# Flask Intro

Download exercise starter code

## Step Zero: 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.

You can configure this in your ***~/.bash_profile***. To confirm, open up this file in VSCode:

```
code ~/.bash_profile
```

Add the following line to it, if you don't have this line already included:

```
export FLASK_ENV=development
```

**Close this terminal window and open a new one.**

Test that this works like this:

```
$echo $FLASK_ENV development
```

## Set Up Your Project

Download the starter code. You'll get a directory with two directories in it:

```
flask-greet-calc/ greet/ calc/
```

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

```
$python3 -m venv venv
```

Start using your venv:

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

Install Flask:

```
(env) $pip3 install flask ...
```

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

```
(env) $pip3 freeze > requirements.txt
```

(you can look at that file with `cat requirements.txt`)

## Set Up Git

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

```
(env) $git init
```

Then, since we **don't** want the ***venv/*** folder put into Git (or send to GitHub), put it in a file called ***.gitignore*** (notice the leading dot!). Inside that file should be this line:

*.gitignore*

`venv/`

(which means "ignore all folders named ***venv/*** anywhere here and below, as far as git is concerned")

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

```
(env) $git status
```

## 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
```

## 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***:

*calc/operations.py*

```
"""Basic math operations."""def add(a, b): """Add a and b.""" return a + b
def sub(a, b): """Substract b from a."""return a - b def mult(a, b):
"""Multiply a and b."""return a * b def div(a, b): """Divide a by b."""return
a / b
```

Import and use these in your routes.

After you've tried out your app, run the unit tests:

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

## Further Study

You probably have a lot of code duplication in your ***calc*** routes, given that you're doing such similar things in each.

Make a single route/view function that can deal with 4 different kinds of URLs:

- ***/math/add***
- ***/math/sub***
- ***/math/mult***
- ***/math/div***

You can write this in one function with one route by using a route parameter for the actual operation ("add", "sub", etc).

As an extra-bonus, see if you can find a way to do this in the route without a whole series of if/elif statements. One good way is to use a dictionary to map operation names to the functions that do the underlying math.

## Solution

View Our Solution