

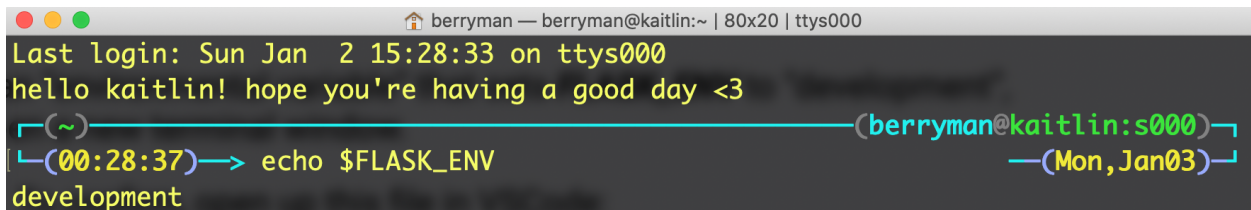
Kaitlin Berryman
Springboard Software Engineering Bootcamp
Flask Greet and Calc Exercise

Setup Your Environment

It will be more convenient if you always have an “environmental variable” that sets FLASK_ENV to “development”, so you don’t have to do that every time you open a new terminal window.

I use zsh / oh-my-zsh for a custom config, so I actually needed to edit .zshrc in ~ instead of .bash_profile or .bashrc. My .zshrc runs every time I open a new terminal.

```
107 # Kaitlin's User Settings
108 echo "hello kaitlin! hope you're having a good day <3"
109 export FLASK_ENV=development
```



```
berryman@kaitlin:~ | 80x20 | ttys000
Last login: Sun Jan  2 15:28:33 on ttys000
hello kaitlin! hope you're having a good day <3
berryman@kaitlin:s000 ~
(00:28:37) → echo $FLASK_ENV
development
```

Set Up Your Project

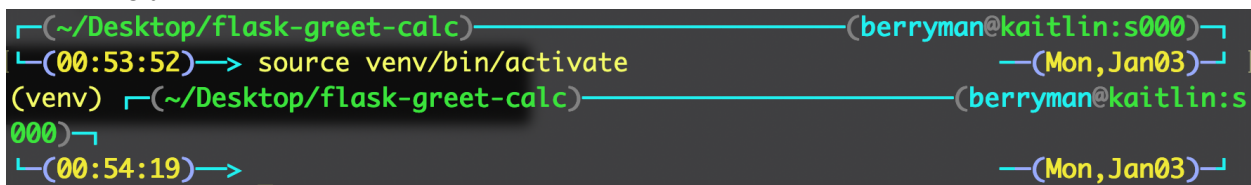
At the top level of this (inside flask-greet-calc), create a virtual environment:

```
(~/Desktop/flask-greet-calc) berryman@kaitlin:s000
(00:53:41) → python3 -m venv venv
```



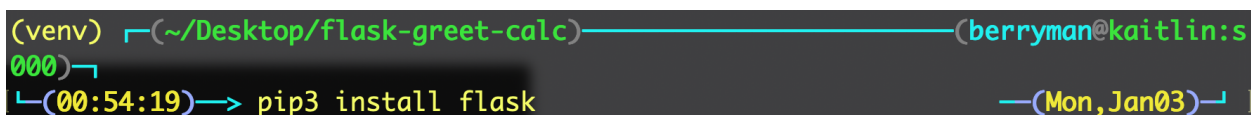
Start using your venv:

```
(~/Desktop/flask-greet-calc) berryman@kaitlin:s000
(00:53:52) → source venv/bin/activate
(venv) (~/Desktop/flask-greet-calc) berryman@kaitlin:s000
(00:54:19) → _
```



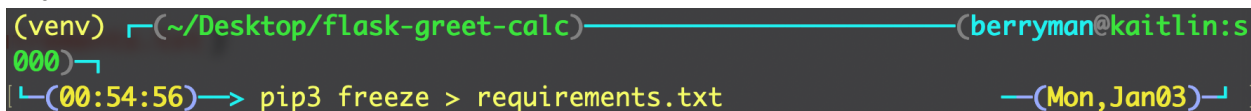
Install Flask:

```
(venv) (~/Desktop/flask-greet-calc) berryman@kaitlin:s000
(00:54:19) → pip3 install flask
```



Make a “requirements.txt” file in this directory with a listing of all the software needed for this project:

```
(venv) (~/Desktop/flask-greet-calc) berryman@kaitlin:s000
(00:54:56) → pip3 freeze > requirements.txt
```



```
(venv) └─(~/Desktop/flask-greet-calc)─(berryman@kaitlin:s000)─
└─(00:55:51)─> cat requirements.txt ─(Mon, Jan03)─
click==8.0.3
Flask==2.0.2
itsdangerous==2.0.1
Jinja2==3.0.3
MarkupSafe==2.0.1
Werkzeug==2.0.2
```

Set Up Git

We want you to add this project to Git, so let's make our project a Git repository:

```
(venv) └─(~/Desktop/flask-greet-calc)─(berryman@kaitlin:s000)─
└─(00:57:00)─> git init ─(Mon, Jan03)─
```

Then, since we don't want the venv/ folder put into Git (or send to GitHub), put it in a file called .gitignore:

```
(venv) └─(~/Desktop/flask-greet-calc)─(berryman@kaitlin:s000)─
└─(00:57:19 on master ★)─> touch .gitignore ─(Mon, Jan03)─
```

```
.gitignore
1  venv/
    .....
```

You should test that Git is ignoring this file by making sure it doesn't appear as an untracked file in git status:

```
(venv) └─(~/Desktop/flask-greet-calc)─(berryman@kaitlin:s000)─
└─(00:57:54 on master ★)─> git status ─(Mon, Jan03)─
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
    calc/
    greet/
    requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Greet

Greet

In the ***greet*** folder, Make a simple Flask app that responds to these routes with simple text messages:

/welcome

Returns "welcome"

/welcome/home

Returns "welcome home"

/welcome/back

Return "welcome back"

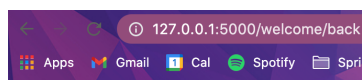
Once you've finished this, run the tests for it:

```
$ python3 -m unittest test.py
```

```
(venv) └─(~/Desktop/flask-greet-calc)─┐
└─(01:01:50 on master ★)─> cd greet
(venv) └─(~/Desktop/flask-greet-calc/greet)─┐
└─(01:01:53 on master ★)─> flask run
* Environment: development
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 131-331-463
```



welcome



welcome back



welcome home

```
(venv) └─(~/Desktop/flask-greet-calc/greet)
000)└─
└─(01:05:12 on master ★)─> python3 -m unittest test.py
...
-----
Ran 3 tests in 0.008s

OK
```

```
.gitignore  ×  app.py
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/welcome')
6  def welcome():
7      return "welcome"
8
9  @app.route('/welcome/home')
10 def welcome_home():
11     return "welcome home"
12
13 @app.route('/welcome/back')
14 def welcome_back():
15     return "welcome back"
16
```

Calc

Calc

Build a simple calculator with Flask, which uses URL query parameters to get the numbers to calculate with.

Make a Flask app that responds to 4 different routes. Each route does a math operation with two numbers, **a** and **b**, which will be passed in as URL GET-style query parameters.

/add

Adds **a** and **b** and returns result as the body.

/sub

Same, subtracting **b** from **a**.

/mult

Same, multiplying **a** and **b**.

/div

Same, dividing **a** by **b**.

For example, a URL like **`http://localhost:5000/add?a=10&b=20`** should return a string response of exactly **30**.

Write the routes for this but **don't hardcode the math operation in your route function** directly. Instead, we've provided helper functions for this in the file **`operations.py`**.

```
1  from flask import Flask, request
2  from operations import add, sub, mult, div
3
4  app = Flask(__name__)
5
6  @app.route('/add')
7  def add_route():
8      a = int(request.args["a"])
9      b = int(request.args["b"])
10     return str(add(a, b))
11
12  @app.route('/sub')
13  def sub_route():
14      a = int(request.args["a"])
15      b = int(request.args["b"])
16      return str(sub(a, b))
17
18  @app.route('/mult')
19  def mult_route():
20      a = int(request.args["a"])
21      b = int(request.args["b"])
22      return str(mult(a, b))
23
24  @app.route('/div')
25  def div_route():
26      a = int(request.args["a"])
27      b = int(request.args["b"])
28      return str(div(a, b))
```