Week 10: PHP Part 2

Agenda

- Overview
    - What's Due?
    - Midterm
- PHP Recap
    - Form Attributes
    - Checking for Form Submission
    - Creating Layouts with PHP
    - Debugging our PHP
- AJAX and PHP
    - AJAX
        - XMLHttpRequest Object
    - Viewing Data in the Browser
    - Combining PHP with AJAX
- AJAX & PHP Exercise
- Website Network Information (Two Posts)
- Next Week

Overview

Last week, we went over what PHP is and how it is used on web pages. This week, we will revisit some of those concepts to make our pages more secure and easier to maintain. Methods for working with external data on the same file server will also be explored.

PHP is a server-side language and our UAlbany server spaces don't allow for us to use those types of pages. 000webhost will be the platform we use for the rest of the semester. You should have created a 000webhost account to complete last week's assignment. If you haven't done that already, please make sure to do so for this week.

Make sure to use a unique password for this website as it has a history of data breaches. The goal is to use it for testing basic PHP features with our web pages. You are free to use other platforms if you'd like, they should have a similar interface.

What's Due

- PHP Exercise 2
- Website Network Information (Two Posts)
- Midterm

Midterm

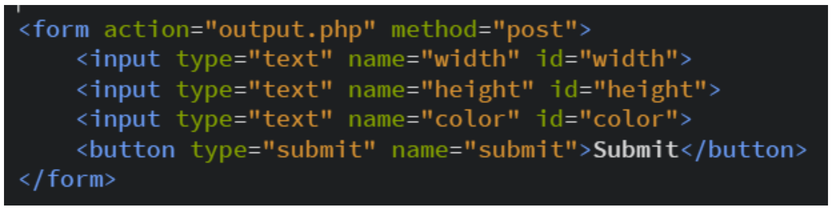
The Midterm was due this week on April 4th before midnight. In total, you need to create 3 mostly done pages. The layouts for these pages should be complete. However, the content doesn't need to be finalized. You can use lorem ipsum for the text and image placeholders. For more details about requirements, please see the Final Project Description file on Blackboard→Course Materials→Final Project.

PHP Recap

In the following sections, I will briefly go over some concepts we learned last week while introducing new methods related to those concepts. Specifically, we'll cover working with forms, creating layouts, and debugging our PHP.

Form Attributes

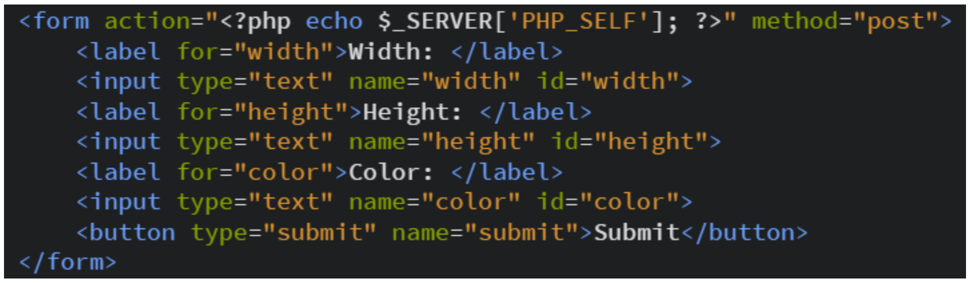
Last week, we started working with forms in PHP. The two key parts of a form are the action and method attributes. If you did the assignment from last week, your form might have looked something like the following snippet of code, but with labels:



```
<form action="output.php" method="post">
    <input type="text" name="width" id="width">
    <input type="text" name="height" id="height">
    <input type="text" name="color" id="color">
    <button type="submit" name="submit">Submit</button>
</form>
```

The action attribute value tells the form where to go. If we set our action attribute to some page, our form will try to pass information to that page and our current browser page will change. In general, a blank action attribute value or no action attribute at all will cause the page to submit to itself. However, HTML5 specification states that we must use a non-empty value for this attribute, or it won't validate.

One technique for having a page submit to itself is using a variable from PHP's `$_SERVER[]` array. Just like `$_POST` and `$_GET`, it is a super global array, but it contains information about the server. The variable that we want in this array is PHP_SELF. We could change our form attribute to look like the code below.

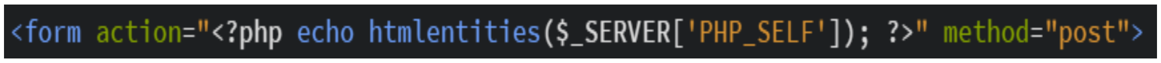


```
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
    <label for="width">Width: </label>
    <input type="text" name="width" id="width">
    <label for="height">Height: </label>
    <input type="text" name="height" id="height">
    <label for="color">Color: </label>
    <input type="text" name="color" id="color">
    <button type="submit" name="submit">Submit</button>
</form>
```

In this code, we are using the echo function to output `$_SERVER['PHP_SELF']` which evaluates to the page's file name. This allows us to submit the page to itself without leaving the attribute empty. This is good for ensuring the HTML5 specifications are met, but it isn't necessarily the most secure way to do things.

The problem is that a user can submit this page from another location and add extra content to it for malicious purposes. This is known as Cross-Site Scripting and is often used to send users to malicious pages or to insert bad data into databases. A hacker could insert JavaScript code snippets into the URL for the action attribute and potentially do damage. However, we can get around this by using the `htmlentities()` function.

Htmlentities takes in a string and then converts any html characters into harmless normal characters to prevent anything from being executed. To use it, we would have the same code as the above screenshot, but we would place the `$_SERVER['PHP_SELF']` part inside of the parenthesis in htmlentities like so:



```
<form action="<?php echo htmlentities($_SERVER['PHP_SELF']); ?>" method="post">
```

This is one of the best ways to have a form submit to itself so that it is more secure and validates by the HTML5 specification.

The second important attribute for forms is method and it lets the server know how to handle the data being passed. "Get" exposes data in the URL and is useful for allowing users to bookmark pages. It only has a 2000-character limit though and is bad for security. If you're pulling data from a database, get is a good option.

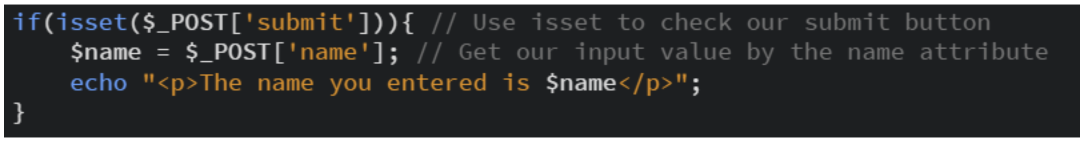
If you are trying to insert data into a database, "Post" is going to be the better option. Form information set through POST is more secure than get as it doesn't appear in the browser URL. However, it's important to note that neither are really secure, and need to be combined with good programming practices to be successful (form validation, data sanitization, etc.).

Whenever you submit info in a form that uses a method such as get or post, the values in the form get placed into an array known as a "super global array." This is an array that is available to all parts of your code. To refer to anything submitted, you'd have to know the method it was submitted through and the value of the name attribute in the tag. For example, if the following field was in a form using the get method, you'd use $_GET['fname'] to retrieve the input value. Likewise, you'd use $_POST['fname'] if it was sent with the post method.

<input name="fname" id="fname" type="text>

$fname = $_GET['fname'];

Checking for Form Submission
In last week's class, we learned about using the isset() function to check to see if a form was submitted based on a button value was set. In some examples, we looked to see if `$_POST['submit']` was set like this:

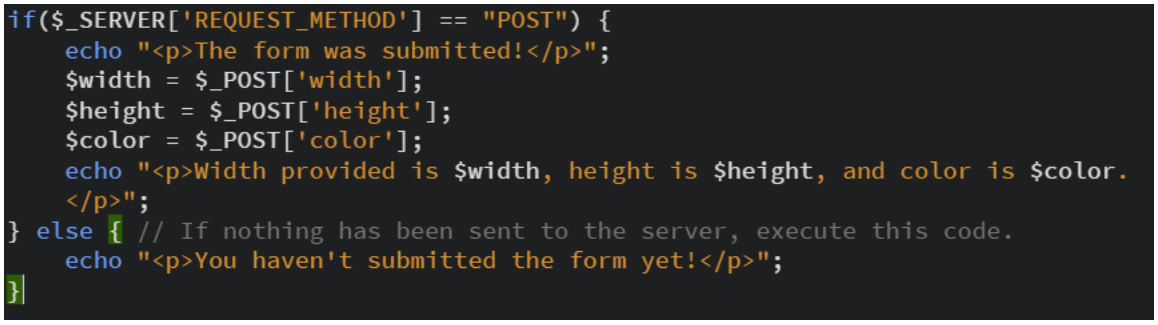


```
if(isset($_POST['submit'])){ // Use isset to check our submit button
    $name = $_POST['name']; // Get our input value by the name attribute
    echo "<p>The name you entered is $name</p>";
}
```

This is one technique to check if a form has been submitted. However, it is not necessarily secure or thorough enough. In older browsers, a user could submit the form by hitting enter inside of a textbox and the button value wouldn't be set inside of the super global array. Also, a hacker or some other page could send input to your page using another method such as GET.

For usability and security purposes, I recommend using the REQUEST_METHOD variable for the `$_SERVER` super global for checking if a form was submitted. Just like the PHP_SELF variable, the REQUEST_METHOD variable contains information from the server. This variable contains information about the method that was used to request the page. In the example below, I check to see if the request method was "POST" and if it isn't, I have a message displayed inside of a paragraph tag. This message is always present until the page receives a post method request.

You could use if/else statements to check or various request methods and have your page respond accordingly.



```
if($_SERVER['REQUEST_METHOD'] == "POST") {
    echo "<p>The form was submitted!</p>";
    $width = $_POST['width'];
    $height = $_POST['height'];
    $color = $_POST['color'];
    echo "<p>Width provided is $width, height is $height, and color is $color.
    </p>";
} else { // If nothing has been sent to the server, execute this code.
    echo "<p>You haven't submitted the form yet!</p>";
}
```

The screenshots above are from this week's class example which can be found linked below the lecture notes.

Creating Layouts with PHP
Using PHP, we can create all sorts of layouts dynamically. To do so, we use words like include or require to pull in files. When you use include, if the file doesn't exist, the page will produce an error, but the script will continue executing. If you use require, the file must exist, or the script will stop running.

```
<?php include 'header.php'; ?>
<?php require 'header.php'; ?>
```

You can also change them to include_once or require_once if you need to make sure the included file is only included at most one time. If you want code inside of the included file to execute, it still must be wrapped in its own set of PHP tags. Otherwise, HTML can just be placed inside of it by itself as shown in the class example.

The most common use for PHP includes are items that are repeated on a website. For the Final Project, you will need to use PHP. I'd recommend using PHP includes for headers, footers, navigation menus, or any other code that will be used multiple times on your pages. Rather than typing that code multiple times, you can have it in one area for editing.

The class examples this week demonstrate how to use the code discussed above to enhance the security of your forms and make your page layouts easier to manage. I would review them and then look over the links below if you want more information.

Debugging PHP
If you have trouble, use Lu's PHP Debugging Guide on Blackboard. You can also check your PHP using the link below:
https://phpcodechecker.com/

It will generally tell you where your error is and even if you don't understand what it means, look at the line numbers around the error and check for syntax problems. You must copy and paste all your code inside of the checker.

A lot of the mistakes you make in PHP will be like mistakes you've done in other programming languages. Missing semicolons, misspelled variables, unclosed curly brackets, and other minor things can cause your code to not work. If you use a text editor with syntax highlighting with color, it will be easier to find your errors. Certain things like variables, strings, and operators get their own colors so you can see where they are in the code.

AJAX and PHP

In the following section, we will go over AJAX and how it can be used on our pages. After explaining AJAX and what it does, we will learn about how to integrate it with PHP.

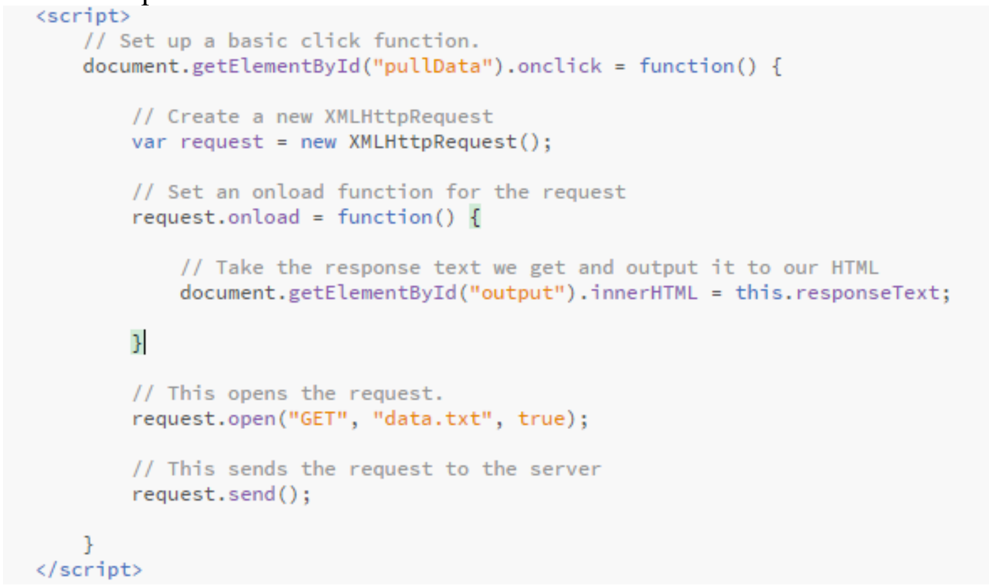
AJAX

This stands for **A**synchronous **J**avaScript **A**nd **X**ML and it serves as a method for accessing web servers from a web page. AJAX is typically used to change the content on a page dynamically, without having to refresh the page. Although it mentions the usage of XML, AJAX is used to transport all sorts of data including plain text or JSON. As opposed to being a programming language, AJAX serves as a method or way to work with data from a web server using HTTP.

A few weeks ago, we worked with data in the browser using the Cookie object. JavaScript has another object which is useful for working with data on servers. This is the XMLHttpRequest object, and we use it to send to requests to web servers to potentially retrieve data.

With AJAX, we can use an event on a web page to trigger a function (button is clicked, key is pressed, etc.). We then create an XMLHttpRequest object with JS. This object would be used to send a request to the web server. The server then processes the request and sends a response back to our web page. Our JS should then read the response that was returned and take an action which may include updating page content or outputting an error message.

A basic request for data from another file on the same server could look like this:



```
<script>
    // Set up a basic click function.
    document.getElementById("pullData").onclick = function() {

        // Create a new XMLHttpRequest
        var request = new XMLHttpRequest();

        // Set an onload function for the request
        request.onload = function() {

            // Take the response text we get and output it to our HTML
            document.getElementById("output").innerHTML = this.responseText;

        }

        // This opens the request.
        request.open("GET", "data.txt", true);

        // This sends the request to the server
        request.send();

    }
</script>
```

We create a standard click event on a button with an id of "pullData." Inside of this function is where the magic with AJAX occurs.

The first line of code creates a new XMLHttpRequest and assigns it to the request variable. After this, we create a function which is triggered when the request loads (request.onload = function). Inside of that function is where we can generate output or do whatever we want with the response text (this.responseText refers to the responseText for the request variable).

The next two lines are what open/send the request. Request.open takes in three parameters: the method to use (GET or POST), the file we want to use (data.txt), and true or false to indicate asynchronous vs synchronous loading. Request.send just sends the request off and the request will return information depending on the server/requested data.

<u>XMLHttpRequest Object</u>
This is used to exchange data with a web server in the background so that page content can be updated without refreshing the entire page. It is built into most browsers, and it has a few methods and properties that will be useful for us. We've seen some of the methods such as open and send, so we'll explore some properties next.

One property that was in the previous example was "onload." This is triggered whenever the request is received or loaded. It is also known as a callback function. These are functions which wait to execute until some condition is met. In this case, the function isn't triggered until the request is loaded.

Another property we can use is onreadystatechange. It is triggered when the readyState property is changed and is another example of a callback function. The readyState property relates to the state of the request in terms of its process.

readyState has 5 possible values which range from the numbers 0-4. Each one relates to a different step in the server request process. I've defined them below:

0: The request has not been initialized
1: A server connection has been established
2: The request was received by the server
3: The request is being processed currently
4: The request is finished. and the response is ready

If you check the "readyState" of your request, it will return one of those numbers. The number you want to look for is "4" since that means the request has been completed and the response is ready.

The last property I want to discuss is status. This is another property you can check for the XMLHttpRequest object. It has many possible values, but some of the most common are 200 and 404. 200 means that the request is OK and was successful. 404 is something you might be familiar with already (404 error: page not found) and represents when the requested page is not found.

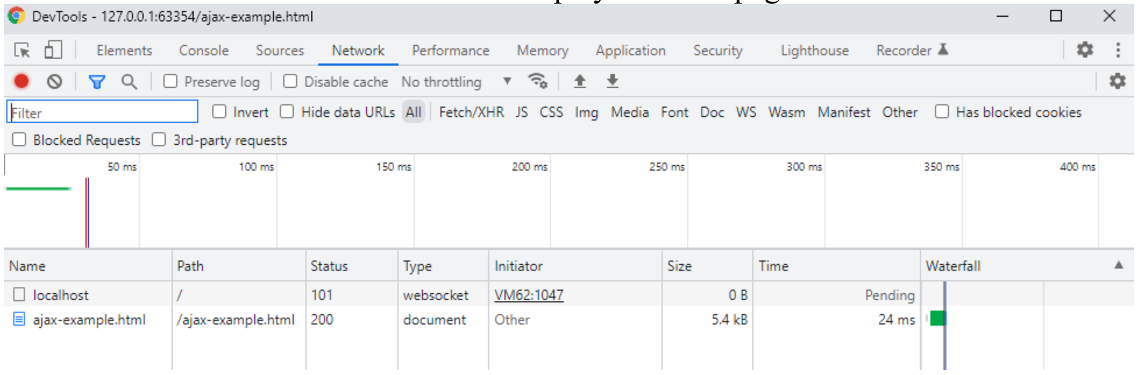
When checking to see if your response is successful, you can combine values from the readyState and status properties. I've demonstrated how that could be done in the screenshot below. The main difference is that we check the status and readyState inside of our function before we generate output to the page.

```
<script>
// EXAMPLE 2
document.getElementById("example2").onclick = function() {

    // Create a new XMLHttpRequest
    var request = new XMLHttpRequest();

    // Set an onreadystatechange function for the request
    request.onreadystatechange = function() {

        // Check if the status is 200 (OK)
        // Check if the readyState is 4 (request is finished and response is ready)
        if(this.status == 200 && this.readyState == 4) {

            // Take the response text we get and output it to our HTML
            document.getElementById("example2output").innerHTML = this.responseText;
        }

    }

    // This opens the request.
    request.open("GET", "data2.txt", true);

    // This sends the request to the server
    request.send();

}

</script>
```

For more info related to the different status messages, visit this link:
https://www.w3schools.com/tags/ref_httpmessages.asp

Viewing Data in the Browser

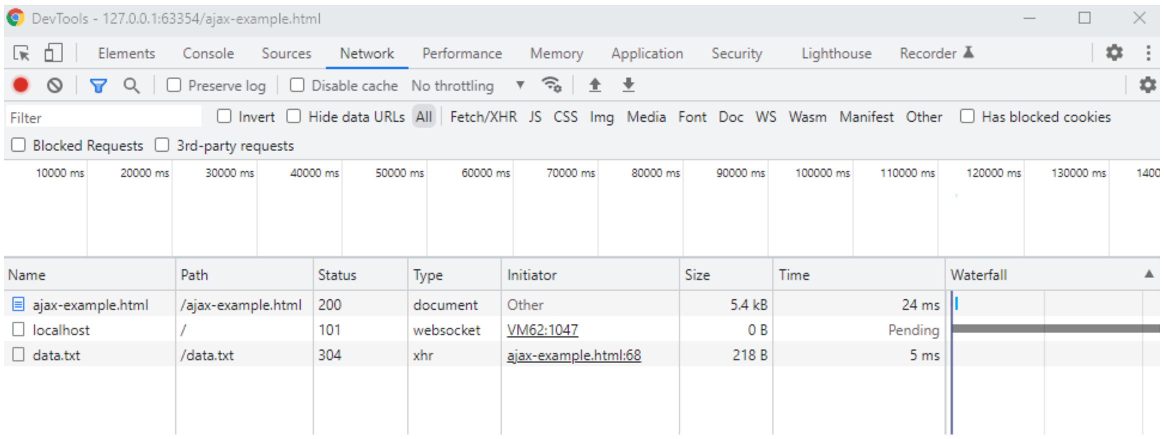
Just like Cookies, you can view the data being handled inside of the browser. To do so, simply right-click the page, and inspect it. From there, you'll go to the "Network" tab which may look something like this when the page first loads. Ajax-example.html is the name of the page and there is a lot of information related to it displayed on the page.



Name	Path	Status	Type	Initiator	Size	Time	Waterfall
localhost	/	101	websocket	VM62:1047	0 B	Pending	
ajax-example.html	/ajax-example.html	200	document	Other	5.4 kB	24 ms	

You'll see that there is data related to the name of the document, the path it comes from, its status, type, and other items. The size and time information tell you how long it took to load the external resources while the type and initiator columns tell you about what kind of resource it is in addition to what initiated/loaded it.

If you're using the example page provided and were to click the very first button for example one, the network tab would look like the screenshot below. Notice that there is a little more data displayed inside of the panel.


DevTools - 127.0.0.1:63354/ajax-example.html
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder
Preserve log Disable cache No throttling
Filter Invert Hide data URLs All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other Has blocked cookies
Blocked Requests 3rd-party requests
10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms 80000 ms 90000 ms 100000 ms 110000 ms 120000 ms 130000 ms

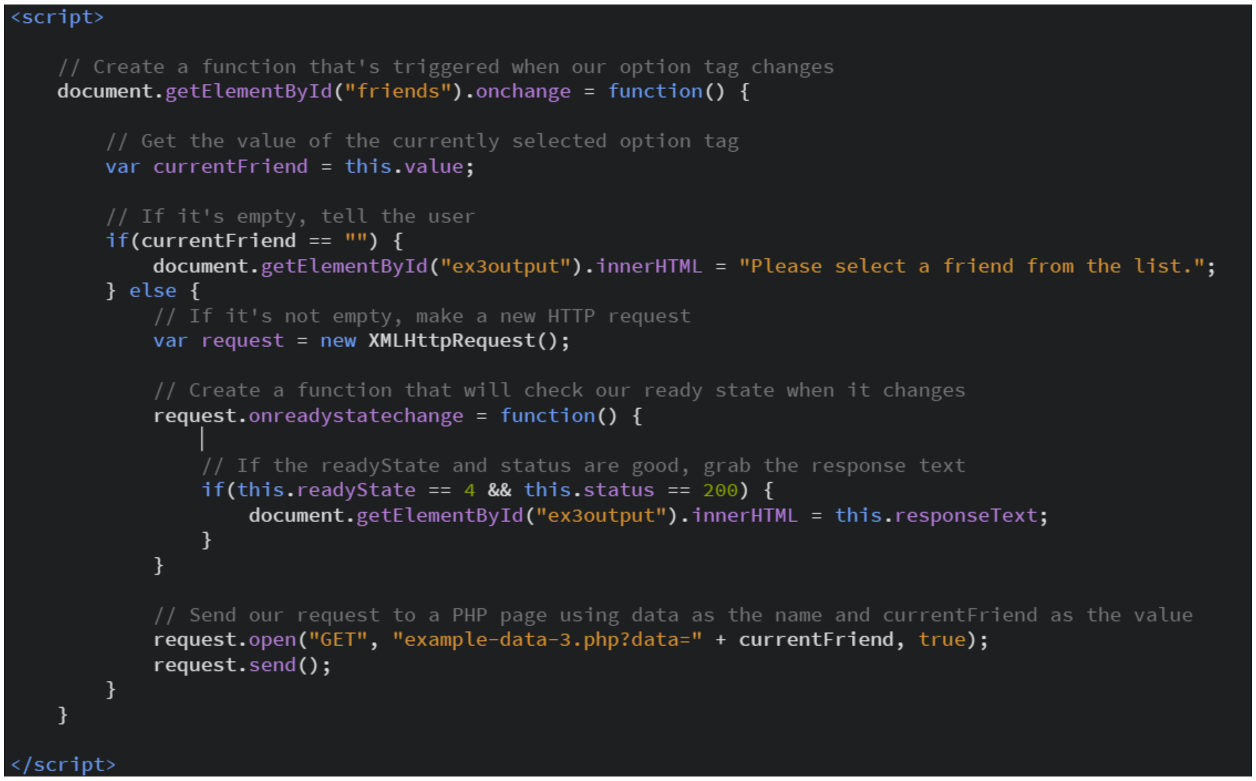
Name	Path	Status	Type	Initiator	Size	Time	Waterfall
ajax-example.html	/ajax-example.html	200	document	Other	5.4 kB	24 ms	
localhost	/	101	websocket	VM62:1047	0 B	Pending	
data.txt	/data.txt	304	xhr	ajax-example.html:68	218 B	5 ms	

The difference in the second screenshot is the presence of data.txt. That's the file which we retrieve data from in example 1. The status would be 200 when that is done for the first time. However, the status will be 304 if we have already retrieved that resource before and it is cached in the browser. 304 just means the resource is cached so we can use what we have already.

For this week's discussion posts, you'll use the Network tab to monitor data in your browser. Make sure to try out the various features and see what's being loaded along with how long they are taking to load.

Combining PHP with AJAX

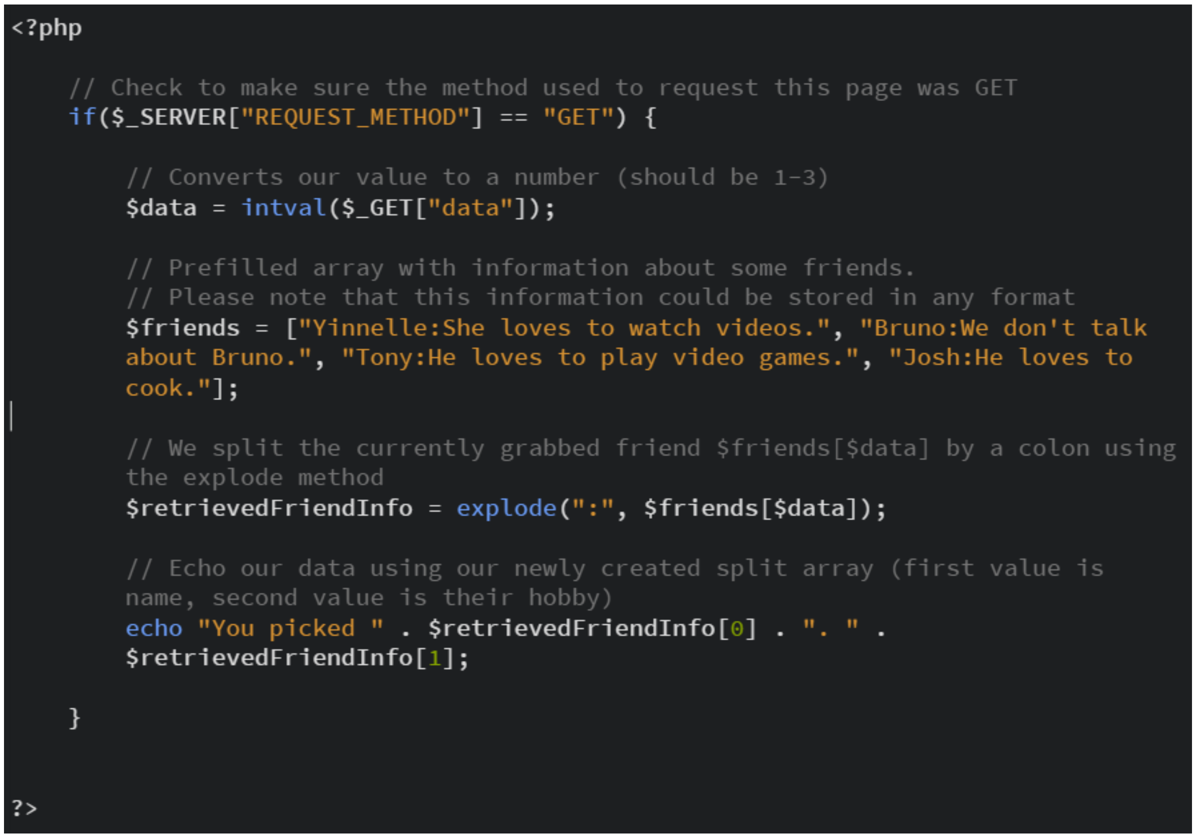
We can use AJAX with PHP in a similar way to what we have done already. In last week's assignment, you used a form to pass data to another page. On that other page, you used PHP to check the data and then output a div with the height, width, and color provided. The same thing can be done with AJAX and PHP. We would use AJAX to make a request to a PHP page. The PHP page can then use code to manipulate the data we received and generate some output on initial page.



```
<script>

    // Create a function that's triggered when our option tag changes
    document.getElementById("friends").onchange = function() {

        // Get the value of the currently selected option tag
        var currentFriend = this.value;

        // If it's empty, tell the user
        if(currentFriend == "") {
            document.getElementById("ex3output").innerHTML = "Please select a friend from the list.";
        } else {
            // If it's not empty, make a new HTTP request
            var request = new XMLHttpRequest();

            // Create a function that will check our ready state when it changes
            request.onreadystatechange = function() {

                // If the readyState and status are good, grab the response text
                if(this.readyState == 4 && this.status == 200) {
                    document.getElementById("ex3output").innerHTML = this.responseText;
                }
            }

            // Send our request to a PHP page using data as the name and currentFriend as the value
            request.open("GET", "example-data-3.php?data=" + currentFriend, true);
            request.send();
        }
    }
</script>
```

On this page, we use “example-data-3.php?data=” + currentFriend as the data to request. What we’re doing is using example-data-3.php as the file. However, we are passing data and currentFriend as a name/value pairing (data=currentFriend). Just like cookies, we can store or pass data in name/value pairs.

The following screenshot is from example-data-3.php. You can see that we grab our data using $_GET[“data”];. The data passed over from our form will have to be referenced by whatever word we attach after the question mark in our request.open() function. In this case, it is “data” being used with GET, so we would use $_GET[“data”]. The data being passed over will be either an empty string, or the numbers 0-3.



```
<?php

    // Check to make sure the method used to request this page was GET
    if($_SERVER["REQUEST_METHOD"] == "GET") {

        // Converts our value to a number (should be 1-3)
        $data = intval($_GET["data"]);

        // Prefilled array with information about some friends.
        // Please note that this information could be stored in any format
        $friends = ["Yinnelle:She loves to watch videos.", "Bruno:We don't talk
        about Bruno.", "Tony:He loves to play video games.", "Josh:He loves to
        cook."];

        // We split the currently grabbed friend $friends[$data] by a colon using
        the explode method
        $retrievedFriendInfo = explode(":", $friends[$data]);

        // Echo our data using our newly created split array (first value is
        name, second value is their hobby)
        echo "You picked " . $retrievedFriendInfo[0] . ". " .
        $retrievedFriendInfo[1];

    }


?>
```

Once we have the data, we turn it into an integer using intval. The $data variable, which should be a number between 1-3, is used to reference someone in the $friends array ($friends[$data]). Then, the explode method is used to split the friendsArray data by a colon.

If the number 0 was sent over by the initial page, “Yinnelle:She loves to watch videos.” would be the index used in my array. The explode function would separate it into “Yinnelle” and “She loves to watch videos.”. These two pieces are referenced on the next line to create output on our page. The result of the echo statement is ultimately what is returned to our first page and output.

Examples

Online Resources

- https://www.w3schools.com/php/php_includes.asp (PHP Includes)
- https://www.php.net/manual/en/tutorial.forms.php (Working with Forms)
- https://html.form.guide/php-form/php-form-action-self/ (Self-submitting Pages)
- https://www.w3schools.com/php/php_ajax_php.asp (AJAX and PHP)

- https://www.php.net/manual/en/language.variables.superglobals.php (PHP superglobals in general)
- https://www.php.net/manual/en/function.htmlentities.php (htmlentities function)

A new video will be released on Tuesday afternoon. This video will cover PHP and AJAX in the context of this week's assignment. When it is available, the link will be added to the assignment description.

My Examples

- https://cinf362.000webhostapp.com/examples/ajax-example.php (AJAX/PHP)
- https://cinf362.000webhostapp.com/examples/secure-input-example.php (Forms/Includes)

PowerPoint documents have also been released under Blackboard → Course Materials → Resources →PHP & SQL. One of them offers a brief introduction to PHP while the other will serve as a debugging guide.

You won't be able to right-click the page and view the source this week to view my PHP code. Since PHP is processed by the server and rendered as HTML, the user can't see the PHP code you're using. The code for the example below is in the "examples" folder that was included with this week's lecture notes zip folder. You can take all the contents of the examples folder and place them somewhere on your 000webhost account and the pages should work. There are comments in the code to explain what is happening. Feel free to look at the live link to see the page in action.

Here's a link to **all** course examples:
https://cinf362.000webhostapp.com/examples/ (PHP/SQL Examples)
https://www.albany.edu/~cv762525/cinf362/examples/ (HTML, CSS, and JS examples)
https://www.albany.edu/~cv762525/cinf362/videos/ (videos)

Before completing the assignment(s) for this week, please read the "Viewing Your Web Pages.docx" file on Blackboard. You will not be able to submit anything for the assignment without completing that portion first. It is located directly in the Lecture Notes folder.

AJAX & PHP Exercise

Due Monday, April 11th at midnight
Download the "AJAX-PHP-Exercise.zip" folder from Blackboard under this week's lecture notes. Inside of the folder will be two PHP files to use for this assignment: assignment-input.php and assignment-data.php.

The input page should function so that when a user types inside of the input box, suggested fruits appear below the input box inside of p tags. To accomplish this, you'll need to add HTML/JS to the assignment-input.php page and PHP to the assignment-data.php page. I've left comments on each page describing what code you need to add. The details below provide a general description of what each page will do and the things you can expect to see.

Assignment-input.php
This page will serve as the input for your content. The user should enter a fruit and have a list of potential fruits appear on their screen.

HTML

- Add an input with a name and id of "fruitInput". It should also have a type of "text"
- Add a div with an id of "fruitOutput". It should have a p tag inside of it with the words "Please enter a fruit."

JS

- Follow the comments inside of the provided script tag
- You'll be using an onkeyup event to trigger a function that will make an AJAX call
- Inside of the function, you'll get the value of the input box
- You'll check the value of the input, if it's empty, you'll let the user know
- If it isn't empty (else), you'll create a new XMLHttpRequest to the PHP page
    - This is covered in example 3 for this week's examples
    - You should be checking for onreadystatechange
        - Make sure readyState is 4 and status is 200
- Generate output to the page using responseText

<u>Assignment-data.php</u>
This page will be used to receive the data from the assignment-input.php page. We'll check the data against an array of fruits and provide results based on what is found. This page should never be in view to the user.

PHP

- Create a variable called returnedFruit
- Check to see if the server used get as the request method and place the following code in it:
    - Create a variable called submittedFruit and store our fruitInput from GET
    - Loop through the $fruitsArray array using foreach
    - Check if our fruit is found in the array using the stripos() method (explained below) – this should be inside of the foreach loop
    - If it is found, add the fruit to itself in a string (using ".=")
- Create an if statement outside of the code above to check returnedFruits
    - If it's not an empty string, echo $returnedFruit
    - If it is an empty string, echo "<p>No fruit found.</p>"

This page will take data from the first page and check it against an array using the stripos() method. This method checks what is known as a needle inside of a haystack and returns a number based on where the needle is located in the haystack. In, our case, we want to check if that value is 0. This would imply it matched the first character and is a perfect match.

For example:
$chLocation = stripos("Chris", "ch");

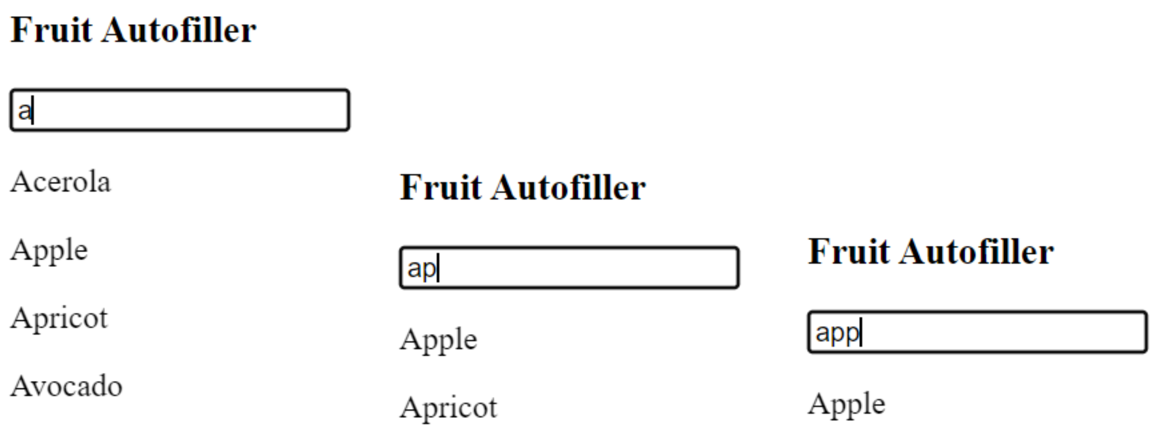
The $chLocation variable would be equal to 0 since ch appears in the first position of "Chris". It's not case-sensitive and the string being search is always based on 0 for the index (first letter is 0, second letter is 1, and so on) just like arrays.

Let's say we wanted to check if a name started with Brad. We could use the following code to do that:

```
$fname = "Christopher";
if (stripos($fname, "Brad") === 0) {
        // do something if there's a match
}
```

We're providing $fname as the "haystack" and Brad as the "needle" to search for in this code. If Brad is found at position 0 in the string, our code would be triggered. However, $fname is "Christopher" and so nothing would occur. You'll use this same format to check the submitted fruit against the fruit from the fruit array in the assignment this week.

If you've completed the assignment correctly (outputting retrieved values in paragraph tags), your output will change as you type. For example, as the word apple is typed the suggested fruits will decrease as the input begins to match apple more and more.



Fruit Autofiller

a

Acerola

Apple

Apricot

Avocado

Fruit Autofiller

ap

Apple

Apricot

Fruit Autofiller

app

Apple

Make sure your PHP files are on a server that supports them. The UAlbany servers don't, so you should be using 000webhost or another platform such as byethost. If your PHP doesn't work, put your code in the PHP Checker link provided in the lecture notes above. It should give you a line number and a reason why it's failing. Check the line number (if provided) and adjust the code as necessary. Also, make sure your file has a .php extension. Without this extension, the browser won't know to treat it like a PHP file.

Your webpage is **due on Monday, April 11th at midnight.** To submit the work for this exercise, visit Blackboard → Course Materials → Lectures Notes for this week's class. You can also go into the "Assignments" folder and the submission area will be titled "AJAX & PHP Exercise You should be submitting a live link to me; it **must be through 000webhost**. UAlbany servers don't allow for PHP to be used and so we must utilize a hosting platform such as 000webshot. The work submitted will be evaluated based on the rubric explained below.

AJAX & PHP Exercise Rubric – 10pts

- Live link submitted – 1pt
- Zip folder submitted – 1pt
- HTML Completed – 1pt
- JS Completed – 3.5pts
- PHP Completed – 3.5pts

Website Network Information (Two Posts)

Initial Post due Friday, April 8th at midnight

Visit a few websites not previously used for discussion posts and open Google Chrome's Web Tools (right-click and inspect or press F12). Click on the Network tab and write about what you are seeing. Try this with complex websites in addition to simpler ones. A lot of the information will be very confusing at first and I don't expect you to understand most of it. The websites listed below contains guides/references to this tab and will be useful in helping you understand what you are seeing.

- https://developers.google.com/web/tools/chrome-devtools/network
- https://developers.google.com/web/tools/chrome-devtools/network/reference
- https://www.keycdn.com/blog/chrome-devtools (more general, cool features mentioned)

After reviewing those links and poking around at least 3 websites, discuss what you saw and your perceptions of it. You can use any of the points below to guide you in your post. Please make sure to write at least 2 short paragraphs, mention the websites you visited, and cover some of the items I mention.

- What was the most interesting thing you saw?
- Did you see any POST or GET methods being used?
- Were any requested items similar across websites?
- Did anything seem to load particularly fast or slow?
- Status codes
    - Which ones were present?
    - What do they mean?
    - Have you encountered any before?
- Disable cache and refresh – did anything change?
- Was there anything confusing that you wanted to understand but couldn't?

Do not answer the questions in a questions/answer format. Write in complete sentences and discuss your findings or perceptions of PHP.

To get started with analyzing what you're seeing, you'll need Google Chrome's Web Tools or to use web tools in another browser. After you're in the Network tab, refresh the page or press CTRL + R. That will refresh the page and start recording network data. You can stop it after however long you'd like. What will appear is a list of items used on the network. You can click any of the objects under "Name" and it will give you more information. The information available will be in tabs (Headers, Preview, Response, Initiator, and Timing); clicking on different ones will show you info about the headers used, download times, etc.

Submission
The initial post is **due Friday, April 8th at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "Website Network Information" where you can post your initial post. You can also visit the Discussion Board area directly from the Course Materials folder.

Response Post due April 11th at midnight
Visit a website another student has discussed that you haven't already mentioned and in one paragraph, compare the results of your findings. Were any of the things you saw on the websites you visited like what their website is showing? What is different? Why do you think they are different or similar? Feel free to use any of the discussion points above to compare findings. This response post doesn't have to be as thorough as the first one, but you should be mentioning specific things and drawing some comparisons.

The response post is **due Sunday, April 11th at midnight**. To submit, go to Blackboard → Course Materials → Lecture Notes for this week's class. There will be a discussion area called "Website Network Information" where you can post your response post. You can also visit the Discussion Board area directly from the Course Materials folder.

Website Network Information Rubric – 2pts
The initial post is worth 1.5pts and the response post is worth .5pt for a total of 2 points. I will be evaluating your posts based on the following criteria:

- Did you mention specific information about what you saw in the inspector?
    - Status codes?
    - POST/GET methods being used?
- Was the information provided useful/interesting?
- Did your response post contribute to the original post?
    - Avoid summarizing the other person's post or simply saying that it was a good post.

Next Week

Next week, we will learn about databases and the language used to manipulate them (SQL). Specifically, we will learn about tables, rows, columns, and basic SQL syntax used to retrieve data sets from tables.

Final Project

The semester is wrapping up quickly as there is about a month left before the Final Project is due. Please make sure you are making progress on it and reaching out to me if you need help with it.