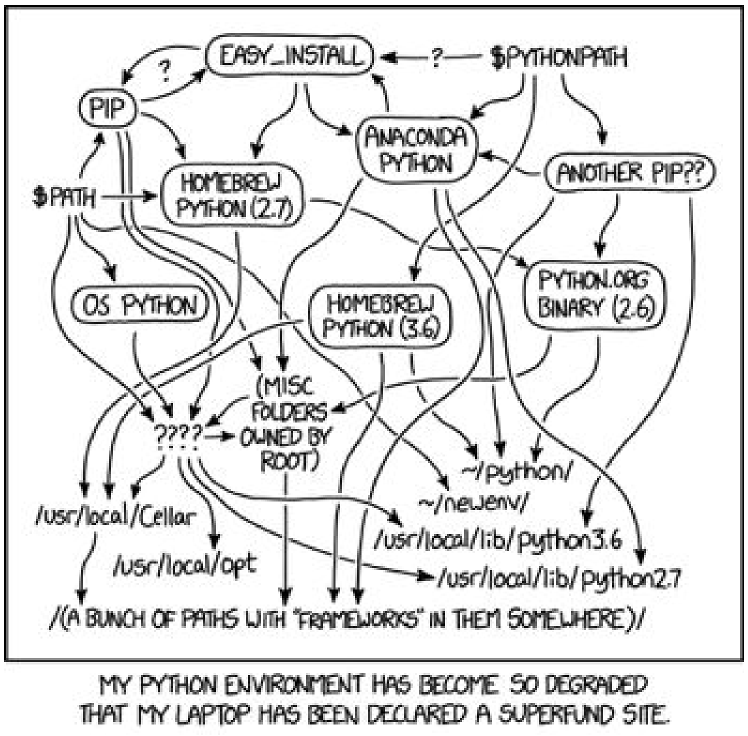
S-14a: Building Web Applications for Data Analysis

Lab 1: Setting Up the Environment



Source: https://xkcd.com/1987/

Learning Objectives

- Use several web development tools
- Work with command-line interface and editor
- Understand version control system
- Install Python, Flask, Heroku, and Postgres
- Create your first Flask web app

INTRODUCTION

Welcome! You are about to enter the world of web applications. Before we start on a journey together, let's install and briefly discuss all necessary dependencies.

Text Editor

A text editor is where we will write the code. A popular text editor is Sublime Text. Sublime is available for OSX, Linux, and Windows and it can be downloaded for free. To install Sublime visit this location: link and click "Download" link at the top of the page.

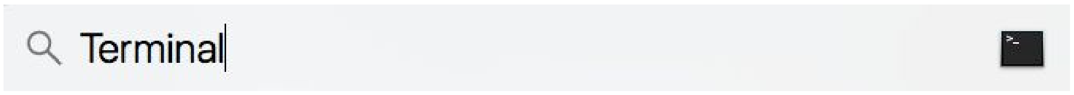
Sublime Text
Download
Buy
News
Forum
Support
A sophisticated text editor for code, markup and prose
DOWNLOAD FOR MAC
Sublime Text 3 *(Build 3176)*
Changelog

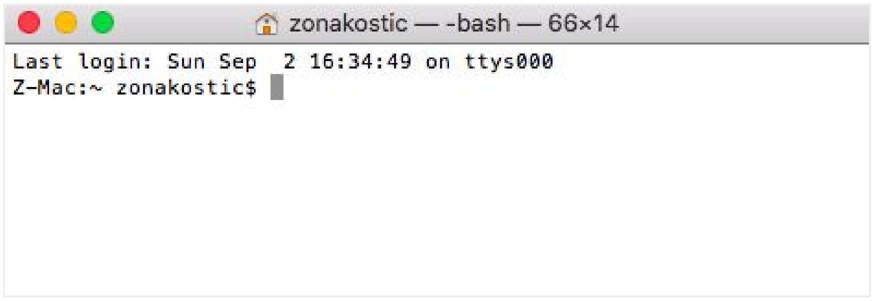
Source: link

From the Sublime page choose the installer for your operating system and install the program on your computer. After installing, open up Sublime to familiarize yourself with the environment. To create another file, go to the "File New". Once you're ready to save a file, go to "File Save" and save the file to your computer. As we get further along with web development we will typically have multiple files open at the same time. Make sure to save all your changes on time.

Use the command-line interface

The command-line is where we will type commands to and from the code. For OSX and Linux there should already be a command line program installed called "Terminal". For OSX you can just click on the *Spotlight Search* and type in `Terminal`. For Windows, look for "Command Prompt".





```
Last login: Sun Sep  2 16:34:49 on ttys000
Z-Mac:~ zonakostic$ 
```

Terminal output

Version Control

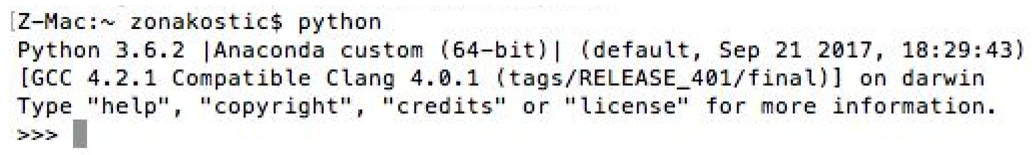
"Version control allows you to keep track of your work and helps you easily explore the changes you have made it data, coding scripts, notes, etc. You are probably already doing some type of version control, if you save multiple files. This approach will leave you with tens, or hundreds, of similar files, it makes it rather cumbersome to directly compare different versions, and is not easy to share among collaborators. With version control software such as Git, the control is much smoother and easier to implement. Using an online platform like Github to store your files means that you have an online backup of your work, which is beneficial for both you and your collaborators."

source: link

If you still don't have your Git installed, the S-14a team strongly recommends opening one! The steps for installation: link.

Python

Now it's time to start installing Python packages. If your operating system does not provide you with a Python package, you can download the installer from the Python official website: link. To make sure your Python installation is functional open a terminal window and type `python3`, or if that does not work, type `python`. Here is what you should expect to see:



```
[Z-Mac:~ zonakostic$ python
Python 3.6.2 |Anaconda custom (64-bit)| (default, Sep 21 2017, 18:29:43)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Terminal output after typing "python"

The Python interpreter is now waiting at an interactive prompt, where you can enter Python statements. This way you have confirmed that Python is installed on your system. To exit the interactive prompt, you can type `exit()` and press *Enter* (on the Linux and Mac OS X versions you can also exit the interpreter by pressing *Ctrl-D*). On Windows, the exit shortcut is *Ctrl-Z* followed by *Enter*.

Project Structure

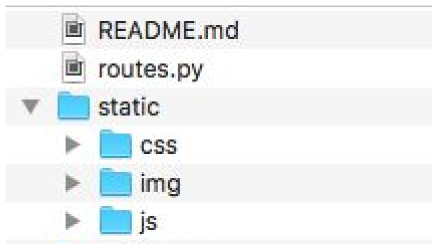
After setting up the development environment we can start building our first Flask app. First, we will create a homepage with mostly static content. Then, we will have a workflow that we can generalize to build more complex and dynamic web pages. Finally, we will run our first app on a local server. Let's start by creating a folders and files structure to keep the project organized. Open your terminal, move to the directory of your course labs, and type:

```
mkdir lab1
cd lab1

mkdir static
mkdir templates

cd static
mkdir css
mkdir img
mkdir js
```

Navigate back to your `lab1` directory and create an empty Python file named *routes.py*:

```
touch routes.py //for Mac
type nul > routes.py //for Win
```

This where we will write the app's main code. Create a file named *README.md*, this is where we are going to put information about the app. Type `touch README.md (type nul > README.md)` and your *README.md* is ready for use. You folder structure should look like this:



Files and folders structure

Virtual Environments

Python uses the concept of virtual environments to address the issue of maintaining different versions of packages. A virtual environment is a complete copy of the Python interpreter. When you install packages in a virtual environment, the Python interpreter is not affected, only the copy of it. So the solution to have complete freedom to install any versions of your packages for each application is to use a different virtual environments for each application.

With your terminal navigate to the directory of the project (*lab 1*) or if you are in the project navigate to *`lab1`* directory and type:

```
python3 -m venv env
```

After that, you will have a directory named *env* where the virtual environment files will be stored. If you are using a version of the Python older than 3.4 (and that includes the 2.7 release), virtual environments are not supported natively. For those versions of Python, you need to download and install a third-party tool called `virtualenv` before you can create virtual environments. You can install and create a virtual environment with the following command lines:

```
$ pip install virtualenv
virtualenv env
```

Now you have to tell the system that you want to use your virtual environment. To activate your virtual environment use the following command lines:

```
$ source env/bin/activate
(env) $ _
```

If you are using a Microsoft Windows command prompt window, the activation command is slightly different:

```
$ env\Scripts\activate
(env) $ _
```

When you activate a virtual environment, the configuration of your terminal session is modified so that the Python interpreter stored inside is the one that is invoked when you type python. Also, the terminal prompt is modified to include the name of the activated virtual environment. The changes made to the terminal session are all temporary and private to that session, so they will not persist when you close the terminal window. If you work with multiple terminal windows open at the same time, it is perfectly fine to have different virtual environments activated on each one.

Now that you have a virtual environment created and activated, you can finally install Flask in it.

Installing Flask

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

Flask
web development, one drop at a time
overview // docs // community // extensions // donate

Source: link

In Python, packages such as Flask are available in a public repository, from where anybody can download them and install them. The official Python package repository is called *PyPI*, which stands for *Python Package Index*. Installing a package from PyPI is very simple - Python comes with a tool called *pip* that does this work. To install a package use *pip* as follows:

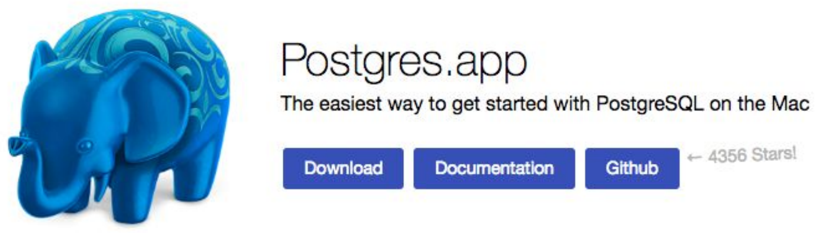
```
$ pip install <package-name>
```

For installing Flask, use this command:

```
$ pip install Flask
```

Installing Database

We'll use PostgreSQL as a relational database to store the flask app's data. PostgreSQL is a powerful, open source object-relational database system with over 30 years of active development that has earned a strong reputation for reliability, feature robustness, and performance.



Source: link

There is a wealth of information to be found describing how to install and use PostgreSQL through the official documentation. To install Postgres, go to link and follow the instructions for your operating system. We will use the app called Postgres.app. Once Postgres is downloaded, you'll need to create a new user account and password and sign in. For Windows users, please follow these steps: link

Deployment Environment - Heroku

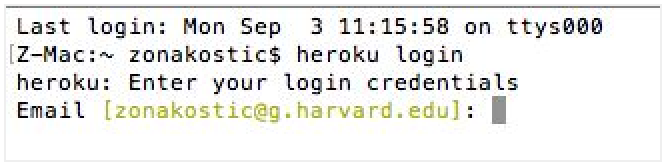
Heroku focuses relentlessly on apps and the developer experience around apps. Heroku lets companies of all sizes embrace the value of apps, not the distraction of hardware, nor the distraction of servers - virtual or otherwise.

Data lies at the heart of any significant app — whether it's customer data or data about the service it provides — an app and its data go hand in hand. Heroku's rich ecosystem of services includes Heroku Postgres, a database service that comes with operational expertise, built-in. Developers shouldn't need to discover how to optimally provision a database through trial and error - but instead have immediate access to a scalable, highly available database with rollback - one that supports their apps and development style.

source: [link]()

The final installation step is a simple way to deploy a Flask application, that is a way to put the Flask application up online so that people can see it. A popular deployment service is Heroku. Heroku's basic option is free. To get started with Heroku go to: [link]() and sign up for a free account.

Once you've done that, go to: [link]() and click *Download*. The Heroku toolbelt is a set of command-line tools that we'll use to interface with Heroku. Log-in to Heroku from the *Terminal* by typing `heroku login`.



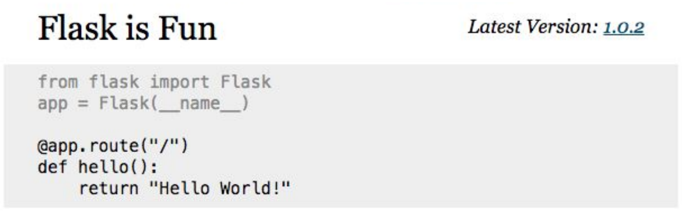
```
Last login: Mon Sep  3 11:15:58 on ttys000
[Z-Mac:~ zonakostic$ heroku login
heroku: Enter your login credentials
Email [zonakostic@g.harvard.edu]: 
```

Terminal output

Put your email and password and you're all set. We'll get into the details of how to deploy Heroku as we develop the application in the *Lab 2*.

“Hello World” Flask Example

Flask website welcomes with a very simple example application that has just five lines of code. Let us create and run our first Flask app in the last section of this Lab.



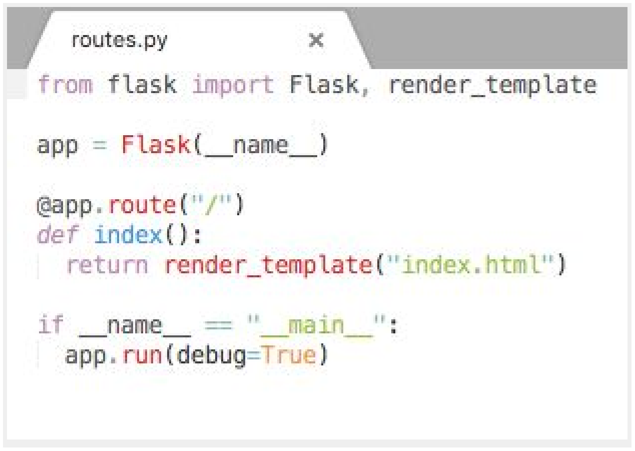
Flask is Fun

Latest Version: 1.0.2

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

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

Source: link

Use the previously created *lab 1* app folder structure to create our first app. Open Terminal and move to your `lab1` directory. Then, create a file called *routes.py*. Open the file in Sublime editor and type:

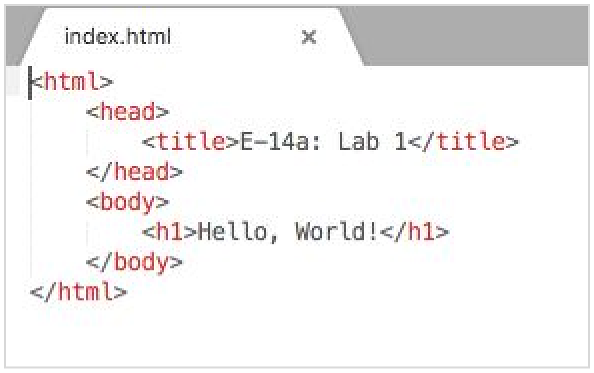


```
from flask import Flask, render_template

app = Flask(__name__)

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

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

Sublime output of routes.py

The routes are different URLs that the application implements. In Flask, handlers for the application routes are written as Python functions, called view functions. View functions are mapped to one or more route URLs so that Flask knows what logic to execute when a client requests a given URL.

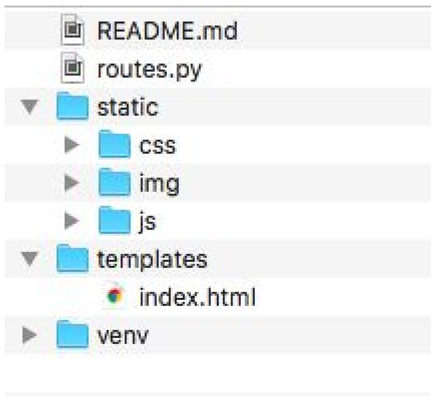
Next, from your Terminal navigate to your *templates* folder and once you are there create *index.html* page. You can simply use touch `index.html`. Open html page in your Sublime editor and type:



```
index.html
<html>
    <head>
        <title>E-14a: Lab 1</title>
    </head>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

Sublime output of index.html

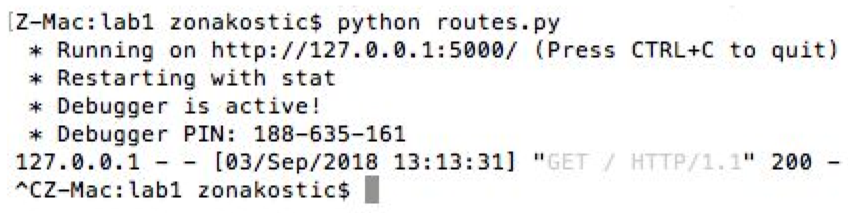
The code you typed in *routes.py* will be discussed in *Lab 2* in more detail. For now, make sure that you have saved your *routes.py* file and that your folder structure looks like this:



```
README.md
routes.py
▼ static
    ▶ css
    ▶ img
    ▶ js
▼ templates
    index.html
▶ venv
```

The final structure of the app's files and folders

In terminal, navigate to `lab1` directory and type *python routes.py* to run the application. After the server initializes it will wait for client connections. The output from Flask run indicates that the server is running on IP address 127.0.0.1, which is always the address of your own computer. This address is so common that also has a simpler name that you may have seen before: *localhost*.



```
[Z-Mac:lab1 zonakostic$ python routes.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 188-635-161
127.0.0.1 - - [03/Sep/2018 13:13:31] "GET / HTTP/1.1" 200 -
^CZ-Mac:lab1 zonakostic$ 
```

Terminal output

Network servers listen for connections on a specific port number. Applications deployed on production web servers typically listen on port 443, or sometimes 80 if they do not implement encryption, but access to these ports require administration rights. Since this application is running in a development environment, Flask uses the freely available port 5000.

Now open up your web browser and enter the following URL in the address field:

http://127.0.0.1:5000/

Congratulations! You have created your own Flask app.

Credits and Additional Resources

The Flask Mega Tutorial book:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Flask Community:
http://flask.pocoo.org/community/poweredby/

Flask Web Development - Developing Web Applications with Python:
https://flaskbook.com/