## Development Web Server

Flask applications include a development web server that can be started with the `flask run` command. This command looks for the name of the Python script that contains the application instance in the `FLASK_APP` environment variable.

To start the *hello.py* application from the previous section, first make sure the virtual environment you created earlier is activated and has Flask installed in it. For Linux and macOS users, start the web server as follows:

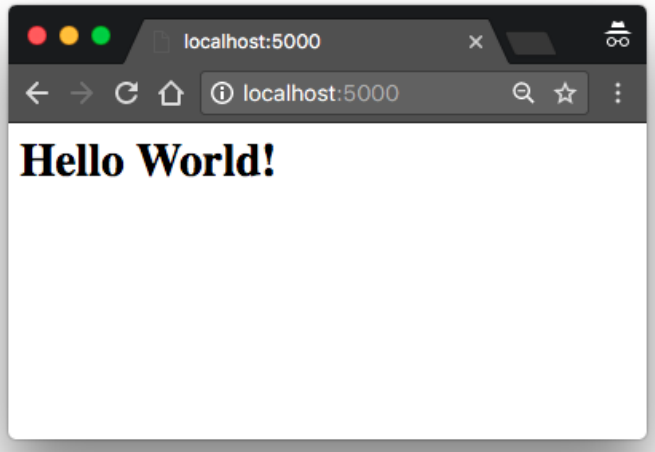
```
(venv) $ export FLASK_APP=hello.py
(venv) $ flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

For Microsoft Windows users, the only difference is in how the `FLASK_APP` environment variable is set:

```
(venv) $ set FLASK_APP=hello.py
(venv) $ flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Once the server starts up, it goes into a loop that accepts requests and services them. This loop continues until you stop the application by pressing Ctrl+C.

With the server running, open your web browser and type **`http://localhost:5000/`** in the address bar. Figure 2-1 shows what you'll see after connecting to the application.



## Debug Mode

Flask applications can optionally be executed in *debug mode*. In this mode, two very convenient modules of the development server called the *reloader* and the *debugger* are enabled by default.

When the reloader is enabled, Flask watches all the source code files of your project and automatically restarts the server when any of the files are modified. Having a server running with the reloader enabled is extremely useful during development, because every time you modify and save a source file, the server automatically restarts and picks up the change.

The debugger is a web-based tool that appears in your browser when your application raises an unhandled exception. The web browser window transforms into an interactive stack trace that allows you to inspect source code and evaluate expressions in any place in the call stack. You can see how the debugger looks in Figure 2-3.

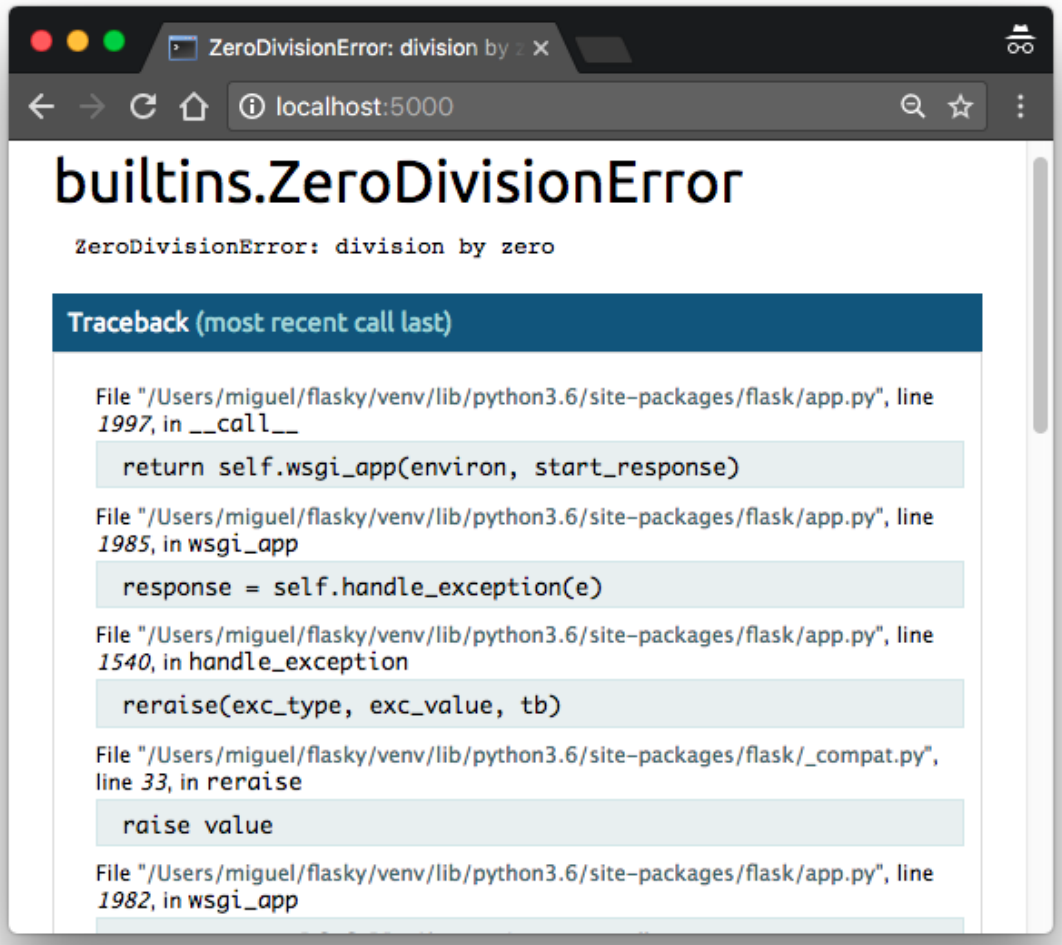



*Figure 2-3. Flask debugger*

By default, debug mode is disabled. To enable it, set a `FLASK_DEBUG=1` environment variable before invoking `flask run`:

```
(venv) $ export FLASK_APP=hello.py
(venv) $ export FLASK_DEBUG=1
(venv) $ flask run
 * Serving Flask app "hello"
 * Forcing debug mode on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 273-181-528
```

If you are using Microsoft Windows, use `set` instead of `export` to set the environment variables.

NOTE

If you start your server with the `app.run()` method,
the `FLASK_APP` and `FLASK_DEBUG`environment variables are not used. To enable debug mode programmatically, use `app.run(debug=True)`.

WARNING

Never enable debug mode on a production server. The debugger in particular allows the client to request remote code execution, so it makes your production server vulnerable to attacks. As a simple protection measure, the debugger needs to be activated with a PIN, printed to the console by the `flask run` command.