HTTP Basics

The Web runs on HTTP, or HyperText Transfer Protocol. This protocol governs how web browsers request files from web servers and how the servers send the files back.

When a web browser requests a web page, it sends an HTTP request message to a web server. The request message always includes some header information, and it sometimes also includes a body. The web server responds with a reply message, which always includes header information and usually contains a body. The first line of an HTTP request looks like this:

```
GET /index.html HTTP/1.1
```

This line specifies an HTTP command, called a method, followed by the address of a document and the version of the HTTP protocol being used. In this case, the request is using the **GET** method to ask for the *index.html* document using HTTP 1.1. After this initial line, the request can contain optional header information that gives the server additional data about the request. For example:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 10.0 [en]
Accept: image/gif, image/jpeg, text/*, */*
```

The User-Agent header provides information about the web browser, while the Accept header specifies the MIME types that the browser accepts. After any headers, the request contains a **blank line** to indicate the end of the header section. The request can also contain additional data, if that is appropriate for the method being used (e.g., with the **POST** method, as we'll discuss shortly). If the request doesn't contain any data, it ends with a blank line.

The web server receives the request, processes it, and sends a response. The first line of an HTTP response looks like this:

```
HTTP/1.1 200 OK
```

This line specifies the protocol version, a status code, and a description of that code. In this case, the status code is "200," meaning that the request was successful (hence the description "OK"). After the status line, the response contains headers that give the client additional information about the response. For example:

```
Date: Sat, 27 Jan 2024 20:25:12 GMT
Server: Apache 1.3.33 (Unix) mod_perl/1.26 PHP/5.0.4
Content-Type: text/html
Content-Length: 141
```

The Server header provides information about the web server software, while the Content-Type header specifies the MIME type of the data included in the response. After the headers, the response contains a blank line, followed by the requested data if the request was successful.

The two most common HTTP methods are GET and POST.

- The GET method is designed for retrieving information, such as a document, an image, or the results of a database query, from the server.
- The POST method is meant for posting information, such as a credit card number or information that is to be stored in a database, to the server.
- The GET method is what a web browser uses when the user types in a URL or clicks on a link.
- When the user submits a form, either the GET or POST method can be used, as specified by the `method` attribute of the `form` tag.

Form Processing

A GET request encodes the form parameters in the URL in what is called a query string:

```
/path/to/chunkify.php?word=despicable&length=3
```

A POST request passes the form parameters in the body of the HTTP request, leaving the URL untouched.

The most visible difference between GET and POST is the URL line. Because all of a form's parameters are encoded in the URL with a GET request, users can bookmark GET queries. They cannot do this with POST requests.

The type of method that was used to request a PHP page is available through `$_SERVER['REQUEST_METHOD']`. For example:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
  // handle a GET request
} else {
  die("You may only GET this page.");
}
```

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope:

- $_SERVER
- $_REQUEST
- $_POST
- $_GET

Task 1:

- At this moment you can use **chunkify.html and chunkify.php** to see how it works.
- Convert the code from POST to GET and observe the changes.

Self-Processing Pages

One PHP page can be used to both generate a form and process it. If the page shown in **temp.php** is requested with the GET method, it prints a form that accepts a Fahrenheit temperature. If called with the POST method, however, the page calculates and displays the corresponding Celsius temperature.

Task 2:

- Now create a **calculator.php** where you can have 4 buttons for +, -, *, / and accept two numbers from user to perform simple mathematical operation.

Sticky Pages

Many web sites use a technique known as sticky forms, in which the results of a query are accompanied by a search form whose default values are those of the previous query. For instance, if you search Google (http://www.google.com) for "Programming PHP," the top of the results page contains another search box, which already contains "Programming PHP." To refine your search to "Programming PHP from O'Reilly," you can simply add the extra keywords.

Use **tempsticky.php** to see the effect.

Task 3:

- Can you also make your **calculator.php** a sticky web page?

Multiple Selection: Select Box and Checkbox

HTML selection lists, created with the `select` tag, can allow multiple selections. To ensure that PHP recognizes the multiple values that the browser passes to a form-processing script, you need to make the name of the field in the HTML form end with `[]`. For example:

```
<select name="languages[]">
  <input name="c">C</input>
  <input name="c++">C++</input>
  <input name="php">PHP</input>
  <input name="perl">Perl</input>
</select>
```

Now, when the user submits the form, $_GET['languages'] contains an array instead of a simple string. This array contains the values that were selected by the user.

Task 4:

- Complete the **selectbox.php** and **checkbox.php** files to see the effect.

Form Validation

When you allow users to input data, you typically need to validate that data before using it or storing it for later use. There are several strategies available for validating data. The first is JavaScript on the client side. However, since the user can choose to turn JavaScript off, or may even be using a browser that doesn't support it, this cannot be the only validation you do.

A more secure choice is to use PHP to do the validation. We will practice form validation by creating a log in system in the next class.