Getting Started with App Engine: Part 2

This guide is intended to help you get started working with AppEngine. If you know just a little bit of Python, HTML, and CSS, you should be able to work through these examples and get a live web application up and running pretty easily.

This chapter focuses on working with form data and the webapp2 platform. You should be familiar with Part 1 (creating an App Engine application with simple outputs) before you start this material.

Table of Contents

- Introduction
- Working with Form Data
  - Defining a Form
  - Handling Form Data
  - Displaying HTML Output

Getting Started with App Engine: Part 2

Introduction

In Part 1 of this guide, we were able to create a simple web application that used Python and HTML to display a simple message to a user. If this were 1995 we'd be able to get by with just that much - simple HTML with little in the way of interactivity. Fortunately for us this is 20 years later and we can do a lot more. In this section we'll handle user input.

Working with Form Data

Handling form data requires a few pieces; we need to first define the form that we want our users to use. Next we'll need to identify what part of our application should process the form data. After that we'll process the data, then we'll display a new page to the user.

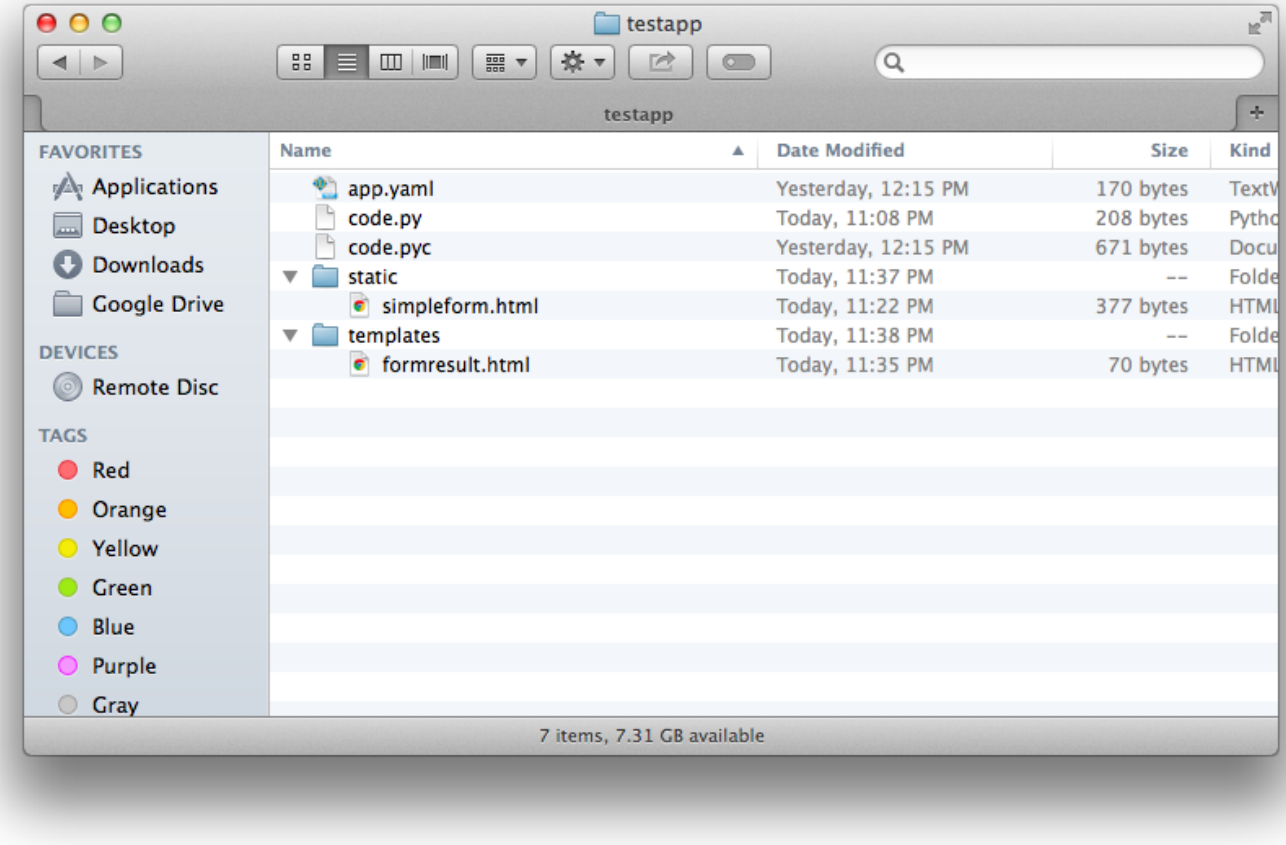
Just about everything we work with on the Internet follows this pattern; we create some HTML (either statically or through some output dynamically assembled via Python, Java, ASP, etc.) that is sent to the user's browser, where it's rendered into some form. The user enters data and submits a form, and we get a new request that can be processed, so that we can send more HTML to the user's browser.

So, in order to accomplish this in App Engine, we'll need to take on a few steps:

1. Define a form in HTML (we'll use a template here).
2. Create a new RequestHandler class in our Python code so that we can handle the form.
3. Map the new RequestHandler class to some URL.
4. Retrieve the user-submitted form data and process it.
5. Display a new HTML page.

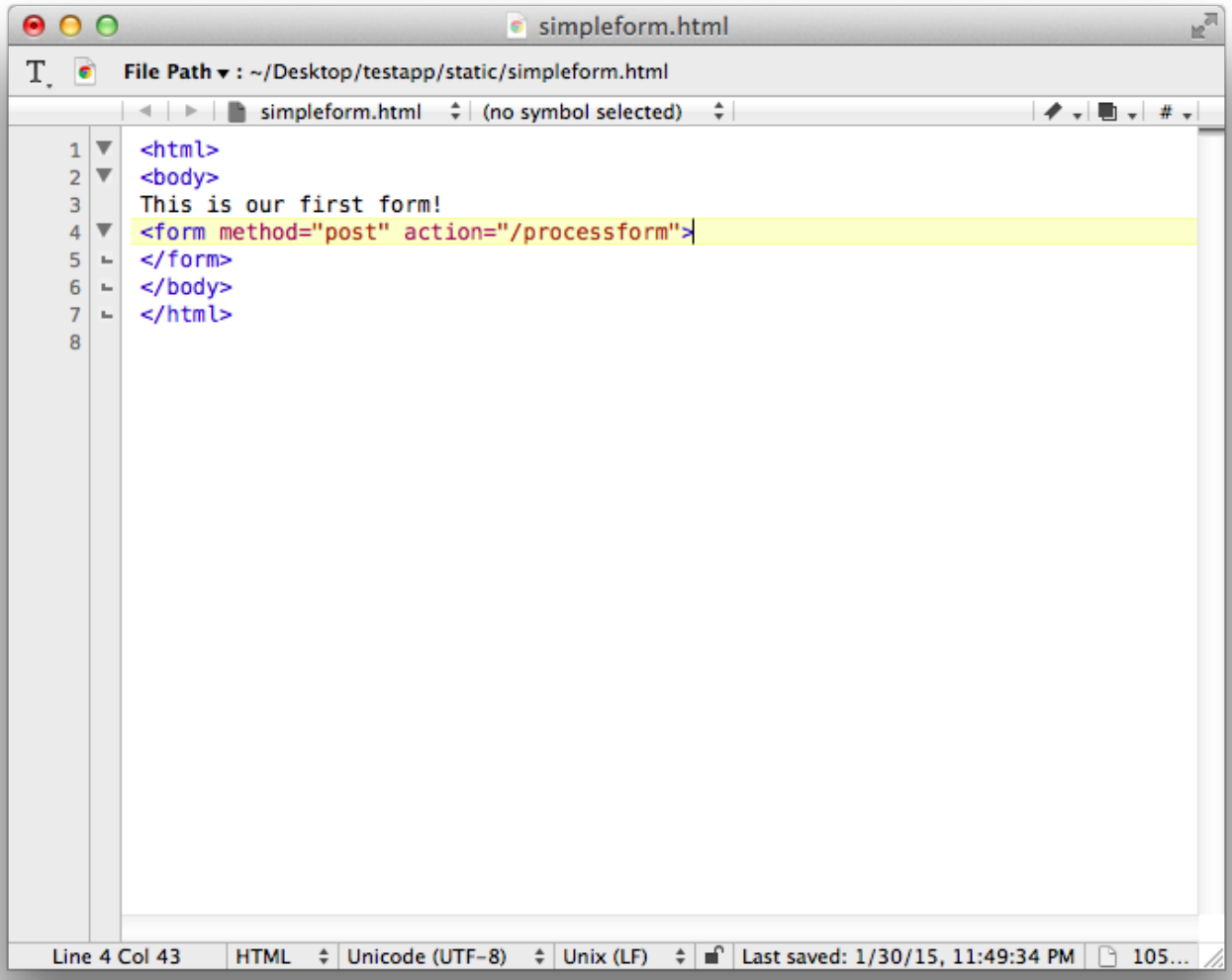
Getting Started with App Engine: Part 2

Let's start by creating a form in HTML. This part is simple. We'll create a new HTML file in our static directory. The YAML file we created in Part 1 should let us use anything in our "static" directory very easily. Let's create a file called simpleform.html and put it in the "static" directory as shown in the screenshot below:



Defining a Form

In order to create a form, we use the “form” tag (it’s aptly named). There are two interesting attributes in this tag - the method=“post” attribute, and the action=”/processform” attribute:



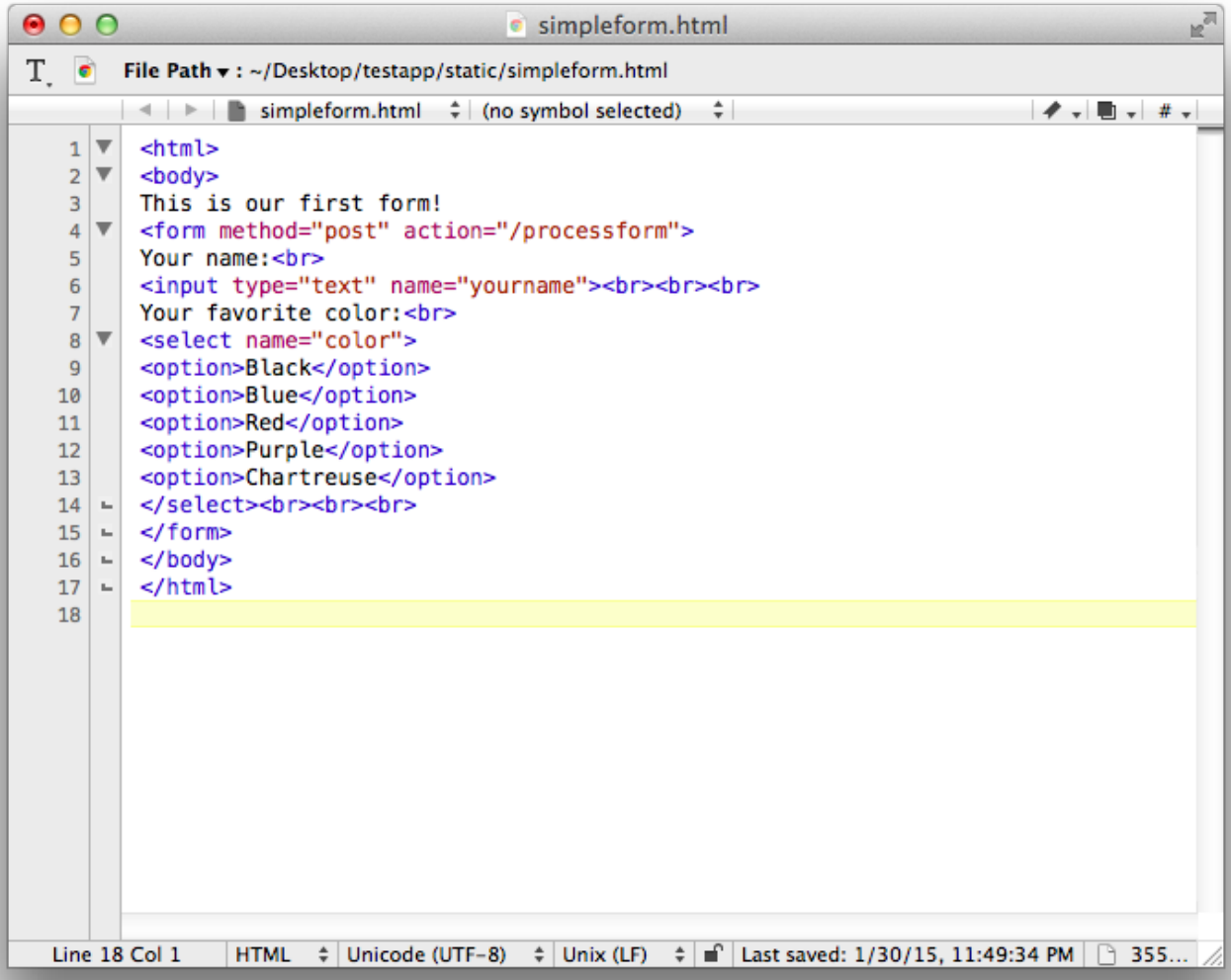
The method attribute identifies how we should submit the form; usually, this is “get” or “post” - the big difference is in parameters. When we define a few form fields here, the values from them will become parameters that the user’s browser will send back to the server when the form is submitted.

For “get” submissions, the parameters become part of the URL. If you’ve ever seen a URL that had a question mark with lots of *parameter=value* parts, that’s what an HTTP GET request looks like. https://www.google.com/?q=GET+request is an example of an HTTP GET request.

For “post” submissions, the parameters are encoded in the HTTP request, so they’re not as obvious or visible to the user.

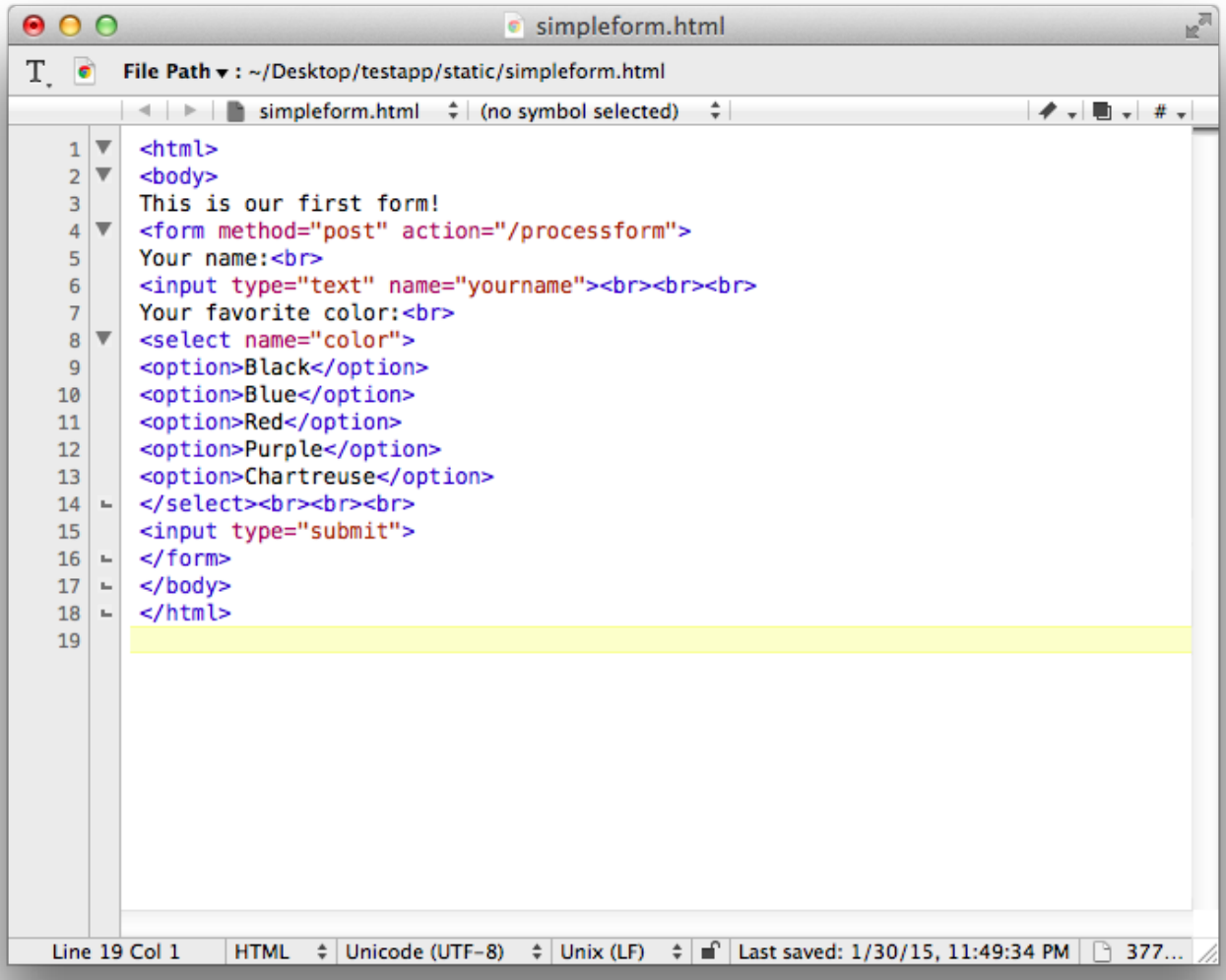
The action=”/processform” attribute is also important; that tells the browser where to send the data.

Now let’s add a few form fields:



The “input” tag here with the type=”text” attribute will show a simple text field in the user’s browser. The “select” tag will show a drop-down menu consisting of 5 colors (Black, Blue, Red, Purple, and Chartreuse).

This will get us most of the way there, but we need to provide the user with some way of submitting the form; that's where the "input" tag with a type="submit" attribute can work:



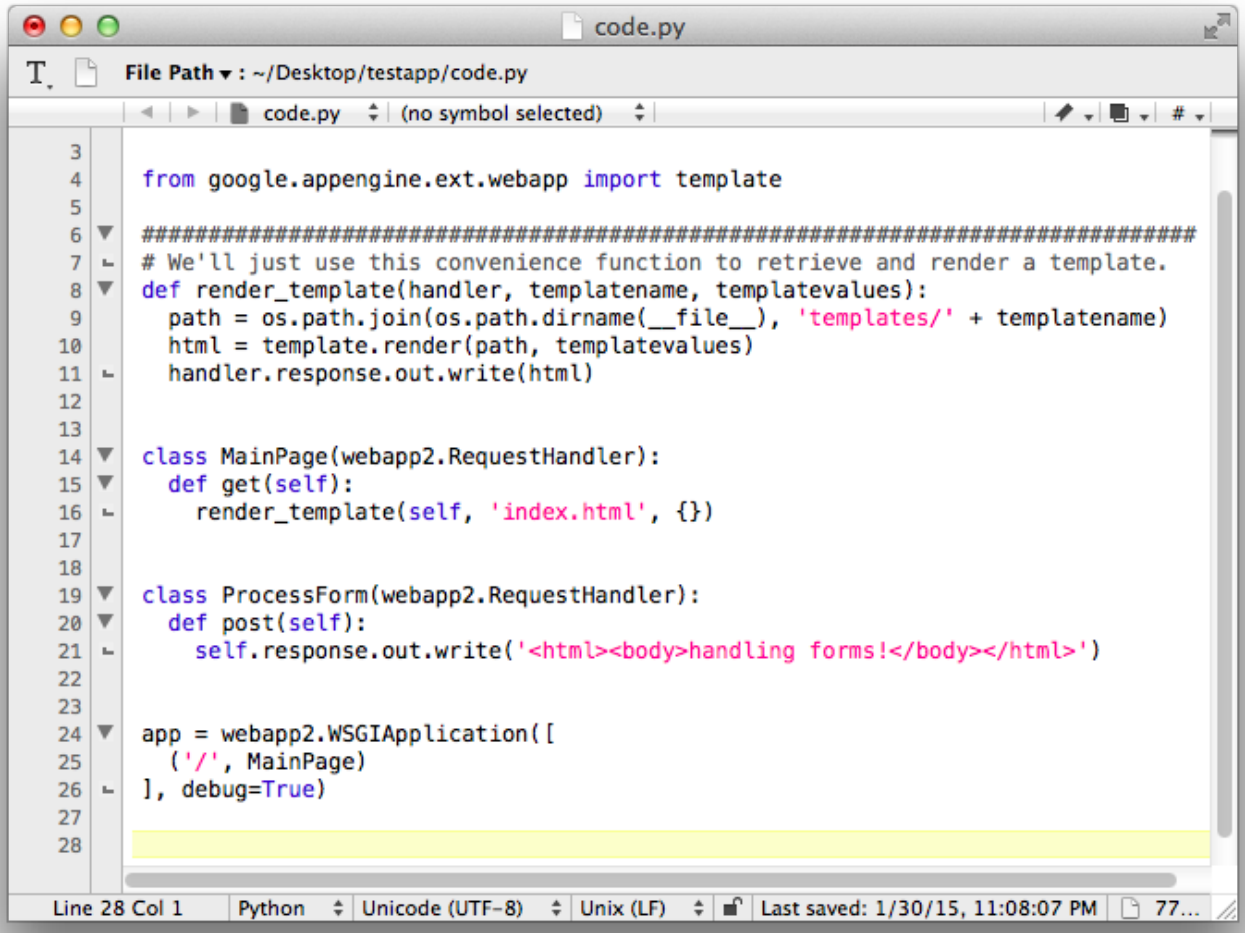
If there are no issues, you should see a simple form that looks like this:

When we submit the form, parameters called “yourname” and “color” are submitted to the server; this is handled by App Engine objects, so that we don’t have to worry about parsing HTTP requests. Note that these names match the “name” attribute in the HTML tags we created.

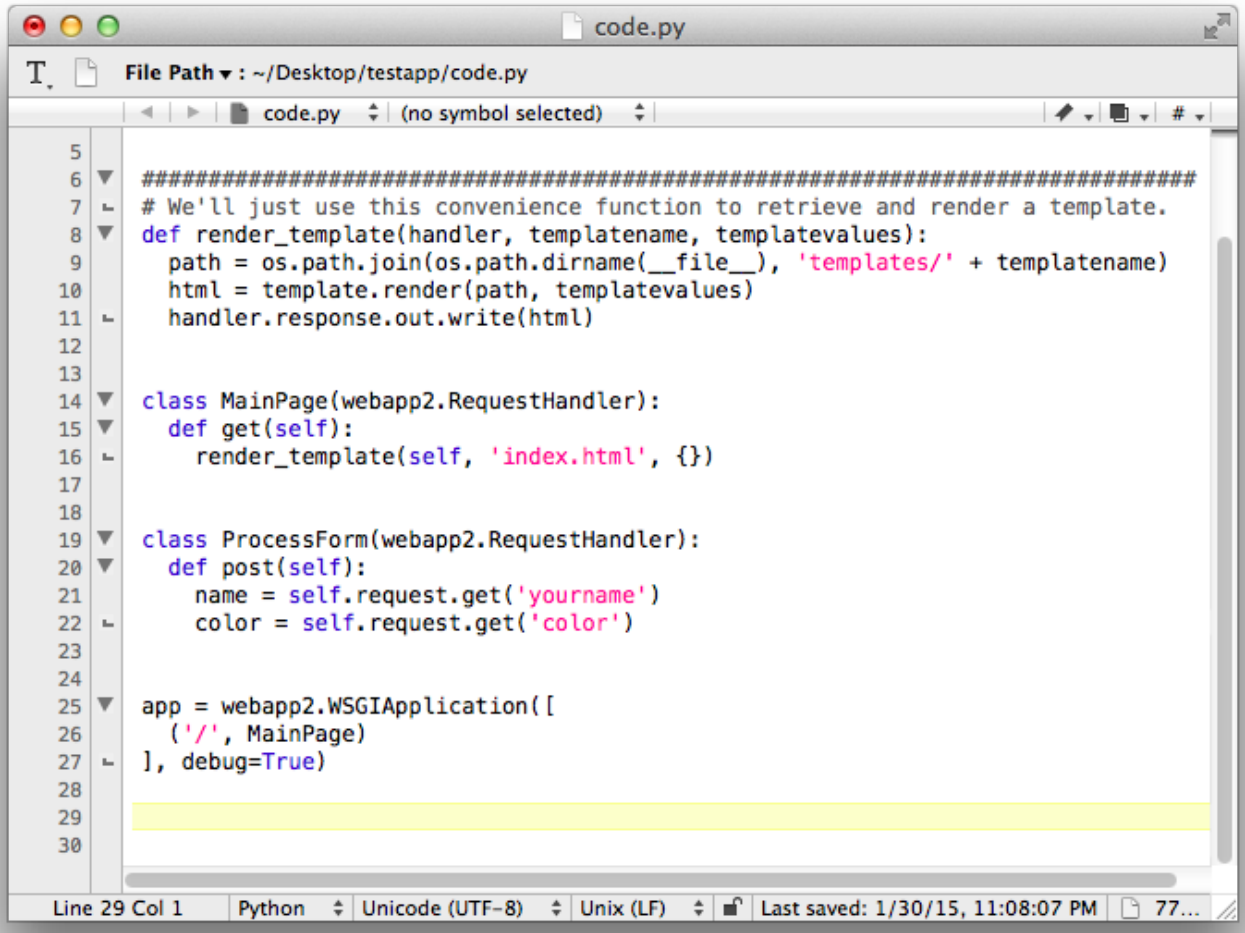
Now that we’ve created the form, let’s move on the to the next steps - creating a RequestHandler and mapping it to the right URL.

Handling Form Data

Open your code.py file to add a new RequestHandler called “ProcessForm” - note that when we created the MainPage RequestHandler, we implemented a “get” method - here we’ll implement a “post” method so that we can handle HTTP POST requests with this handler. Your code should look something like this:

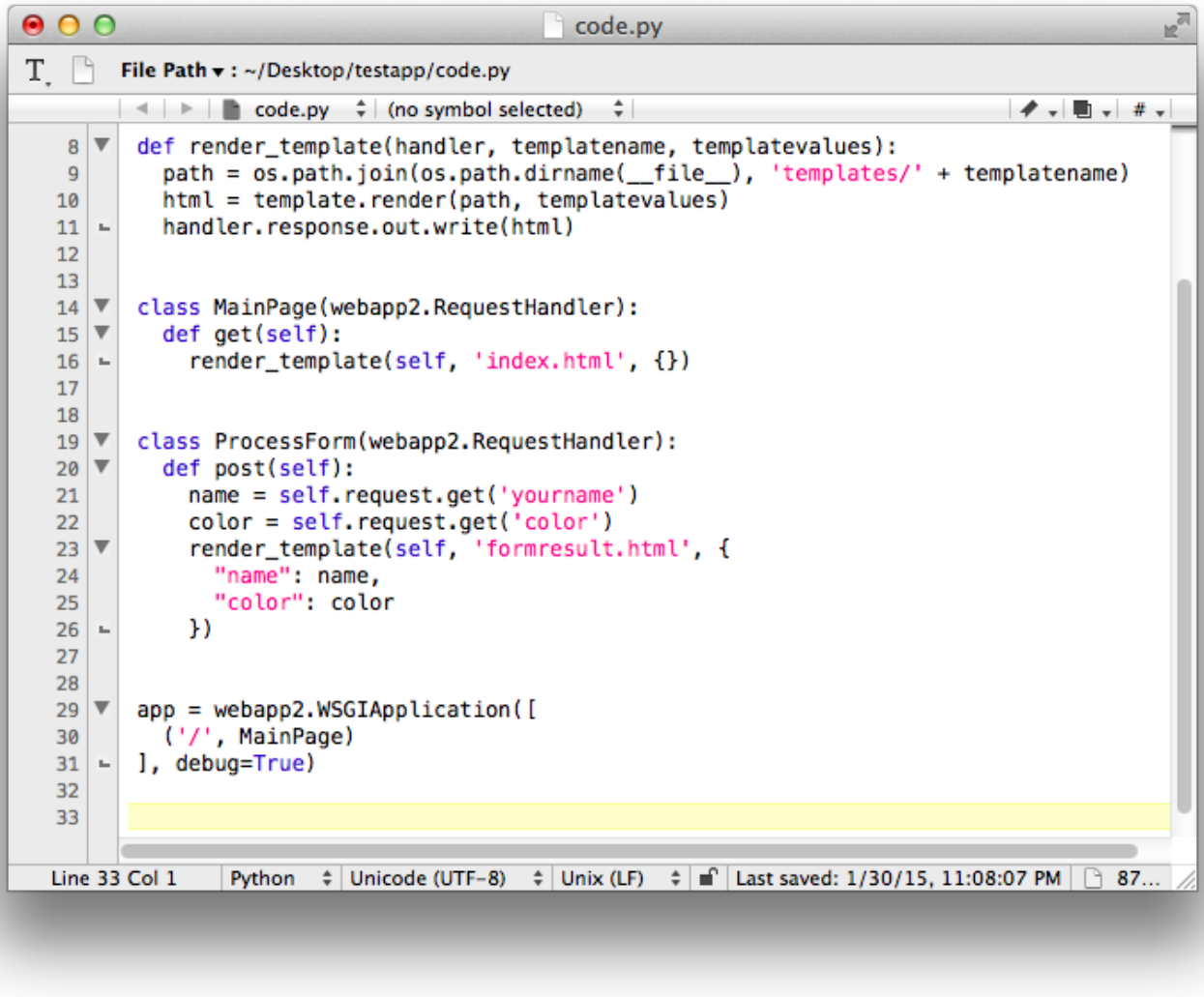


That handles the basic definition of the RequestHandler, but we'd ultimately like it to do a little more than that. We can use the self.request.get('paramname') method to retrieve actual parameters. Let's do that here:




```
5
6   ################################################################################
7   # We'll just use this convenience function to retrieve and render a template.
8   def render_template(handler, templatename, templatevalues):
9     path = os.path.join(os.path.dirname(__file__), 'templates/' + templatename)
10    html = template.render(path, templatevalues)
11    handler.response.out.write(html)
12
13
14  class MainPage(webapp2.RequestHandler):
15    def get(self):
16      render_template(self, 'index.html', {})
17
18
19  class ProcessForm(webapp2.RequestHandler):
20    def post(self):
21      name = self.request.get('yourname')
22      color = self.request.get('color')
23
24
25  app = webapp2.WSGIApplication([
26    ('/', MainPage)
27  ], debug=True)
28
29
30
```

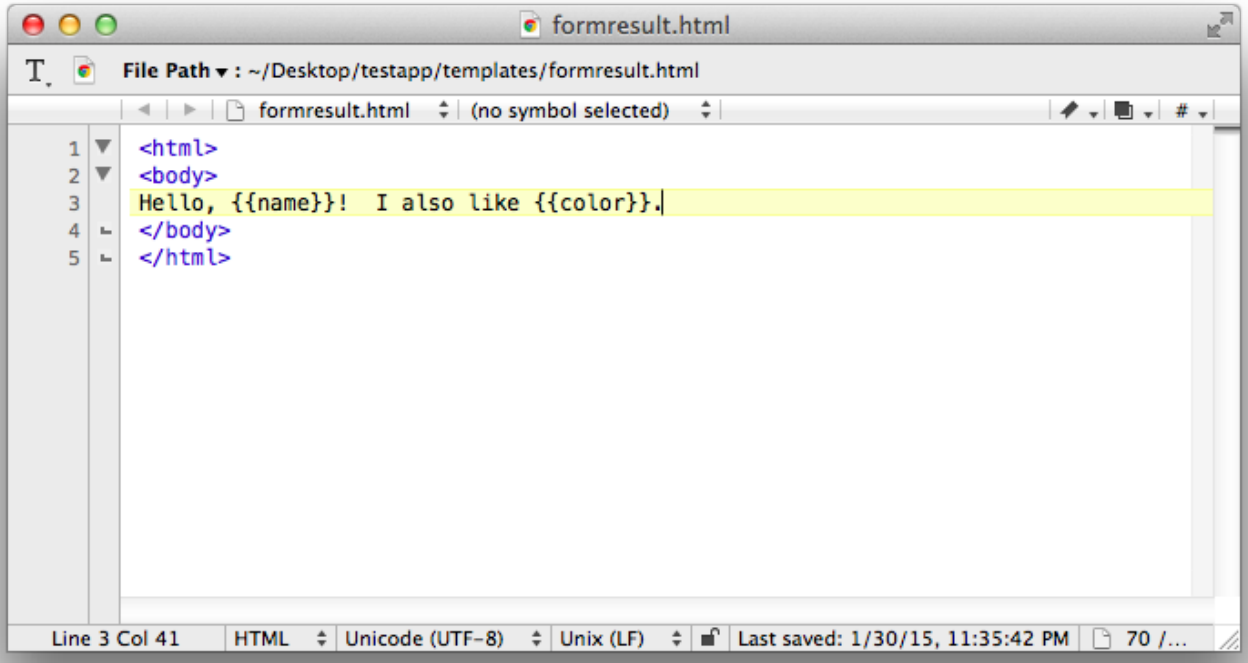
Once we have them, we'll send them to another HTML template (we'll still need to define this). Let's call that one "formresult.html" and we'll specify that here, along with the parameters:



This page defines a few pieces; we're actually creating a dictionary on the fly in order to provide the template parameters. We'll call the name parameter "name" and we'll use "color" for the color parameter.

Displaying HTML Output

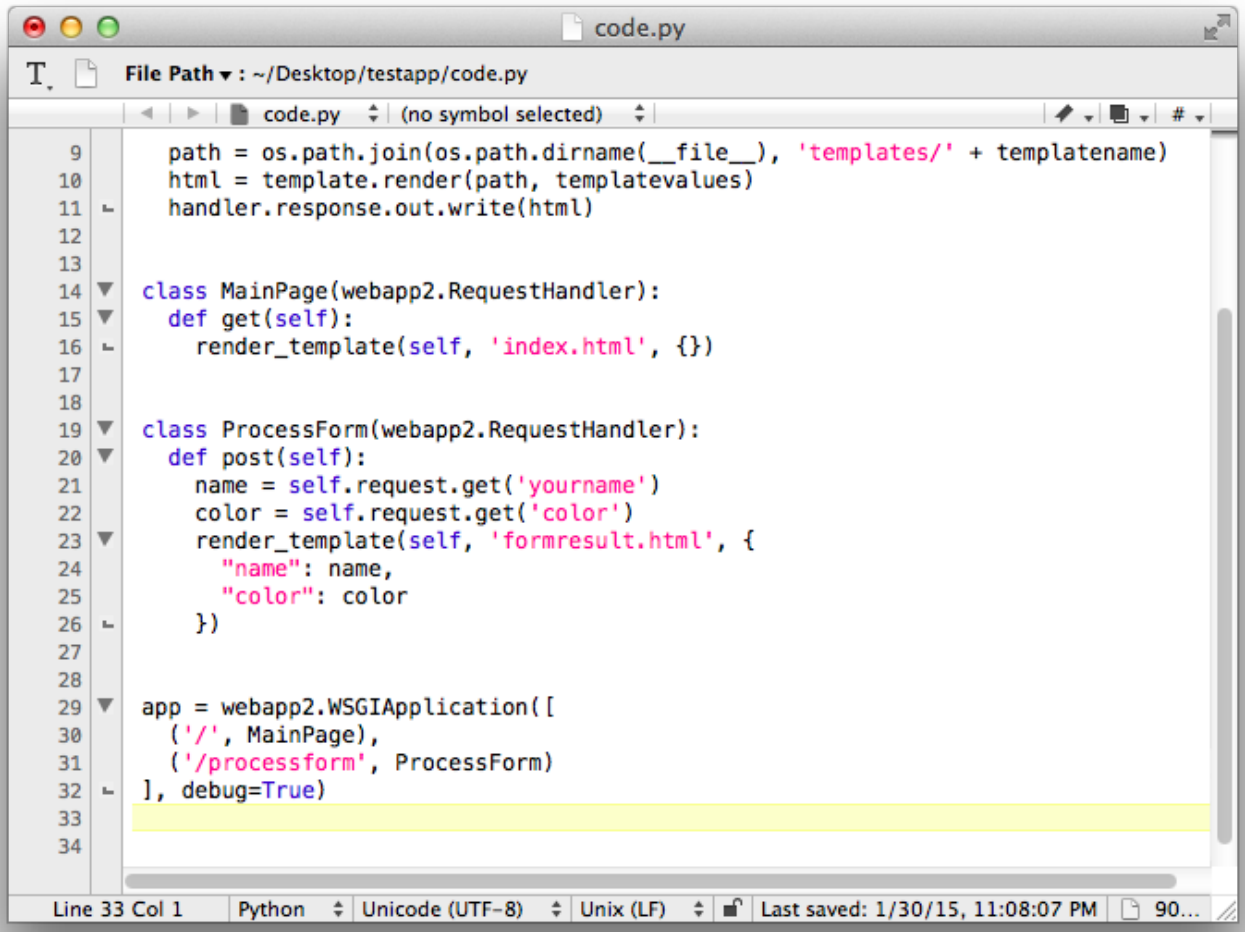
Next, we'll create the "formresult.html" template. Create this in your "templates" directory. It should look something like this:



The areas that say {{name}} and {{color}} won't be shown to the user in a literal sense; the code we use to render the template will actually insert the "name" and "color" values that we passed into the function as a dictionary.

Getting Started with App Engine: Part 2

The last step here is to map our requests. In code.py, add the mapping below (seen on line 31):



If everything works out OK, you should see something like this when you navigate to http://localhost:8080/static/simpleform.html and enter the form.

Hello, Tim! I also like Chartreuse.

That does it for this lesson!