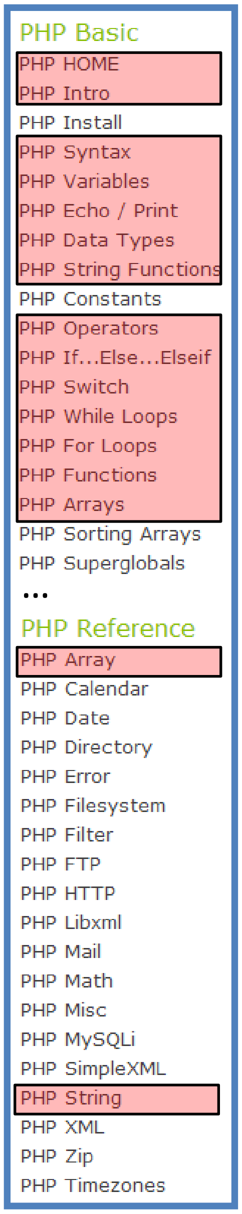
PHP for Sheridan Students

1. Introduction

In *JavaScript for Sheridan Students* you learned how to write JavaScript programs to generate and change the DOM after a web page is loaded. This allows the creation of **dynamic pages** – i.e. pages that change after loading, either on their own or in response to user input.



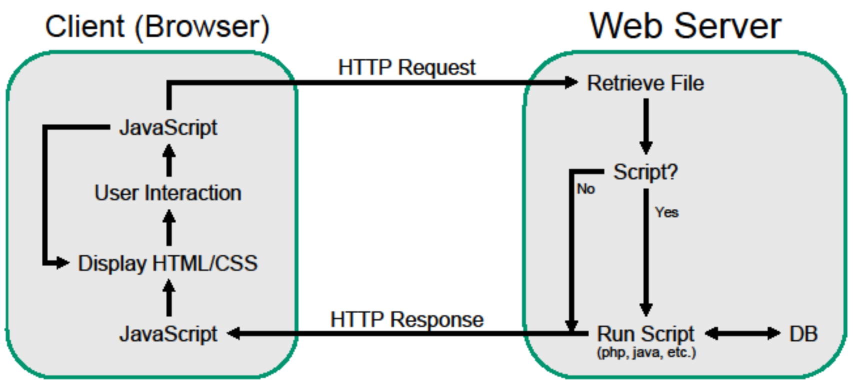
In this chapter, we will introduce server-side scripting. Server-side scripting is a way of generating **custom page content** – i.e. content that can be tailored to specific users. Scripting on the server side keeps your code out of site and allows you to access private or proprietary content from a server-side database.

1.1 How to Use this Text

This text is meant as a complement to other more complete reference and tutorial sources on the web. In particular, I recommend you use http://w3schools.com. The official PHP manual at http://www.php.net/manual/en/ can also be very useful. The graphic on the right highlights the sections of w3schools that will be covered here.

1.2 Web App Architecture

Up until now you have exclusively been exploring the client side of the client-server architecture pictured below. Now you will start exploring how to write programs that run on the server side using the world's most popular server-side scripting language, **PHP** (**P**HP: **H**ypertext **P**reprocessor). Note that PHP is not the only server-side scripting system in use today (Java and Active Server Pages are also popular choices)[1] and is often not the best choice for large scale apps (although though Facebook is one notable exception, making make heavy use of PHP scripts).



[1] http://w3techs.com/technologies/overview/programming_language/all

1.3 Software

Server-side scripts do not run in the user's browser. They run on the server side. This means that in order to test your PHP scripts you have to get access to a PHP- and MySQL-enabled web server.

One way to do this is to turn your PC into a web server using WampServer (http://www.wampserver.com/en/) or XAMPP (http://www.apachefriends.org/index.html). Another way is to use an external server like http://mobile.sheridanc.on.ca/ to develop and debug your code on line.

Whichever option you choose, you should be able to configure your IDE to automatically transfer your project to the server and run it. Your instructor will let you know his or her preferred option and will walk you through the necessary setup.

If you are using the mobile.sheridanc.on.ca server, you should create a file called ".htaccess" with the following contents in your public_html folder:

```
php_flag display_errors 1
php_value error_reporting 32767
php_flag xdebug.default_enable on
```

This will make sure that you get a full report on any syntax or runtime errors produced by your PHP programs.

1.4 Your First PHP Program

1.4.1 The `index.php` Template

Whichever IDE you are using, you ought to be able to create a new project with an index.php file which will contain some template code. Here is the default NetBeans template. You get this when you select "New PHP Web Page" (the "New PHP File" option gets you a much more minimal template).

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
        // put your code here
        ?>
    </body>
</html>
```

If your first php file doesn't look like the one above, you might want to quickly type this in before continuing. Note that the default template does not contain a viewport meta tag:

```
<meta name="viewport" content="width=device-width">
```

Without this, any pages created might not look very good on mobile devices. You might want to consider adding this tag to your IDE's default template if it is not already there.

1.4.2 "Hello, World!"

It's time for the Hello World program! In the template from your IDE, type the following between the `<?php` and `?>` tags:

```
        echo "<p>Hello, World!</p>";
```

Then run the program. The page should just say "Hello, World!" and nothing else.

View the page source in the browser to take a look at the output of your PHP script. Notice that the PHP code you wrote above is nowhere to be seen in the page source. This is because the code was run on the server and only the output of this code was sent to the browser. The `echo` command causes the specified string to be written into the output of the page.

Congratulations! You are now a server-side scripter. ☺

Try It Yourself: The "Hello, World!" Remixes

Take a look at **HelloWorld.php**, **HelloWorldNoPHP.php** and **HelloWorldAllPHP.php** from the **Chapter1Examples.zip** file. Run them yourself (either through your IDE or by uploading them to your own web space) or try them on line at http://mobile.sheridanc.on.ca/~scottsam/webdev2/. Look at the page source in each case and answer the following questions:

1. Why can't you see the PHP code?
2. If there's no PHP then in what sense are you really viewing the "page source"?
3. What are the differences in the page source of the three pages?
4. Can you explain the differences in the page source of the three pages? Make a note of your ideas and keep them handy as you read the next section so you can check if you got it right.

2. PHP Basics

A PHP script's job is to respond to an HTTP request by helping the server to produce an HTTP response (usually containing HTML, CSS and/or JavaScript). An HTTP response consists of headers and a body. In this course you usually won't have to worry about the headers – they will be created automatically for you. So the job of the PHP script will usually just be to output the HTML, CSS and JavaScript code that will form the body of the HTTP response.

A PHP script can access files and databases on the server side to create custom content for the user. When an HTTP Request for a PHP file arrives at the server, the PHP file is retrieved, the PHP scripts are executed, and the output is sent back to the client as part of the HTTP Response.

PHP files in practice usually look like a mixture of HTML, CSS, JavaScript and PHP. But it is perhaps better to think of them as entirely containing PHP script. The examples below may help to make this clearer.

2.1 The "Pure" PHP Hello World Program

Here's an example of a "pure" php program (**HelloWorldPurePHP.php** in the example pack).

```
<?php
     echo "<!DOCTYPE html>";
     echo "<html>";
     echo "<head>";
     echo "<meta charset='UTF-8'>";
     echo "<title>Hello World</title>";
     echo "</head>";
     echo "<body>";
     echo "<p>Hello, World!</p>";
     echo "</body>";
     echo "</html>";
```

All PHP scripts must start with `<?php`,and the `echo` statement is PHP's basic print statement that causes a string to be written into the output. Note that as in other web languages, strings can be specified in either single or double quotes as needed. However double-quoted strings are usually better because they support a number of escape sequences (like `"\n"` if you want to put a carriage return into your output) and will also make the outputting of variables easier (see section 2.4.3).

The output of the above program (visible in the page source of the browser when you run it) will be a classic HTML hello world page, like this:

```
<!DOCTYPE html><html><head><meta http-equiv="Content-Type"
content="text/html; charset=UTF-8"><title>Hello
World</title></head><body><p>Hello, World!</p></body></html>
```

Question: Why is everything on one line in the output? How could we change that?

2.2 The "Hybrid" Hello World Program

Most of the output of PHP programs is static (unchanging) HTML. As a shortcut (so you don't have to type echo all the time) PHP supports the mixing of HTML and PHP scripts like this (**HelloWorld.php** in the example pack):

```
<html>
    <head>
        <meta charset='UTF-8'>
        <title>Hello, World!</title>
    </head>
    <body>
        <?php
        echo "<p>Hello, World!</p>";
        ?>
    </body>
</html>
```

In the above example, the only part that is in PHP format is the part in bold. Notice that when we mix HTML and PHP we need both an opening PHP tag `<?php` and a closing tag `?>`. The closing tag was not necessary in the pure PHP version.

One way to view this situation is that the entire file is a PHP program where any line not enclosed in `<?php ?>` tags is converted to an `echo` statement.

The other, perhaps more common way to think of this program is that it contains a mix of HTML and PHP. However this way of thinking can be a little bit dangerous and could lead to confusion, since PHP is executed on the server side while the output of the PHP script is interpreted and/or run on the client side.

From the PHP developer's point of view HTML, CSS and JavaScript is all just data to be sent to the client.

2.3 Do We Really Need PHP?

You might be asking yourself what the point of the PHP element is in the "hybrid" version of Hello World. And you're right, we don't need it at all. So the entire PHP program could just look like this (**HelloWorldNoPHP.php** in the example pack):

```
<html>
    <head>
        <meta charset='UTF-8'>
        <title>Hello World</title>
    </head>
    <body>
        <p>Hello, World!</p>
    </body>
</html>
```

In this case, we should probably convert this file from a `.php` file to a `.html` file so that the server doesn't waste time treating it like a script. (Don't worry, we'll soon get into PHP programs that can't just be replaced by their HTML equivalents.)

2.4 Important Features of PHP

We will get to a more interesting PHP program in a moment, but first some notes on the details of the PHP language. As with JavaScript, PHP syntax is based on C, and thus is very close to Java syntax. Functions are declared and used just like in JavaScript and assignment, expressions, if statements, loops, etc. all work pretty much the way you would expect from both Java and Javascript, with a few exceptions as outlined below.

2.4.1 Variables and Functions

Same as JavaScript

PHP's variables and functions are **weakly typed**: Values have types but function and variable types do not have to be declared in advance. The PHP language also uses **aggressive implicit type casting** to convert from one type to another as necessary.

The syntax for function declaration is the same as for JavaScript:

```
function function_name(parameter list) {
     code
}
```

Not the Same as JavaScript

Variables are never declared. They are created when first used and can be global or local to a particular function depending on where they are first used.

Not the Same as JavaScript

Variable names must start with a $ character.

```
$numRows = 0;                ← legal
$numRows = $y * 3 + 2;       ← legal
numRows = 0;                 ← no good!

function foo ($a, $b, $c) {← legal
     …
     return $d;
}
```

Not the Same as JavaScript

All variables are assumed to be local. If you want to use a global variable in a function you have to indicate it by using the `global` keyword as shown in the examples below:

```
$x = 1;          ← creates a global variable $x
function foo() {
     $y = 2;     ← creates a local variable $y
     echo $x;    ← attempt to access undefined local variable $x
     echo $y;
}

function foo2() {
     $y = 3;
     global $x; ← indicates that $x will refer to the global var
     echo $x;    ← access global variable $x
     echo $y;
}
```

PHP functions can also declare static variables that keep their values for the next call of the function.

See the **globalVariableTests.php** file in **Chapter1Examples.zip** or run it on line at http://mobile.sheridanc.on.ca/~scottsam/webdev2/.

2.4.2 The Script vs. the Expression Tag

Sometimes you want to write a whole bunch of PHP code. That's what the `<?php ... ?>` tag is for, also known as the **PHP tag**. But sometimes you just want to echo the contents of a variable, the return value of a function or the result of an expression. For that you can use the **expression tag**: `<?= ... ?>`.

A tag like this…

```
<?= expression ?>
```

… is shorthand for this …

```
<?php
   echo expression;
?>
```

Here are some examples:

```
<?= $myVar ?>          ← echoes the contents of variable $myVar
<?= $x + $y ?>         ← echoes the result of adding $x and $y
<?= foo(1,2,3) ?>      ← echoes the return value of foo(1,2,3)
```

2.4.3 Strings

PHP programs by their very nature are string processing programs. This makes string data very important.

Not the Same as JavaScript

The concatenation operator is the . (dot) operator, not the + operator as in Java and JavaScript. It is actually a very good thing for a language to use to different operators for these two very different operations, but it can cause headaches to people used to JavaScript and Java.

The function below returns a string which is the concatenation of strings $a, $b and $c (if these variables do not contain strings, they will be automatically cast to the string type and then concatenated).

```
function($a, $b, $c) {
   return $a.$b.$c;
}
```

On the other hand, the function below will return the numeric addition of $a, $b, and $c. If they are Strings, it will cast them as numbers first (defaulting to 0 if they are badly formatted) and will return a number that represents their addition.

```
function($a, $b, $c) {
   return $a+$b+$c;
}
```

Troubleshooting tip: If you see a 0 in your output where you expected a longer string, you probably used + instead of the dot for concatenation.

See **stringConcatenationTests.php** in **Chapter1Examples.zip** or run it on line at http://mobile.sheridanc.on.ca/~scottsam/webdev2/.

Not the Same as JavaScript

You can embed variable names inside double-quoted strings. So instead of this:

```
echo "Hello " . $name . ". It's " . $weekday . " today.";
```

You can write this:

```
echo "Hello $name. It’s $weekday today";
```

If you want to echo the value of a more complicated variable expression, like an associative array reference, you can use curly braces inside the string, like this:

```
echo "Hello {$names["owner"]}!";
```

Note that this does not work inside single quoted strings. The following will simply echo *Hello {$names["owner"]}!* to the screen:

```
echo 'Hello {$names["owner"]}!';
```

Not the Same as JavaScript

Strings are not objects. So don’t expect the `.` operator to work for dereferencing and accessing methods, as in `$s.length()`. Instead you must use built-in global functions like `strlen($s)`. (Even if strings were objects in PHP, the dot operator would not be how you access the methods since the dot is for concatenation in PHP.)

Same as JavaScript

There are lots of string manipulation functions available. You can find a list of them in the w3Schools **PHP String** section of **PHP Reference** on w3Schools.

2.4.4 Arrays

PHP programs also have to do a lot of array manipulation. Parameters you get from HTTP Requests are stored in arrays, and so are the results of database queries.

Same as JavaScript

PHP supports both numeric and associative arrays. You can create an array and an array index just by assigning something to it, you can mix and match integer and string indices and you can mix and match the types of the values stored in an array (though this is not usually a good idea in terms of code clarity):

```
$a[0] = 12;      ← These lines create an array with 2 entries
$a["hi"] = "lo";
```

You can also create an array with numeric indices (starting at 0) like this:

```
$cars = array("Saab","Volvo","BMW","Toyota");
```

And you can create an associative array like this:

```
$ages = array("Peter"=>32, "Quentin"=>30, "Joe"=>34);
```

Not the Same as JavaScript

Arrays are not objects. So don’t use `$a.length`. Instead you manipulate arrays with built-in global functions like `count($a)`. You can find a list of array functions in the **PHP Array** section of **PHP Reference** on w3Schools.

2.4.4 Control Structures

Same as JavaScript

All the basic control structures (if, else, switch, while, for, do, etc.) work in exactly the same way as their Java and JavaScript counterparts with one exception…

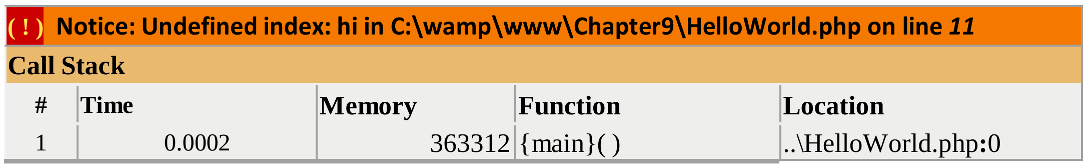
Not the Same as JavaScript

PHP contains a special `elseif` keyword to use when stringing together if statements, like this:

```
if ($a == 6) {
…
} elseif ($a == 7) {
…
} elseif ($a == 8) {
…
} else {
…
}
```

2.4.5 Error Reporting

Unlike JavaScript, PHP can be configured so that syntax and runtime errors will actually show up as part of the output (Hallelujah!). For example, if you try to access an undefined array index, you'll get an error a bit like the one below returned as part of the web page in the HTTP response. (The exact details of how it looks, or whether it appears at all, can be configured by the system administrator for the web server.)



(!) Notice: Undefined index: hi in C:\wamp\www\Chapter9\HelloWorld.php on line *11*

Call Stack				
#	**Time**	**Memory**	**Function**	**Location**
1	0.0002	363312	{main}( )	..\HelloWorld.php**:**0

Note that sometimes PHP error reporting is turned off as a security measure. You might have to do something extra to turn it on for your server (see section 1.3).

2.5 JavaScript vs. PHP

The table below compares some of the important features of JavaScript and PHP side by side. Read this in conjunction with the client-server diagram at the beginning of the chapter and make sure you understand what each row of the table is telling you and why each language has the features and limitations that it has.

	JavaScript	PHP
Runs On:	Browser	Server
Access to client machine:	Some	None
Access to server machine:	None	Full
Access to database:	None	Full
Access to DOM:	Full	None
Input:	User interaction	HTTP Parameters, Files, and Databases
Output:	Pop-ups, changes to DOM	Text (HTML, CSS, JavaScript)
Comments:	Java-like, visible in page source	Java-like, not visible in page source
Coding Styles:	Imperative, Event-Driven, Functional, Object-Oriented	Imperative, Functional Object-Oriented
Syntax:	Java -Like	Java -Like
Variables:	Weakly Typed	Weakly typed (names start with $)
Array Support:	Numeric and Associative	Numeric and Associative

2.6 Exercises

Note: PHP scripts are supposed to be for creating custom content using external input. The two main sources of input to a PHP script are parameters in the user's request and content stored in a server-side database. We haven't learned how to deal with either source of input yet, so for now all your scripts will use variables or constants at the top of the program that you can tweak to "customize" the content displayed. In the next lesson, you will learn how to set the values of these variables from user input.

1. **Basics:** Write a PHP script that creates an HTML page to say hello to a user by name. Include a `$name` variable at the top of the script that can be modified to change the output of the script. Write two versions – one using a PHP script tag and one using a PHP expression tag.

2. **Numeric Types:** Write a PHP script that computes the amount of tax for a three-item restaurant bill (burger, fries and drink) and then displays the items in the bill with their prices, the subtotal, tax (13%) and total, nicely formatted. Include `$burger`, `$fries`, and `$drink` variables at the top of the script to hold the prices of each item.

3. **Functions:** Modify the question 2 program so that the three variables are passed to a function which outputs the bill.

4. **Loops:** Write a PHP script that creates an HTML page to display all the even numbers from `$start` to `$end`, each in its own `<p>` element, where `$start` and `$end` are variables defined at the top of the script.

5. Write a PHP script to determine whether or not an integer is a prime number. The prime number check should be done by a function that you define somewhere in the script. Include a `$n` variable at the top of the script to set the number you are checking. The prime number check should be done by a function that returns `true` or `false`.

6. Write a PHP script that uses the prime number checking function you wrote in question 5 to fill an array with the first 100 prime numbers and report them in an HTML document.

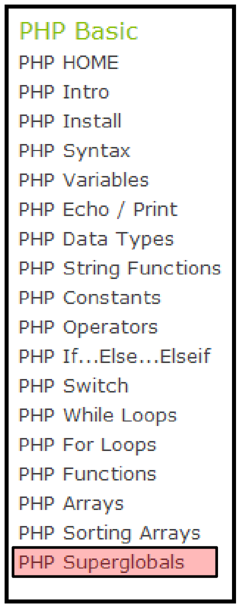
7. Write a PHP script that creates a nicely-formatted multiplication table in an HTML page. Include two variables at the top of the script that can be changed to alter the number of rows and columns displayed.

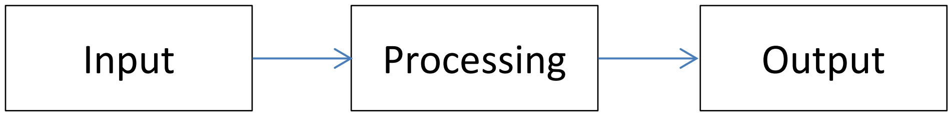
8. Write a PHP script that displays an HTML form that could be used to allow a user to update their information on your spam server. The form should have fields for name, phone number and email address as well as checkboxes for at least 3 email subscriptions. Include variables at the top of the script to hold default values for each of these fields (including whether or not each box should be checked). Make sure these default values are displayed when the page loads. (Reminder: Use the `value` attribute of text boxes to set a default value and use the `checked` attribute to auto-check checkboxes.)

9. Write a PHP script that simulates a simple slot machine. When the page is loaded, the program should randomly choose three different fruit images to display for the user. If two of them match, they win. If all three match they get the jackpot. Otherwise they lose. Include a button to reload the page so they can try again. You can use the fruit images in the **fruit.zip** file. You will also need to use the PHP `rand` function – `rand(`*`min, max`*`)` returns a random integer between `min` and `max`.

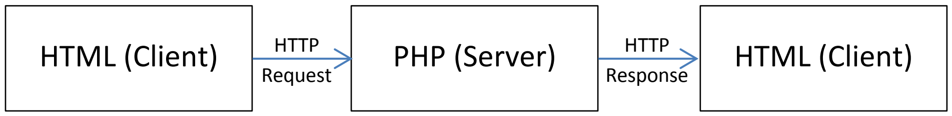
3. Input-Processing-Output with PHP Programs

The Input-Processing-Output model is one of the simplest model of computer software: A user provides input data which is processed by the program and then the results of that processing are sent as output back to the user.





In simple PHP programs, the user input comes from a web form submitted to a PHP program, the processing is done in PHP and the output is an HTML page sent back to the client as an HTTP Response.



3.1 Input with HTML GET Parameters

The most common way to get user input into a PHP program is through HTTP Request parameters, also known as GET and POST parameters (we will just use GET parameters for now). You can send HTTP Request parameters from HTML forms (as you learned in your Web Dev 1 course) but as a quick starting point or for testing purposes you can also just type GET parameters directly into a URL.

The URL below contains a list of GET parameters (underlined). The start of the list is signified by the ? character. The format is `paramName=paramValue`. If there is more than one parameter, they are separated by the & character.

```
http://localhost/helloClient.php?firstName=Sam&lastName=Scott
```

The above URL contains two GET parameters called `firstName` and `lastName`. They have the values "Sam" and "Scott" respectively.

This URL could also have been generated automatically by the following form (**helloClientUserInfo.html** in the example pack):

```
<form action="helloClient.php" method="GET">
     First Name: <input type="text" name="firstName"><br>
     Last Name: <input type="text" name="lastName"><br>
     <input type="submit">
</form>
```

The important attributes above are:

- The `action` attribute tells the browser what URL to load when the submit button is pressed;
- The `method` attribute should be "GET";
- The `name` attributes on each input element, which cause a parameter with that name to be sent

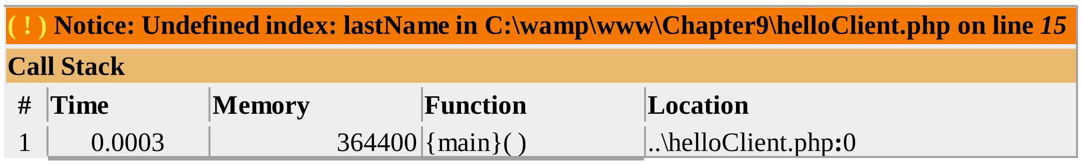
when the user presses the submit button (the value of each parameter comes from the contents of the input element).

3.2 Processing and Output (Using the Parameters in PHP)

A PHP script always has access to the GET parameters that came with the HTTP Request. They are stored in the **superglobal**[2] associative array $_GET respectively (note the underscore). In the above example, the `helloClient.php` program could access `$_GET["firstName"]` and `$_GET["lastName"]`, like this (**helloClient.php** in the examples on line).

```
echo "<p>Hello, Mr. {$_GET["lastName"]}. May I call you
      {$_GET["firstName"]}?</p>";
```

If you load this program by typing its URL into a browser but fail to specify parameters in the URL, you should see an error message like the one below (the exact appearance of this error message will depend on your PHP server settings).



(!) Notice: Undefined index: lastName in C:\wamp\www\Chapter9\helloClient.php on line *15*

Call Stack				
#	**Time**	**Memory**	**Function**	**Location**
1	0.0003	364400	{main}( )	..\helloClient.php:0

To get avoid this ugly error, you can use the extremely handy `boolean` function `isset()` to check whether a variable has a value before using it (see **helloClient2.php** in the example pack).

```
<?php
if (isset($_GET["lastName"])) {
    echo "<p>Hello, Mr. {$_GET["lastName"]}.";
    if (isset($_GET["firstName"]))
        echo "May I call you {$_GET["firstName"]}?</p>";
    else
        echo "</p>";
} else
    echo "<p>Hello. You forgot to specify your name.</p>";
?>
```

The above code recovers gracefully if one or more parameters are missing.

[2] The term “superglobal” is used for variables that can be accessed in functions without using the `global` keyword (see section 2 for a discussion of the `global` keyword).

3.3 Another Example

In the example below, the code says hello to the user one or more times, as specified with the parameter `numHellos` (**helloRepeat.php** in the example pack).

```
if (isset($_GET["numHellos"]))
   $n = $_GET["numHellos"];
else
   $n = 1;

for ($i = 0; $i < $n; $i++)
   echo "<p>Hello, World!</p>";
```

3.4 A Note on Interleaving PHP and HTML

In a case like 9.3.2, some coders would prefer to use plain HTML rather than the echo statement to place the "Hello World" text. This can be done like this (**helloRepeat2.php** in the examples on line).

```
<?php
if (isset($_GET['numHellos']))
   $n = $_GET['numHellos'];
else
   $n = 1;
for ($i = 0; $i < $n; $i++) {
   ?>                        ← close PHP tag
   <p>Hello, World!</p>      ← HTML to be repeated
   <?php                     ← re-open PHP tag
}
?>
```

Whether or not to do this is largely a matter of personal taste, but if taken too far, heavy interleaving like this can make the code much harder to read (for examples, see **helloClient3.php** and **helloClient4.php** in the examples on line).

No matter how heavily you interleave, if you hide the details of HTML inside `echo` statements, it can be hard for NetBeans to be sure that your syntax is correct and that you have not used deprecated tags. No matter how you write your code, you should always inspect your HTML output using *view page source* and the *developer tools* in your browser to make sure the output of your PHP script consists of well-formatted HTML, CSS, and JavaScript.

3.5 Exercises

1. **Basics:** Modify each of the exercises 1 through 6 from section 2.6:
    a. Make them get the values of their input variables from GET parameters specified in the URL. If any parameters are missing, either assign default values or change the output so that the user sees an error page.
    b. Create an HTML form in a separate file so that you can specify their parameter values more easily.

2. **Multiplication Tables**: Modify exercise 7 from section 2.6
    a. Make it get the number of rows and columns from parameters. If the parameters are missing, assume a default value of 10 for each. Put a warning on the page in red text if either parameter is missing.
    b. Create an HTML form in a separate file so that you can specify the parameters for exercise 3 more easily.

3. **Multiplication Tables**: Modify exercise 3 above so that the user can also specify the following using the same form that specifies the number of rows and columns:
    a. The title for the table
    b. Foreground, background, and heading background colors.
    c. Addition, subtraction, or multiplication as the operation (should also change some text on the page as appropriate).
    d. Default widths and heights for the <td> elements.

4. **Rock Paper Scissors:** Make a page that plays rock/paper/scissors with the user. It should take a parameter that specifies the user's move and then generate a random move of its own. It should display appropriate images for the computer's move and the player's move and then state who won.

5. **Slot Machine (tricky):** Modify the slot machine example from the exercise in section 2.6. Make it so that when the user first arrives they get a credit of 10 tokens (displayed on the page) and a form element that lets them choose how many tokens they want to bet. Then make the SPIN button reload the page and send the current credit as a parameter. When the page reloads, they should win 5 times their bet for three matching symbols or 2 times their bet for 2 matching symbols. Otherwise they lose a credit.

3.6 Form Elements with Multiple Values

There is a lot more that can be said about using forms to send data to the server, but you know most of it already from your Web Development 1 course. You can use radio buttons, drop down lists, text areas, color choosers, number elements, time and date elements and more.

From the server's perspective, these input elements almost all operate just like text inputs – that is, they each send a string parameter to the server. But some elements allow the user to specify multiple values, and these need to be treated differently.

3.6.1 Check Boxes

Check boxes are typically used to send multiple parameters under the same name. For example, the code below will cause between 0 and 4 parameters to be sent to the server, each named `activity`.

```
<input type="checkbox" name="activity" value="tennis">tennis
<input type="checkbox" name="activity" value="hockey">hockey
<input type="checkbox" name="activity" value="bowling">bowling
<input type="checkbox" name="activity" value="darts">darts
```

If the user checks both “tennis” and “hockey” the parameter string will be

```
target.php?activity=tennis&activity=hockey
```

`In the PHP program, $_GET["activity"]` will hold either “tennis” or “hockey”, but not both.

To get around this, you have to name the check boxes in a way that signals to PHP that they should be stored in an array. You do that by adding `[]` to the end of the `name` attribute, like this:

```
<input type="checkbox" name="activity[]" value="tennis">tennis
<input type="checkbox" name="activity[]" value="hockey">hockey
<input type="checkbox" name="activity[]" value="bowling">bowling
<input type="checkbox" name="activity[]" value="darts">darts
```

On the server side, this will come through as an array parameter. So `$_GET["activity"]` will be an array with the indices 0, 1, and so on set to the values the user checked.

To find out how many values the user checked, use the global `sizeOf()` function to retrieving the size of the array:

```
sizeOf($_GET["activity"])
```

To access a particular array element, specify the numeric index in square brackets:

```
$_GET["activity"][0]
```

The above will return the first item in the array as a string.

Careful! You will get errors if you try to access an array that doesn’t exist (i.e. the user checked no boxes). Don’t forget to check for the existence of every parameter using the `isset()` function before you try to access it.

3.6.2 The `multiple` Attribute

You can also cause a `<select>` element to allow the user to select more than one value by including the `multiple` attribute, like this:

```
<select name="carMake[]" multiple>
     <option>Saab</option>
     <option>Audi</option>
     <option>BMW</option>
     <option>Maybach</option>
</select>
```

Just as with the check box, you should use `[]` to tell PHP to read the values into an array, and you should use `isset` to find out whether the user selected any values at all.

3.7 Exercises

1. **Even or Odd:** Create an HTML page with a form that allows the user to select multiple integers (from 1 to 10) to send to the server. You can use checkboxes or a `<select>` element. The form should redirect to `checkInts.php`.

2. **Even or Odd (part 2):** Now create the `checkInts.php` file. This program should output an unordered list of the integers it receives (using `<ul>` and `<li>`) stating for each integer whether it is odd or even. Then it should output the sum of all integers. If no integers were selected, the message "nothing selected" should appear and the sum should show as 0.

3.8 Data Validation

It's really important to validate and clean up any data coming into your server from a form. If you don't validate and clean up form data you can end up looking bad when the page crashes because the user typed a bad input. But worse yet, you open yourself and your users up to possible attacks by hackers.

To see a simple hack in action, try the file **hackMe.php** from the example pack. Send the following GET parameter by typing it into the URL when you run the file:

```
hackMe.php?param=<script>location.reload()</script>
```

This benign hack causes a script element to be placed on the page that constantly refreshes it, effectively disabling the page and wasting browser cycles. At the time of writing, this hack works on the current version of Firefox and IE but not on Chrome, which is smart enough to detect it (check the messages in the JavaScript console of Chrome).

This is just a simple example. If you don't properly sanitize your data and you are using a database, you can also be susceptible to an SQL Injection Attack – more on this when we get to the SQL part of the course.

3.8.1 Where to Validate

All data should be validated twice.

The first line of defense is in the client browser. You should check the form data before you allow it to be submitted to the server. This will be good enough to stop most errors from happening, and checking on the client side is usually a better experience for the user. But it is not enough to keep you safe.

You also need to check the data you receive in your PHP program. There are at least two good reasons to duplicate the data validation effort like this. The first is that different browsers (especially older ones) may not be able to run your client-side validation code. The second reason is that Hackers will always be able to find a way to disable or circumvent client-side validation, so even if your client-side data validation is working perfectly, your PHP programs could still receive bad data.

3.8.2 HTML5 Client-Side Validation

HTML5 contains almost everything you need to fully validate form data before it is sent.

The following HTML5 input types can be used to prevent badly formatted data from being submitted:

Input type	**Validation**
`number`	Must be an integer
`range`	Must be an integer
`email`	Must be well-formatted email address
`url`	Must be well-formatted URL
`color`	Must be a hex color string (#RRGGBB)
`date`	Must be a legal date (YYYY-MM-DD)
`time`	Must be a legal time (HH:MM)

The following attributes can be added to form elements in HTML5 to further control the user:

Attribute	**Input Type / Form Element**	**Validation**
`required`	`all`	Must be filled in
`min`, `max` and `step`	`number, range, date, time`	Value must be in specified range, with step setting the distance between legal values (e.g. if min=10, max=20 and step=3, only 10, 13, 16, and 19 are valid)
`maxlength`	`text`	Sets a maximum number of characters.
`pattern`	`text`, `search`, `url`, `email`, and `password`	Specifies a regular expression that must be matched.
`title`	`all`	Defines a hint the user will get when they hover over the form element and when they try to submit bad data.

The attributes in the table above are mostly fairly straightforward. For example, the code below forces the user to enter an even integer in the range 0 to 100:

```
<input title="Even number from 0 to 100" type="number"
       name="even" min="0" max="100" step="2" required>
```

The following forces the user to enter a code between 1 and 10 characters long:

```
<input title="Between 1 and 10 characters long" type="text"
       name="code" maxlength="10" required>
```

The pattern attribute for regular expressions takes us beyond the scope of this course, but you can learn on your own how to use its powerful verification methods to control the user even more strongly. Here's an example that forces the user to enter a legal Java variable name (starts with a letter and contains only letters, digits and underscore characters).

```
<input type="text" name="variableName" required
       pattern="[a-zA-Z][a-zA-Z0-9_]"
```

```
            title="Enter a legal Java variable name.">
```

If you want more information, there's a nice regular expressions tutorial here: http://regexone.com/.

The above examples can all be found in **clientSideVal.html** file in the example pack.

3.8.3 JavaScript Client-Side Validation

Before HTML5, one of the biggest uses of JavaScript in web applications was form validation. The basic technique was to create a validation function which returns `true` if the form data is ok and `false` otherwise, then add it to the `<form>` element like this:

```
        <form name="form1" onsubmit="return validate()"
              action="checkInt.php">
```

The `validate` function could check all the fields, issue messages to the user and return `false` to prevent form submission if something was wrong.

With HTML5 you almost never need to use JavaScript in this way anymore, although JavaScript functions can still be used to do live validation and pop up hints and feedback as the user types. There's a nice example of that here: http://alistapart.com/d/forward-thinking-form-validation/.

You still also need JavaScript to do checks that involve more than one field (for example when you ask the user to enter their email address twice, you need to check that both fields contain the same contents).

3.8.4 Server-Side Validation

On the server side, the first thing you should always do is make sure the user gave you the parameters you want. You can use the `isset` function for this:

```
     if (isset($_GET["name"])) …
```

You also might want to make sure that the parameter is not the empty string with the `empty` function:

```
     if (empty($_GET["name"])) …
```

And if you were expecting a number, you should check that with the `is_numeric` function:

```
     if (is_numeric($_GET["age"])) …
```

Then there are a number of things you should probably do to clean up any parameter value you get from the user, especially if it might be echoed back to the user in the HTTP response.

1. Trim the data to remove leading and trailing whitespace
   ```
   trim($data);
   ```

2. Remove slashes which are sometimes inserted as escape characters by the browser
   ```
   stripslashes($data);
   ```

3. Replace html special characters with browser-safe equivalents (e.g. replaces “<” with “&LT;”)
```
      htmlspecialchars($data);
```

Here’s a boilerplate function from w3schools that can be used to sanitize any data string from the user:

```
function clean($data)
{
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

You would use it like this:

```
$name = clean($_GET["name"]);
```

Finally, PHP contains a large number of string functions that you can use to check all kinds of properties of the string (its length, whether it contains a particular substring, etc.) Here are a few ideas, but you should look at w3schools for the full PHP String reference.

Function	**What it does**
`strpos($s1,$s2)`	Find the location of $s2 in $s1
`stripos($s1,$s2)`	As above, but case insensitive
`substr($s,$start,$length)`	Extract a substring
`strtolower($s)`	Change to lowercase
`strtoupper($s)`	Change to uppercase
`strlen($s)`	Get string length
`strcasecmp($s1,$s2)`	Case insensitive comparision of $s1 and $s2
`preg_match($s)`	Find the first occurrence of a substring matching the given regular expression pattern.

Regular expressions in PHP work in more-or-less the same way as they do in HTML5. For a tutorial, go here: http://webcheatsheet.com/php/regular_expressions.php.

To see PHP form validation in action, take a look at **serverSideVal.php** and **serverSideVal.html** in the example pack.

3.9 Exercises

1. **Basics:** Go back to the programs you created in exercise 1 from section 3.4 (which in turn came from exercises 1 through 6 of section 2.6). Make sure you fully validate and clean up all user data on both the client and server side. Add code on the client and server side to make sure that numeric values entered are in an acceptable range. Note: Once you have tightened up the user input on the client side, you will need to test server-side validation either by typing parameters into the URL or by making a new HTML form where each input element is of type `text`.

2. **Even or Odd:** Add client and server-side data validation and cleanup to `checkInts.html` and `checkInts.php` (from the section 3.7 exercises). From the server, you should print error messages into the HTTP response if anything is wrong. To test server-side validation you can type parameters into the URL and/or make a new HTML form where each input element is of type `text`.

3. **Even or Odd with One File:** There is no particular reason why `checkInts.html` and `checkInts.php` (from exercise 2 above) have to be separate files. You could have one combined file that displays the form at the top, and then the result of the last form submission underneath it. Just create a new PHP web page, paste in the entire HTML form file, and then put the PHP code where it needs to be underneath the form and tweak as necessary.

4. **Multiplication Tables:** Go back to the multiplication table exercises from section 3.5 (and 2.6). Add HTML5 form checking to enforce the following limits: the custom title of the page must be 40 characters or less, the size of the table should be between 1 and 20 for both rows and columns. The cell width and height should have a minimum of 12 and a maximum of 30 pixels.

   Add PHP data validation to make sure the parameters (including the colors) are present and contain valid values before you use them.

5. **The Briefcase:** Make a PHP file that presents the user with an image of a closed briefcase and two sets of three input elements to enter two three digit codes (each digit can range from 0 to 9) to unlock it.

   When the user submits their guess the page should reload and the guess should be sent to a PHP script. If the digits they entered match the hard-coded combination, the briefcase will open. Make sure you validate their inputs in HTML5 and also check and sanitize on the server side.

   Try to make the input look as much like a briefcase code would look in real life. You can use the images provided in the example pack to show the briefcase (or you can find your own).

6. **The Briefcase: Extra Features**
   a. Display a running list of past codes they have tried (store these codes in input elements of type "hidden" so they keep getting resubmitted to your PHP program).
   b. Give hints in JavaScript alert boxes (number of correct digits, higher or lower, etc.)
   c. If they get one of the locks open, disable those input fields and display the combination that opened the lock in the fields. Then let them keep working on the second lock.