Application Setup

A Flask application is an instance of the **`Flask`** class. Everything about the application, such as configuration and URLs, will be registered with this class.

The most straightforward way to create a Flask application is to create a global **`Flask`** instance directly at the top of your code, like how the "Hello, World!" example did on the previous page. While this is simple and useful in some cases, it can cause some tricky issues as the project grows.

Instead of creating a **`Flask`** instance globally, you will create it inside a function. This function is known as the *application factory*. Any configuration, registration, and other setup the application needs will happen inside the function, then the application will be returned.

The Application Factory

It's time to start coding! Create the `flaskr` directory and add the `__init__.py` file. The `__init__.py` serves double duty: it will contain the application factory, and it tells Python that the `flaskr` directory should be treated as a package.

```
$ mkdir flaskr
```

`flaskr/__init__.py`

```
import os

from flask import Flask


def create_app(test_config=None):
    # create and configure the app
    app = Flask(__name__, instance_relative_config=True)
    app.config.from_mapping(
        SECRET_KEY='dev',
        DATABASE=os.path.join(app.instance_path, 'flaskr.sqlite'),
    )

    if test_config is None:
        # load the instance config, if it exists, when not testing
        app.config.from_pyfile('config.py', silent=True)
    else:
        # load the test config if passed in
        app.config.from_mapping(test_config)
```

```
    # ensure the instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    # a simple page that says hello
    @app.route('/hello')
    def hello():
        return 'Hello, World!'

    return app
```

`create_app` is the application factory function. You'll add to it later in the tutorial, but it already does a lot.

1. `app = Flask(__name__, instance_relative_config=True)` creates the **`Flask`** instance.
    - `__name__` is the name of the current Python module. The app needs to know where it's located to set up some paths, and `__name__` is a convenient way to tell it that.
    - `instance_relative_config=True` tells the app that configuration files are relative to the instance folder. The instance folder is located outside the `flaskr` package and can hold local data that shouldn't be committed to version control, such as configuration secrets and the database file.
2. **`app.config.from_mapping()`** sets some default configuration that the app will use:
    - **`SECRET_KEY`** is used by Flask and extensions to keep data safe. It's set to `'dev'` to provide a convenient value during development, but it should be overridden with a random value when deploying.
    - `DATABASE` is the path where the SQLite database file will be saved. It's under **`app.instance_path`**, which is the path that Flask has chosen for the instance folder. You'll learn more about the database in the next section.
3. **`app.config.from_pyfile()`** overrides the default configuration with values taken from the `config.py` file in the instance folder if it exists. For example, when deploying, this can be used to set a real `SECRET_KEY`.
    - `test_config` can also be passed to the factory, and will be used instead of the instance configuration. This is so the tests you'll write later in the tutorial can be configured independently of any development values you have configured.

4. **`os.makedirs()`** ensures that **`app.instance_path`** exists. Flask doesn't create the instance folder automatically, but it needs to be created because your project will create the SQLite database file there.
5. **`@app.route()`** creates a simple route so you can see the application working before getting into the rest of the tutorial. It creates a connection between the URL `/hello` and a function that returns a response, the string `'Hello, World!'` in this case.

Run The Application

Now you can run your application using the `flask` command. From the terminal, tell Flask where to find your application, then run it in development mode. Remember, you should still be in the top-level `flask-tutorial` directory, not the `flaskr` package.

Development mode shows an interactive debugger whenever a page raises an exception, and restarts the server whenever you make changes to the code. You can leave it running and just reload the browser page as you follow the tutorial.

For Linux and Mac:

```
$ export FLASK_APP=flaskr
$ export FLASK_ENV=development
$ flask run
```

For Windows cmd, use `set` instead of `export`:

```
> set FLASK_APP=flaskr
> set FLASK_ENV=development
> flask run
```

For Windows PowerShell, use `$env:` instead of `export`:

```
> $env:FLASK_APP = "flaskr"
> $env:FLASK_ENV = "development"
> flask run
```

You'll see output similar to this:

```
* Serving Flask app "flaskr"
* 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: 855-212-761
```

Visit http://127.0.0.1:5000/hello in a browser and you should see the “Hello, World!” message. Congratulations, you’re now running your Flask web application!

Continue to Define and Access the Database.