Homework 6: Server-side Scripting

1. Objectives

- Get experience with the PHP programming language
- Get experience with Google Geocode and the Forecast.io APIs
- Get experience using an XML and a JSON parser in PHP.

2. Description

In this exercise, you are asked to create a webpage that allows you to search for weather information using the Google Geocode and Forecast.io APIs, and the results will be displayed in tabular format.

2.1. Description of the Search Form

A user first opens a page, called **forecast.php (or any valid web page name)**, where he/she can enter an address (street, city, and state) and select the temperature unit (Fahrenheit, Celsius). An example is shown in Figure 1. The State field includes a list of all US states, which are provided in section 3.4. Also, The form should include a forecast.io disclaimer, linking to 'http://forecast.io/'.

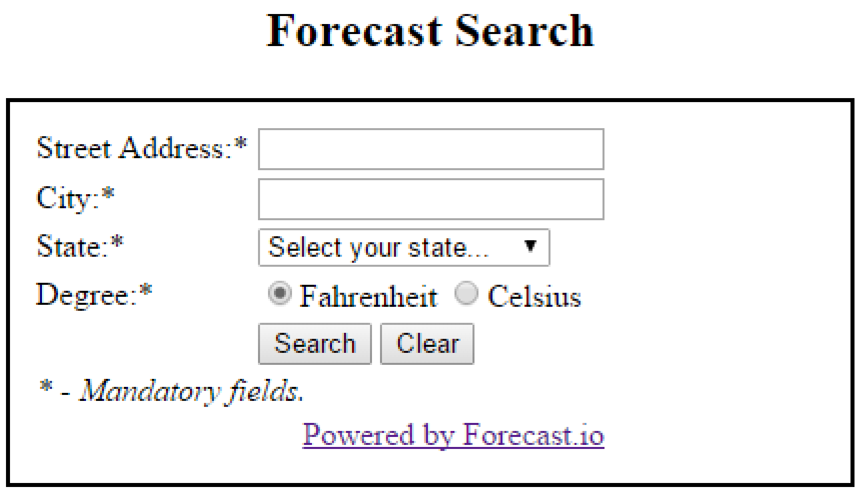
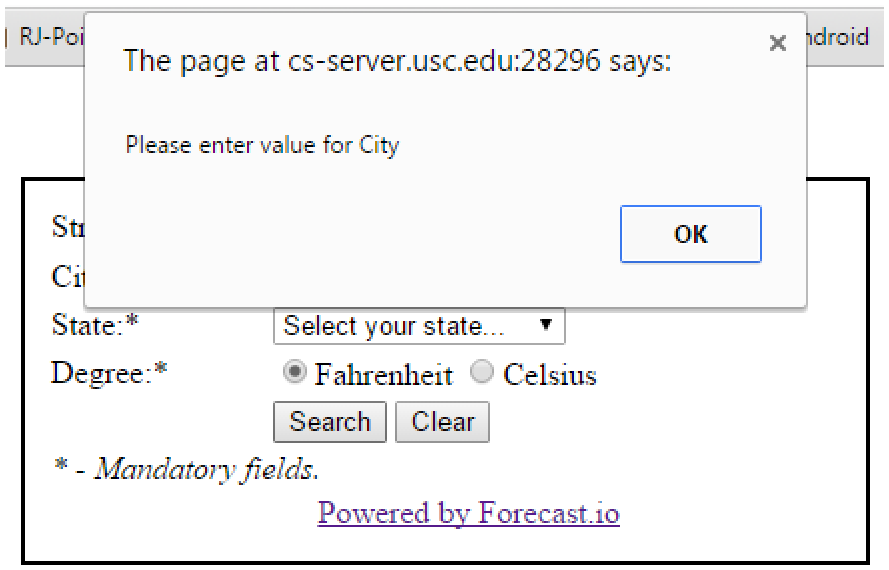


Figure 1: Initial Search Screen

The search form has two buttons:

- **Search** button: This button validates whether the user provided values for street address, city and state. The validation should be implemented in a JavaScript function. If the user did not enter one of the data items, then an alert should be shown with an appropriate message prompting the user to provide complete information. An example of the alert is shown in Figure 2, and an example of valid input is shown in Figure 3. Once the user has provided valid data, your script should send a request to your web server for **forecast.php (or whatever your valid web page name is)** with the form data. You can use either GET or POST to transfer the form data to the web server. A PHP script will grab the data and send it to the Google GeoCode and Forecast.io web services in sequence.

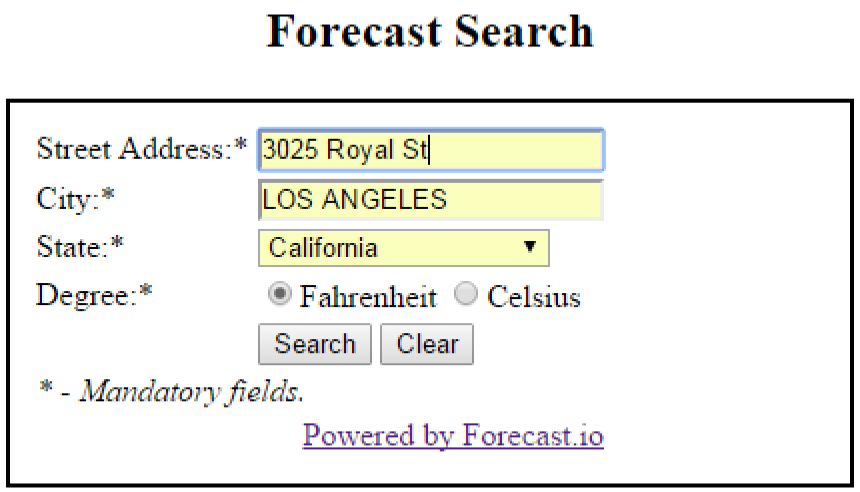
- **Clear** button: This button must clear the result area, all text fields, unselect the State value and reset the "Degree" field to its default value of Fahrenheit. The Clear operation is done using a JavaScript function.



The page at cs-server.usc.edu:28296 says:

Please enter value for City

Figure 2: An Message Error When Proving Incomplete Address (e.g., the values of city and state fields are missing)



Forecast Search

Street Address:*	3025 Royal St
City:*	LOS ANGELES
State:*	California
Degree:*	(● Fahrenheit) Celsius

Figure 3: An Example of Valid Input

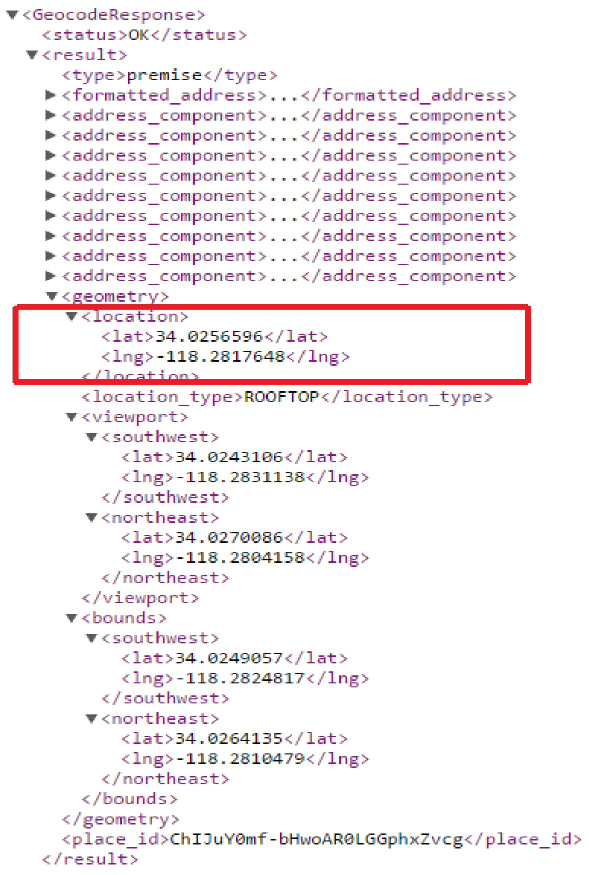
2.2. Displaying Results

In this section, we explain how to use the form data to construct web services calls to Google GeoCode and Forecast.io APIs and display the result in your page.

The PHP script (**forecast.php or whatever your valid page name**) will use the address information (street, city, and state) to construct a web service URL to query the Google Geocode API appropriately:

http://maps.google.com/maps/api/geocode/xml?**address**=*3025 Royal St*,*Los Angeles*,*CA*

The response of this URL is an XML-formatted object. One key piece of data returned is the latitude and longitude values for the given address. Figure 4 shows an example of the XML returned in the Google GeoCode web service response. You will see in the next step that you must use the extracted latitude and longitude to look up the weather for a given location.



```
▼<GeocodeResponse>
   <status>OK</status>
  ▼<result>
     <type>premise</type>
    ▶<formatted_address>...</formatted_address>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▶<address_component>...</address_component>
    ▼<geometry>
      ▼<location>
         <lat>34.0256596</lat>
         <lng>-118.2817648</lng>
       </location>
       <location_type>ROOFTOP</location_type>
      ▼<viewport>
        ▼<southwest>
           <lat>34.0243106</lat>
           <lng>-118.2831138</lng>
         </southwest>
        ▼<northeast>
           <lat>34.0270086</lat>
           <lng>-118.2804158</lng>
         </northeast>
       </viewport>
      ▼<bounds>
        ▼<southwest>
           <lat>34.0249057</lat>
           <lng>-118.2824817</lng>
         </southwest>
        ▼<northeast>
           <lat>34.0264135</lat>
           <lng>-118.2810479</lng>
         </northeast>
       </bounds>
     </geometry>
     <place_id>ChIJuY0mf-bHwoAR0LGGphxZvcg</place_id>
   </result>
```

Figure 4: A Sample Result of Google GeoCode Query

Next, using the latitude and longitude values, another web service URL needs to be constructed to query the Forecast.io API to get the weather information for the given location such as:

https://api.forecast.io/forecast/**YOUR_APIKEY/LATITUDE,LONGITUDE?units=units_value&exclude=flags**

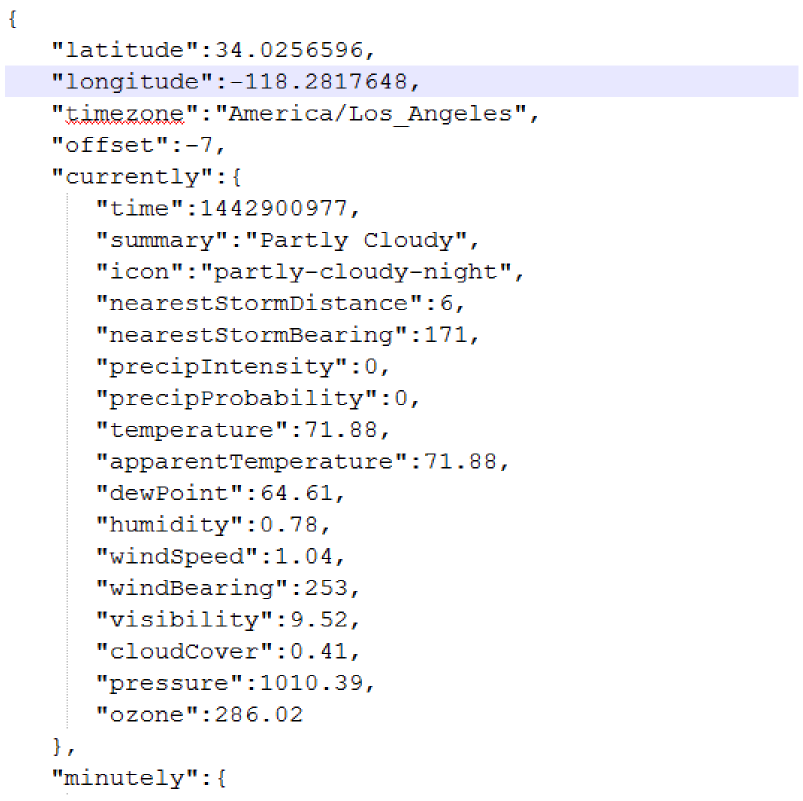
When constructing the Forecast.io web service API call, you should provide five parameters:

- The first parameter is your Forecast.io API key (a.k.a. the Dark Sky API key). How to create this key is explained in section 3.1.
- The second and the third parameter are the latitude and longitude values which are extracted from the XML-formatted data returned by the Google Geocode API.
- The name of fourth parameter is units. The value of this parameter is either “us” or “si”. If the temperature is in degree Celsius, *units=si* and if the temperature is in degree Fahrenheit, *units=us*.
- The name of the fifth parameter is *exclude*. The value of this parameter is “*flags*”.

The response from a query of the Forecast.io web service is a JSON-formatted object. An example of a returned JSON-formatted object from the Forecast.io API is available at:

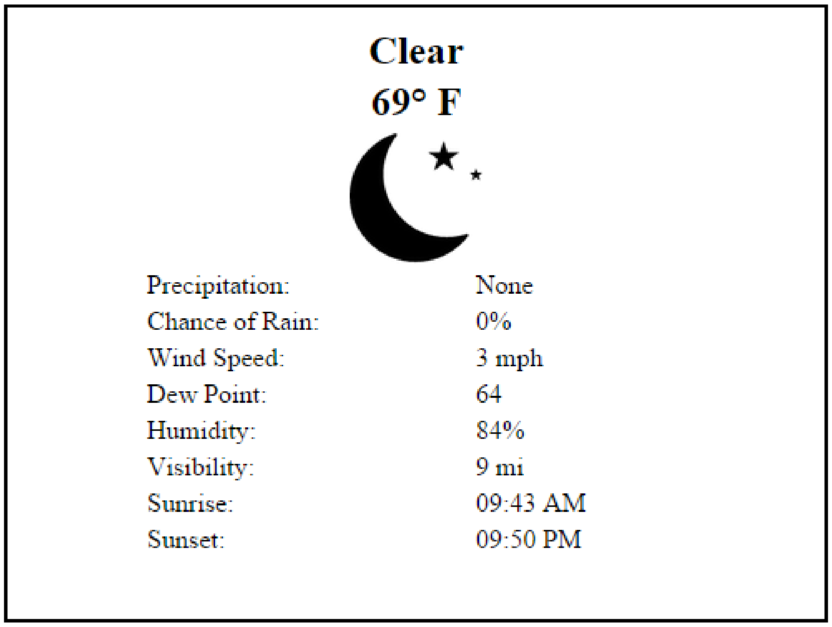
http://cs-server.usc.edu:45678/hw/hw6/sample_json_output.json

Figure 6 shows part of this file. You need to parse the returned JSON-formatted object and extract some fields. After extracting the data, the PHP script should display the data in a tabular format below the search form. A sample output is shown in Figure 7.



```
{
   "latitude":34.0256596,
   "longitude":-118.2817648,
   "timezone":"America/Los_Angeles",
   "offset":-7,
   "currently":{
      "time":1442900977,
      "summary":"Partly Cloudy",
      "icon":"partly-cloudy-night",
      "nearestStormDistance":6,
      "nearestStormBearing":171,
      "precipIntensity":0,
      "precipProbability":0,
      "temperature":71.88,
      "apparentTemperature":71.88,
      "dewPoint":64.61,
      "humidity":0.78,
      "windSpeed":1.04,
      "windBearing":253,
      "visibility":9.52,
      "cloudCover":0.41,
      "pressure":1010.39,
      "ozone":286.02
   },
   "minutely":{
```

Figure 6: Part of Sample JSON-formatted object returned from Forecast.io API call



Clear

69° F

Precipitation:	None
Chance of Rain:	0%
Wind Speed:	3 mph
Dew Point:	64
Humidity:	84%
Visibility:	9 mi
Sunrise:	09:43 AM
Sunset:	09:50 PM

Figure 7: Search Result

You need to map the data extracted from the result of the Forecast.io API call to your table output using the following table:

Table Column	**Data from result of Forecast.io API call**
Table Header	The weather condition is the value of *summary* in the *currently* object. The weather temperature is the value of *temperature* in the *currently* object. The temperature should be printed in integer format. Then, display an icon describing the weather condition. The displayed icon depends the value of *icon* in the *currently* object. ● icon value = “clear-day” → display “clear.png” ● icon value = “clear-night” → display clear_night.png” ● icon value = “rain” → display “rain.png” ● icon value = “snow” → display “snow.png” ● icon value = “sleet” → display “sleet.png” ● icon value = “wind” → display “wind.png” ● icon value = “fog” → display “fog.png” ● icon value = “cloudy” → display “cloudy.png” ● icon value = “partly-cloudy-day” →

	display “cloud_day.png” ● icon value = “partly-cloudy-night” → display “cloud_night.png” The images are available at http://cs-server.usc.edu:45678/hw/hw6/images. The Weather Icon must be displayed, and the alt text and title must be the text of the weather condition.
Precipitation	The value of *precipIntensity* in the *currently* object. It is possibly one of the following values {0, 0.002, 0.0017, 0.1, 0.4}. ● *precipIntensity* value = 0 → display “None” ● *precipIntensity* value = 0.002 → display “Very Light” ● *precipIntensity* value = 0.017 → display “Light” ● *precipIntensity* value = 0.1 → display “Moderate” ● *precipIntensity* value = 0.4 → display “Heavy”
Chance of Rain	The value of *precipProbability* in the *currently* object. You should multiply the value by 100 and display the percentage “%” character.
Wind Speed	The value of *windSpeed* in the *currently*

	object. The value should be displayed in integer format. The unit is “mph”.
Dew Point	The value of *dewPoint* in the *currently* object. The value should be displayed in integer format.
Humidity	The value of *humidity* in the *currently* object. You should multiply the value by 100 and display the percentage “%” character.
Visibility	The value of *visibility* in the *currently* object. The value should be displayed in integer format. The unit is “mi”.
Sunrise	The value of *sunriseTime* is the first object of data array in the *daily* object. The value is a Unix timestamp so it needs to be converted to the “two-digits-hour:two-digits-minute AM/PM” format. The hour should be in 12-hour format. Example is 01:15 PM.
Sunset	The value of *sunsetTime* is the first object of data array in the *daily* object. The value is a Unix timestamp so it needs to be converted to the “two-digits-hour:two-digits-minute AM/PM” format. The hour should be in 12-hour format. Example is 01:15 PM.

In summary, the search mechanism to be be implemented behaves as follows:

- Based on the input data in the search form, construct a web service URL to retrieve the XML-formatted output from the Google GeoCode API.
- Parse the returned XML and extract the latitude and longitude values.
- Call the Forecast.io API (latitude and longitude are parameters in the web service URL) and retrieve the JSON-formatted output.
- Parse the returned JSON-formatted output and extract the weather information.
- Display the weather information in tabular format.

2.3. Saving Previous Inputs

In addition to displaying the results, your page should maintain the provided values to display the current result. For example, if one searches for "***Street:*** *3025 Royal St*, **City:** Los Angeles, **State:** CA", one should see what was provided in the search form and the corresponding results. It follows that you need to keep the whole search box/input fields and buttons even while displaying results/errors.

3. Hints

3.1. How to get Forecast.io API Key

Go to https://developer.forecast.io/. Click on "Register". You should fill out the form as shown in Figure 9. The API Key (Dark Sky API key) is displayed as shown in Figure 10. Copy this key as you will use it in the Forecast.io web service API call.

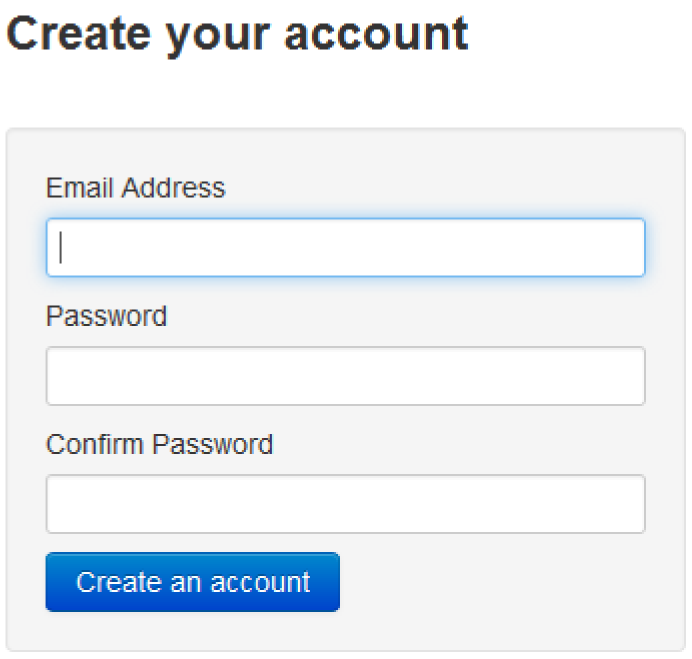
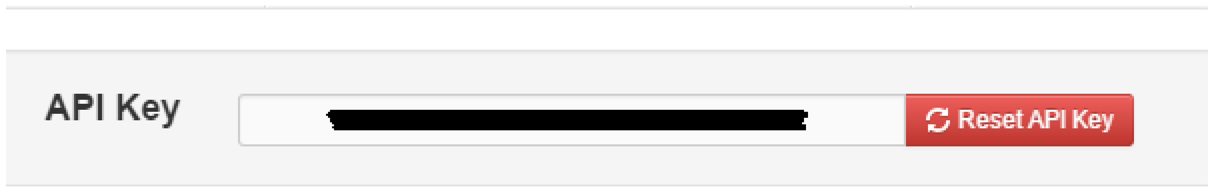


Figure 9: Forecast.io Registration Form



[redacted]

Figure 10: Forecast.io Application Key

3.2. Parsing XML files in PHP

You are free to choose any XML parsing library, but we recommend the SimpleXML library. The SimpleXML library provides a simple way of getting an XML element's name, attributes, and text. As of PHP 5, the SimpleXML library functions are part of the PHP core. No installation is required to use these functions. The following two tables show a set of functions which you may use. For more detailed information, please read:

- http://www.w3schools.com/php/php_xml_simplexml.asp
- http://php.net/manual/en/book.simplexml.php
- http://www.w3schools.com/php/php_ref_simplexml.asp

PHP 5 SimpleXML Functions

Function	Description
__construct()	Creates a new SimpleXMLElement object
addAttribute()	Adds an attribute to the SimpleXML element
addChild()	Adds a child element the SimpleXML element
asXML()	Formats the SimpleXML object's data in XML (version 1.0)
attributes()	Returns attributes and values within an XML tag
children()	Finds the children of a specified node
count()	Counts the children of a specified node
getDocNamespaces()	Returns the namespaces DECLARED in document
getName()	Returns the name of the XML tag referenced by the SimpleXML element
getNamespaces()	Returns the namespaces USED in document
registerXPathNamespace()	Creates a namespace context for the next XPath query
saveXML()	Alias of asXML()
simplexml_import_dom()	Returns a SimpleXMLElement object from a DOM node

simplexml_load_file()	Converts an XML file into a SimpleXMLElement object
simplexml_load_string()	Converts an XML string into a SimpleXMLElement object
xpath()	Runs an XPath query on XML data

PHP 5 SimpleXML Iteration Functions

Function	Description
current()	Returns the current element
getChildren()	Returns the child elements of the current element
hasChildren()	Cheks whether the current element has children
key()	Return the current key
next()	Moves to the next element
rewind()	Rewind to the first element
valid()	Check whether the current element is valid

3.3 Parsing JSON-formatted data in PHP

In PHP 5, you can parse JSON-formatted data using the “*json_decode*” function. For more information, please read http://php.net/manual/en/function.json-decode.php.

To read the contents of a JSON-formatted object, you can use the “*file_get_contents*” function. Because in this homework the JSON-formatted object is retrieved using the HTTPS protocol, your Apache server should be configured with a PHP compiler enabled with Curl and OpenSSL as described in HW#5.

3.4. List of US States and Their Two-Letter Abbreviations

Two-Letter Abbreviation	State
AL	Alabama
AK	Alaska

AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado
CT	Connecticut
DE	Delaware
DC	District Of Columbia
FL	Florida
GA	Georgia
HI	Hawaii
ID	Idaho
IL	Illinois
IN	Indiana
IA	Iowa
KS	Kansas
KY	Kentucky
LA	Louisiana
ME	Maine
MD	Maryland
MA	Massachusetts
MI	Michigan
MN	Minnesota

MS	Mississippi
MO	Missouri
MT	Montana
NE	Nebraska
NV	Nevada
NH	New Hampshire
NJ	New Jersey
NM	New Mexico
NY	New York
NC	North Carolina
ND	North Dakota
OH	Ohio
OK	Oklahoma
OR	Oregon
PA	Pennsylvania
RI	Rhode Island
SC	South Carolina
SD	South Dakota
TN	Tennessee
TX	Texas
UT	Utah
VT	Vermont

VA	Virginia
WA	Washington
WV	West Virginia
WI	Wisconsin
WY	Wyoming

4. Files to Submit

In your course homework page, you should update the **HW6 link** to refer to your new initial web page fro this exercise. Also, Submit your files (likely only a single .php file) electronically to the csci571 account so that they can be graded and compared to all other students' code via the MOSS code comparison tool.

****IMPORTANT**:**
All discussions and explanations in Piazza related to this homework are part of the homework description. So please review all Piazza threads before finishing the assignment.