CS253 - Unit 2: Getting and Processing User Input

Contents

1. Forms

In this unit we are going to talk about **Forms**, and how you get data from the browser to your web server.

To get us started, open the text editor of your choice and enter some text, for example:

Hello, world!

[Note, Steve is using Sublime Text 2 as his text editor in this lesson]

Save the file as play.html and then open the file in your web browser of choice.

What you will see is the text that you entered in the file displayed in your browser. If you go back to the file and change the text, then save it and refresh your web browser, you will see the changed text displayed in the browser.

This is a handy way of experimenting with HTML in a real browser.

1.1. The Form Tag

We will now introduce a new HTML tag, the FORM tag:

```
<form>
</form>
```

If we were to enter this into out text editor, save it and then open the file in our browser we wouldn't see anything. That is because there is no content in our form.

So, what about the content that can go into an HTML form? Well, there are a few things that can go into the form. One of these is the <input> element:

```
<form>
   <input>
</form>
```

If you enter this in your text editor and then open the document in the browser you should see an input box into which you can type text.

The <input> tag can take a number of attributes, one of which is called **name**. The element would have the form:

```
<input name="name">
```

1.2. Quiz: User Input

Enter the html for an input tag whose name parameter is q.

1.3. Naming Input

We can now add the name attribute to the file that we have been editing:

```
<form>
  <input name="q">
</form>
```

If we save this and refresh the file in our browser, we see that the appearance of the text box doesn't change.

1.4. Entering Input

Match the letter to the description:

Put some text in the input box in your browser and press enter. What happens?

- Nothing.
- The URL changes with my text.
- An error message Appears.
- The text disappears.

1.5. Submitting Input

Instead of requiring users to press the enter key every time, let's provide them with a button to submit our form. We can do this by using the input element, but with a new attribute, called **type**. The element will appear something like this:

```
<input type="submit">
```

This will cause the browser to show a button which a can be used to submit the form. We can add this to our form and view the result in a browser:

```
<form>
  <input name="q">
  <input type="submit">
</form>
```

1.6. Quiz: Submitting Input

Is clicking Submit any different than pressing enter?

1.7 The Action Attribute

By default, an HTML form submits to itself. While this may be handy for testing elements in a browser, isn't so handy if we want it to submit the data to our server. Let's look at how we can make the form submit to somewhere else.

We are going to add a new attribute to our form. The attribute is called **action**, and it controls where a form submits to. The attribute looks something like this:

```
<form action="/foo">
   <input …>
   <input …>
</form>
```

The value of the action attribute will be the URL of where you want the form to submit the data.

1.8 Action Attribute Quiz

Add an **action** attribute to your <form> element whose value is **http://www.google.com/search**. Click Submit. What Happens?

- Nothing.
- The URL changes with what we entered.
- My browser goes to Google search results.
- I've written an entire search engine.

1.9 URL Encoding Quiz

Type better flight search into the form and click submit. What is the value of the q parameter from the URL?

1.10 URL Encoding

A URL cannot contain spaces. If the user input into a form contains spaces, the browser has to convert that input into something that doesn't contain spaces before appending it as a query parameter. It does this by replacing spaces with the '+' character. Other reserved characters (e.g. the exclamation mark '!') are also replaced with codes in a similar manner. This is called "URL encoding" or "Form Encoding" or similar terms.

2. Hello Webapp World

Let's move on to some live web applications. By the end of the last homework you should have had Google App Engine running on your machine, and you should also have had a basic app online. We will start with the simple Hello World example that Google has on their site. This is the main Hello World Python file from the Google App Engine example page:

```
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write('Hello, webapp World!')

app = webapp.WSGIApplication([('/', MainPage)],
                                          debug=True)
```

The file has two main sections. Lets start with the section at the bottom, the URL mapping section:

```
app = webapp.WSGIApplication([('/', MainPage)],
                                      debug=True)
```

In this case, we have just the one URL, '/', which maps to a handler named MainPage.

MainPage is defined in the class in the section above, called MainPage. It inherits from webapp2.RequestHandler which is the generic request handler from Google.

[If you aren't familiar with classes, they are basically a convenient way of grouping together functions and data that are related to the same thing. You can learn more about them offline, for example here: http://docs.python.org/tutorial/classes.html]

The class MainPage is a function named **get**, which takes a parameter named **self**. (self is the common first parameter to most Python methods) The function does two things:

- First it takes self.response, which is the global response object used by this framework, and it sets the Content-Type header to be 'text/plain'. By default, the Content-Type is 'text/html', but in this case, the class sets the header to 'text/plain'.
- Next it writes the string 'Hello, webapp World!'

2.1 Content Type

Let’s take the form that we created in our text editor and put it in our application.

What we will do is create a new variable at the top of the file called form, and paste in the form from our text editor:

```
import webapp2

form = “””
<form action=”http://www.google.com/search”>
   <input name=”q”>
   <input type=”submit”>
</form>
“””

class MainPage(webapp2.RequestHandler):
   def get(self):
      self.response.headers['Content-Type'] = 'text/plain'
      self.response.out.write(form)

app = webapp.WSGIApplication([('/', MainPage)],
                                        debug=True)
```

Note the use of the **triple-quotes** which allow us to enter longer strings.

We have also replace the string 'Hello, webapp World!' with **form** so that it will print out the string of the form rather than the string 'Hello, webapp World!'.

Saving this and then displaying the file in our browser actually displays the HTML for our form, rather than the form we wanted!

2.2 Content Type Quiz

Why do we see raw HTML instead of our text box?

- Google App Engine can only send raw HTML.
- Our browser is mis-configured.
- We’re sending the wrong Content-Type header.
- Our HTML is invalid.

The reason we see the raw HTML is because of the line in our code that set the Content-Type header::

```
self.response.headers['Content-Type'] = 'text/plain'
```

If we remove this line, either by deleting it or just commenting it out, and save the file, it should display correctly in our browser.

2.3 More Handlers

Let's make some changes to our file. First, we will remove the Google search action from the form and replace it with "/testform". This will cause the form to be submitted to /testform instead of to Google when we click the Submit button.

Now, since our application can only respond to '/', we are going to need to add a handler for '/testform' as well.

'/testform' will be dealt with by a handler called TestHandler. This handler doesn't exist yet, so we will need to create it. It will have the form:

```
class TestHandler(webapp2.RequestHandler):
    def get(self):
        q=self.request.get("q")
        self.response.out.write(q)
```

What this does is to run a function called **get**, that sets a variable called q to the value self.request.get("q"). "request" is the object that represents the request that came from the browser, and you can call "get" on that object to extract different parameters. The last thing the handler does is to return the value of q to the browser.

Our app now looks like this:

```
import webapp2

form = """
<form action="/testform">
    <input name="q">
    <input type="submit">
</form>
"""

class MainPage(webapp2.RequestHandler):
    def get(self):
        #self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write(form)


class TestHandler(webapp2.RequestHandler):
    def get(self):
```

```
        q=self.request.get("q")
        self.response.out.write(q)

app = webapp2.WSGIApplication([('/', MainPage),
                               ('/testform', TestHandler)],
                               debug=True)
```

If we save this and refresh our browser, we won't see any difference in our form. However, if we type the text 'Hello, World!' into the box and click Submit, the text is displayed in the browser window with the URL something like:

http://localhost:8080/testform?q=Hello%2C+World!

Note that the exclamation mark was been encoded as %2C in this instance.

Now, I will show you something neat!

If we edit the TestHandler handler as follows:

```
class TestHandler(webapp2.RequestHandler):
    def get(self):
        #q=self.request.get("q")
        #self.response.out.write(q)

        self.response.headers['Content-Type'] = 'text/plain'
        self.response.out.write(self.request)
```

So, we have commented out the original actions and then set the Content-Type header to be text/plain, and then added the line:

```
self.response.out.write(self.request)
```

We will save this and then run it in the browser.

2.4 Handlers Quiz

What just happened?

- We see garbage.
- We see the HTTP request.
- We see a message from the future.
- Nothing changes.

You will recognise much of the request displayed in the browser from the work we did on HTTP requests in Unit 1:

```
GET /testform?q=Hello%2C+World%21 HTTP/1.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Connection: keep-alive
Host: localhost:8080
Referer: http://localhost:8080/
User-Agent: Mozilla/5.0 (Macintosh; IntelMac OS X 10_7_3)
AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.79
Safari/535.11
```

The first line is the actual request, followed by a series of headers, some of which we have met before.

The Referer header is interesting. This refers to the URL that sent the request. Notice that “Referer” is mis-spelt. In the original HTTP specification the word was spelled wrong, but it has lived on for almost 20 years for backward-compatibility reasons.

2.5 The Method Attribute

In Unit 1 we mentioned that the most important methods in HTTP requests are **get** and **post**. The default method is get. We can change the method which is used by specifying the method attribute.

We now add the attribute method=”post” to our form:

```
import webapp2

form = “””
<form method=”post” action=”/testform”>
   <input name=”q”>
   <input type=”submit”>
</form>
“””

class MainPage(webapp2.RequestHandler):
   def get(self):
      #self.response.headers['Content-Type'] = 'text/plain'
      self.response.out.write(form)

class TestHandler(webapp2.RequestHandler):
   def get(self):
      q=self.request.get(“q”)
      self.response.out.write(q)

      #self.response.headers['Content-Type'] = 'text/plain'
      #self.response.out.write(self.request)
```

```
app = webapp.WSGIApplication([('/', MainPage),
                              (‘/testform’, TestHandler)]
                              debug=True)
```

2.6 Method Attribute Quiz

What happens when we launch the file in a browser window?

- The text we entered was displayed.
- An error was displayed.
- The request headers were displayed.
- Ran a Google search with our request.

The error occurs because we are using the method post for our form, but TestHandler is only configured to use get! We can change TestHandler to use post as follows:

```
class TestHandler(webapp2.RequestHandler):
    def post(self):
        q=self.request.get(“q”)
        self.response.out.write(q)
```

Now, when we refresh the browser, the text entered in the text box is displayed just like before, except that the q parameter isn’t shown in the URL.

2.7 Methods and Parameters Quiz

Where did our q parameter go?

(Hint: Look at the request)

- In the URL.
- After the HTTP response headers.
- In a second request.
- In the HTTP request headers.

2.8 Differences Between Get and Post

Let’s talk a little bit about the differences between get and post.

We have already seen one difference in that get parameters are included in the URL while post parameters are included the body. Another difference is that get requests are normally used to fetch documents and get parameters describe the document to be fetched, while post is often used for updating data on the server.

Because get parameters form part of the URL, they are subject to the maximum length of URL permitted in the browser whereas posts, by default, have no maximum length.

A further difference is that get parameters are generally OK to cache. Post parameters are almost never cached as they are probably updating data on the server.

Get requests shouldn't change the data on the server. You can make the same get request multiple times and the server shouldn't change. Post requests, on the other hand, are OK to change the server. That is what they are generally used for.

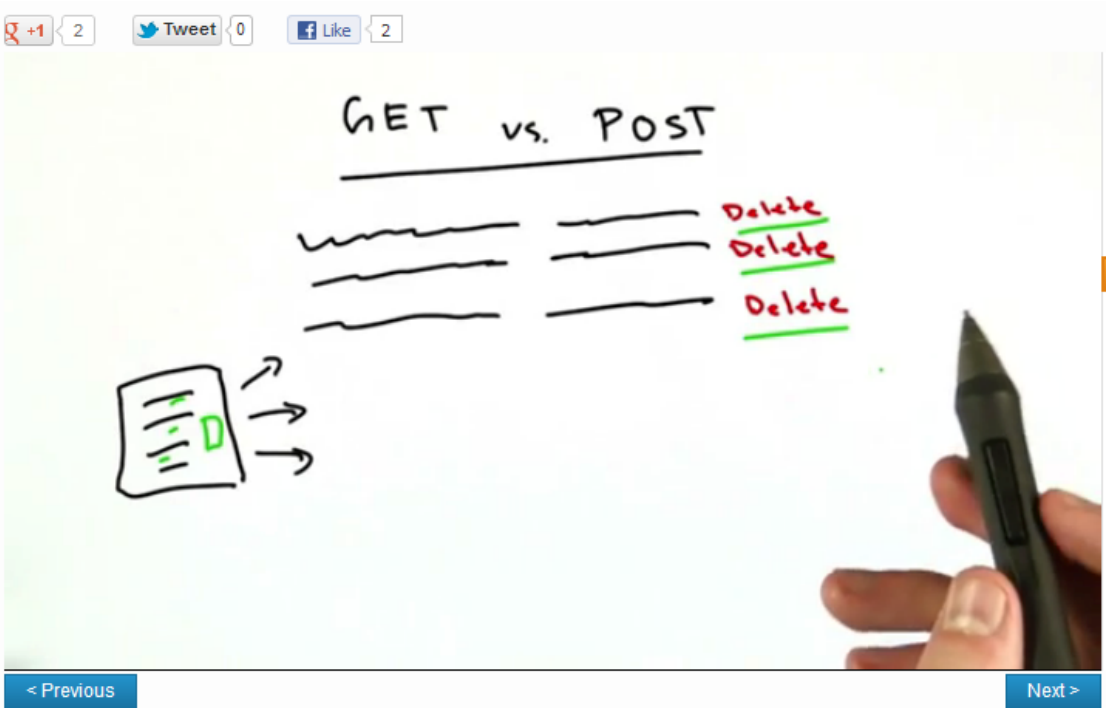
The differences are summarised below:

GET	POST
- Parameters in URL	- Parameters in body
- Used for fetching documents	- Used for updating data
- Maximum URL length	- No maximum length
- OK to cache	- Not OK to cache
- Shouldn't change the server	- OK to change the server

2.9 Problems With Get

We've talked about some of the differences between get and post, now let's look at what happens if you follow the rules for using get and post correctly.

Some years ago there was an online program for organising to-do lists and similar tasks called Basecamp, made by a company called 37Signals. Basecamp displayed the to-do list with a 'Delete' link alongside each item in the list.



Now, as we have seen, a normal link using the anchor tag makes get requests. These had been designed to make post requests to delete the item on the to-do list.

Another program which was out at that time was Google Web Accelerator. This was a browser plug-in that sat in the browser and while you were browsing the Google Web Accelerator would make the requests for any links on the page behind the scenes so that when you clicked the links the pages were ready to go.

The problem in this case was that the Google Web Accelerator was hitting the 'Delete' links in Basecamp, so users would go to their Basecamp page, look at their to-do list, and find that their items were deleting on their own. This was a major problem, and an example of what can go wrong if you use get requests to update data on the server.

2.10 When to Use Post and Get Quiz

Which of these are appropriate for GET requests?

- Fetching a document.
- Updating data.
- Requests that can be cached.
- Sending a lot of data to the server.

2.11 Passwords

We now want to introduce some more input types. The default type for an input element is "text". We left this out earlier, but we should really be specific with our types, especially as we are now going to introduce some more types.

If we go back to our basic play.html document, adding the type="text" attribute to the first input element gives us:

```
<form>
   <input type="text" name="q">
   <input type="submit">
</form>
```

The first new input type that we want to introduce is type="password". When the input field is a password, any text typed in the field appears as a row of dots:

```
<form>
   <input type="password" name="q">
   <input type="submit">
</form>
```

2.12 Passwords Quiz

When we submit this form, what is the value of the q parameter in the URL?

- Whatever I typed in the box.
- The parameter won't appear at all.
- hunter2

It is important to be aware that the password entered in a password field will appear in the URL and it not sent securely to the server.

2.13 Checkboxes

Another input type is type="checkbox". You won't be surprised to learn that this will create a checkbox in your form:

```
<form>
  <input type="checkbox" name="q">
  <input type="submit">
</form>
```

If the checkbox is checked, the parameter appended to the URL will be q=on

2.14 Checkboxes Quiz

What is the value of the q parameter when I submit the form with the checkbox unchecked?

- Off.
- The parameter doesn't appear at all.
- The parameter is blank.
- hunter2.

2.15 Multiple Checkboxes.

Now we have seen how a single checkbox works, but you rarely see a single checkbox. They more commonly appear in groups. In the form below I have added two more checkboxes, named r and s, and also added a line break so we have a little separation from the Submit button:

```
<form>
   <input type=”checkbox” name=”q”>
   <input type=”checkbox” name=”r”>
   <input type=”checkbox” name=”s”>
   <br>
   <input type=”submit”>
</form>
```

2.16 Multiple Checkboxes Quiz

When I select the first two checkboxes and click submit, what does the query section of the URL look like?

2.17 Radio Buttons

Radio buttons are also often used in forms. In this case the type=”radio”. The form below will display three radio buttons named q, r and s:

```
<form>
   <input type=”radio” name=”q”>
   <input type=”radio” name=”r”>
   <input type=”radio” name=”s”>
   <br>
   <input type=”submit”>
</form>
```

When you load this in the web browser you will see three radio buttons. However, they do not behave as you would expect them to. Normally when we have radio buttons they behave as a group. and you can only select one button. What we have here are essentially checkboxes that cannot be unchecked.

2.18 Grouping Radio Buttons Quiz

Which of these yields the “group” behaviour from the radio buttons?

- Give them all the same **id**.
- Give them all the same **name**.
- Give them all the same **group**.
- include the **<input>**s in a **<group>** element.

2.19 Radio Button Values

So, if we give all the radio buttons in a group they will behave as we expect radio buttons to do:

```
<form>
   <input type=”radio” name=”q”>
   <input type=”radio” name=”q”>
   <input type=”radio” name=”q”>
   <br>
   <input type=”submit”>
</form>
```

However, a problem with this set up as it stands is that the parameter passed to the server will be q=on, whichever of the 3 buttons is selected. We can solve this problem using the **value** parameter:

```
<form>
   <input type=”radio” name=”q” value=”one”>
   <input type=”radio” name=”q” value=”two “>
   <input type=”radio” name=”q” value=”three “>
   <br>
   <input type=”submit”>
</form>
```

2.20 Radio Button Values Quiz

What is the value of the of the **q** parameter when the form is submitted with the second button selected?

2.21 Label Elements

In the previous example, we had our three radio buttons, but how is a user to know what the buttons represent and which button to select? That is what the label element is for.

In order to label each button we need to enclose it in a label element:

```
<label>
   One
   <input type=”radio” name=”q” value=”one”>
</label>
```

This will cause the label (in this case “One”) to appear next to the button. Our complete form will now be:

```
<form>
   <label>
      One
      <input type=”radio” name=”q” value=”one”>
   </label>
   <label>
      Two
      <input type=”radio” name=”q” value=”two”>
   </label>
   <label>
      Three
      <input type=”radio” name=”q” value=”three”>
   </label>
   <br>
   <input type=”submit”>
</form>
```

The label doesn’t have to match the value of the radio button, but they often do (or at least are often closely related).

2.22 Dropdowns

There is one last form element I would like to expose you to before we move on, and this is the dropdown.

A drop down has a form that looks like this:

```
<select name="q">
  <option>One</option>
  <option>Two</option>
  <option>Three</option>
</select>
```

The dropdown starts with a **select** element with a name attribute (we have been using **q**, so why break a winning streak), followed by a number of **option** elements, and finally the closing **select** element. Our form will now appear as:

```
<form>
  <select name="q">
    <option>One</option>
    <option>Two</option>
    <option>Three</option>
  </select>
  <br>
  <input type="submit">
</form>
```

2.23 Dropdowns Quiz

When we select **Two** from the dropdown, what is the value of the **q** parameter?

2.24 Dropdowns and Values

I you want to have more descriptive text in the dropdown, but a different parameter to appear in the URL, you can use the **value** parameter.

```
<form>
  <select name="q">
    <option value="1">Number One</option>
    <option value="2">Number Two</option>
    <option value="3">Number Three</option>
  </select>
  <br>
  <input type="submit">
</form>
```

Now, the text displayed in the dropdown will be Number One, Number Two…, but if we select Number Two from the list the query parameter attached to the URL will be simply q=2.

2.25 The Number One Quiz

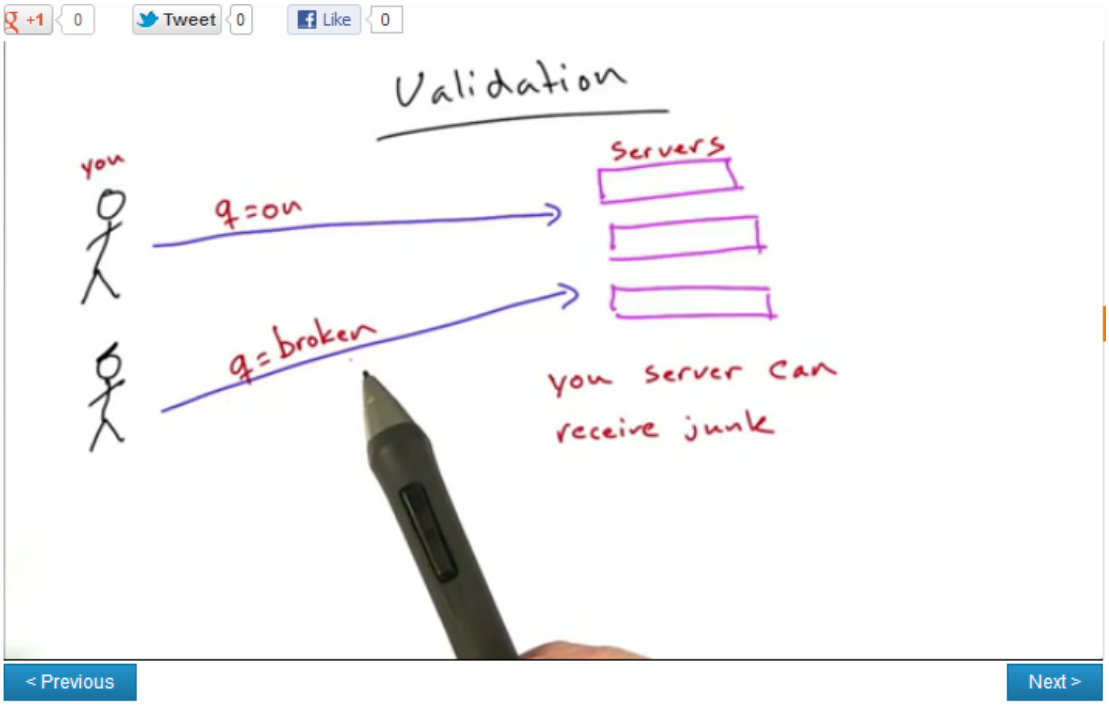
What will **q**'s value in the URL be when we submit the form with the value Number One selected?

- Number One
- 1
- Nothing
- hunter2

3. Validation

Let’s introduce a new concept. The concept of validation. Validation basically means verifying on the server-side that what we received is what we expected to receive.

Imagine the situation where we have set up our form, and the servers are programmed to respond in a particular way when q=on, and in another way when q is absent. But how will they respond if they receive a request where q=broken? The smart thing to do would probably be to treat the request as if q was unchecked in this case.



However, the point remains that just because we have designed a form to limit the responses that can be sent to our servers doesn’t mean that a rogue user can’t send a request directly to our servers with almost any arbitrary junk in it. It is up to us to make sure our servers can handle it.

3.1 What is Your Birthday?

Let’s go back to our very simple form as shown below:

```
import webapp2

form = “””
<form method=”post”>
   <input name=”q”>
   <input type=”submit”>
</form>
“””

class MainPage(webapp2.RequestHandler):
   def get(self):
      self.response.out.write(form)

app = webapp.WSGIApplication([('/', MainPage)], debug=True)
```

Notice that we removed the action /testform and so no longer need the handler which dealt with /testform.

Instead of a form with just a text-field, we are going to modify the form to ask for the user’s birthday, and then validate their inputs. The form now looks like this:

```
<form method=”post”>
   What is your birthday?
   <br>
   <label> Month
      <input type=”text” name=”month”>
   </label>
   <label> Day
      <input type=”text” name=”day”>
   </label>
   <label> Year
      <input type=”text” name=”year”>
   </label>
   <br><br>
   <input type=”submit”>
</form>
```

3.2 What is Your Birthday? Quiz

What happens when I hit Submit?

- The form clears.
- We get an **Error 405** because we never added a **post()** handler for **/**.
- We see the form values in the URL.
- The app counts down to my birthday.

3.3 Handling Posts

So let's add a post function to our page.

```
import webapp2

form = """
<form method="post">
  What is your birthday?
  <br>
  <label> Month
    <input type="text" name="month">
  </label>
  <label> Day
    <input type="text" name="day">
  </label>
  <label> Year
    <input type="text" name="year">
  </label>
  <br><br>
  <input type="submit">
</form>
"""

class MainPage(webapp2.RequestHandler):
  def get(self):
    self.response.out.write(form)

  def post(self):
    self.response.out.write("Thanks! That's a totally valid day!")


app = webapp.WSGIApplication([('/', MainPage)], debug=True)
```

This will return the string "Thanks! That's a totally valid day!" to the browser when we post from the form, but what if the date isn't valid?

3.4 Handling Bad Data Quiz

What are some possible solutions to users entering bad data into our form?

- Use dropdowns to limit what users can actually enter.
- Assume that users will only enter good data.
- Verify what the users enter and complain if the data is bad.
- Make values up.

3.5 Valid Month Quiz

Write a function that, given input from the user, returns whether or not it is a valid month.

3.6 Valid Day Quiz

Next, I'd like you to write a function that, given what the user types in for the day, returns whether that day is valid or not.

3.7 Valid Year Quiz

OK, one last function to write, this time to determine whether what the user entered for the year is valid.

3.8 Checking Validation

Now that we have the functions to check the users input, let's look at how they fit into the process.

We have to do three things:

1. Verify the user's input.
2. On error, render the form again.
3. Include an error message

3.9 Responding Based on Validation

We can now modify our post function to use the validation functions that we have just written and respond to the request appropriately based on the results of the validation. The new post function looks like this:

```
def post(self):
    user_month = valid_month(self.request.get('month'))
    user_day = valid_day(self.request.get('day'))
    user_year = valid_year(self.request.get('year'))

    if not(user_month and user_day and user_year):
        self.response.out.write(form)
    else:
        self.response.out.write("Thanks! That's a totally valid day!")
```

The first line of the function calls the valid_month function that we just wrote, and sends the month parameter from the request as the function parameter. Similarly, the next two lines validate the day and the year of the user input. If these are not all valid, the function resends the form. If they are all valid, the function writes the string "Thanks! That's a totally valid day!".

4 String Substitution

Before we go any further, I want to teach you a quick little Python thing about how to do string substitution. This will make generating our HTML a little bit more convenient.

If we have a string in Python something like this:

“ some bold text”

This is exactly the kind of string that we are likely to be returning in our web app – a little bit of HTML with some contents. If we have a lot of bold text, this will get to be rather a pain if we have to generate this entire string every time. Instead, we can do something like this:

“%s”%VARIABLE

We still have the same basic structure of the string with the tags creating some bold contents, but now that contents is represented by the %s. What this does is to substitute the %s with the variable %VARIABLE following the string. This is really convenient as we can now have a function that substitutes some variable into the string meaning that we don’t have to keep building the string over, and over again.

4.1 String Substitution Quiz

Write a function 'sub1' that, given a string, embeds that string in the string: "I think X is a perfectly normal thing to do in public." where X is replaced by the given string.

If we wanted to substitute multiple strings, the procedure is very similar and will have a format similar to this:

“text %s text %s” %(VAR1, VAR2)

We can have multiple instances of %s, but we also need multiple variables. If you don’t include enough variable, Python may get mad at you!

4.2 Substituting Multiple Strings Quiz

Write a function 'sub2' that, given two strings, embeds those strings in the string: "I think X and Y are perfectly normal things to do in public.", where X and Y are replaced by the given strings.

4.3 Advanced String Substitution

What happens if you want to include the same variable twice in the same string? We can do this by including a name identifier in parentheses:

“text %(NAME)s text” %{“NAME: value}

Now, instead of just having the variables at the end of the string, we can include a dictionary that maps NAME to a value. NAME can appear in the string multiple times, and we can have multiple names.

4.4 Advanced String Substitution Quiz

Write a function 'sub_m' that takes a name and a nickname, and returns a string of the following format: "I'm NICKNAME. My real name is NAME, but my friends call me NICKNAME."

4.5 Substituting Into Our Form

Let’s improve our form by making it a little more user friendly if the user enters invalid data.

First, let’s modify the form itself to include a placeholder for the error message. We will use a element as shown, and inside the we will use string substitution to print our error message:

```
<form method=”post”>
   What is your birthday?
   <br>
   <label> Month <input type=”text” name=”month”></label>
   <label> Day <input type=”text” name=”day”></label>
   <label> Year <input type=”text” name=”year”></label>
   <div style=”color: red”>%(error)s</div>
   <br><br>
   <input type=”submit”>
</form>
```

Notice that we have used a simple css format attribute to make text within the (the error message) appear in red.

Since we will be writing the form in a couple of places, it makes sense to create a function and then call the function when we need to write the form:

```
def write_form(self, error=“”):
  self.response.out.write(form %{“error”: error})
```

The first parameter for all the functions inside a class should be **self**. The default value of the **error** parameter taken by the function is the empty string “”. We can now use this function in the get and post functions where we need to write our form:

```
import webapp2

form = """
<form method="post">
   What is your birthday?
   <br>
   <label> Month <input type="text" name="month"></label>
   <label> Day <input type="text" name="day"></label>
   <label> Year <input type="text" name="year"></label>
   <div style="color: red">%(error)s</div>
   <br><br>
   <input type="submit">
</form>
 """

class MainPage(webapp2.RequestHandler):
   def write_form(self, error=""):
      self.response.out.write(form %{"error": error})

   def get(self):
     self.write_form()

   def post(self):
     self.write_form("That doesn't look valid to me, friend.")


app = webapp.WSGIApplication([('/', MainPage)], debug=True)
```

We have replaced the default empty string with our desired error message in the post function.

4.6 Preserving User Input

It would also be nice if, when we entered invalid data, the form left our original data so that we don't have to re-enter everything again. There is a way to do that, but first we need to learn a little bit more HTML.

We have seen the value attribute applied to some form elements, now we are going to consider how it works with the text input element. A text input element has the form:

```
<input type="text" value="cool">
```

The value of the value attribute (“cool” in this example) is the default value for this element. This is the value that will appear in the text box when it is rendered in the browser. This could be used to render the form with the values that the user just typed in, so that they don’t have to keep re-typing them. You can also use this to fill out form fields with default values if you know what the user is likely to enter into the box.

4.7 Preserving User Input Quiz

Which of theses is a correct input for preserving the user’s month?

- <input value=”%(month)s”>
- <input name=”month”>%(month)s</input>
- <input name=”month”>
- <input name=”month” value=”%(month)s”>

4.8 Problems With HTML Input

We are now in a position to modify our code the preserve the user inputs:

```
import webapp2

form = “””
<form method=”post”>
   What is your birthday?
   <br>
   <label> Month <input type=”text” name=”month” value=”%(month)s”></label>
   <label> Day <input type=”text” name=”day” value=”%(day)s”></label>
   <label> Year <input type=”text” name=”year” value=”%(year)s”></label>
   <div style=”color: red”>%(error)s</div>
   <br><br>
   <input type=”submit”>
</form>
 “””

class MainPage(webapp2.RequestHandler):
   def write_form(self, error=“”, month=””, day=””, year=””):
      self.response.out.write(form %{“error”: error,
                                     “month”: month,
                                     “day”: day,
                                     “year”: year })

   def get(self):
     self.write_form()

   def post(self):
      user_month = self.request.get(‘month’)
```

```
        user_day = self.request.get('day')
        user_year = self.request.get('year')

        month = valid_month(user_month)
        day = valid_day(user_day)
        year = valid_year(user_year)

        if not(month and day and year):
            self.write_form("That doesn't look valid to me, friend.", user_month,
                            user_day, user_year)
        else:
            self.response.out.write("Thanks! That's a totally valid day!")

app = webapp.WSGIApplication([('/', MainPage)], debug=True)
```

So, we have added values to our text input fields, and edited the write_form function (and its associated dictionary) to use the new parameters, **month**, **day**, and **year**.

Next, we edited the post function to use the new variable. To do this, we had to restructure the function and split the act of fetching the parameter out of the request from the act of testing the validity of the user data. We also had to change the validity check to use the correct updated variables.

Lastly, we added the variables **user_month**, **user_day** and **user_year** to the write_form call in the post function.

4.9 Problems With HTML Input Quiz

Suppose I enter the text **bar”>what!** into one of the fields.

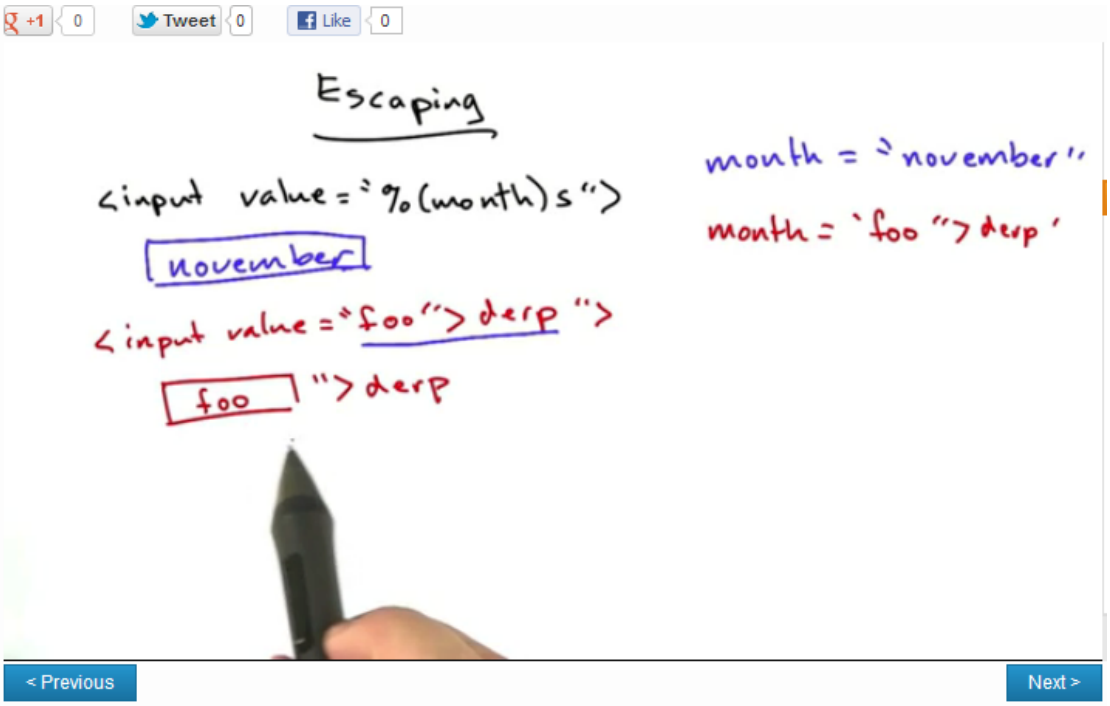
What happens when I click submit?

- We see an error message and our input is in the form.
- We see an error message, but the quote messes up our HTML.
- There’s an error on the server side.
- The page renders blank.

4.10 Handling HTML Input

Consider the code that creates the month field in our form:

<label> Month <input type=”text” name=”month” value=”%(month)s”></label>



If we enter the text **November** into the month field is, the python code renders the value attribute to give the input field:

<input type=”text” name=”month” value=”November”>

Which is what we would expect.

However, if we entered **foo”>derp** into the field, the Python code renders the attribute to give the field:

<input type=”text” name=”month” value=”foo”>derp”>

The section of the element highlighted in red will be interpreted by the browser as the complete HTML tag!

Clearly, this isn’t the behaviour we want. Worse than that, it allows an unscrupulous user who knows how our website works to enter HTML into the field and compromise the operation of the site. This is something we can’t allow.

4.11 HTML Escaping

The problem is that we want to prevent the characters **“>** being used in our inputs. We can check for this, but the nicest way to fix this is through a method called **escaping**.

HTML allows you to change certain characters, or “escape” them. so that when the user types in the quote mark we can still show it in the text box. What we do is, instead of returning the quote in the HTML, we convert it to **"**

Similarly, we can convert the closing angle-bracket, >, into **>**

Some common escape characters are shown below:

“	"
>	>
<	<
&	&

Using these escape codes what is displayed is the relevant symbol, but they aren’t actually HTML.

4.12 Using HTML Escaping

Let’s take a look at some of these substitutions in practice.

Type the following text into the text editor of your choice and save it as an html file:

What if I want to talk about HTML in HTML?

When you open the file in your browser you will see the text just as you typed it. Simple text is rendered as text in the browser. No surprises there.

Now, suppose you enclose the first HTML in angle brackets:

What if I want to talk about <HTML> in HTML?

When you load this into your browser, the <HTML> is not displayed. This is because your browser will have interpreted it as an HTML tag. However, if we use the escape codes from the table above:

What if I want to talk about <HTML> in HTML?

The text will now appear as we wanted it to, with the angle brackets in place, in the browser window.

4.13 Using HTML Escaping Quiz

What is the correct way to include the text **&=&** in rendered HTML?

4.14 Implementing HTML Escaping Quiz

Implement the function escape_html(), which replaces :

> with >
< with <
“ with "
& with &

4.15 Problems Reinventing the Wheel

In fact, Python has a built in function to implement HTML escaping. in the cgi module. We can therefore implement the function much more easily as follows:

```
import cgi

def escape_html(s):
    return cgi.escape(s, quote = True)
```

Generally, it is a good idea to use the built in functions since they are often better and normally bug free. As Steve says: “A word to the wise: use the built-in pre-written escape functions whenever you can.”

4.16 Current Limitations

Now that we have the escape_html() function we can use it in the write_form() function of our form application:

```
def write_form(self, error="", month="", day="", year=""):
    self.response.out.write(form %{"error": error,
                                   "month": escape_html(month),
                                   "day": escape_html(day),
                                   "year": escape_html(year) })
```

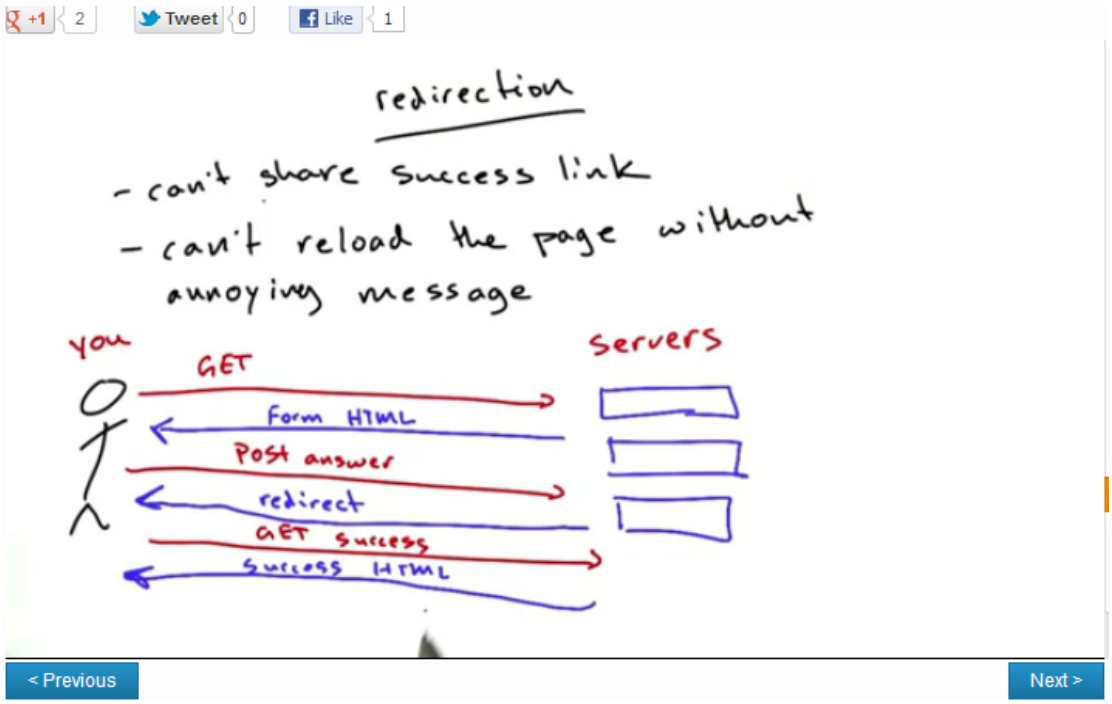
As you can see, we have applied the escape_html() function to each of the user inputs.

Now that we have the error handling more or less sorted, what can we do to improve the behaviour of the application when the user enters valid data?

In its current form the result page displayed when the user enters valid data has a number of limitations. Firstly, you can't link to it. Secondly, if you try to reload the page, your browser will display a message warning you that it needs to re-submit the data to refresh the page. This is because you are trying to reload a page that was created using a post.

4.17 Redirection

The way to work around the problems we were having with the success page is to use a redirect. Instead of rendering the result in a post, we send them to another page that says "Thanks!".



If the user's post is successful, the server sends a redirect message that causes the browser to get the "success" page.

4.18 Redirection Advantages Quiz

Why is it nice to redirect after a form submission?

- Because POSTs can't return HTML.
- So that reloading the page doesn't resubmit the form.
- To remove the form parameters from the URL.
- So we can have distinct pages for forms and success pages.

4.19 Implementing Redirection

Let's make the change to our application.

To make the change, we need to do three things. We need to:

1. make a "thanks" handler.
2. add the /thanks URL.
3. redirect to the /thanks URL.

The handler will look like this:

```
class ThanksHandler(webapp2.RequestHandler):
   def get(self)
      self.response.out.write("Thanks! That's a totally valid day!")
```

Next, we add the /thanks URL to the mapping area:

```
app = webapp.WSGIApplication([('/', MainPage), ('/thanks'.ThanksHandler)
                                          debug=True),
```

Now we can modify the post() function to redirect to the /thanks URL:

```
def post(self):
    user_month = self.request.get('month')
    user_day = self.request.get('day')
    user_year = self.request.get('year')

    month = valid_month(user_month)
    day = valid_day(user_day)
    year = valid_year(user_year)

    if not(month and day and year):
       self.write_form("That doesn't look valid to me, friend.", user_month,
                       user_day, user_year)
    else:
       self.redirect("/thanks")
```

This gives the final version of our application below:

```
import webapp2

form = """
<form method="post">
   What is your birthday?
   <br>
   <label> Month <input type="text" name="month" value="%(month)s"></label>
   <label> Day <input type="text" name="day" value="%(day)s"></label>
   <label> Year <input type="text" name="year" value="%(year)s"></label>
   <div style="color: red">%(error)s</div>
   <br><br>
   <input type="submit">
</form>
 """

class MainPage(webapp2.RequestHandler):
   def write_form(self, error="", month="", day="", year=""):
      self.response.out.write(form %{"error": error,
                                     "month": escape_html(month),
                                     "day": escape_html(day),
                                     "year": escape_html(year) })


   def get(self):
     self.write_form()

   def post(self):
      user_month = self.request.get('month')
      user_day = self.request.get('day')
      user_year = self.request.get('year')

      month = valid_month(user_month)
      day = valid_day(user_day)
      year = valid_year(user_year)

      if not(month and day and year):
         self.write_form("That doesn't look valid to me, friend.", user_month,
                         user_day, user_year)
      else:
         self.redirect("/thanks")

class ThanksHandler(webapp2.RequestHandler):
   def get(self)
      self.response.out.write("Thanks! That's a totally valid day!")


app = webapp.WSGIApplication([('/', MainPage), ('/thanks'.ThanksHandler)
                                       debug=True),
```

Answers

1.2. Quiz: User Input

<input name="q">

1.4. Entering Input

Put some text in the input box in your browser and press enter. What happens?

- Nothing.
- **The URL changes with my text.**
- An error message Appears.
- The text disappears.

The URL should now appear with the text you entered as a query parameter. For example, if you had entered the text ‘udacity’, your URL would now be something like:

play.html?q=udacity

1.6. Quiz: Submitting Input

No.

1.8 Action Attribute Quiz

- Nothing.
- The URL changes with what we entered.
- **My browser goes to Google search results.**
- I’ve written an entire search engine.

1.9 URL Encoding Quiz

better+flight+search

2.2 Content Type Quiz

Why do we see raw HTML instead of our text box?

- Google App Engine can only send raw HTML.
- Our browser is mis-configured.
- **We're sending the wrong Content-Type header.**
- Our HTML is invalid.

2.4 Handlers Quiz

- We see garbage.
- **We see the HTTP request.**
- We see a message from the future.
- Nothing changes.

2.6 Method Attribute Quiz

- The text we entered was displayed.
- **An error was displayed.**
- The request headers were displayed.
- Ran a Google search with our request.

2.7 Methods and Parameters Quiz

- In the URL.
- **After the HTTP response headers.**
- In a second request.
- In the HTTP request headers.

2.10 When to Use Post and Get Quiz

- **Fetching a document.**
- Updating data.
- **Requests that can be cached.**
- Sending a lot of data to the server.

2.12 Passwords Quiz

When we submit this form, what is the value of the q parameter in the URL?

- **Whatever I typed in the box.**
- The parameter won't appear at all.
- hunter2

2.14 Checkboxes Quiz

What is the value of the q parameter when I submit the form with the checkbox unchecked?

- Off.
- **The parameter doesn't appear at all.**
- The parameter is blank.
- hunter2.

2.16 Multiple Checkboxes Quiz

q=on&r=on

2.18 Grouping Radio Buttons Quiz

- Give them all the same id.
- **Give them all the same name.**
- Give them all the same group.
- include the <input>s in a <group> element.

2.20 Radio Button Values Quiz

q=two

2.23 Dropdowns Quiz

Two

2.25 The Number One Quiz

- Number One
- **1**
- Nothing
- hunter2

3.2 What is Your Birthday? Quiz

- The form clears.
- **We get an Error 405 because we never added a post() handler for /.**
- We see the form values in the URL.
- The app counts down to my birthday.

3.4 Handling Bad Data Quiz

- **Use dropdowns to limit what users can actually enter.**
- Assume that users will only enter good data.
- **Verify what the users enter and complain if the data is bad.**
- Make values up.

3.5 Valid Month Quiz

```
months = ['January',
          'February',
          'March',
          'April',
          'May',
          'June',
          'July',
          'August',
          'September',
          'October',
          'November',
          'December']

def valid_month(month):
    if month:
        cap_month = month.capitalize()
        if cap_month in months:
            return cap_month
```

3.6 Valid Day Quiz

```
def valid_day(day):
    if day and day.isdigit():
        day=int(day)
        if day > 0 and day <=31:
            return day
```

3.7 Valid Year Quiz

```
def valid_year(year):
    if year and year.isdigit():
        year = int(year)
        if year > 1900 and year < 2020:
            return year
```

4.1 String Substitution Quiz

```
given_string = "I think %s is a perfectly normal thing to do in public."
def sub1(s):
    return given_string %s
```

4.2 Substituting Multiple Strings Quiz

```
given_string2 = "I think %s and %s are perfectly normal things to do in public."
def sub2(s1, s2):
    return given_string2 %(s1, s2)
```

4.4 Advanced String Substitution Quiz

```
given_string2 = "I'm %(nickname)s. My real name is %(name)s, but my friends call
me %(nickname)s."
def sub_m(name, nickname):
    return given_string2 %{'name': name, 'nickname': nickname}
```

4.7 Preserving User Input Quiz

- <input value="%(month)s">
- <input name="month">%(month)s</input>
- <input name="month">
- **<input name="month" value="%(month)s">**

4.9 Problems With HTML Input Quiz

- We see an error message and our input is in the form.
- **We see an error message, but the quote messes up our HTML.**
- There's an error on the server side.
- The page renders blank.

4.13 Using HTML Escaping Quiz

&=&

4.14 Implementing HTML Escaping Quiz

```
def escape_html(s):
    for (i, o) in (('&', '&amp;'),('>', '&gt;'),('<', '&lt;'),('"', '&quot;')):
        s= s.replace(i, o)
    return s

print escape_html("test")
```

4.18 Redirection Advantages Quiz

- Because POSTs can't return HTML.
- **So that reloading the page doesn't resubmit the form.**
- To remove the form parameters from the URL.
- **So we can have distinct pages for forms and success pages.**