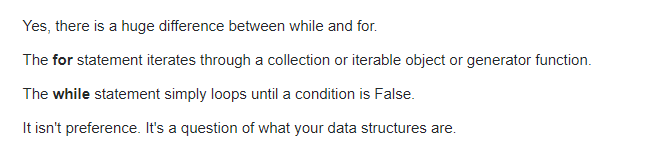


* + Loops:
    - For loop:



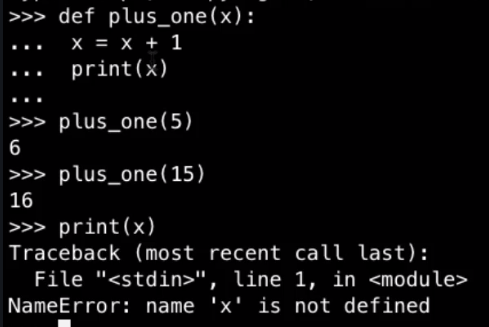
* + - While loop:



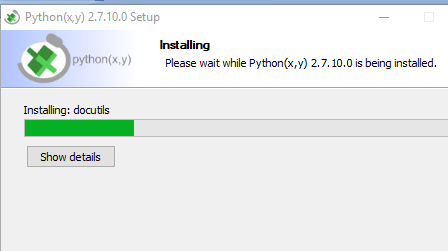




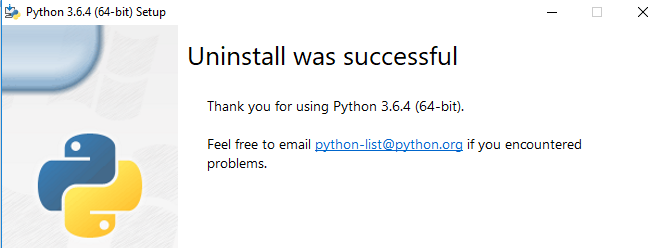
* + Functions: (since x is a local variable, this is called scoping)

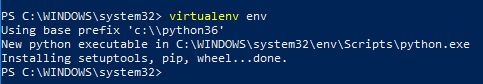


1. INSTALL PYTHON(X, Y) NOW: just search for it in search bar and open .exe



* + Go to C:\Python27\Scripts and look for spyder.bat file and create a desktop shortcut
  + If you have python36 etc too installed, clicking on the desktop icon won’t work
  + Go to control panel, systems, advanced settings, environment, and for the PATH variable have C:\Python27 and delete all those paths with Python36 per se
  + Now, click on the desktop icon and run as admin – you have your spider open

1. WHEN I TRIED DELETING PYTHON2.7 and 3.4.6 AND RE-INSTALL EVERYTHING. I WAS NOT ABLE TO UNINSTALL THEM VIA CONTROL PANEL, PROGRAMS, PROGRAMS AND FEATURES. SO I USED CMD. Run it as administrator. Then type in WMIC. Then type in GET PRODUCT NAME. And then you have
   * 
   * 
   * Couldn’t remove python 3.6.4 using cmd. Then, went to control panel, repair, and then uninstall worked!
   * https://stackoverflow.com/questions/15828294/problems-in-fully-uninstalling-python-2-7-from-windows-7
   * 
   * 
2. After having installed virtualenv. Again open powershell as admin and type in virtualenv env. This creates a virtual environment. New python, pip exe which doesn’t affect the one we installed initially
   * <https://virtualenv.pypa.io/en/stable/userguide/>
   * http://pymote.readthedocs.io/en/latest/install/windows\_virtualenv.html



* + Set the execution policy—if you remote sign the scripts, it will allow these scripts written by it’s owner to run bcoz otherwise they aren’t trusted

 and A

* + Then type in this command to activate the virtual environment env you created (**NOTE S OF SCRIPTS IS SMALL** : it’s env\scripts\activate)

 So, you see (env) :D

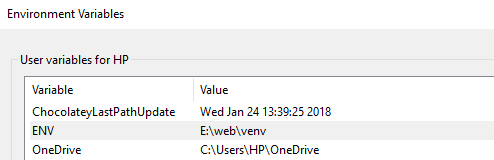
* + Add a new user environment variable ENV to the path to env

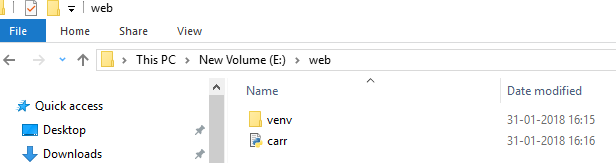


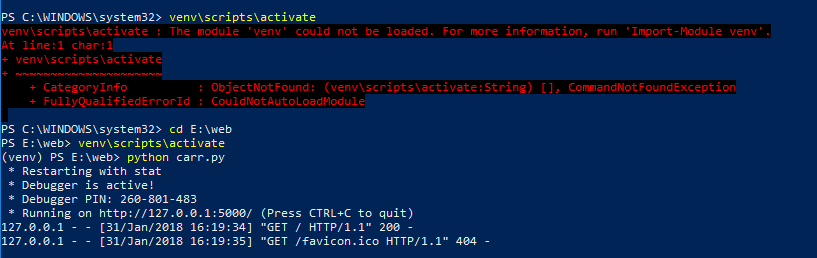
* + 
  +  --this will open it
  + And you can open jupyter in chrome too using the url they provide in the powershell..
  + I installed spyder too ..by pip install spyder and then you open it by typing in spyder3
  + 
  +  in python3, spyder is installed as spyder3

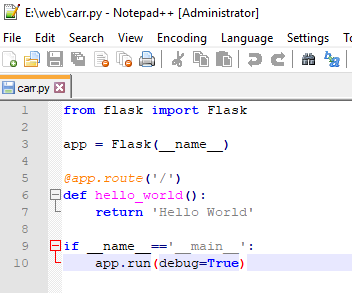
1. **I am using Notepad++ as Editor**

Was not able to run the .py file i created in notepad++ as admin so, copying the env folder here and updating ENV variable (from **C:\Windows\System32\env** to **E:\web\venv**)

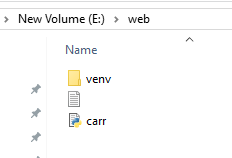






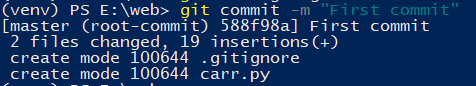


This @sign is called decorator which means that modify the next line to a certain behaviour

1. Starting with Git:
   * 
   * 
   * Then, create a txt document names gitignore inside web folder, open it in notepad++ and add the code below. Then, save and close the file and rename it using powershell
     + Go to <https://gist.github.com/fromzeroedu>
     + Click on gitignore
     + Click on raw
     + Copy the code
     + Paste it in the gitignore.txt file open in your notepad++
   * 
   * Thus, you will have this—
   * 
   * 

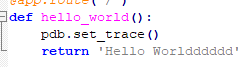






1. Debugging:

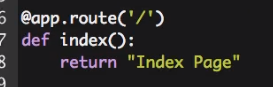
When in debug mode (i.e. Debug=True)—

* + If you make changes to your file, you can see the changes by just refreshing your website and you don’t need to run python carr.py everytime you make changes
  + The whole error shows up on the website
  + Is slower than production mode
  + You can use pdb, to stop the cursor on that line while debugging and then **use n (for next)** and enter to go to the next line and **use** **c to continue**
  + 
  + I got this error—
  + 
  + bcoz i used tab sometimes and other times i was using spaces. You have to be consistent.

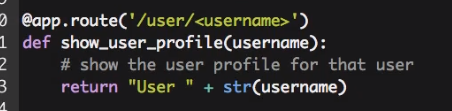
1. Routing:

Route is the access point for that function

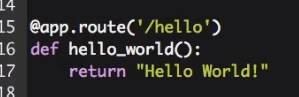
* + When i do what is shown below, Index Page is seen at 127.0.0.1/



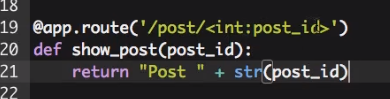
* + When you want to pass a variable, you use <>. Thus, at 127.0.0.1/user/dfgneri you will see User dfgneri where you can pass a string/int/etc



* + You see Hello World at https:/127.0.0.1/hello



* + This implies that you will need to pass an integer only. So you see Post 3 at https:/127.0.0.1/post/3



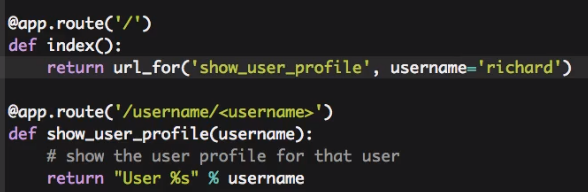
* + - If not an integer you get an error—



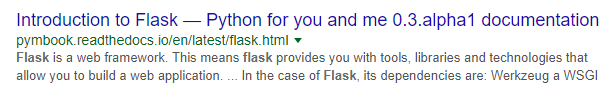
* + In python you do this instead of using say, str(number) etc and make the code look lengthy. **%s implies a string, %d implies a digit**



1. Using url\_for:
   * To get the url for a function, we use it



1. What is flask?

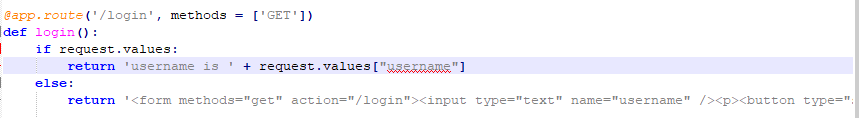


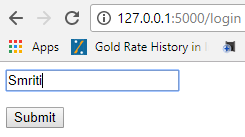
1. The GET Method—

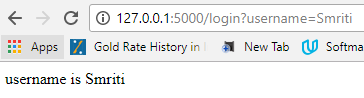
Get and post methods are used to send information entered in the form to server

https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests

Import request. It lets you know the stuff user has entered in the form. In our form—first we have a textbox, then we have a button. Thus, we have input for textbox. Thus we use input and then we use button. Type for input is text and that for button is button

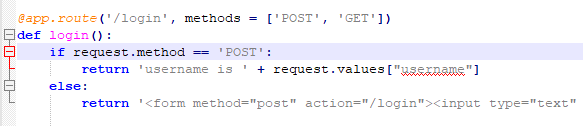






1. The POST Method:

We need to define both get and post in the decorator. Get is used for login-home page

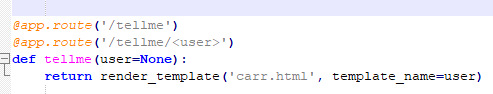


1. Templates:

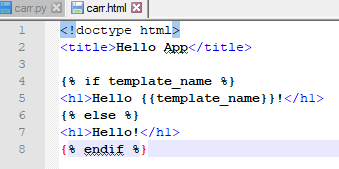
MVC – controller is the logic so, .py files. View is the template, model is the database which stores all the data

Create a folder, templates inside your ‘web’ folder. Create a carr.html file via Notepad++ in that templates folder.

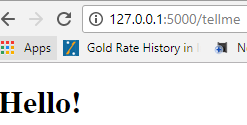
Carr.py:

* + You can define one function for more than one decorators e.g. here:
  + 
  + It is called render\_template bcoz render means these will/ can be modified
  + 

1. Carr.html
   * Use {{ }} when you are getting the value from a function
   * We used if-else here, so that it doesn’t show **Hello None!** when no user is passed





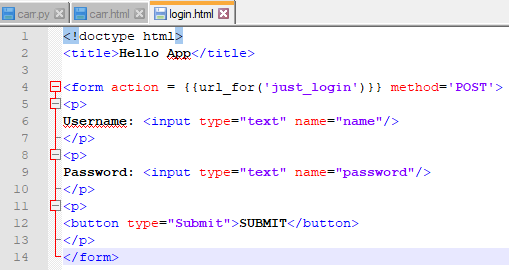


1. The login template:

Create a new .html file

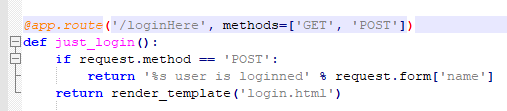
Login.html:

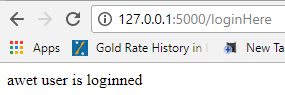
* Form “action” is used to give the path for action
* Button type should be submit, if you have it as button then nothing will happen if you click the button



Carr.py:

* + Get is used when you have the input form in front of you. To get the information. Therefore, we have return render\_template(‘login.html’). this is the empty template
  + And when we have got all the information, then the post method comes into play and here we are posting the data as %s user is loginned







1. The Login Function:

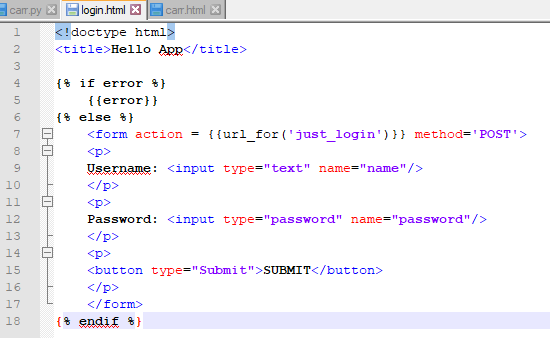
Carr.py

* + Define a function w/o any decorator
  + After user has input the data, i.e. get has been used, then, when it goes to post method to check the input, and if it is an invalid input, you set error = ‘Incorrect...’ and return the error message you have set in .html file



Login.html

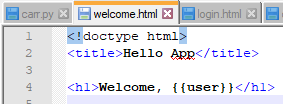
* + If username is not valid then we don’t let it post the data, thus error checking comes at the stage of get. If error is not none, then show the value of error & not the form
  + Input type for password is password, thus, you can’t see it



1. Redirect after POST:

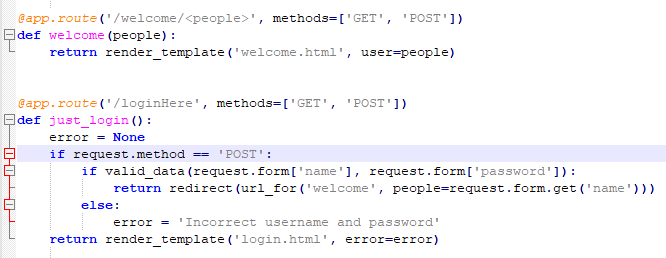
Welcome.html:

* + Making a new template that we want to be appearing after login is successful



Functions in carr.py:

* + Defining this function, welcome which calls the welcome.html with a \*\*different url\*\* because we want the user to get directed to a different url after login is successful
  + Redirect(url) – it redirects you to the url you are passing into redirect()



1. Flash messages:

Carr.py

* + **flash(“successfully login”)** -This stores the all flash messages in a session cookie



* you establish a secret key. Then, a cookie is created with all the info like name, date etc. Then this secret key is appended with this cookie which is then sent to the browser.

https://stackoverflow.com/questions/22463939/demystify-flask-app-secret-key

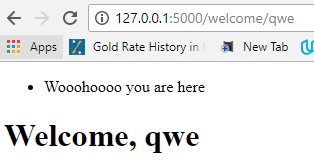


welcome.html:

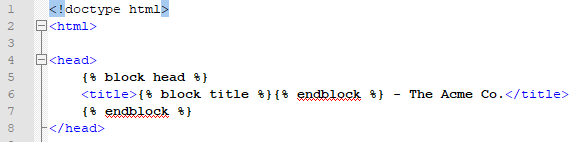
* + get\_flashed\_messages() is an inbuilt function in flask that stores all the flash messages in the form of a list/dict.



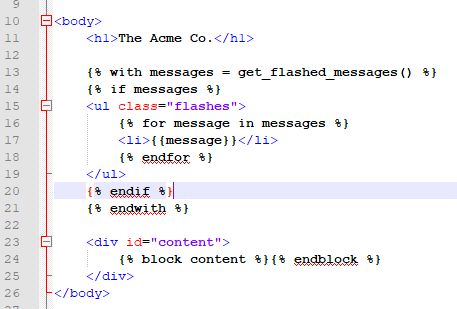
Looks like this—



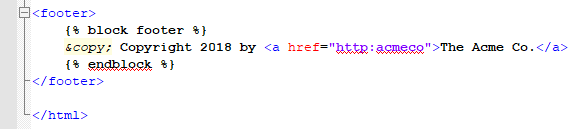
1. Template Inheritence:
   * In base template we have **{% block title %}{% endblock %},** and in the child template we write everything between {% block title %} and {% endblock %}, which replaces {% block title %}{% endblock %} in the base template. DID YOU GET IT?
   * Whatever is inside **{% block title %}{% endblock %}** inbase.html will get replaced by **{% block title %}{% endblock %}** in children.html
   * You now just need to have {% block .... %} ....... {% endblock %} in the children.html
   * Base.html:
     + First you start with <html> </html>
     + Then comes <head></head>. Indent everything inside it
     + We added {% block head %} so that if you want to override this whole head section with the child.html you could.
     + We override {% block title %} with the child.html, so you can have Welcome – The Acme Co. Or Login – The Acme Co. etc



* Now we add <body></body>
* Since logic to show flash messages is going to be valid for all children files, we add it here
* Then you add content block so that one can add any content specific to the particular child file



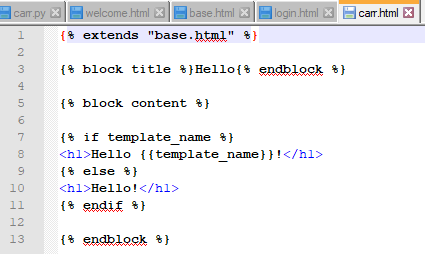
* We add footer for every page, we have added footer block too so that one can override this based on every page’s need



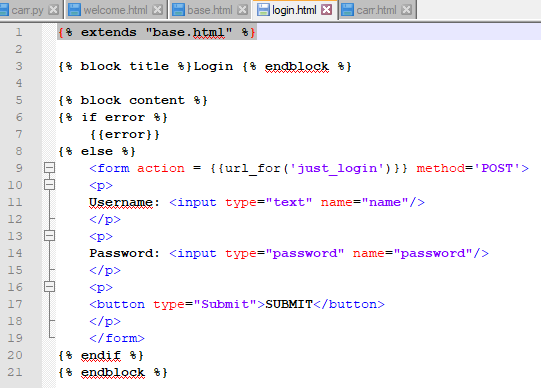
**Children—**

In the child file, you first add {% extends “base.html” %} so that base file is used and overridden by this child file wherever blocks are defined. **So, here, we just define blocks**. Block title and block content in this case

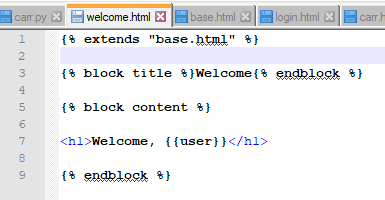
* + Carr.html:



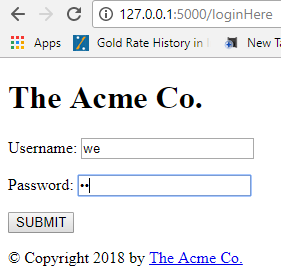
* + Login.html:



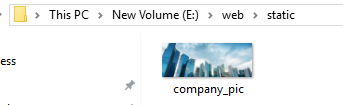
* + Welcome.html:



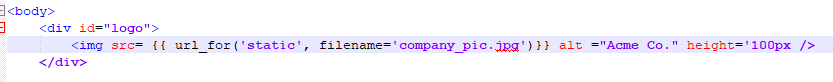
**Output**—

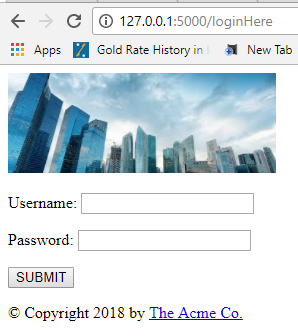


1. Introduction to cookies:
2. Introduction to sessions:
3. Adding static files:
   * Flask uses ‘static’ directory by default to locate static files. So we create a new folder named ‘static’ and keep our .jpg file inside it



* + So, flask creates a route /static/<filename> for the file in static folder. There is an inbuilt function named ‘static’ set at that route. Add this to your base.html so that you can see the logo on every page



* + 

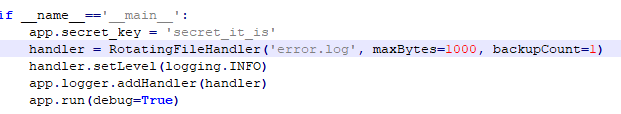
1. Logging:
   * Rotating file handler --- means a new log file is created/ used as soon as the current log file reaches the given number of bytes



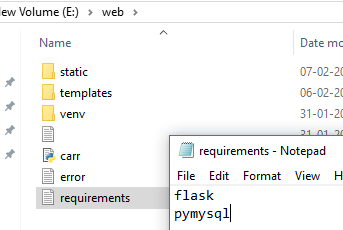
* + Add the below line wherever you want to add logs. I added it in just\_login() function. There are many levels to logger—warning/error/etc



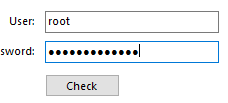
* + Defining the error log file name, max bytes then it moves to a new log file, we keep 1 backup file for our current log file. Logger level is set to INFO. So a log file will be created in the ‘web’ folder



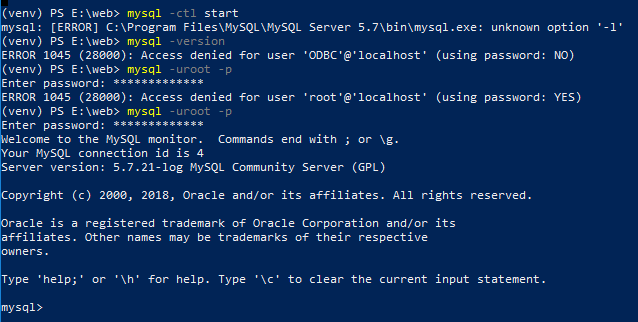
1. Introduction to MySQL:
   * Make a .txt file to write all the softwares required, e.g. flask, mysql



* + Install mySQL: <https://www.youtube.com/watch?v=WuBcTJnIuzo>
    - Install mysql-installer-community not the –web-community one
    - Mysql **root password is Highway!!@@11** new user (smriti) password is Highway!!@@22



* + Put this path in environmental variables 
  + Run powershell and run this—
    - mysql –uroot –p
    - enter the password for root which is Highway!!@@11



* + To know the databases that are already there in mysql server



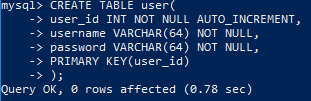
* + To create a new database—



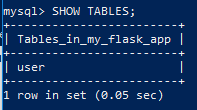
* + A database is a collection of tables. We need to go inside this database and create a table

So, 

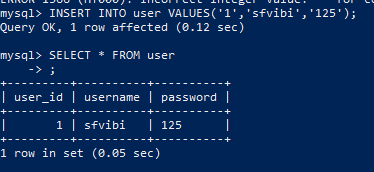
* + Now we create a table inside this database we are in—

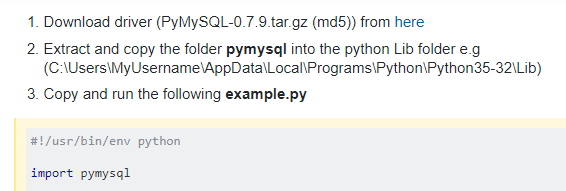
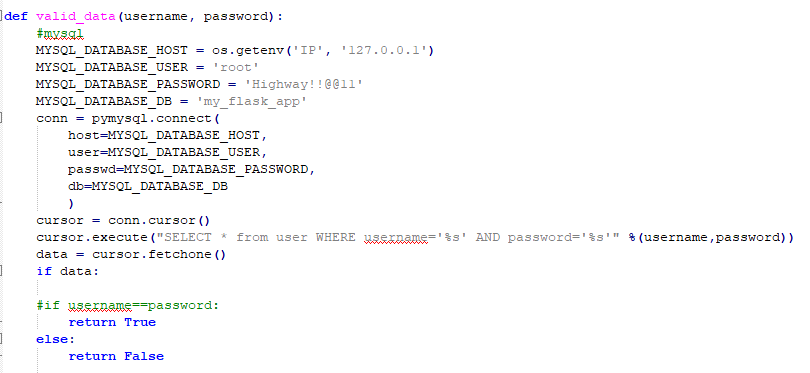


* + We can check the table we created by using the command—

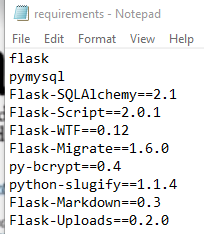


1. Our first table:
   * Insert a row, and then, do select \* from user to view the entire table

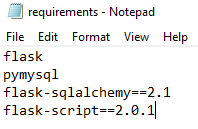


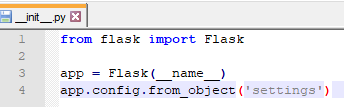
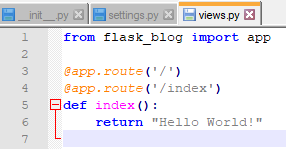
* + To exit the sql mode—
  + 
  + Now, you check the requirements.txt and install the packages mentioned in there. Since flask is already installed, so it will install pymysql
    - Pymysql is the library which will help mysql to talk to flask
  + 
  + Go to carr.py and 
  + I was getting this error – no module named ‘pymysql’
  + <https://stackoverflow.com/questions/35684400/how-to-use-python-3-5-1-with-a-mysql-database>
  + 
  + So, in the above, you have to copy the folder in E:/web/env/Lib
  + Now we add a logic in valid\_data() that if the username and password entered by user matches with what we have in our database, then return True.
  + 

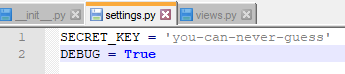
1. Important: Requirements.txt file- just update it, not executing it right now:



1. Introduction to flask\_blog:
   * Create a new folder flask\_blog in E:/
   * Create a virtual env in it – this creates an executable python3.exe
   * and activate it
     + 
     + 
     + 
   * Create a requirements.txt inside the flask\_blog folder to track the lib we will need----
     + **SQLAlchemy** helps you to interact with the SQL database using the object oriented python language. It is an ORM (object relational mapper). We will use this instead of what we did in previous section

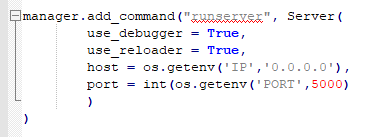
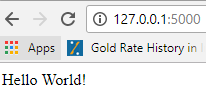


* + Create an \_\_init\_\_.py in the flask\_blog folder
    - We create an flask application instance named, app
    - Then, we set the configuration of the app from settings file which can be different for different environment, for local and production
    - 
  + Keep different folders for different parts of your application to keep their .py files separate and make them use
    - So, create a ‘home’ folder inside E:/flask\_blog --- this home folder is to store the .py files for our home-page
      * Then, create a views.py file in it
      * ‘from flask\_blog import app’ means -- From the \_\_init\_\_ file (in virtual env) in flask\_blog import the flask application instance, app that we created
      * So, this is our home-page
      * 
  + Create a settings.py file in flask\_blog folder to define the settings for this blog/web-app

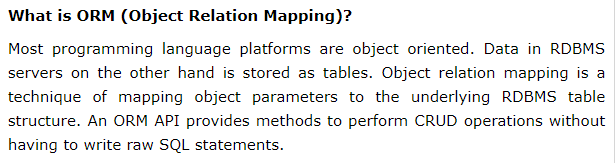


* + Create a manage.py file in flask\_blog folder

manage.py

* + Import os – for host and port
  + Import sys – to get absolute value of the path made by joining two paths
  + 
  + Flask-script extension i.e. **flask.ext.script** provides support to write external scripts in flask, the cmd-tasks that belong outside web application itself
  + Manager class keeps track of all the commands and handles how they are called from the cmd
  + We create an instance of manager class. For it, we need to pass in the instance of our web app, which is app
  + Server in an in-built command
  + Since, we are just using the in-built command, Server. We will just update it. Then, this command needs to be added to the manager instance
  + By default, flask-script has debug set to false, so we will set it to true. Also, we will redefine the defaults set to host and port
  + 
  + 
  + 
  + 
  + So, first we will install flask-script. For that we will just let it install all the libraries mentioned in our requirements.txt. this will install flask-script and sqlalchemy
  + 
  + We will now run ‘runserver’ command from manage.py
  + 
  + 
  + All the explanation is here -- https://flask-script.readthedocs.io/en/latest/

1. Setting up the ORM:



* + First let’s create a blog folder and move home folder in there



* + Go to mysql cmd line interface by doing this—



* + We create a new database named blog and then, exit—

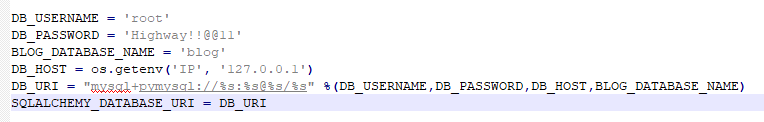






* + We will add to our settings.py file so that we can access sql database from our flask application. So we add the following to our settings.py file

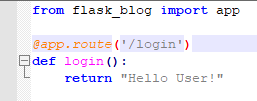




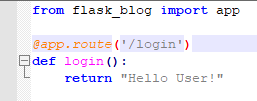
* + We will add sqlalchemy to our init.py file. And, create an instance of it which can be used elsewhere
    - Alchemy means magically transforming. So, SQLAlchemy is a python toolkit which gives the full power of SQL to developers.
    - flask-sqlalchemy is a flask extension which adds support for sqlalchemy to the flask application
    - 
    - We add the below line in the \_\_init\_\_.py so that we can access this object named db from anywhere in the application. Db calls our database. We need to pass the object of our flask app to sqlalchemy

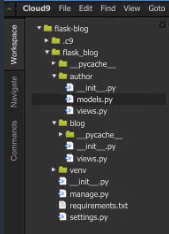


* + Now, create a new folder, ‘author’ in E:/flask\_blog. This is bcoz users will first login before entering our blog
    - Create \_\_init\_\_.py, views.py, models.py in the folder—author
    - Add this to the views.py file

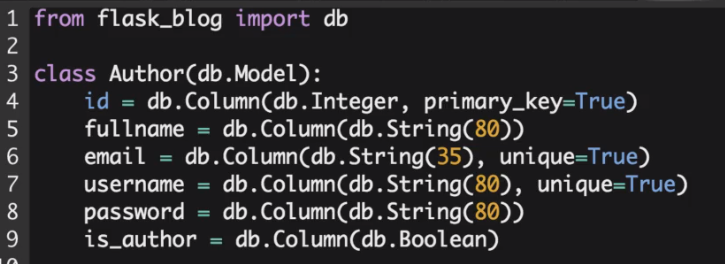


* + Since you created a new folder and a new views.py file, so add/import it to the flask\_blog init i.e. the main web app init file
    - 
  + add \_\_init\_\_.py to blog folder too
  + Why do we need to add \_\_init\_\_.py file to every folder?
    - This tells python that it is not just a folder but, a package. So, when you let’s say import the blog folder or author folder, it knows where to search, which author folder to import.
  + What is MVC?
    - Model – is the database, the sql database we have
    - Template is the view. The presentation of the blog
    - View- is the controller-- is the route. i.e. why we add the below code to the view file we created for author folder

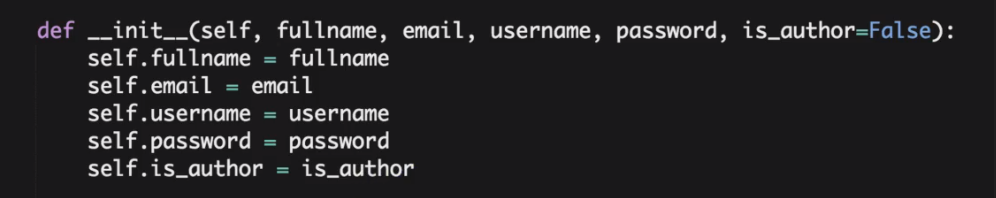


* + 
  + So,
    - in views.py we define routes
    - in models.py we define tables/ databases

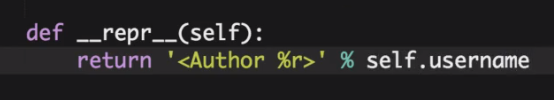
1. The Author Model: http://flask-sqlalchemy.pocoo.org/2.3/models/#models
   * Here, we will be defining the models.py of Author folder we created
   * Base class for all the models is db.Model, i.e. why we pass it as a base class while create a table class, like, author()
   * Table names are derived from class names, e.g. Author will be author, CalC will be cal\_c

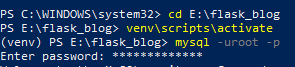
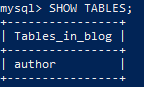


* + We have to then define \_\_init\_\_ which basically what happens when the object is first created. So the username (etc) are passed into the object directly



* + When you call the object in a terminal how should it represent



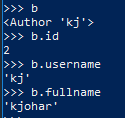
* + We had created an instance of database, db. We now need to create tables, based on the classes we made in models.py. so, tables will be created for all those functions written in author.models.py imported
    - 
    - 
    - 
  + Then, you can open another powershell as admin to check if the table(s) were created or not
    - 
    - 
    - 
    - 
    - 
  + Now, coming back to our first powershell--- we create an object of class Author and pass the input parameters.
    - So, there are 3 steps to insert data into database:
    - Create python object, add it to the session, commit the session. Session here is a beefed up version of a database transaction. Calling add() function adds the object. It will issue an INSERT statement for the database. You need to commit it to get it inserted



* + - Lets add one more entry:

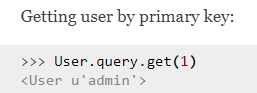
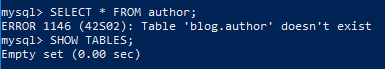


* + - You can play with it—

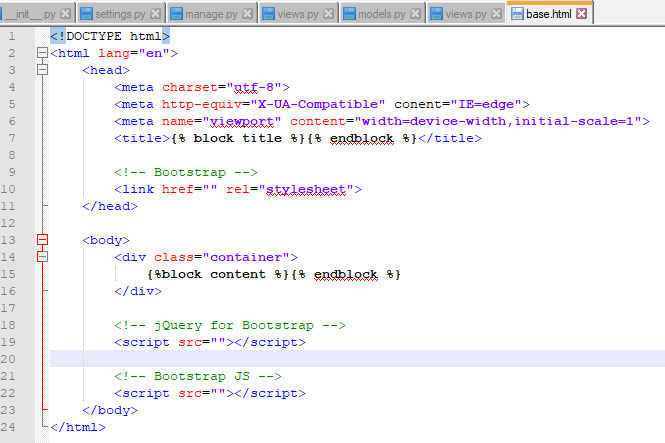


* + - Now, we check if these 2 entries have been added to the table or not in the 2nd powershell



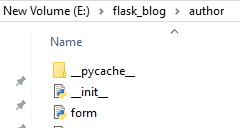
* + Now we again go to the 1st powershell:
    - To retrieve user by fullname, and get the first user you do the below. You get back data from database/table using query
    - 
    - 
    - 
    - Now we will delete the table(s) we created under author.models.py. for this first you need to do db.session.commit() just in case any pending db operations. Then you use drop\_all()
    - 
  + Then, go to the 2nd powershell – the mysql powershell to check what happened in the database
    - 
  + <http://flask-sqlalchemy.pocoo.org/2.3/queries/> for ref

1. The Base template and Bootstrap:
   * Bootstrap=frontend, flask=backend
   * we will add a base template for the look and feel of our blog i.e. frontend or html
   * Create a new folder named ‘templates’ inside E:/flask\_blog
   * In the templates folder, create base.html



* + Add the bootstrap cdn values in there—
    - So, you get bootstrap css and js from here <https://www.bootstrapcdn.com/> and bootstrap jQuery from e.g. jQuery 2.x minified <http://code.jquery.com/>
    - 
    - 
    - 

1. Introduction to WTForms:
   * We will need to install wtf forms. This library allows you to manage forms in flask. So, add flask-wtf to the requirements.txt and you install flask-wtf 
   * Wtforms has a class named Form which we will pass in as our base class
   * Create form.py in author folder-



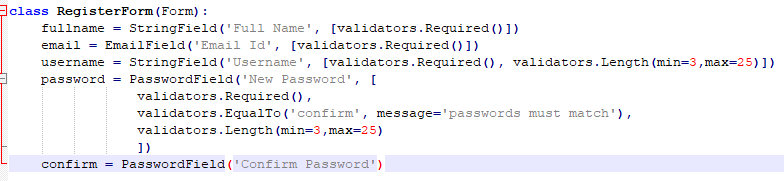
* **In form.py:**
  + - So, you import Form class from wtf forms-



* + - Then, we import validators which validate the data before sending it to database. We import stringfield, passwordfield, emailfield to enter username as a string, passwords, email ids. e.g. Emailfield will make sure that the entered data is in the form of [exrt@example.com](mailto:exrt@example.com)



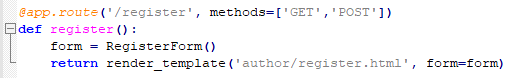
* + - Then we define a class RegisterForm and pass in the base class ‘Form’
    - Here we defined variables for all the variables we had defined in models.py
    - ‘Full Name’ is what gets displayed
    - Validators.required() sends a message if the input is not string e.g.
    - Validators.length(...) - to check if length of the input is b/w 3 and 25
    - The field for password gets displayed as ‘New Password’
    - Validators.EqualTo(...) – to check if the password entered in the variable named ‘confirm’ matches with the value in for the variable ‘password’. And, show the message if they don’t match
    - Confirm = PasswordField(‘Confirm Password’) – to provide display name to it and make it a password type field



* + **In views.py we add the route where we display this form**:
    - pass the form as RegisterForm but where is RegisterForm? so, we import it
    - since, form.py can be at other places/in other folders too, so we use author.form



* + - @app.route(...) we have defined the route of the function
    - Then, we pass the form to register.html inside author folder. This way, the form can be used in the html file. The html templates are kept in the templates folder only. Thus, we create a folder named ‘author’ in it and then, a file named ‘register.html’ inside it





* + **Now we work on the register.html inside E:/flask\_blog/templates/author**
    - So, first, we say, it extends the base template



* + - Add the title



* + - Now, we add content. We set div as ‘row’ as this makes the content display in rows



* + - md 🡪medium devices. They have 12 columns so, col-md-6 implies divide the screen into 6/12 i.e. 50-50. Offset by \* columns. So, -offset-3 implies you offset the column by 3 columns. https://v4-alpha.getbootstrap.com/layout/grid/#offsetting-columns



* + - now, we add a header



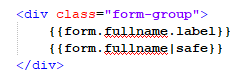
* + - now, we start to define our form-



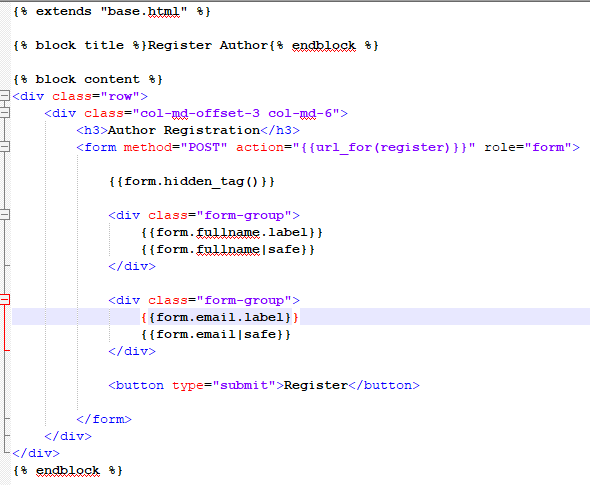
* + - To prevent csrf attacks. https://docs.spring.io/spring-security/site/docs/current/reference/html/csrf.html



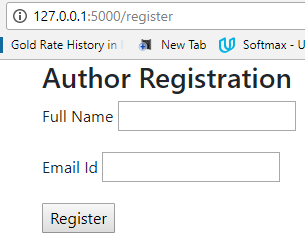
* + - Div class=”form-group” – form-group is a class that ensures proper spacing between label and value
    - Since we pass in form=form, so we access form fields by form.fullname. .label provides the label for that field. E.g. ‘Full Name’ in this case. Form.fullname is the value entered by user. We use |safe to let html know that whatever is being passed is safe



* + - **The complete register.html file will look like**:



1. Handling errors:
   * Run 



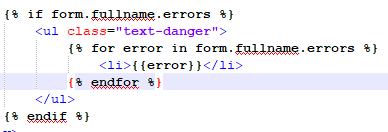
* + If all validators are satisfied, redirect user to the success page.

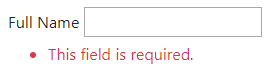


* + - Success looks like:



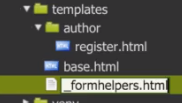
* + Since, we had html5 for EmailField, this comes with an additional message that pops up if the email doesn’t include @
  + Now, what happens if there is an error. We can display it
    - If there are any errors/ validation fails i.e. full name has not been entered, as in our case. Show the value of the error {{error}}. Class=”text-danger” is a class in bootstrap that makes the text red. The text assosciated with validators.required() might be/ is the field is required. So this message will be shown in red



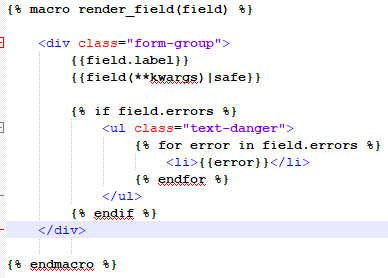


* + You will repeat the same code for email field too in register.html
    - Now, since we are repeating the code lines. We need to avoid it. This can be done using “**macros**”. DRY-don’t repeat yourself

1. Template Macros:
   * I will make a macro file in templates folder bcoz it might get used in other modules too. Their name starts with a \_. So, we create \_formhelpers.html



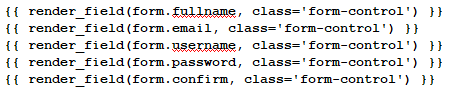
* + - So, this is the macro you make. You replace form.username with field. For the value for that specific field you use \*\*kwargs
    - \*args and \*\*kwargs are used when you are not certain about the number of variables passed into a function



* + Now, in register.html-
    - You import the macro

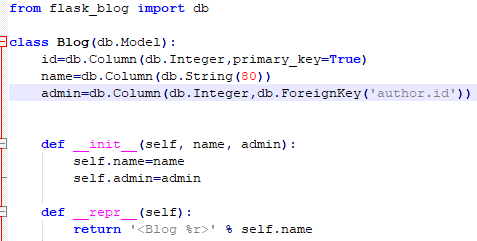


* + - Pass in the field in render\_field() and you also pass a kwargs which is class=”form-control”. Form-control set the value field width to 100% by default, and the boxes appear below the field label. Bcoz we have had field(\*\*kwargs)|safe i.e. this kwarg is being set for the input box

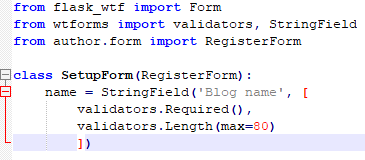


* + 

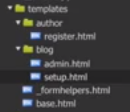
1. The Blog Model and Form:
   * Create a models.py and a form.py in the folder named ‘blog’
   * Models.py
     + A foreign key is a field in a table that matches another field of another table



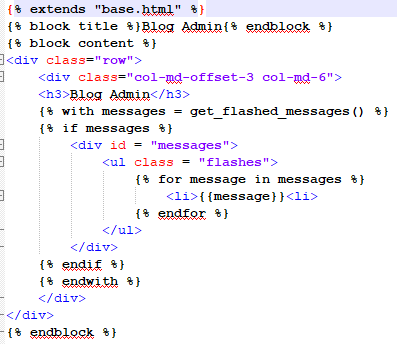
* + Form.py
    - We will keep everything, fullname, username, email, password etc and we will add just one variable that is ‘blog name’. So, we import the whole ‘RegisterForm’. We needed to pass the base class ‘Form’ into our new class ‘SetupForm’, passing ‘RegisterForm’ will do bcoz it already is a subclass of ‘Form’.



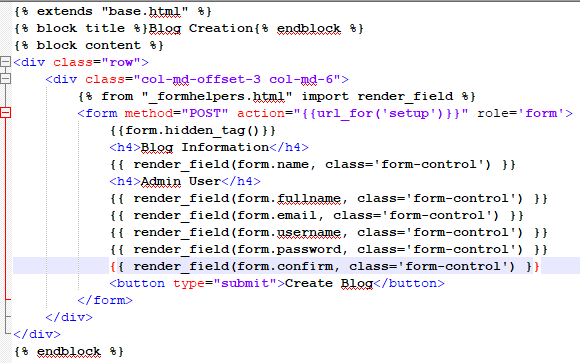
1. Blog Admin and Setup Templates:
   * Now, we create the html templates for blog. Admin is the template that gets displayed after you have setup the blog. So, we are actually more interested in making the setup template.
   * All the html templates are made under templates folder.
   * So, we create admin.html and setup.html in blog under templates



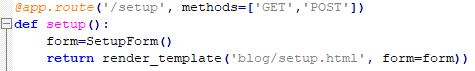
* + Admin.html:
    - We will just display a title and flash messages

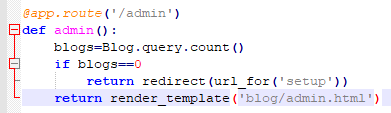
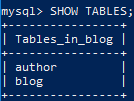


* + Setup.html:
    - We display the blog name and the other specifics, fullname, username, password etc



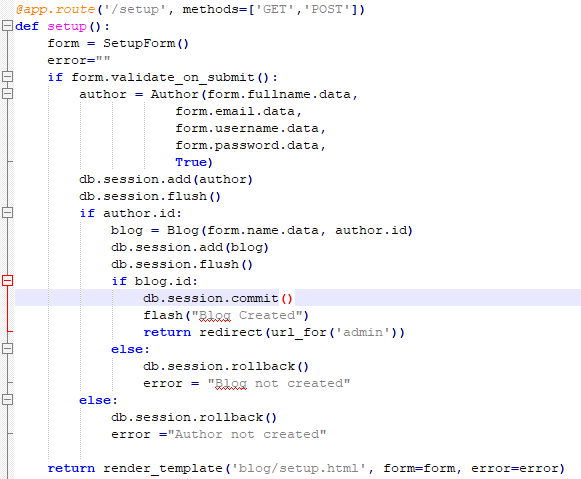
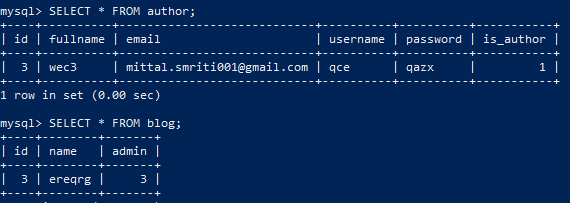
* + Now, we add routes for these templates in /blog/views.py:
    - In views.py, we add route for admin and a route for setup.
    - At /admin, first we get the count of blogs from database. If it’s zero, then redirect user to setup page. Otherwise to admin page.
    - Setup page is a form as we defined it in setup.html
    - Admin page is just a title and flask messages
    - Render\_template() always searches for a template inside ‘templates’ folder. That is why say, ‘blog/setup.html’ bcoz we have a folder named ‘blog’ inside the ‘templates’ folder
    - Since, ‘setup’ is a form. Thus, we will have methods=’GET’, ‘POST’ in it’s route and pass form inside render\_template



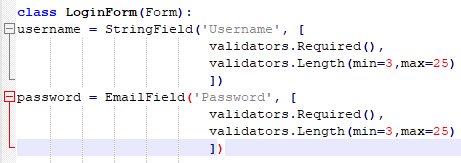
* + - Since, ‘admin’ doesn’t involve any form, so we need not use methods=’GET’,’POST’
    - Also, since we query from db- the table named ‘blog’ we will need to import Blog from blog.models. if there are no entries in the ‘blog’ table, we redirect user to the setup/ blog creation page. Otherwise go to admin template
    - 
  + Although, we are query the table named ’blog’ but, we haven’t actually created the table yet. So, we create it
    - 
    - Since, we are using author.id in our model ‘Blog’ which we import here in views.py. but the table ‘author’ hasn’t been created yet. So to be able to create table ‘blog’ we need to first create table ‘author’
    - 
    - And then, you can check if ‘blog’ was created or not when you executed the above command to create author. MOST LIKELY, if should also have been created
    - Now, open another powershell, and check the db-
    - 
    - 
    - 
    - 
  + Before we are actually able to submit this blog creation form, we need to add database operations-which we do in the next video

1. Blog Creation Database:
   * If you get this error, that means you didn’t add methods=[‘GET’, ‘POST’] in route
   * 
   * So, when user fills the form to create blog, we need to store everything in db and if nothing fails then, we direct it to admin again.
   * Also, we need to first make sure ‘author’ table has been populated correctly, without any errors; bcoz ‘blog’ table is based on it, which is assured by having an id for it. So, we check for author.id
   * db.session.flush() creates the id for the input data. So, we don’t need to commit it before we have checked that input data
   * db.session.rollback()
   * so, we commit only when author.id and blog.id has been created
   * i had not passed a Boolean, True for is\_author. That is why i got this error



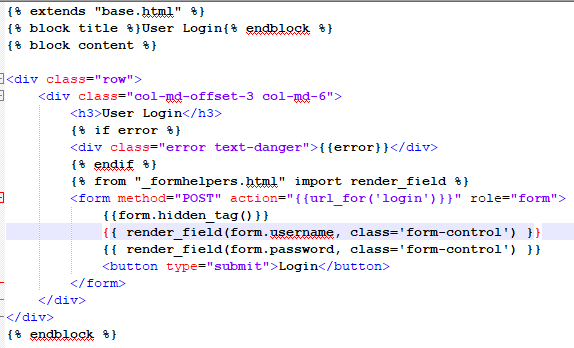
* + 
  + Output in db:
  + 

1. Author Login:
   * After registration of the user is done, we need to have a login page. and where we check if the data entered by user matches with what is stored in db (via registration)
   * So, we create a login form in forms.py under ‘author’ folder



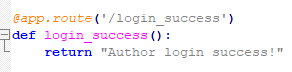
* + Next, we create a login.html template





* + Now, we get this form implemented in author.views.py
  + Since, author stores all the info, while blog stores only the blog-name and admin key. We query author
  + You store username in session, so that you can use it when one switches from say, ‘login page’ to ‘success page’ bcoz otherwise info is lost when one goes from one page to another

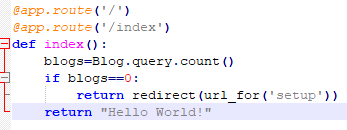




1. Our First Decorator:
   * We want to make sure that if a user is logged in only then, they are able to access a route of the blog
   * <http://flask.pocoo.org/docs/0.12/patterns/viewdecorators/>
   * https://stackoverflow.com/questions/19797709/what-is-a-self-written-decorator-like-login-required-actually-doing
   * **f is by convention** the function on which we are applying this decorator.
   * So, if user is logged in then, do show the page they have requested. Otherwise direct them to login page
   * A decorator replaces the function it is applied on. Since, it returns that function.
   * So, we do f(\*\*args,\*kwargs), bcoz we don’t the number of parameters the function will have. Different functions, different no. of params
   * Ok so- admin was our final page that gets displayed when user has created their blog. So, one should only be able to access it when they are logged in. So, we are adding a decorator to this function
   * In blog/views.py



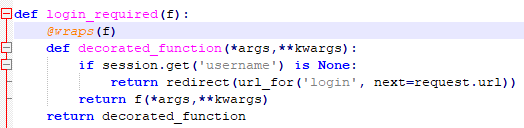
* + Also, we are moving this from admin to index or main page. so that if user is on main page, we check if they have created a blog or not. If not, then they can set it up



* + We will create a decorators.py for our decorators in E:/flask\_blog/author bcoz it is for the login required for authors
  + decorators.py:
    - from the library ‘functools’ import the function ‘wraps’
    - import session since we will use what we stored in session in previous lecture (session[‘username’]=form.username.data)



* + - the decorated\_function returns the f function, the function we applied the login\_required() decorator to. Since, function might have any no. of arguments, we passed \*args,\*\*kwargs. Otherwise, we redirect user to login page and after they have logged in, the page that user had requested initially should come upppppp
    - ALSO, we will have to modify our ‘login’ function so that it could take the ‘next’ argument



* + In blog/views.py:
    - We import the decorator we just defined



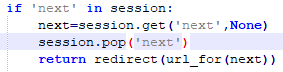
* + We now modify the ‘login’ function for it to be able to accept ‘next’. We go to author.views.py where we have defined our ‘login’ function
    - * author.views.py:
      * we import this bcoz we want to add this decorator to login\_success function too



* + - * if we do have a ‘next’ in request --- i.e. let’s say user requested ‘admin’ page which we can ONLY know during ‘GET’ so remember to put this logic before let’s say, form.validate\_on\_submit(), then, store it in session



* + - * when author has entered his login details and they match with db
      * then, we direct user to what is stored in ‘next’, not the login\_success
      * if there is no ‘next’ then direct to login\_success
      * ‘next’ – is the url for the page user has requested originally
      * Session.pop is used to remove the entry in session



* Oh this is --



Bcoz next itself is the url

* And don’t use none here –



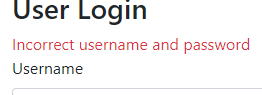
* + 
  + Now, when i requested /admin, i was directed to login page. where everything after login\ is the cookie. You can check it in developer tools ctrl+shift+i



1. Introduction to Migration:
   * Let’s first just display an error message if username/password doesn’t match with data in db. You don’t want to give away too much information to hackers so you just say both are incorrect.
   * author.views.py:
     + if author.count() is zero then this. And pass the error message into the form-



* + yayy!



1. Migrations --- Is the way you handle changes to the database. How do you keep track of all those changes to databases? How do you make changes to db, you added a new column let’s say? That too, frequently. How do you do that in a more scalable way?
   * In flask we have a package: flask-migrate. We add it to requirements.txt



* + We are going to migrate using migrate command. So, we add it to our init file
    - Go to \_\_init\_\_.py of our app
    - Pass app, db so that migrate knows what the apps is, and what the db is-





* + We are going to run it from manage.py so we need to make a change there too
    - **[[e.g. we have been running python manage.py runserver only . now we will be running python manage.py db too. Server and MigrateCommand are the inbuilt commands. We wanted to change the default settings for Server so we did it there. Runserver was to run the website, MigrateCommand is to run the database and modifications]]**
    - 
    - 
  + We will delete the tables we created manually. Migrate command will take care of it all from now onwards. ☺ **[[but, note that we WILL still need to create a database MANUALLY]]**

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database

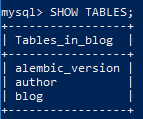
* + - 
    - 
    - In another powershell-
    - 
    - 
    - You can exit from shell to return to your virtual env
    - 
  + So, we run this command to create a repository of migrations. This will store all the versions of db or tables pushed to prod



* + First we always do ‘db migrate’, this provides a version no. to all the tables we have in our models.py. and store the version in versions folder. You can then check the files generated and see if everything is OK. Then, we ‘upgrade’ it to actually update the database



Output:



**[[we defined the database to use/ upgrade in our SETTIINGS.py file]]**

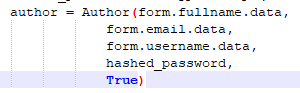
1. More Secure Passwords:
   * Add ‘py-bcrypt’ to requirements.txt file and install it. But, i get this failure message-
     +  https://github.com/Kotti/Kotti/issues/528
     + So, i instead pip install bcrypt
   * go to blog/views.py
     + 
     + generate a salt say, it’s qv3h



* + - when you use this salt to generate a hash for some password, the hash will be **salt.**nrn.34u9. so, it will be qv3h. nrn.34u9



* + - We want to store the hashed/ encrypted password in db. So, we pass this pashed\_password to author



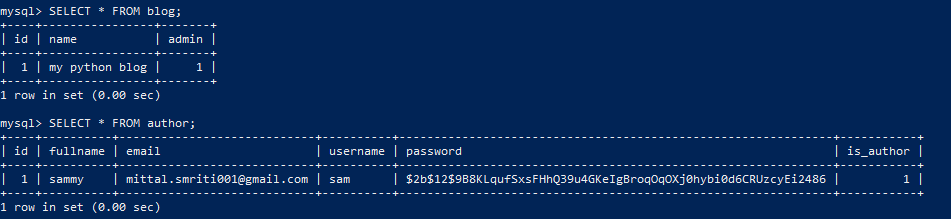
* + We will change the password length from 80 to 60 in author.models.py.



* + - but it looks like that there is a bug in a migration library so for column changes to get reflected in db, we will need to add compare\_type=True to env file in ‘migrations’ folder on line #75
    - we migrate and upgrade db bcoz we made changes to models.py
    - 
    - 
  + **[[Note: all are tables are empty as of now]]**
  + Lets’ run server and create a blog so that it gets stored in ‘author’ and ‘blog’ tables
    - 
    - Go to 127.0.0.1/setup
    - Fill in details for blog creation
    - You get this error bcoz html form sends in the Unicode strings and so, you need to first encode the password. Both input password and salt must be bytes strings—
    - 
    - To correct this, you do this---- add ---- .encode(‘utf8’)



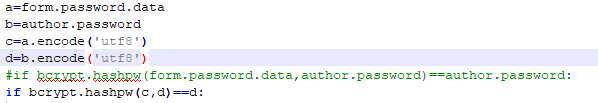
* + - Now, you have this ☺



* + We will need to update author.views.py too to check input passwords against **hashed** passwords in our db
    - So, essentially- salt is a randomly generated number. Hash is just the salt.xyz.abc. when comparing two hashpw, salt is taken out and that is what is compared. So,
    - **Hashed\_password =hashpw(registeredpw, salt) = salt.xyz.abc**
    - **Hashpw(input,hashed\_password) = hashpw(input,salt...)=hashpw just uses the salt part to calculate the hash**
    - <https://stackoverflow.com/questions/13023361/how-does-node-bcrypt-js-compare-hashed-and-plaintext-passwords-without-the-salt>



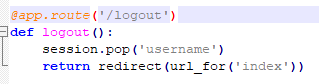
* + - I was getting this Unicode error, which got solved by-



* + Let’s create a logout function to avoid deleting the cookie each time by ctrl+shift+i which will have a delete cookie and redirect user to index page
    - On blog/admin.html page we will add a logout link



* + - Add this to author/views.py



1. Checking Is Author:
   * The idea is that people can/ will register. but only authors can’t edit/ create a blog-post/ go to admin, others can only add comments
   * Firstly, like before all this, i will make my code look better. I will use first() instead of using author[0] in author/views.py



* + Secondly, i will store the ‘is\_author’ in session. WHY? Session creates a cookie for it. This is how e.g, facebook remembers us without having us logged into every page.
    - Also, login form only has username and password. So, we will get ‘is\_author’ from author



* + In blog/views.py import session, abort. Abort is used to direct to a forbidden page

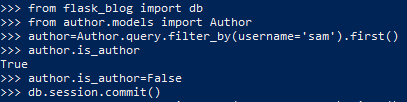


* + We will forbid admin page if ‘is\_author’ is False

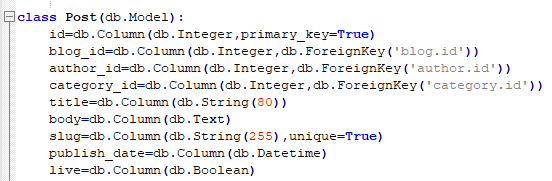


* + I will go to shell to change ‘is\_author’ to False and then, revert the changes

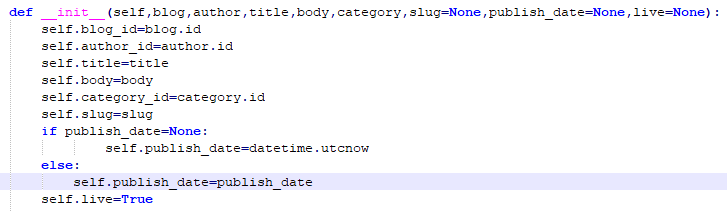




1. The Post and Category Models:
   * Let’s create models for our blog-post and category
   * blog/models.py-
   * Post:
     + We need to create a class for every table we make
     + Id—to create a numbering by default
     + Blog\_id, author\_id, category\_ids --- are being added to the table to link the post with the author identity
     + Title---is the title of the blog-post
     + Body—is the content of the blog and is kept ‘Text’
     + Slug--- is the url for the blogpost
     + Publish date ---is the date when this post gets published, if is none, then, we use the time of posting as the publish date
     + Live—we don’t delete the blogposts from db if we don’t want them to be shown, instead tag them, live=False. Then, they don’t appear on the blog



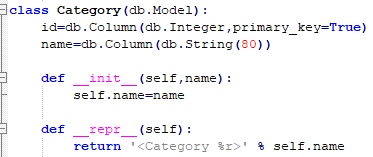
* + - We define the init for this table. We pass the blog, author, category objects rather than passing their ids bcoz blog id can change so we better search by blog than it’s id. \*args (params) comes first and then comes the \*\*kwargs (parameters with keywords)



* + - This is the representation of the table.



* + - Similarly we create the ‘Category’ class, e.g. category can be ‘python’



* + - RELATIONSHIPS: we add this to our class ‘Blog’. This doesn’t get added to the database but is of help to sqlalchemy to search for a post/similar posts when user wants them
    - This is to tell db that when i do blog.posts, then, please return back all the posts related to that blog. So, ‘Post’ is the table you connect ‘Blog’ to. Backref is the table posts is connected to, and lazy=dynamic is to tell db to not show ALL the posts, but a few

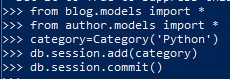


1. Post Migration and Testing:
   * Error correction: it’s DateTime, not Datetime
     + 
   * Migrating the models we created to database
     + 
     + 
   * Now, let’s start with testing the tables:
     + We import all the models we had – blog/models.py and author/models.py

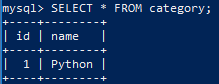




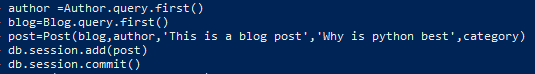
* + - We will add one table row to ‘Category’ table
      * Create the object, db.session.add(), and then, db.session.commit()



* + - * Just checking db:



* + - Now, we add a blog post-



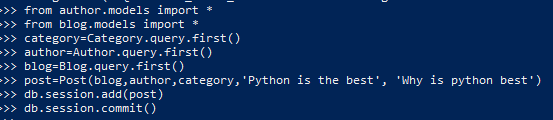
* + - * **[[NOTE: If you make corrections in code for sql/db, then you need to restart mysql..basically, like refresh db]]**
      * I was getting an error



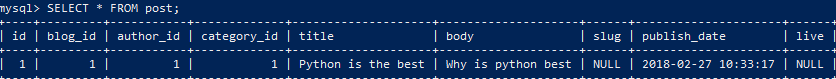
* + - * + It is datetime.utcnow(), not .utcnow

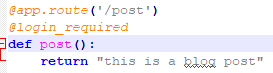
****

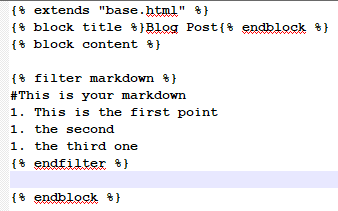
* + - * + Pass the variables in the same order in init in the order in which they are defined as columns for the table



* + - * Let’s check the database-

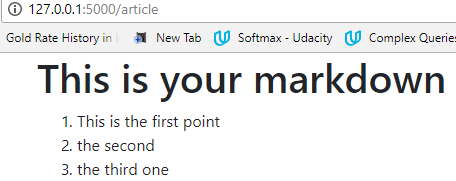


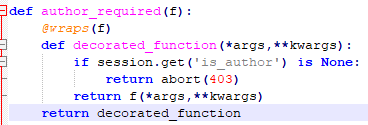
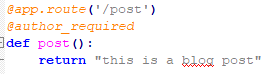
* + - * Now, let’s delete the test cases. We can delete from mysql> or using shell. If we use mysql to delete the posts we do:
        + 
      * Let’s create a route for blog posts in blog/views.py
        + 

1. Introduction to Markdown:
   * https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
   * I need two more libraries.
     + python-slugify – to generate slugs w/o having to worry abt generating them
     + flask-markdown – is a language with a set of formatting rules
     + 
   * Let’s create a placeholder for markdown, in templates we create article.html
     + article.html--
       - we use {% filter markdown %}{% endfilter %} to render markdown
       - 
   * Create a function in blog/views.py:
     + Since an article can be read by anyone, so we don’t need to add ‘login\_required’
     + 
   * We will need to add markdown in \_\_init\_\_.py too (BUT WHYYYYY? ☹)
     + 
     + 
   * 
     + I was getting some errors—
       - Bcoz i had created article.html in /templates folder itself. I moved it to templates/blog/article.html



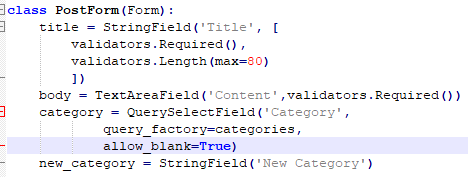
* + Output: see how the # showed H1 and all 1.s got converted to 1,2,3



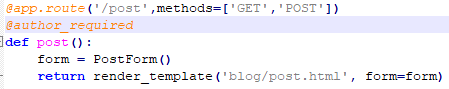
* + Now, let’s add one more decorator – for ‘is\_author’. Even if user is logged-in, i want to check if it is an author or not. If not, it can’t post blog-posts just can comment
    - 
    - Let’s add it to blog.views.py:
      * Only author can post
      * Anyone can see the article
      * 
      * 

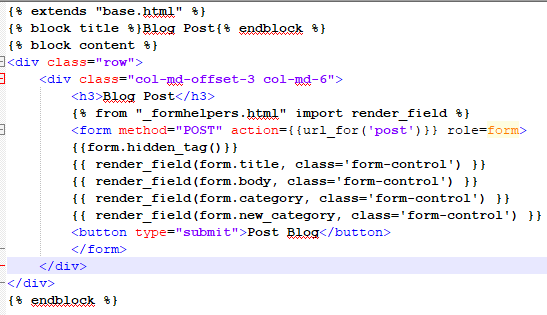
1. Blog Post Form:

Blog/form.py—

* + We need a title for your blog-post
  + Content is a text-area-field
  + Category is the tag, let’s python. We query category table to get the options available to us. Thus, the field is QuerySelectField(). Query\_factory is the parameter for the query, so we will define a function named ‘categories’ which will return the query. Allow\_blank –is if we are ok if user doesn’t select anything from this drop-down menu
  + New-category field—if the user wants to add their own new category
    - 
  + The function to query db:
    - 
  + Importing the relevant stuff:
    - 
    - 

Now, let’s define the function in blog/views.py and the template:

* + Blog/views.py:
    - 
  + Blog/post.html:
    - rows is the number of rows a text-area-field can have



* + a few errors i got while trying to run it—
    - this error implies that i have missed [..] brackets for validators.required()



* + - and this error. Which got resolved by -https://github.com/flask-admin/flask-admin/issues/1588#issuecomment-358096050 – i.e. by adding **sqlalchemy==1.2.0b3** to your requirements.txt file and getting it installed

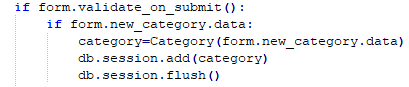
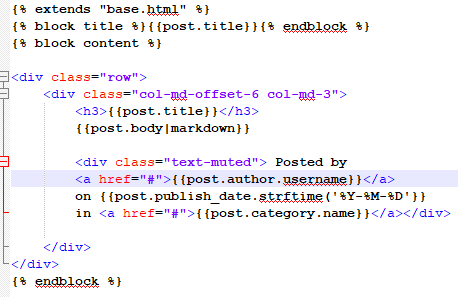


* + - you can get <Category ‘Python’> changed to Python by updating your \_\_repr\_\_ function from-



To-



1. Saving the Post to the Database:
   * We add logic to blog/views.py to post()
   * If user has input a new category of their own, then use it to create an object for Category and add it to the category table. We will commit only once, so now we just use db.session.flush() to generate an id for it
     + 
   * If user has chosen from the drop down, get the primary key and use it to get the object from category table to be used as input post Post
     + 
   * If category is blank and new\_category is also left blank, then do nothing
     + 
   * Get the blog object. Assuming that there is only one blog.
     + 
   * Get author object filter by username so that you get the logged in user
     + 
   * Get title, body
     + 
   * Create slug automatically using the title for the post
     + 
   * Create ‘post’ object with all these inputs
     + 
   * Add to db and commit
     + 
   * All this we did, if form was validated. Thus, we will redirect user to ‘admin’ page
     + 
   * Otherwise keep showing the post template
     + 
2. The Article View:
   * We will render an article using ‘/article’. We pass in the ‘slug’ to it so as to let it know the article we want. Also, remember, slug is unique. It works perfectly fine as an id for an article.
     + 
   * We search the post by the slug
     + 
   * And we pass the post to the template
     + 
   * Now, we will need to modify the ‘blog/article.html’ template
     + First we show the title. Then, the content. Then, we have the footer.
     + 
     + Also, here date should be defined like this—
       - 
   * NOTICE THAT IN THE ABOVE, you had just passed the ‘post’ object, but you are able to get data from ‘author’ and ‘category’ tables as well.
     + e.g. we jump from post to author, post to category. This is called linking. And is done via relationships. So, it loads many posts in our case
     + http://flask-sqlalchemy.pocoo.org/2.3/models/
     + since, we can have same category name for many posts, thus--



* + - we can have one author posting many posts, thus—



* + We will also modify the ‘post’ function. So, once the user has filled out the form correctly, and everything is stored in database, we redirect them to the article they just published.
    - 
  + You can now runserver and go to /post to post a new post and it will direct you to your new article with a slug

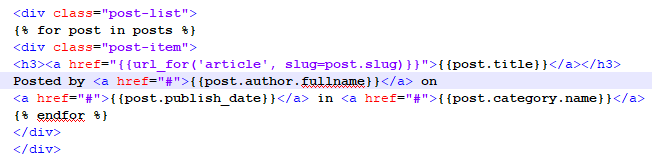
1. List Articles View:
   * Show list of articles on the index page of the blog
   * Firstly we need to pop the session for ‘is\_author’ too when we logout. So that when people logout, that cookie is also erased
     + 

In blog/views.py:

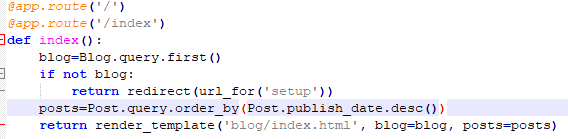
* + We will do a query, get all posts ordered by their publish-date and pass these posts to the admin template
    - 

In article.html:

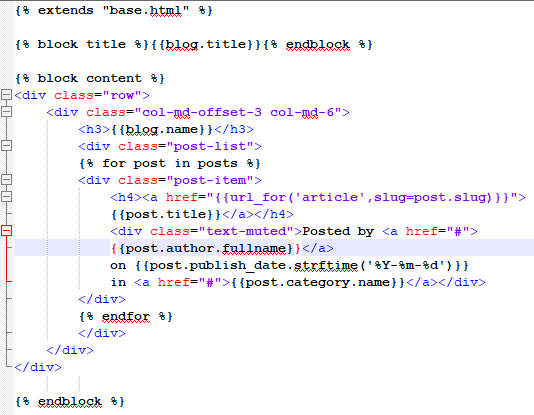
* + Posts are shown as list items. First we show the post-title which is hyperlinked to the actual article. Followed by the fullname, publish-date, category for each of the posts

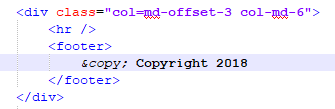


* + I will do the similar thing for the index page
    - Assuming that we only have one blog. So, if there is no blog, then, redirect to setup a blog. Otherwise, redirect to index.html

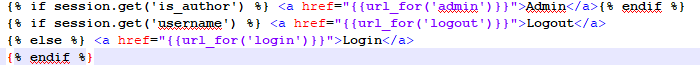


* + - let’s make the index.html:
      * we display a list of post with items as -- post-title and posted by ...



* + Next, we add footer to every page. so for that, we will add it to base.html template. Add this to the container class itself, to align it with the block content. i.e. the post-titles
    - 

1. Creating a footer:
   * In base.html --- We also add a hyperlink for logout if loggedin, login if not-loggedin already. If is\_author, then option to go to admin. we have access to session without having to import it in the html pages



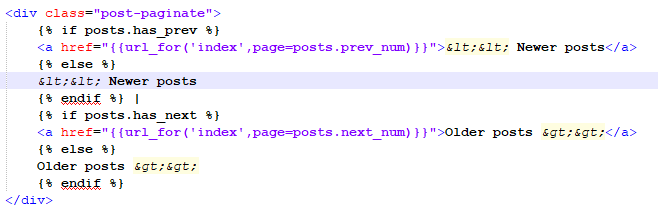
* + Also, in ‘login’ function, we redirect to ‘index’ page not this, login-success page now
    - 

1. Pagination:
   * Add this to blog/views.py since, for the time being we just want to be applied to this view. Otherwise, we could have added it to settings.py
     + 
   * For ‘index’ page: add pagination() to posts
     + Page=1 -- if user calls the ‘index’ page w/o any page no. we will use page=1, otherwise it will use what user has passed as the page in ‘/index/<int:page>’
     + False – if the page passed doesn’t exist then don’t return a 404



In index.html:

* + Now, we have a list of posts, each list item has 5 posts in it
    - 
  + If prev page exists, then, display ‘newer posts’ in hyperlink, otherwise, no hyperlink to ‘newer posts’. Similarly for next page



1. Setting Up Flask Uploads:
   * Install flask-uploads, it is used to upload a file/image
     + In requirements.txt add ‘Flask-Uploads==0.2.0’ and –
     + 
   * We will add a path to settings.py where images go when a user upload some images
     + Uploaded image destination in Settings.py:-



* + - Add a new folder, ‘static’ in flask\_blog and add a new folder, ‘images’ to ‘static’ folder
    - Add this to settings.py file, url for the images
  + \_\_init\_\_.py
    - An ‘upload set’ is a single collection of files. Since, files can by any type of files. So, we specify ‘IMAGES’
    - Configure\_uploads is used to configure the uploaded set, like all the files uploaded for that app. You pass the app instance, which is ‘app’ and the instance of uploaded set





* + let’s add a column to our table, ‘Post’ where we will store the name of the file uploaded. So, it will be a string bcoz filename
    - 
    - 
    - \_\_init\_\_





* + - We will add a property to get the image url.
      * https://www.journaldev.com/14893/python-property-decorator
      * We use @property, the inbuilt decorator in python when we don’t want to add a new parameter to the existing definition that is affected by the change in another parameter
      * Uploaded\_images.url() gets the url of the filename we pass in. Since, self.image stores the filename, we pass in self.image
      * 
  + Now, let’s migrate it to db.
    - When i try to migrate i get the below error

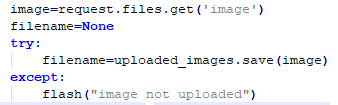


* + - Fixed by removing ‘module’ import in E:\flask\_blog\venv\Lib\site-packages\flask\_uploads.py
    - <https://github.com/maxcountryman/flask-uploads/issues/1>
    - 
    - 
  + Till now, we have stored the image filename in database

1. Adding Images to Blog Posts:
   * We have views, forms.py and html template to modify
   * In the form.py—we will add imageField to get the image file from the user
   * In the post.html – this template will just be used to place this imagefield
   * In the view.py –
     + if form.validate\_on\_submit, then, we will use .save() and pass the uploaded file to get the filename <https://pythonhosted.org/Flask-Uploads/#flaskext.uploads.UploadSet.save>
     + So, we will pass this filename into the ‘Post’ object to get it stored in db
   * So, here we go---
     + Blog/form.py
       - 
       - Add this before the title in ‘PostForm’. Type of files allowed are png,jpg. And the error message that get displayed if not png/jpg



* + - Post.html
      * You need to pass multipart/form-data. There is a separator which tells if from this part to this part is a file, other part is other fields
      * 
      * 
    - Blog/views.py
      * Get the file uploaded by user
      * A file is get via request.files.get(). Pass in the name of the field on the form i.e. ‘image’
      * And get the filename via .save(the uploaded file)



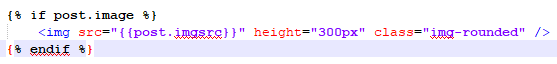
* + - * And pass this filename to post object
      * You will also need to import, request and uploaded\_images





* + Now, let’s try to run this:
  + 
    - I was getting this error – ‘TypeError 'tuple' object is not callable’ just after if form.validate\_on\_submit()
    - Fixed this by—removing an extra ‘(‘ for validators of image in blog/forms.py
    - 
  + Now, we do see an image in /static folder and image name saved in db
  + Next, we work on how to display this image on the actual post/article

1. Displaying the Image in the Article:
   * Since, it goes to article.html after post.html. and we pass in the slug to get the post which is passed into the article.html template
   * So, in article.html you add:



* + The image wasn’t getting displayed. So i checked the source-code. Which looked like this



* + - I fixed it by adding a ‘/’ in settings path for url

1. Deleting an Article:
   * On every article, we want to display a hyperlinked ‘delete’ below the title. He can only delete a post if he is an author!!
   * So, we go to blog/article.html, and add this line—



* + So, if a user clicks on ‘delete’, it is taken to ‘/delete/post\_id’ and we get which post to delete by filtering the posts by the ids.



* + Admin can see the deleted posts too bcoz he can choose make them live again as he is an author, but index page can only have live posts.
  + So, in ‘index’ in blog/views.py you filter\_by live



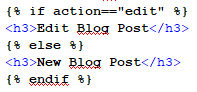
* + You will just display ‘deleted’ in front of the post on admin page



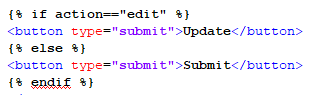
1. Editing an Article:
   * Add this to blog/article.html—



* + We edit the blog/post.html to –
    - Make it display edit blog post as the header if we are editing



* + - Update button, instead of submit



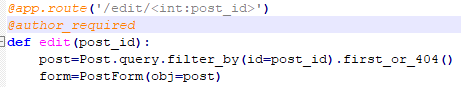
* + - Action it on ‘edit’ function if action==edit



* + - Display the original image on the top if there was any image in the original blog post



* + Now, let’s update blog/views.py
    - We add ‘edit’ function
    - Get the post from the post-id that was passed
    - **Pass the post object you got into the form, so that the PostForm loads the original input and you can just edit it and ‘update’**



* + - Return render\_template



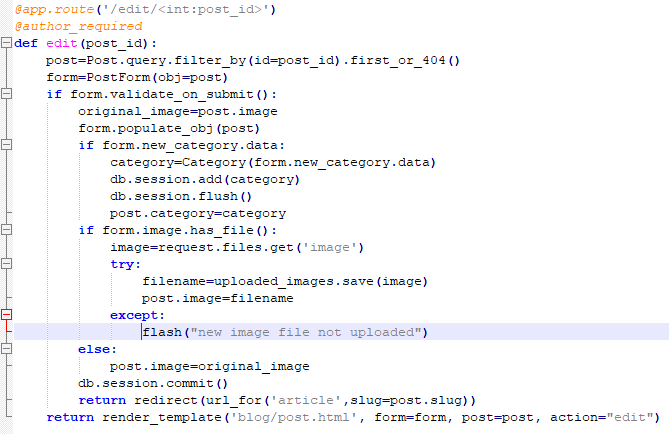
* + - Once, the user has updated the form, we get all the info and store in db



* + - **So, now we have got the updated form. That is why we are able to do form.validate. (bcoz e.g. we always send an empty form, form.validate will be satisfied only when for has been filled, implying this form is the updated form. But, we need to update our post object so that we can put it into db. S, first we wil get the image from the original post object incase user doesn’t add any image file this but, wanted to keep the original image file. And then, we will update the post object using the updated data in form. We will need to get the original image first, so as to avoid the new post object w/o any image removing the original image**

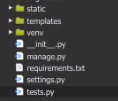


* + - **Now, we are actually ready to commit this new-updated post object. But, before that, we will need to check for image and update the Category table if user has added a new category. So we do that now. If there is a new image that user has added, then we will need to store it in the static folder**



* + I was getting the error ‘method not allowed’
    - 

1. Introduction to Unit Testing:
   * Don’t test things that are external to your app, e.g. session setting. Flask checks/ have it’s own tests to check it.
   * Create tests.py and we use the flask library unittest for tests



* + Add the path to the directory my file is in <https://stackoverflow.com/questions/21005822/what-does-os-path-abspathos-path-joinos-path-dirname-file-os-path-pardir>



* + Import models



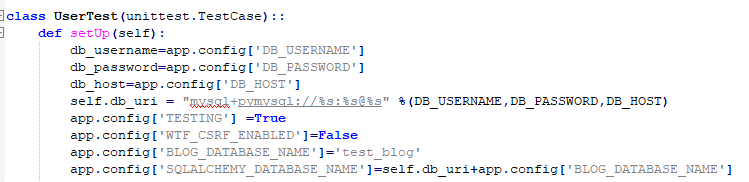
* + Import unittest library, and sqlalchemy



* + Import app, db



* + setUp --
    - It will create a temporary db, so we don’t use the production db. So we can destroy this db as the test is done via tearDown. we will setup the connection to a temporary db the same way, we did it settings.py
    - All inputs will remain same, except the db name
    - CSRF token is only enabled when you are running it on a web browser. So, we turn it off right now



* + - Engine is a sqlalchemy instance that we can talk to



* + - Then, we connect to that engine



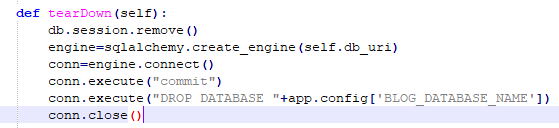
* + - Then, we create db with that blog database name



* + - We do db.create\_all() so that all the tables are created

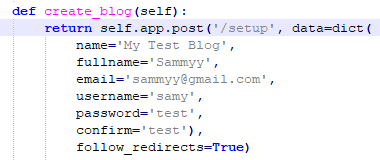


* + tearDown –delete the database we created

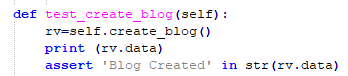


* + now, lets create the test to test if blog is created or not

1. Create Blog Test:
   * You post the data as dict to the url for setup. And, set follow\_redirects as true, so that if setup works fine, it goes on to admin page. meaning setup is completed



* + Then, we create the test for create\_blog.
    - Unittest will search for test\_ to know and run the test.
    - We will catch the return value (rv) of the ‘create\_blog’ function.
    - And how do i know it returns ‘Blog Created’?
    - we pass a flash message when setup is complete and this flash appears on the admin page. implying that setup is done and user has been redirected to ‘admin’ page. – check admin.html



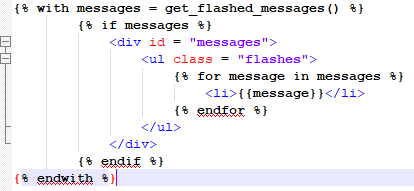
* + We check if name=main – it is kind of checking if we are calling it from the terminal



* + 



* + Let’s create a separate file for flash-messages and include it wherever needed
    - \_flashmessages.html



* + - And add this to admin.html

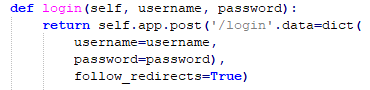


* + But, we always send True for ‘is\_author’ in our setup function. And admin requires @author\_required. So, this should work normally.. let’s check the db. Oh! I can’t now. Bcoz it has created and destroyed it already. So, let’s print rv.data inside test\_create\_blog()

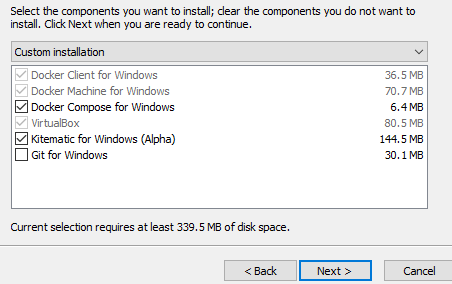


* + Oh this should be print (rv.data). but now it has already created the db and got this error while at checking assertion. So, when we run it again, we will get an error that db already exists. Cant create a duplicate copy of it. That is why, you will need to manually delete/drop the db and re-run...NO NO NO test failed. So creating and dropping of db is done. So this is not the actual problem. THE PROBLEM WAS THAT- i had set email, user and fullname to be unique. So every time it creates the db and destroys it. But the enteries inside the ‘author’ and ‘blog’ tables keep accumulating. So, setting different user/fullname/email worked!!!!
  + Fix this---

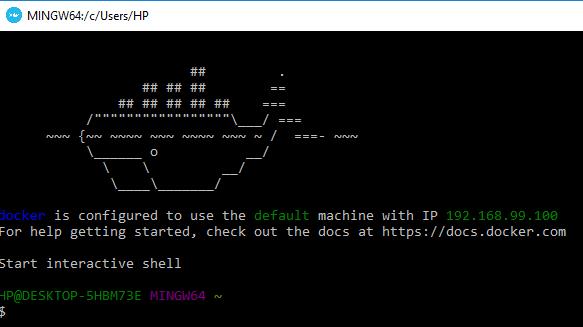


1. User Tests:
   * Instead of being able to call a function with one-specific values. Let’s create a generic login-logout func. We use post, bcoz we need to post values to the form. We used get for logout func bcoz we don’t need to input there anything
     + 
     + 
   * Create test for it. We need to use ‘assert’ when we are testing. I have added a flash() inside login function which will get shown on index page when user gets redirected to it
     + 

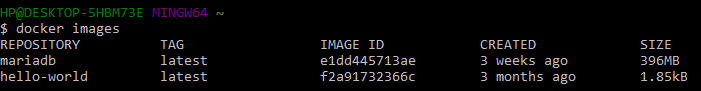
**BONUS SECTION ----- RUNNING OUR FLASK APPLICATION WITH DOCKER**

1. Docker Introduction:
   * e.g. cloud9 is a development environment, same way docker is. If your app works with docker locally, it will work on your target environment e.g. google cloud etc. Apps in docker use the machine’s OS. Unlike that in cloud9 or notepad++ which use a guest OS/ virtual env for every app. READ MORE PLEASE!
2. Docker Installation:
   * **DON’T DO THIS! FIRST CHANGE THE LOCATION FOR VIRTUAL HARD DISK FILES TO BE STORED. THAT IS WHERE THE DOCKER SHOULD BE INSTALLED. OTHERWISE IT MIGHT NOT RUN BCOZ OF THE NOT ENOUGH SPACE PROBLEM**
   * When you install it, a virtual box is created
   * <https://docs.docker.com/docker-for-windows/install/>
   * .exe asks you to sign out of windows. READ ON WHY?
   * Hyper-V and containers features will need to be enabled. It will ask you if it can enable them. Will require restarting your laptop. READ MORE ON IT
   * <https://github.com/docker/for-win/issues/185>
   * Change the location-- <https://blogs.technet.microsoft.com/canitpro/2012/09/11/default-file-locations-in-hyper-v/>
     + 
     + And so on
   * Install--
     + When you install it, a virtual box is created
     + <https://docs.docker.com/docker-for-windows/install/>
     + .exe asks you to sign out of windows. READ ON WHY?
     + Hyper-V and containers features will need to be enabled. It will ask you if it can enable them. Will require restarting your laptop. READ MORE ON IT
   * STILL NOT WORKING DUE TO MEMORY SPACE ISSUE!!!!
   * **Install docker toolbox instead**. It is used when you get memory issues while installing docker for windows. Ok. On it. <https://docs.docker.com/toolbox/toolbox_install_windows/>
   * Right click, and run as administrator. It will install all this-
     + 
   * Click on the docker quickstarter –

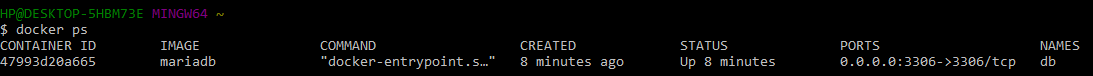
C:\Users\HP\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Docker

* + Getting this error –
    - Error with pre-create check: "This computer is running Hyper-V. VirtualBox won't boot a 64bits VM when Hyper-V is activated. Either use Hyper-V as a driver, or disable the Hyper-V hypervisor. (To skip this check, use --virtualbox-no-vtx-check)
  + Fix –
  + <https://jayvilalta.com/blog/2016/04/28/installing-docker-toolbox-on-windows-with-hyper-v-installed/>
  + Go to docker toolbox folder E:\VirtualHardDiskFiles\Docker Toolbox
  + And open start.sh file inside it in an editor (notepad++) as admin
  + Add this --virtualbox-no-vtx-check to make #60 look like this -- "${DOCKER\_MACHINE}" create -d virtualbox --virtualbox-no-vtx-check $PROXY\_ENV "${VM}"
  + Save and close the file
  + Now, again open the quickstart terminal
  + Now, as i start it. MY LAPTOP SHUTS DOWN BY ITSELF!!
  + So, i will try switching off hyper-V instead <https://stackoverflow.com/questions/36885985/cannot-start-docker-after-installation-on-windows>
  + Search for windows features in start menu
  + <http://www.poweronplatforms.com/enable-disable-hyper-v-windows-10-8/>
  + Uncheck hyper-V and restart your pc
  + C:\Users\HP\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\
  + Run the docker quickstart as admin
  + You have it! ☺
    - 
  + Docker commands start with ‘docker’. Run is to run the container. ‘hello-world’ is an inbuilt container. We will run this command to see if everything’s ok
    - 
  + To check the images (e.g. hello-world) that were installed(?) or run is all good/not
    - 

1. Docker Images and Containers:
   * Images are blueprints for containers
   * We will use mariadb (vs mysql)
   * 
     + Docker run – running docker
     + What will be the name of the image?db -- name db
     + –e is for environment variables. So, we set the password for the db as ‘test’ MYSQL\_ROOT\_PASSWORD=test
     + –d is for daemon. i.e. keep it running, don’t let it die down the way hello-world dies.
     + We expose the port –p 3306:3306
     + The name of the image is mariadb
   * Size of mariadb image is very small as compared to mysql



* + To check if the container is running or not : **docker ps**



* + This container is an instance of class mariadb

1. Interacting with the MySQL container:
   * Image is an executable which contains everything needed to run that application-code, config files, env variables
   * Container is an instance of the image, taking no more memory than any other executable, making it lightweight
   * Why do you need containers?
     + <https://www.cio.com/article/2924995/software/what-are-containers-and-why-do-you-need-them.html>



* + - How do containers resolve the above problem?
    - What other benefits compared to virtual machines?
      * Size is ~100MB while size of VM is ~10GB
      * Applications can be open almost instantaneously
  + 3306 is the default MySql port so that all the apps can interact with it
  + Now, we are using the mariaDB container. We can’t use mysql just like that. So, we will need to connect to mysql using the port 3306 via this mariadb container
  + We create a new container inside mariaDB container. We link it with mysql-client (name it as mysql)
  + This new container will expose us to the interactive shell of mysql. It will have all the libraries of mysql. Will just connect us to mysql
  + We use rm – meaning remove the container once you have exit using it
  + Mysql\_root\_password was set ‘test’
  + And host is ‘mysql’
  + So, we have the image ‘mariadb’. We will create an instance for it, i.e. the container
    - I ran this—



* + - I got this error—



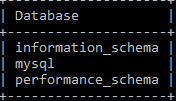
* + - So, i needed to run the container (db) for mariaDB –



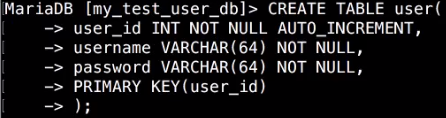
* + - And re-run the mysql-client, linking command and you get the interactive shell just like mysql –



* + - So, let’s play with it a bit –
      * 
        + Output –



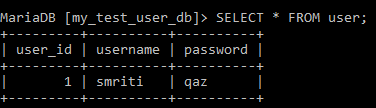
* + - * 
      * 
      * Now, let’s create a table inside this db



* + - * Insert values into it—



* + - * Show the table—



* + - * Now, exit;



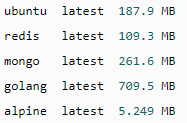
* + - * Now, we check if the mysql-client container is still working or not. No, it’s not. Bcoz of the –rm flag we used there



1. Preparing Our Dockerfile:
   * We want to create an image for our flask-blog. Which we will connect to mariaDB
   * The dockerfile will be a blueprint for the image
   * Now, change directory to E:/flask\_blog. So that we are inside ‘flask\_blog’ folder



* + Next we download ‘ATOM’
    - Go to <https://atom.io/>
    - Download the .exe for windows and install it
    - Why atom? To create a dockerfile and docker image for flask\_blog (EXPLAIN)
    - Atom is a text editor similar to cloud9. It shows all the files of a folder in a tree-structure
  + You use a parent image, the image our image will be based on. Each subsequent declaration in the dockerline modifies it.
  + *That all depends on preference. Lots of people are comfortable with Debian-based distros so debian or ubuntu are frequently recommended. alpine base is usually fewer MBs to push around but has a lot of caveats (e.g., musl libc) and is harder to get started with. Long story short, if you like 'apt-get', use ubuntu or debian. If you like 'yum' use centos. If you want smaller images use alpine but be prepared for it to be a timesink.*

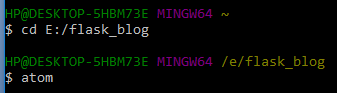


* + - You could use centos –operating system or ubuntu. If you like ‘yum’ then use ‘centos’ if you like ‘apt-get’ then use ‘ubuntu’
    - Yum, apt-get are python package managers- they help install python packages
  + The base operating system, a type of linux and the version used
    - 
  + A few python packages we will need to install. Which cloud9 had inbuilt, but for ATOM we will need to install them using python package manager –‘yum’
  + We need python-devel to get headerfiles to build python extentions and
  + mysql-devel to get headerfiles and libraries to develop mysql client applications
  + mysql-connector-python to connect python with mysql
  + gcc is a compiler



* + Then we install pip
    - 
  + We make a new directory. /opt/ is used to put files on server side
    - 
  + And it makes work-directory as that directory. So from this point on, any operations are going to be pointed to this dir
    - 
  + Add requirements.txt from our local computer to that workdir
    - 
  + Since, the whole server is yours, unlike in virtual environments. So i install this server-wide
    - 
  + Copy all the files you are running these commands from, to server-side
    - 

1. Creating the flask-blog container:
   * Cd E:/flask\_blog
   * atom
   * But, then you get this error
   * 
   * https://github.com/bradtraversy/bs4starter/issues/2
   * so, you add ‘E:/atom’ folder containing the atom exe to PATH in env variables
   * this didn’t really work-
   * open ATOM, go to packages, then, view settings, then, install packages
   * https://github.com/dformoso/docker-atom-tutorial
   * search for ‘platformio’ and install platformio-ide-terminal at C:\Users\HP\.atom\packages.
   * Once you see this package installed at the specified path
   * Reopen the docker and cd E:/flask\_blog and then enter atom
   * This will open ATOM with the working dir as E:/flask\_blog



* + Then, create a new file, ‘Dockerfile’ by right-clicking on flask\_blog in the tree
  + Copy-paste the content for dockerfile
  + Now, go back to docker quickstart terminal
    - We do-
    - 
    - -t flask\_blog is for the name of the image for our application
    - . is to tell it to look for a dockerfile in the current directory
    - And build will build the dockfile. Step by step
    - Get this error at yum install. So, my centos image has been downloaded



* + - * Then based on this solution <https://stackoverflow.com/questions/36630718/docker-as-a-builder-cant-install-systemd-header-files> add this line to your dockerfile



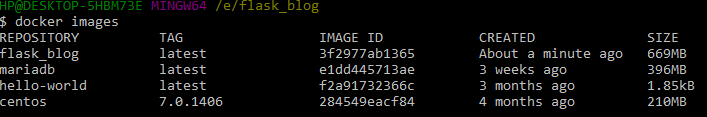
* + - And rerun—



* + - * I got this btw—



* + - After successfully creating image. You can check if a new image as been created or not— ‘docker images’



* + Now, we need to restart the database container. db – we will need it.
  + db is the container name
    - 
  + Now, we create/ run the container for image ‘flask\_blog’
    - Docker run
    - –id – interactive shell and run as daemon meaning, it should continue running in the background
    - –p – is the port to be exposed. Flask apps use port 5000
    - –v – the mount. So we want to make changes inside ATOM locally and want them to be displayed inside the container. /opt/flask\_blog is the directory inside container
    - –name is the name of the container. which is ‘blog’
    - –link you link to link your current container with mariadb. db is the container for mariadb and we will call it ‘mysql’ inside our ‘blog’ container
    - Flask\_blog is the image name we are making container for
    - Bash is used to let it know that we will connect to it



* + - I was getting an error when i had **E:/flask\_blog**. Modified it to **E\flask\_blog**
    - We do this to check if the new container is working
      * 

1. Editing Files:
   * Now, how to get inside the container?
   * 
   * Now, let’s modify the settings.py file in ATOM and save it. changes should get reflected in container
   * Now, ping mysql to see if we can connect to it or not
     + 