**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **26-May-2020** | **Name:** | **Russell D’souza** |
| **Course:** | **DSP** | **USN:** | **4AL15EC023** |
| **Topic:** | **Fourier series & gibbs phenomenon using python. Fourier transformation, its derivative And its convolution Intuition of Fourier transform and Laplace transform Laplace transform of first order. Implementation of lt using mat lab code. Application of z transformation Find z transformation using mat lab code** | **Semester & Section:** | **8th sem & ‘A’ section** |
| **Github Repository:** | **Russell1005** |  |  |

|  |
| --- |
| **MORNING SESSION DETAILS** |
| **Image of session** |

**DAILY ASSESSMENT FORMAT**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Date:** | **26-5-2020** | **Name:** | **Russell D’souza** | | **Course:** | **Python programming** | **USN:** | **4AL15EC023** | | **Topic:** | **Build a personal website with python and Flask** | **Semester & Section:** | **8th A** | | **Github Repository:** | **Russell1005** |  |  | |
|  |
|  |
| **AFTERNOON SESSION DETAILS** | |
| **Image of session** | |

# Step #1: Start with Flask

I was new to [Flask](http://flask.pocoo.org/) so I followed a tutorial to get started (I would recommend [this one](https://youtu.be/MwZwr5Tvyxo) if you’re interested), but the general idea is to create a simple myapp.py (you can name this file however you like) which should look something like this:

|  |
| --- |
| from flask import Flask, render\_template |
|  | app = Flask(\_\_name\_\_) |
|  |  |
|  |  |
|  | @app.route("/") |
|  | def home(): |
|  | return render\_template('home.html') |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | app.run(debug=True) |

You also want to create a very basic home.html template:

|  |
| --- |
| <!DOCTYPE html> |
|  | <html> |
|  | <head></head> |
|  | <body> |
|  | <div> |
|  | <h1>Jane Doe</h1> |
|  | <p>Hello, I'm a cereal killer.</p> |
|  | </div> |
|  | </body> |
|  | </html> |

You can (and probably should) make it fancier, i.e. pass variables via render\_template and use them in the html file, reference an external CSS file instead of some controversial in-line CSS, and so on. However, if you go ahead and run python myapp.py, you should see that the website is up and running on your localhost — if this is too minimal, you can always go back and improve it incrementally later.

# Step #2: Static-ify with Frozen-Flask

If you think once again about it, what we want to build is somewhere where to display our bio and some links, and really nothing more. In other words, our end goal is to make a [static website](https://en.wikipedia.org/wiki/Static_web_page), and we can leverage [Frozen-Flask](https://pythonhosted.org/Frozen-Flask/) to freeze a Flask application into a set of static files.

Frozen-Flask is only about deployment: instead of installing Python and Flask on your server, you can use Frozen-Flask to freeze your application and only have static html files on your server. **It’s magic!**(Ok, not really magic, but that’s quite impressive, right?)

I found [this](http://www.dougalmatthews.com/2017/Jan/13/static-websites-with-flask/) article quite clear and helpful, but in a nutshell, after installing the Frozen-Flask package, go ahead and create a freeze.py that looks something like this (again, you can name this file however you like):

|  |
| --- |
| from flask\_frozen import Freezer |
|  | from myapp import app |
|  |  |
|  | freezer = Freezer(app) |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | freezer.freeze() |

Running this script via python freeze.py will create a build directory containing your application’s content, frozen into static files, which is exactly what we want.

# Step #3: Deploy with Netlify

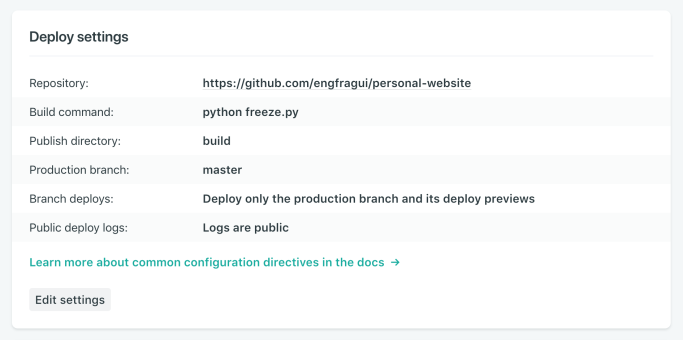
The final step is to host the website somewhere on the internet — and not only on our localhost. To do so, we can leverage [Netlify](https://www.netlify.com/), whose main purpose is indeed to build, deploy, and manage modern web projects.

As suggested by the [official documentation](https://www.netlify.com/docs/), it’s usually a good idea to create a requirements.txt and a runtime.txt. The first will tell Netlify which packages are needed by your application, and can be created via:

pip freeze > requirements.txt

The second file is simply a record of the Python version that you want to run (I went with 3.6).

When you’re ready to deploy your website, head over to the Netlify website, sign up, and choose “create a new site from GitHub”. The Deploy settings are especially interesting and might not be the most straightforward ones to fill out:



Deploy settings that I used for my personal website

Pay attention in particular to:

* Build command, a.k.a. the command we want Netlify to run. In our case, since we want to simply freeze our html so that it creates a static website for us, let’s set it to python freeze.py.
* Public directory, a.k.a. the directory with the static content of the website. We set this to build because that’s where Frozen-Flask puts all the static content after generating it.

After saving these settings, you can finally trigger the build of your website, and ta-da! After few seconds, your personal page will be available at <site-name>.netlify.com, but if you want, you can also buy your own domain if you want to make it look even fancier.

**CSS STYLING:**

body {

margin: 0;

padding: 0;

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

color: #060;

}

/\*

\* Formatting the header area

\*/

header {

background-color: #DFB887;

height: 35px;

width: 100%;

opacity: .9;

margin-bottom: 10px;

}

header h1.logo {

margin: 0;

font-size: 1.7em;

color: #fff;

text-transform: uppercase;

float: left;

}

header h1.logo:hover {

color: #fff;

text-decoration: none;

}

/\*

\* Center the body content

\*/

.container {

width: 1200px;

margin: 0 auto;

}

div.home {

padding: 10px 0 30px 0;

background-color: #E6E6FA;

-webkit-border-radius: 6px;

-moz-border-radius: 6px;

border-radius: 6px;

}

div.about {

padding: 10px 0 30px 0;

background-color: #E6E6FA;

-webkit-border-radius: 6px;

-moz-border-radius: 6px;

border-radius: 6px;

}

h2 {

font-size: 3em;

margin-top: 40px;

text-align: center;

letter-spacing: -2px;

}

h3 {

font-size: 1.7em;

font-weight: 100;

margin-top: 30px;

text-align: center;

letter-spacing: -1px;

color: #999;

}

.menu {

float: right;

margin-top: 8px;

}

.menu li {

display: inline;

}

.menu li + li {

margin-left: 35px;

}

.menu li a {

color: #444;

text-decoration: none;

}

**Steps to deploy a static Flask website to Heroku**

1. Create an account on www.heroku.com if you don't have one already.

2. Download and install Heroku Toolbelt from https://devcenter.heroku.com/articles/heroku-cli

3. Install gunicorn with "pip install gunicorn". Make sure you're using pip from your virtual environment if you have one.

4. Create a requirement.txt file in the main app directory where the main Python app file is located. You can create that file by running "pip freeze > requirements.txt" in the command line. Make sure you're using pip from your virtual environment if you have one. The requirement.txt file should now contain a list of Python packages.

5. Create a file named "Procfile" in the main app directory. The file should not contain any extension. Then type in this line inside: "web: gunicorn script1:app" where "script1" should be replaced with the name of your Python script and "app" with the name of the variable holding your Flask app.

6. Create a runtime.txt file in the main app directory and type "python-3.5.1" inside.

If you're using Python 2, you may want to type in "python-2.7.11" instead.

7. Open your computer terminal/command line to point to the directory where the Python file containing your app code is located.

8. Using the terminal, log in to Heroku with "heroku login"

9. Enter your Heroku email address

10. Enter your Heroku password

11. Create a new Heroku app with "heroku create myawesomeappname"

17. Initialize a local git repository with "git init"

18. Add your local application files to git with "git add ."

19. Tell git your email address with "git config --global user.email "myemail@hotmail.com"". Make sure the email address is inside quotes here.

20. Tell git your username (just pick whatever username) with "git config --global user.name "whateverusername"". The username should be in quotes.

21. Commit the changes with "git commit -m "first commit"". Make sure "first commit" is inside quotes.

22. Before pushing the changes to Heroku, tell heroku the name of the app you want to use with "heroku git:remote --app myawesomeappname"

23. Push the changes to Heroku with "git push heroku master"

26. That should do it. Go ahead and open your app with "heroku open".