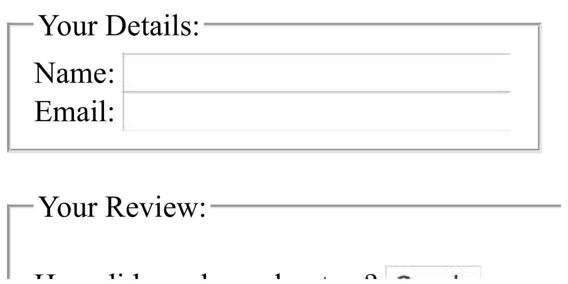
HTML Forms

Intro

In this section we will discuss HTML forms and some of the behind the scenes workings of them. Pretty much any time you are submitting data via a website you are using a form. If you are creating a new account at Amazon, logging into OSU's website or uploading a file, that is all happening with forms. So it is important to get a good grasp of what is going on on the HTML side of things to submit data before we talk about actually processing that data on the server. Below is an example form that we will use to discuss the various parts of a form.

Edit in JSFiddle

- Result
- HTML


Your Details:
Name:
Email:
Your Review:

The Form Tag

When dealing with forms everything will live within a `form` tag. This tag must have an opening and closing tag. In general any content can show up inside a `form` tag except other forms. In addition to general content like paragraphs, headers or images forms almost always include `input` elements. These elements are where the user will actually enter in or select options to submit. There are other things like a `textarea` which function in much the same way as an `input` element but are just called something different.

If you look at the HTML of the form above you will see at the outermost level is a `form` tag, then within that there are many `input` elements and other elements like `fieldset` or `label` to help organize the form.

Form Attributes

The form tag itself has two very important attributes. The first is the `action` . The `action` attribute specifies where the form should be sent. It is similar to an address on a piece of mail. When the form is submitted the browser will package up all the contents of the form and send it to the location specified in the action. The second important attribute is the `method` . This specified how the data will be sent. There are two primary ways a web browser will submit a form one is a `get` the other is a `post` . They both will send the same data but it will be handled differently on both the browser and the server.

Inputs

The Input Tag

The `input` tag is the tag most commonly used to gather data within a form. It stands out a little from other elements because it has an attribute `type` which vastly changes how it is displayed, how it is used and even what other attribute tags can be used. In addition to have this `type` attribute it is also critical that it have a `name` attribute. The `name` is how the server will know which part of the form the data is associated with. For example, in the above form the email field has the attribute `name="email"` it happens to also match the value given to the `type` attribute but this is just a coincidence. If one were to enter the email *foo@bar.com* and submit the form the server would receive something which looked like this `email=foo@bar.com` so it would know the field named email had the value *foo@bar.com* entered into it.

There are *a lot* of different kinds of inputs and attributes. I would recommend Mozilla's Input Reference (https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input) if you ever need an answer to "Is there an input for this?". In addition it enumerates all the other possible attributes which can do things like limit length or do some types of validation.

Radio Buttons and Check Boxes

These inputs need their own special section as their use is not entirely intuitive. Check boxes need both a `name` and `value` just like other inputs. However the server will only ever know about the existence of the check box if it is checked. For example this check box `<input type="checkbox" name="foo" value="bar">` will send the server `foo=bar` if it is checked. If it is unchecked the server will not get any indication that the check box even exists.

Radio buttons add on a little twist. If no radio button is selected, nothing will get sent, just like with a check box. But every radio button that has the same `name` will be in a radio button group. Within that radio button group only one radio button can be selected. In the example above for the "Would you visit again?" portion of the form those radio buttons are all in the same group. So you cannot select both yes and no at the same time. If they had different `name` attributes then they could all be selected at once.

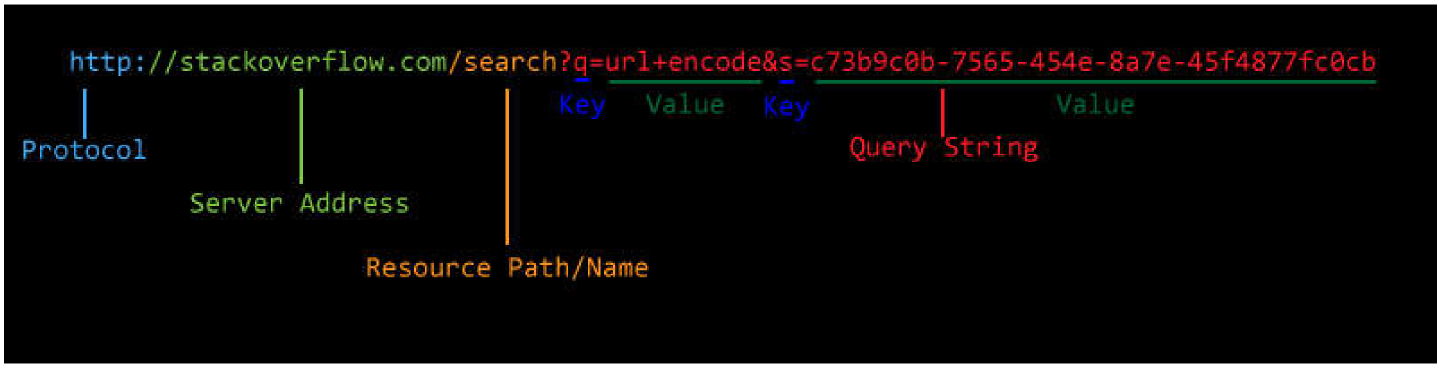
GET and POST

GET and POST requests are the primary way for the browser to send data other than the data usually sent in the headers to a server. When submitting standard HTML forms these are the only options for sending the form data to the server. It is important to understand the differences and to know when one would be preferred over another.

Both GET and POST are going to send key value pairs. On the server side these values are often accessed via an associative array. The server will look up values by supplying a key. For example `username=alice` has the key `username` associated to the value `alice` . This course will often talk about the data send from both GETs and POSTs assuming they are represented as an associative array.

GET

A GET request sends the key value pairs as parts of the URL. This is the primary identifying attribute of a GET request sent from a server. Here is an example URL of a search on StackOverflow which is submitted via a GET request `http://stackoverflow.com/search?q=url+encode&s=c73b9c0b-7565-454e-8a7e-45f4877fc0cb` . We will use this as an example to look at the syntax of a GET request.


http://stackoverflow.com/search?q=url+encode&s=c73b9c0b-7565-454e-8a7e-45f4877fc0cb
Key
Value
Key
Value
Protocol
Query String
Server Address
Resource Path/Name

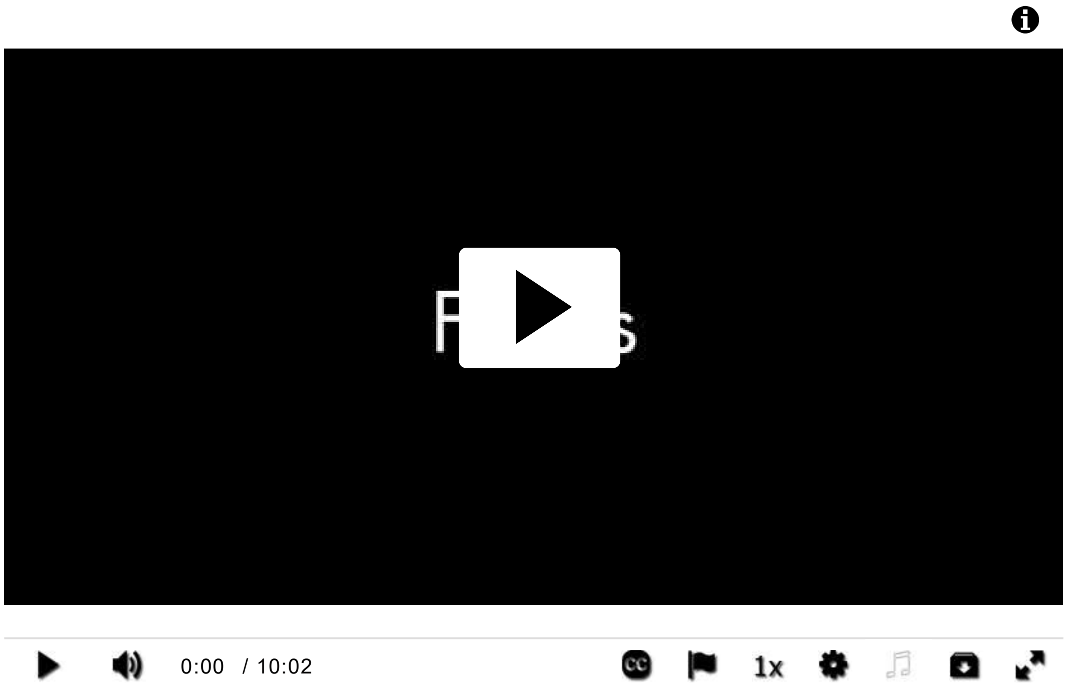
Lets break this down piece by piece `http:` is the protocol. It tells us this is using Hyper Text Transfer Protocol. Often times you will see `https:` which is the secure version of the protocol. Next is `//stackoverflow.com`. This is the address of the server we are trying to contact. After that will come the path to the resource we are requesting. In this case we are only one directory deep in `/search` and the server is able to interpret that as a resource. Other times you will see a directory chain ending in a file like `/foo/bar/kitten.jpeg`. Now what comes next is how we can tell this is probably a GET request. The `?` indicates that it is the start of a *query string*. In that query string there is going to be a specific format for key value pairs. That format is `key1=value1&key2=value2&keyn=valuen`. So keys are assigned values using `=` and key value pairs are separated using `&` symbols. So in this instance the `q` key has the value `url+encode` and the `s` key has the value `c73b...`. In this case “q” probably stands for query and “s” for session but there is no real convention here.

POST

A POST sends the same kinds of data as a GET, a list of key value pairs. What is different is that instead of sending it as part of the URL it is send as part of the request body. That means that you cannot tell what data (if any) was sent to a server via a POST request by just looking at the URL. It will still arrive at the server as part of a query string in the same format as a GET query string. So it will have key value pairs separated by `&l;` symbols.

GET vs POST

Remember that the location of the query string, in the URL vs the request body is the main difference between a POST and a GET. With that in mind think about what the advantages and disadvantages of each might be.



Activity

Create a simple form with a few inputs (I would encourage you to play around with some of the more complex types like check boxes and radio buttons). Set the forms action to `http://web.engr.oregonstate.edu/~zhangluy/tools/class-content/form_tests/check_request.php` . When you submit the form this page will tell you what type it was (GET or POST) and also show you all the variables and their values.

Review

The purpose of this module is to introduce you to forms. You should be comfortable creating a form to take common types of data and submit it to a website somewhere. Later we will be learning more about the specifics of how GET and POST are handled. For now you should be aware that GET values are sent in the URL and POST values are sent in the body of the request. In the future forms will be the main way you send user date to either your server or other servers.