Getting Started with Flask

GitHub Repo with Example Files

Go to https://github.com/rfuller250/cs329e-flask-example to see the files used in this tutorial.

The HTML and CSS files are slightly modified versions of the files presented during class. The app.py file shows the structure of a simple web application using Flask.

Project Directory Structure

Your web application will have a directory for templates (HTML pages) and a directory for static content (CSS, images, and JavaScripts). The app.py program using Flask should be located outside of these two directories in the root project folder.

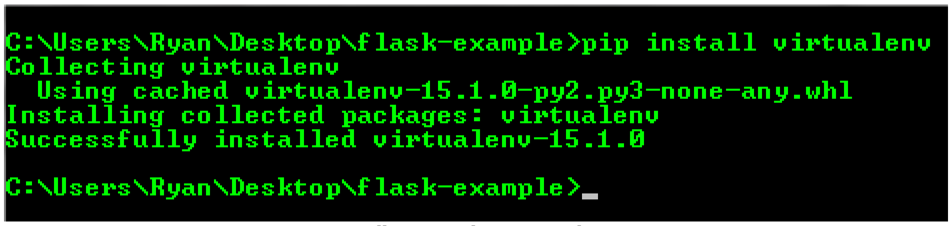
The initial directory structure should look like this:

```
cs329e-idb/
    static/
        css/
        images/
        js/
    templates/
    app.py
```

Install virtualenv (Non-CS Machines)

If you are not working on a CS machine you will probably need to install virtualenv. Using the version of pip associated with python3.5 run: `pip install virtualenv`

This tool allows you to create an isolated Python environment and to install only the packages required for your project without disturbing the base installation of Python.



```
C:\Users\Ryan\Desktop\flask-example>pip install virtualenv
Collecting virtualenv
  Using cached virtualenv-15.1.0-py2.py3-none-any.whl
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0

C:\Users\Ryan\Desktop\flask-example>_
```

Installing virtualenv in Windows

Create a Virtual Environment

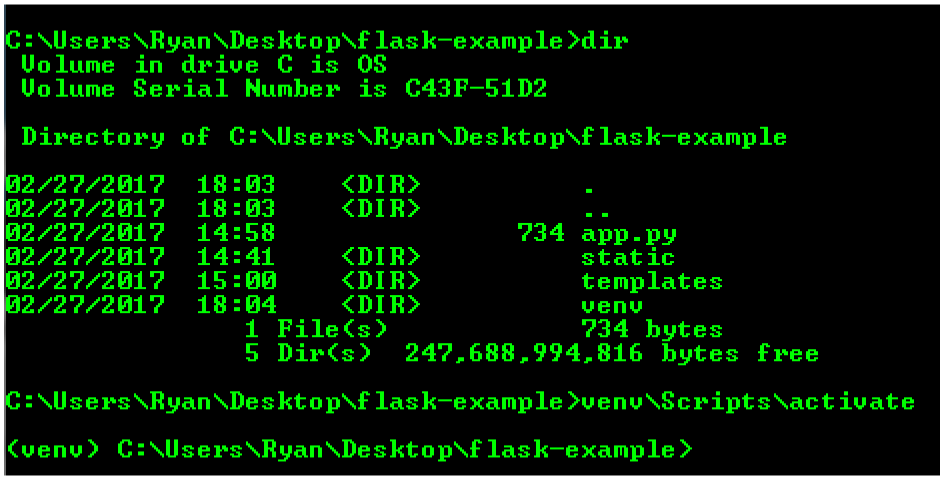
Even if you are working on a CS machine you will need to use a virtual environment when developing and running your web application. (The CS machines do not have Flask installed, but you can install it through a virtual environment.)

Navigate into your root project directory (cs329e-idb/) and run: `virtualenv venv`
If you are on a CS machine use: `virtualenv –p python3.5 venv`

A folder named venv/ should be created in your project directory. This process may take a bit of time, especially on Windows.

Activate the Virtual Environment

Windows: `venv\Scripts\activate` (must use \)
Linux/Mac: `source venv/bin/activate`



```
C:\Users\Ryan\Desktop\flask-example>dir
 Volume in drive C is OS
 Volume Serial Number is C43F-51D2

 Directory of C:\Users\Ryan\Desktop\flask-example

02/27/2017  18:03    <DIR>          .
02/27/2017  18:03    <DIR>          ..
02/27/2017  14:58               734 app.py
02/27/2017  14:41    <DIR>          static
02/27/2017  15:00    <DIR>          templates
02/27/2017  18:04    <DIR>          venv
               1 File(s)            734 bytes
               5 Dir(s)  247,688,994,816 bytes free

C:\Users\Ryan\Desktop\flask-example>venv\Scripts\activate

(venv) C:\Users\Ryan\Desktop\flask-example>
```

Activating venv in Windows

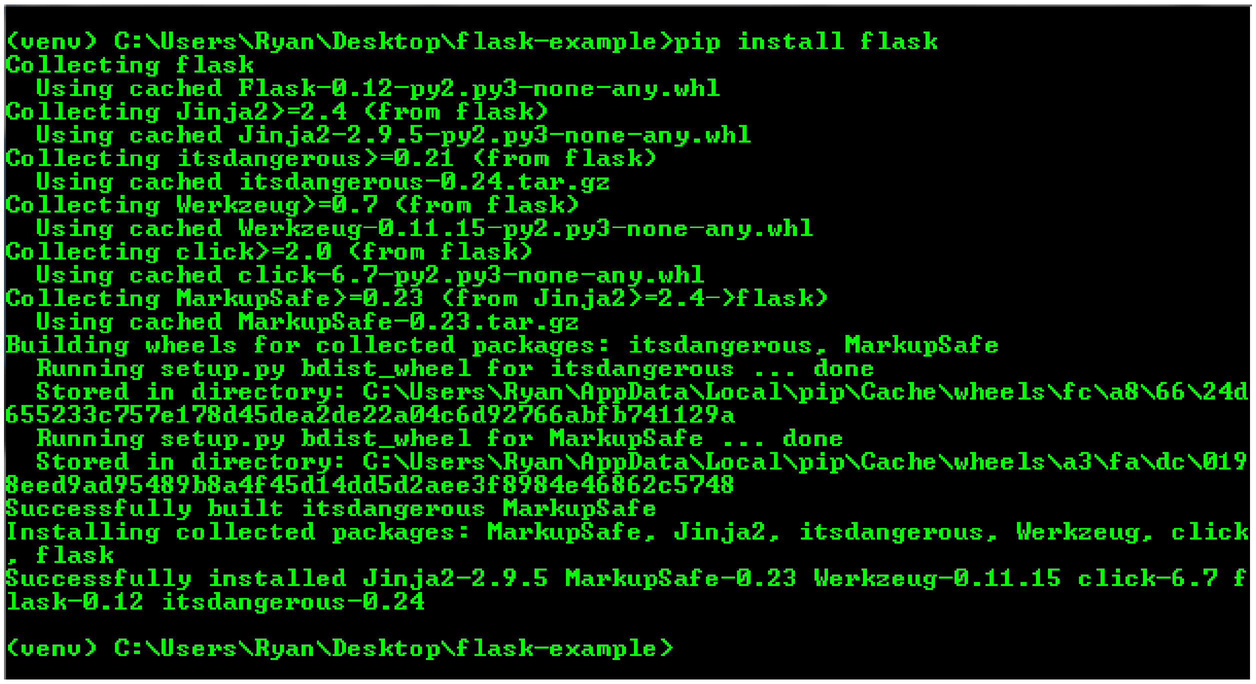
If the environment activates correctly the name will appear in parentheses on the left side of the command prompt.

To deactivate: `deactivate` or close the terminal

Install Flask

With the virtual environment activated, you can use pip to install Python packages. These packages will only be useable from within your virtual environment and do not affect the base installation of python.

To install Flask: pip install flask



```
(venv) C:\Users\Ryan\Desktop\flask-example>pip install flask
Collecting flask
  Using cached Flask-0.12-py2.py3-none-any.whl
Collecting Jinja2>=2.4 (from flask)
  Using cached Jinja2-2.9.5-py2.py3-none-any.whl
Collecting itsdangerous>=0.21 (from flask)
  Using cached itsdangerous-0.24.tar.gz
Collecting Werkzeug>=0.7 (from flask)
  Using cached Werkzeug-0.11.15-py2.py3-none-any.whl
Collecting click>=2.0 (from flask)
  Using cached click-6.7-py2.py3-none-any.whl
Collecting MarkupSafe>=0.23 (from Jinja2>=2.4->flask)
  Using cached MarkupSafe-0.23.tar.gz
Building wheels for collected packages: itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for itsdangerous ... done
  Stored in directory: C:\Users\Ryan\AppData\Local\pip\Cache\wheels\fc\a8\66\24d
655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe ... done
  Stored in directory: C:\Users\Ryan\AppData\Local\pip\Cache\wheels\a3\fa\dc\019
8eed9ad95489b8a4f45d14dd5d2aee3f8984e46862c5748
Successfully built itsdangerous MarkupSafe
Installing collected packages: MarkupSafe, Jinja2, itsdangerous, Werkzeug, click
, flask
Successfully installed Jinja2-2.9.5 MarkupSafe-0.23 Werkzeug-0.11.15 click-6.7 f
lask-0.12 itsdangerous-0.24

(venv) C:\Users\Ryan\Desktop\flask-example>
```

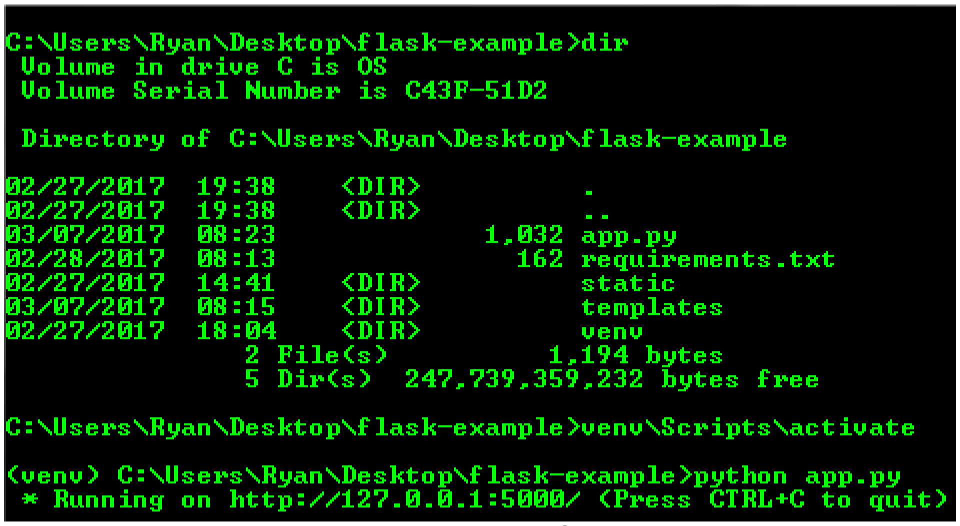
Installing Flask in Windows

Running a Flask Application

Flask comes with a development server so an application can be tested locally without having to deploy to a web server. The development server hosts the application on the localhost address with a default port number of 5000.

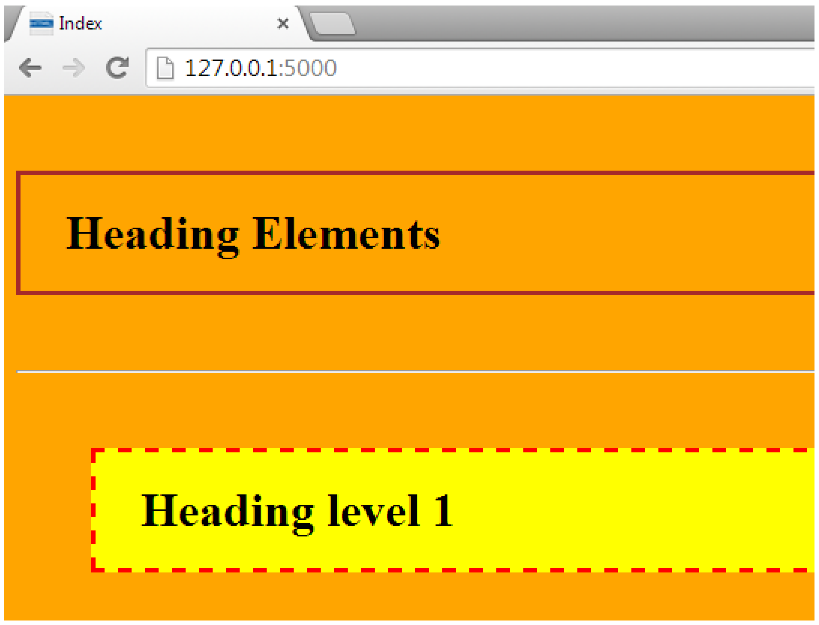
In the main project folder, with the virtual environment activated, run python app.py. You should get a message saying the application is running, and if you navigate in a web browser to http://127.0.0.1:5000, the index.html page should pull up.

Here's a few screenshots showing that process:



```
C:\Users\Ryan\Desktop\flask-example>dir
 Volume in drive C is OS
 Volume Serial Number is C43F-51D2

 Directory of C:\Users\Ryan\Desktop\flask-example

02/27/2017  19:38    <DIR>          .
02/27/2017  19:38    <DIR>          ..
03/07/2017  08:23             1,032 app.py
02/28/2017  08:13               162 requirements.txt
02/27/2017  14:41    <DIR>          static
03/07/2017  08:15    <DIR>          templates
02/27/2017  18:04    <DIR>          venv
               2 File(s)          1,194 bytes
               5 Dir(s)  247,739,359,232 bytes free

C:\Users\Ryan\Desktop\flask-example>venv\Scripts\activate

(venv) C:\Users\Ryan\Desktop\flask-example>python app.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Running app.py in Windows

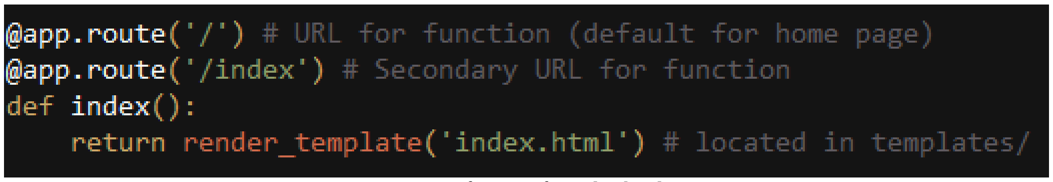


http://127.0.0.1:5000 in Web Browser

Basic Flask Routing using Templates

Take a look at app.py to see the basic structure of a web application using flask.

Each of your HTML pages will have a function that ends with a call to return render_template('name.html'). The @app.route( ) decorators defined above each function specify which URLs will result in a particular HTML page being displayed.

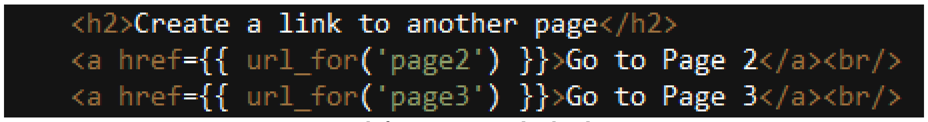


```
@app.route('/') # URL for function (default for home page)
@app.route('/index') # Secondary URL for function
def index():
    return render_template('index.html') # located in templates/
```

Routing function for index.html

For example, the screenshot above shows the routing function for index.html. The @app.route( ) decorators specify that URLs ending in "/" or "/index" will result in this function being called. Once the function is called the index.html page is returned and rendered in a web browser.

To ensure that these functions are called properly, all <a href> tags between your HTML pages (only your HTML pages, not external links) need to be reformatted to work with Flask.

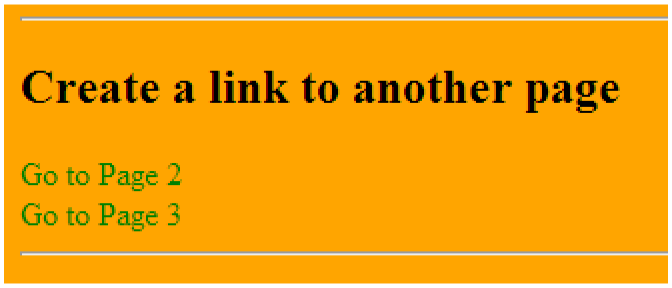


```
    <h2>Create a link to another page</h2>
    <a href={{ url_for('page2') }}>Go to Page 2</a><br/>
    <a href={{ url_for('page3') }}>Go to Page 3</a><br/>
```

Link formatting in index.html

Flask uses Jinja2 along with AngularJS to render templates. The "{{ }}" notation signals Jinja2 to render the information contained within the double brackets. The above screenshot shows two links within index.html. Each link will call the url_for( ) function and is provided the name of the HTML page that is being linked to. This will ensure that the proper routing function in app.py is called.

This section of index.html discussed above will appear as follows in a web browser:

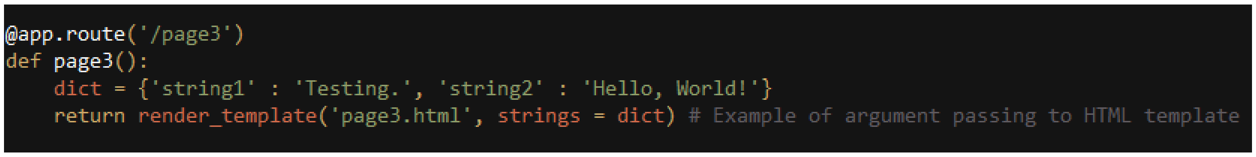


Create a link to another page

Go to Page 2
Go to Page 3

Links between pages in index.html

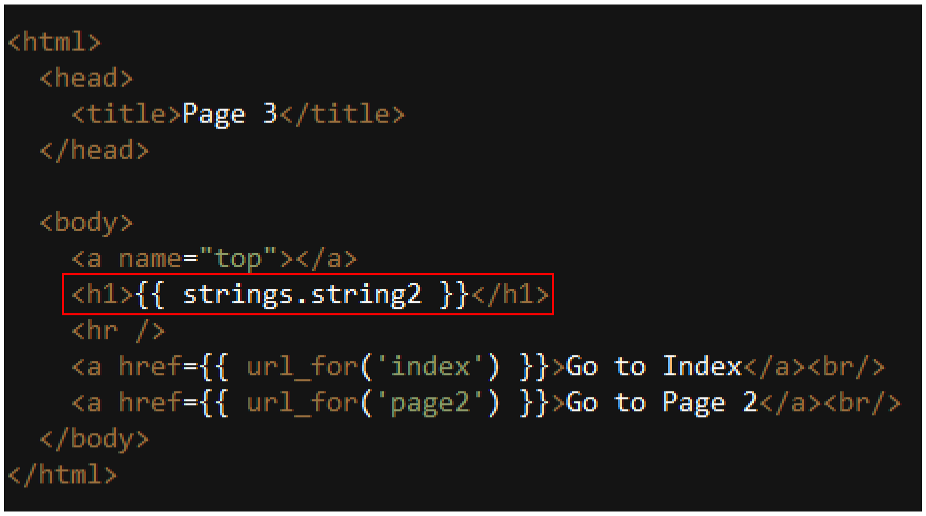
You can also pass information as a parameter in the calls to render_template( ) in the routing functions of app.py. Here's what the page3( ) routing function in app.py looks like:



```
@app.route('/page3')
def page3():
    dict = {'string1' : 'Testing.', 'string2' : 'Hello, World!'}
    return render_template('page3.html', strings = dict) # Example of argument passing to HTML template
```

page3( ) routing function, passing dictionary to HTML template

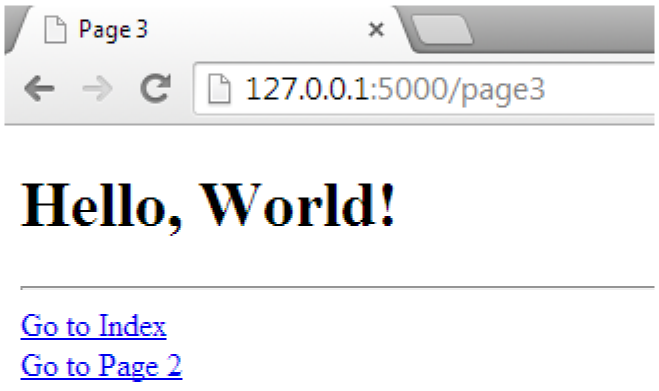
A dictionary named "dict" is created containing two strings. This dictionary gets passed in the call to render_template( ) and is assigned the name "strings". Each parameter passed in the call to render_template( ) must be assigned a name, since that name will be used to access the parameter from within page3.html.



```
<html>
  <head>
    <title>Page 3</title>
  </head>

  <body>
    <a name="top"></a>
    <h1>{{ strings.string2 }}</h1>
    <hr />
    <a href={{ url_for('index') }}>Go to Index</a><br/>
    <a href={{ url_for('page2') }}>Go to Page 2</a><br/>
  </body>
</html>
```

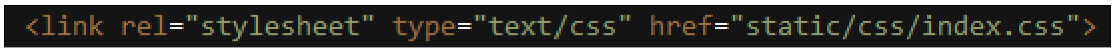
Accessing passed parameter in page3.html

The above screenshot shows the section of page3.html that accesses the dictionary received as a parameter from the routing function. To access the value associated with the key "string2" in the dictionary, you use the notation <parameter_name>.<key>. This will display the selected string in the top header of Page 3 as shown below:

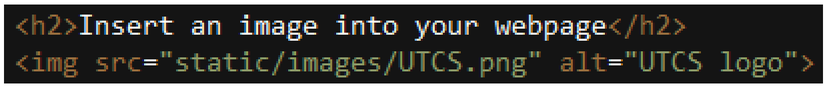


String from dictionary displayed on Page 3

Using Flask also affects the path you should use in your HTML files to embed images and add CSS files. Flask will expect these files to be inside of the static/ directory within your main project directory. Here’s a screenshot showing the path to the index.css file and UTCS image used in index.html:



```
<link rel="stylesheet" type="text/css" href="static/css/index.css">
```



```
<h2>Insert an image into your webpage</h2>
<img src="static/images/UTCS.png" alt="UTCS logo">
```

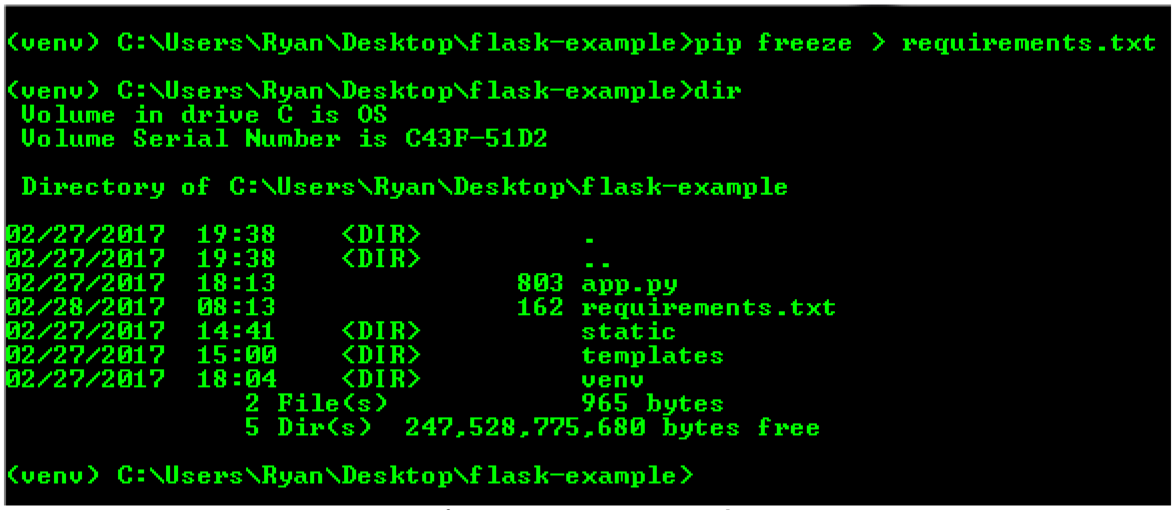
Paths to static content in index.html

The basic structure of the links to static content will be “static/<path-to-file>”.

Flask can be used to accomplish a lot more than the basic features described here. You can find links to a few good online tutorials at the end of this document.

Save/Restore Packages in the Virtual Environment

In the root directory of your project and with the virtual environment activated, you can use pip freeze > requirements.txt to save the list of packages installed within the virtual environment. This will create the file requirements.txt, which can be used to reinstall the packages if you want to run the web application on a different machine.



```
(venv) C:\Users\Ryan\Desktop\flask-example>pip freeze > requirements.txt

(venv) C:\Users\Ryan\Desktop\flask-example>dir
 Volume in drive C is OS
 Volume Serial Number is C43F-51D2

 Directory of C:\Users\Ryan\Desktop\flask-example

02/27/2017  19:38    <DIR>          .
02/27/2017  19:38    <DIR>          ..
02/27/2017  18:13               803 app.py
02/28/2017  08:13               162 requirements.txt
02/27/2017  14:41    <DIR>          static
02/27/2017  15:00    <DIR>          templates
02/27/2017  18:04    <DIR>          venv
               2 File(s)            965 bytes
               5 Dir(s)  247,528,775,680 bytes free

(venv) C:\Users\Ryan\Desktop\flask-example>
```

pip freeze > requirements.txt in Windows

To install your project's packages on a different machine, clone your project repo from GitHub and set up a new virtual environment in the root directory using the above instructions. Then, instead of having to "pip install" each package one at a time, use pip install –r requirements.txt. Make sure the virtual environment is activated before attempting to install.

Links

- CS 373 Flask Example Wiki (ignore the database parts)
- Flask Tutorial at TutorialsPoint
- Python Virtual Environments
- The Flask Mega-Tutorial
- virtualenv & Flask on Windows