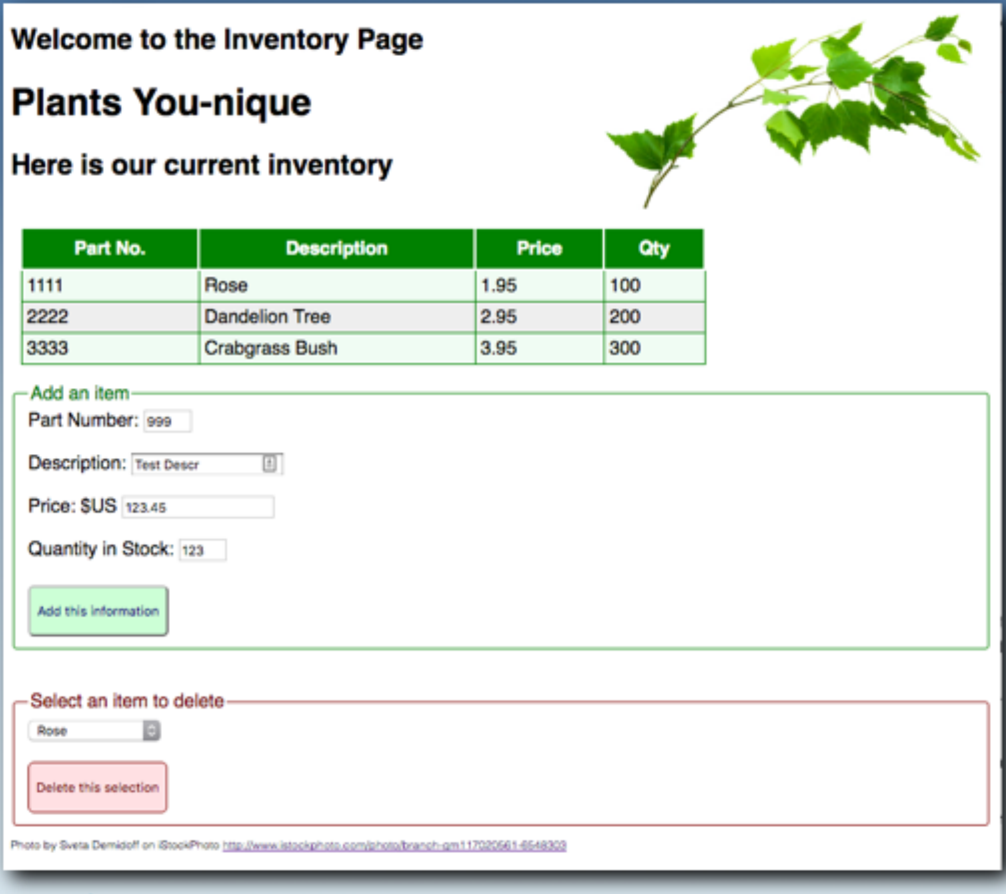
Array, Session, & Forms

Using PHP for Interactive Forms



Welcome to the Inventory Page

Plants You-nique

Here is our current inventory

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Add an item

Part Number: 999

Description: Test Descr

Price: $US 123.45

Quantity in Stock: 123

Select an item to delete

Rose

Photo by Sveta Demidoff on iStockPhoto http://www.istockphoto.com/photo/branch-gm117020561-6548303

web explorations

Summary

The client's story...

Your client has asked you to create an online inventory program. You've decided that PHP is the most appropriate language to prototype the application.

The client wants to track dozens of items but in the initial meeting, you decide on four important ones.

- Product ID
- Product Description
- Price
- Quantity in Inventory

Here are some of the things you need to figure out to make the app possible:

- Create a two-dimensional array holding the data. (Later this will be done using a database).
- Create a form that will allow the user to select an item and delete it from the list.
- Create a form that will allow the user to add new information to the list.
- You want to be able to sort the list automatically by Product ID
- You decide to have the PHP page call itself so you can have all the code in one place.

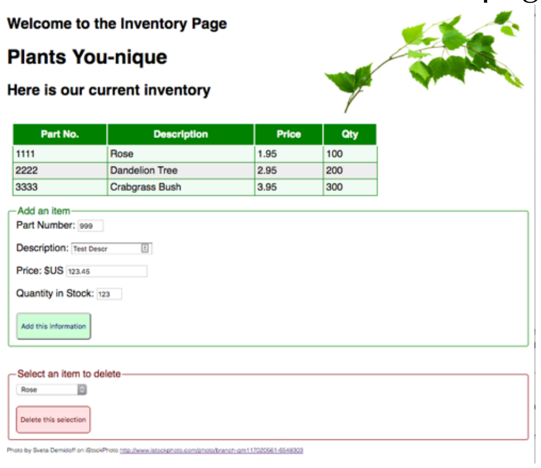
This lab will help you experience how to:

- Set up a working PHP page and build on it a step at a time.

- Set up and use a two-dimensional array.
- Display a table of data using PHP and foreach loops.
- Create a dynamic drop-down list from an array.
- Use function stubs for faster development.
- Have a PHP page call itself - First time user or returning user?
- Use Sessions to "remember" user information.
- Use multiple forms on one page.
- Delete records from an array using unset( )
- Adding records to an array.
- Formatting forms for greater accessibility
- Using CSS to improve the UX - User eXperience

Visualize

Here is a view of the finished page:



Welcome to the Inventory Page

Plants You-nique

Here is our current inventory

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Add an item

Part Number: 999

Description: Test Descr

Price: $US 123.45

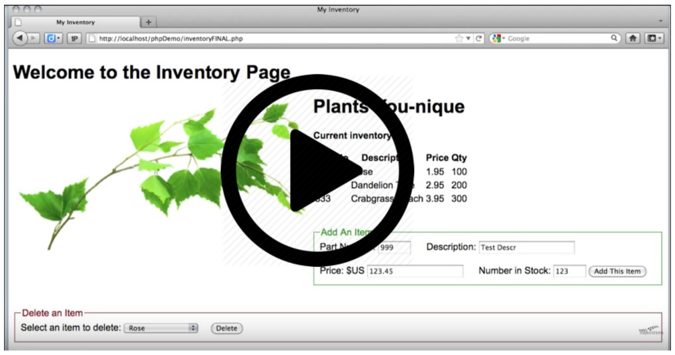
Quantity in Stock: 123

Select an item to delete

Rose

Photo by Sveta Demidoff on iStockPhoto http://www.istockphoto.com/photo/branch-gm117020561-6548303

Here is a short video, from an earlier version of the program, that was created to show the client a first-look.



My Inventory

http://localhost/phpDemo/inventoryFINAL.php

Welcome to the Inventory Page

Plants You-nique

Current inventory

	Descript	Price	Qty
	se	1.95	100
	Dandelion	2.95	200
33	Crabgrass ach	3.95	300

Add An Item

Part N 999 Description: Test Descr

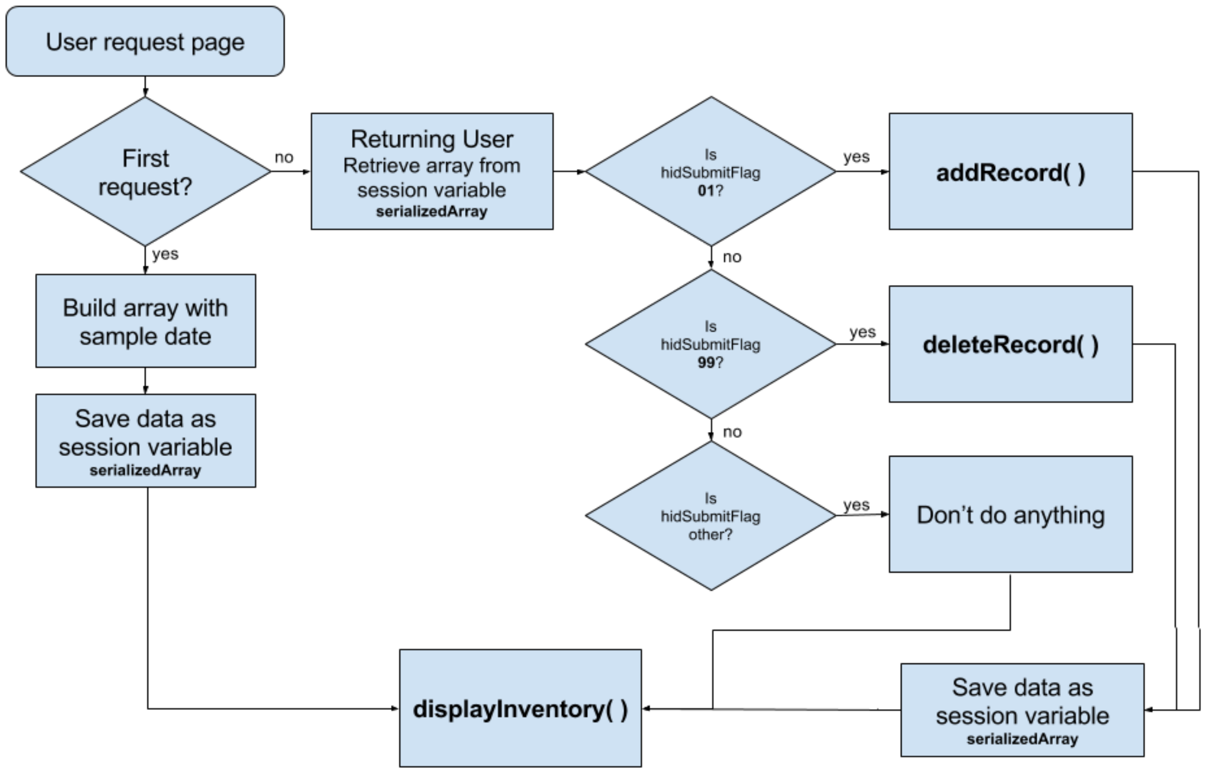
Price: $US 123.45 Number in Stock: 123

Delete an Item

Select an item to delete: Rose

Here is a flow chart showing how the program will respond to the user.

Program flow for inventory.php


User request page
First request?
no
yes
Returning User Retrieve array from session variable serializedArray
Build array with sample date
Save data as session variable serializedArray
Is hidSubmitFlag 01?
yes
no
addRecord( )
Is hidSubmitFlag 99?
yes
no
deleteRecord( )
Is hidSubmitFlag other?
yes
Don't do anything
displayInventory( )
Save data as session variable serializedArray

Setup up a PHP Page

Smart programmers work in stages.

First, hard-code in what you want to display on the page.
Then, convert that into PHP code.

Start off by creating a standard HTML page displaying a heading and a small hand-coded table with a header row, a data row, and four columns. Make sure to use the .php extension when naming the file and save it in the www folder of your localhost server.

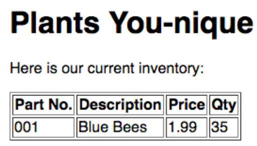
Type out this code. You will probably want to name the file **inventory.php** and save it in the **www folder** of your localhost.
You will learn and understand more if you type in the code. Just using copy/paste does not allow your brain to learn from the details.

You will want to change the heading comment with your own information.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
<!-- inventory.php -  Keep track of inventory
  Peter Johnson - peterk@webExplorations.com
  Written:   08-08-11
  Revised: PKJ 04-29-16 updated course presentation
  -->
 <title>My Inventory</title>

</head>
<body>
<h1>Plants You-nique</h1>

<p>
Here is our current inventory:<br />
<table border="1">
   <tr>
      <th>Part No.</th>
      <th>Description</th>
      <th>Price</th>
      <th>Qty</th>
   </tr>

   <tr>
      <td>001</td>
      <td>Blue Bees</td>
      <td>1.99</td>
      <td>35</td>
   </tr>
</table>
</p>
</body>
</html>
```

Before you add any PHP code, make sure to validate your web page to make sure all your HTML code is valid.

Here is what the output should look like:



Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
001	Blue Bees	1.99	35

Hard-code a 2D Array

Hard-code in a two-dimensional array.

A 2D array is basically an array of rows. Inside each row is another, mini-array for the columns. Think of an Excel spreadsheet.

We will hard-code in some initial data for now, but later, as you get more adept with PHP you'll want to use a database instead.

We want this array object to be global so we will put it just inside the <body> element of the page. Notice how the name of the variable tells everyone that this is an array.

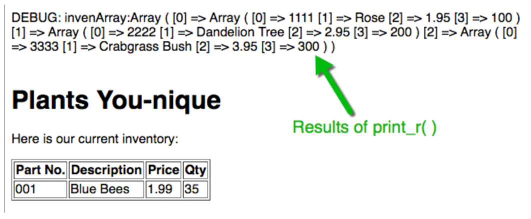
Quickly view the contents of an array using print_r( )

Use the function print_r( ) to display the contents of your array as a DEBUG statement.

Type in the following code right between the <body> and the <h1> tags on your page. (Don't duplicate the <body> and <h1>. Those are included for your reference.)

```
<body>

<?php
   // Create the inventory array
   $invenArray = array( );
   $invenArray[0][0] ="1111";
   $invenArray[0][1] ="Rose";
   $invenArray[0][2] ="1.95";
   $invenArray[0][3] ="100";

   $invenArray[1][0] ="2222";
   $invenArray[1][1] ="Dandelion Tree";
   $invenArray[1][2] ="2.95";
   $invenArray[1][3] ="200";

   $invenArray[2][0] ="3333";
   $invenArray[2][1] ="Crabgrass Bush";
   $invenArray[2][2] ="3.95";
   $invenArray[2][3] ="300";

echo("DEBUG: invenArray:");
print_r($invenArray);
echo("<br /><br />");
?>

<h1>Plants You-nique</h1>
```

When you view the page using localhost here is what it should look like:



```
DEBUG: invenArray:Array ( [0] => Array ( [0] => 1111 [1] => Rose [2] => 1.95 [3] => 100 )
[1] => Array ( [0] => 2222 [1] => Dandelion Tree [2] => 2.95 [3] => 200 ) [2] => Array ( [0]
=> 3333 [1] => Crabgrass Bush [2] => 3.95 [3] => 300 ) )
```

Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
001	Blue Bees	1.99	35

Notice the sample data. The consistency "1111", "2222", "3333" will help out later when you are viewing the array in a table. It will be easy to see if a row or column is out of place.

Write stub functions to build the code framework

By visualizing the program and planning ahead we know that we are going to be doing three main things. 1) Display the inventory, 2) Add new items, and 3) delete items.

Set up the structure of your program now by stubbing in these functions. You don't have to write the code right now. You are simply building the framework much like a carpenter will frame in the walls before putting adding on the windows, doors, and wallboard.

Inside the same <?PHP ?> markers where you have the array, add in the stubs for your functions. Notice the comment markers at the end of each function. This becomes helpful later after you've added lots of code to any of the functions.

Also, for readability, include two blank lines between each function. That will make them easier to find later, as you quickly scroll through your code.

```
<?php
   // Create the inventory array
   $invenArray = array( );
   $invenArray[0][0] ="1111";
   $invenArray[0][1] ="Rose";
   $invenArray[0][2] ="1.95";
   $invenArray[0][3] ="100";

   $invenArray[1][0] ="2222";
   $invenArray[1][1] ="Dandelion Tree";
   $invenArray[1][2] ="2.95";
   $invenArray[1][3] ="200";

   $invenArray[2][0] ="3333";
   $invenArray[2][1] ="Crabgrass Bush";
   $invenArray[2][2] ="3.95";
   $invenArray[2][3] ="300";

// echo("DEBUG: invenArray:");
// print_r($invenArray);
// echo("<br /><br />");
```

```
/* =========================================
          Functions are alphabetical
   ========================================= */
function addRecord( )
{
   echo("addRecord( ) has not been written yet.<br />");
} // end of deleteRecord( )


function deleteRecord( )
{
   echo("deleteRecord( ) has not been written yet.<br />");
} // end of deleteRecord( )


function displayInventory( )
{
   echo("displayInventory( ) has not been written yet.<br />");
} // end of displayInventory( )
?>
```

Display Contents of An Array

Work on the displayInventory( ) function first.

You already have the table hard-coded on the page. Now we just need to set up some loops that will do the same thing using the data from the array.

This code uses several individual lines for clarity. However, most PHP programmers will stuff several HTML elements into a single echo statement. However you write your code, always think about maintainability and readability. Your future self will be glad you did.

Replace the hard-coded table on the page with a single line of PHP code calling the function:
(You may want to comment out the hard-coded table so you have it as a reference.)

```
<p>
Here is our current inventory:<br />
<?php displayInventory( ); ?>
</p>
</body>
</html>
```

If you view the page now in the browser you will just see the message display: displayInventory( ) has not been written yet.

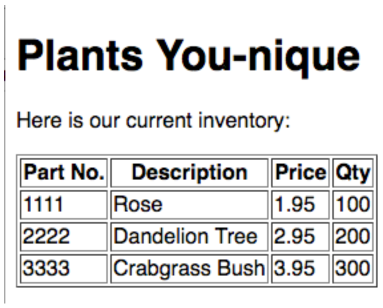
Inside the displayInventory( ) function add this code:

```
function displayInventory( )
{
   global $invenArray;
   echo("<table border='1'>");

   // display the header
   echo "<tr>";
   echo "<th>Part No.</th>";
   echo "<th>Description</th>";
   echo "<th>Price</th>";
   echo "<th>Qty</th>";
   echo "</tr>";
   // Walk through each record or row
   foreach($invenArray as $record)
   {
      echo "<tr>";
      // for each column in the row
      foreach($record as $value)
      {
         echo "<td>$value</td>";
      }
      echo "</tr>";
   }
   // stop the table
   echo "</table>";
} // end of displayInventory( )
```

Notes on the code:

- **global $invenArray** - Unlike other languages, in order to utilize global objects we need to tell PHP what we want to access. The keyword: global does this.
- **Use an echo statement to output the HTML code for the table.** Watch out for the quotes. The echo function requires all strings to be inside of quotes, so any inner quotes such as border='1' have to be in alternating quotes. If you use double quotes for the echo statement then you need to use single quotes inside that String.
- The <table> and <th> elements are simply echoed to the screen.
- **Displaying rows and columns** - Use a foreach loop inside a foreach loop to display the rows and columns. The outside foreach will display each row ($record). Then, inside each row, all the columns ($value) will be displayed wrapped inside a <td> element. After all the columns for a row are output then a </tr> is echoed. And finally, when all the rows have been displayed, a </table> is echoed.

Here is what the output should look like if you used the same sample data in the array.



Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Hard-code a drop-down list

We want to give the client the option to delete an item from the inventory. To do that we will create a form on the page with a drop-down list showing all the items in the array.

In the <body> section of the page, after the array is displayed, add a form with a drop-down listbox. Hard-code in some values to start with. After you get it working you will make this dynamic by using PHP to populate the list box items from the array itself. (This is very similar to what we did by hard-coding a table and then populating it using PHP.)

Type in the HTML code, shown in bold, right after displayInventory( ) is called.

```
<h1>Plants You-nique</h1>

<p>
Here is our current inventory:<br />
<?php displayInventory( ); ?>
</p>

<form action="inventory.php"
      method="POST"
      name="frmDelete">

   Select a product:
   <select name="lstItem" size="1">
      <option value="111" selected>Rose</option>
      <option value="222">Dandelion Tree</option>
      <option value="333">Crabgrass Bush</option>
   </select>
   <br /><br />
   <input name="btnSubmit" type="submit" value="Delete" />
</form>
</body>
</html>
```

Notes on the Code:

- **The <form> element has three important attributes.**
  action=" " has the name of the file that will be requested when the user clicks on the submit button. In this case, it is the same file being

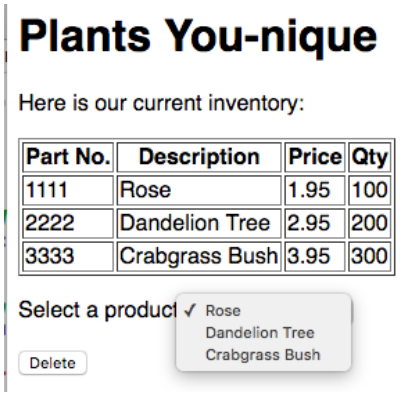
displayed.

method=" " describes how the data from the form will be sent to the server when the user clicks on the submit button. It can be either POST or GET. POST packages up the data and GET adds it to the end of the URL.

name=" " gives the form a specific name. We plan to have two forms on this page, each with its own submit button, so this is an important detail. Notice the prefix "frm". This tells us what type of object this is, a form.

- The <select> element should have a name. The size="1" attribute displays the options as a single line, a listbox. Notice the prefix "lst" telling us this is a list box.
- If you want to have an item pre-selected for the user, simply include the word "selected" inside that particular option. You can see that in action for the first item, "Rose".
- Each <option> in a list has two types of information, the actual value that the server will use and the display value that the user sees. For example, when the form information is sent to the server, if the user selected "Dandelion Tree", then the value "222" will be sent to the server. These two things are often the same, but, as this example shows, the value can also be different than what is displayed on the screen to the user.
- Following the naming conventions, the Submit button is named using the prefix "btn".

When you view the page using your localhost, here is what the screen should look like with the list being activated by the user.



Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Select a product

Make the drop-down list dynamic

We want the list to be dynamic, including only the items that are in the array. As the user adds/deletes items the list should automatically update. PHP allows us to do that.

You may want to comment out the HTML code for the <select> element so you have a reference.

Type in this PHP code shown in bold text, using a foreach loop and the data from the array to populate each <option>.

```
   Select an item to delete:
   <select name="lstItem" size="1">
      <?php
         // Populate the list box using data from the array
         foreach($invenArray as $index => $lstRecord)
         {
            // Make the value the index and the text displayed the description from 
the array
            // The index will be used by deleteRecord( )
            echo "<option value='" . $index . "'>" . $lstRecord[1] . "</option>\n";
         }
      ?>
   </select>
```

Notes on the Code:

- Notice how the <select> tags remain as HTML. The PHP code goes inside this element.
- Use a foreach loop with an index => value pair in order to get the index from the array as well as each row of data.
- Inside the loop insert the current $index value of the loop as the value for each <option>. For the information displayed on the screen show the second column ($lstRecord[1]) from the array. Look at the array. Column zero is the part number and column one is the description. $lstRecord[1] will display the description.
- Use the escape newline character "\n" to format the HTML code. This will make it easier to debug your HTML page when you "view source"

of the web page.

When you display the page, using localhost, the listbox should contain the data from the array.

Have a PHP page call itself

Each time the user clicks the submit button, a new request is sent to the server, using the filename from the action=" " attribute of the form.

```
<form action="inventory.php"
      method="POST"
      name="frmDelete">
```

PHP maintains several arrays, on the server, keeping track of many useful variables. One of these is the $_SERVER[ ] array which has over 35 items. (Do a web search for: PHP $_SERVER[ ] to see a list of these.) We will use $_SERVER['PHP_SELF'] to always have the page request itself.

At the top of the file, just inside the first <?PHP marker, add this line and comment:

```
// The filename of the currently executing script to be used
// as the action=" " attribute of the form element.
$self = $_SERVER['PHP_SELF'];
```

Then, replace the value of the action=" " attribute in the <form> with this bit of PHP code:

```
<form action="<?php $self ?>"
      method="POST"
      name="frmDelete">
```

This technique is especially useful if you happen to change the name of the file you are working on. For example, you may be doing a bit of version control by naming your files inventory1.php, inventory2.php, inventory3.php. Using $_SERVER['PHP_SELF'] will ensure that each time you run the file, it will call itself.

Use Hidden Form Fields

When the user requests a page the server asks, "What is your state?"

With the client/server model we have a regular conversation going on with the server.

1. The page is first requested via URL and displayed in the browser.
2. Viewing this page, the user clicks on the submit button.
3. A new request is made, along with the data from the form.
4. The server uses the form data, creates a new web page, sending it back to the user.

Back and forth, client request/server responding.

Each time the user clicks on the submit button, a new request is made to the server, processed, and a 'new' page is returned to the browser. To the user, it looks like the same page with just some minor changes.

There are several states the server needs to know about.

1) If this is a first-time visitor. Create some new, sample data.
or
2) Is this a returning visitor. (The user clicked the submit button.) If so, use the current data.
and later, when we have two or more forms
3) Is this a returning visitor and if so, which submit button was used? (Should the server add or delete a record?)

In order to track the state of the web page, we will include a hidden field. The user won't see it, but the server will!

Inside the <form>, right after the </select> tag, type in the following code and comment:

```
  </select>
```

```
   <!-- This field is used to determine if the page has been viewed already
        Code 99 = Delete
   -->
   <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='99' />
   <br /><br />
   <input name="btnSubmit" type="submit" value="Delete this selection" />
</form>
</body>
</html>
```

Notes on the Code:

- The name of this form object includes the prefix "hid" designating it as a hidden field.
- The value=" " attribute will be used by the server to deside what to do.

Checking the State

Here is the trick: Data from a form is only included with server requests if the user clicks a submit button. If the page is requested via URL (the original request by the user) then no form data will be included. PHP can use this to determine if a request is new or a returning user.

Type in the following code, at the top of the file, right after `$self = $_SERVER['PHP_SELF'];` moving the array code inside the else clause.

```
<?php
// The filename of the currently executing script to be used
// as the action=" " attribute of the form element.
$self = $_SERVER['PHP_SELF'];

// Check to see if this is the first time viewing the page
// The hidSubmitFlag will not exist if this is the first time
if(array_key_exists('hidSubmitFlag', $_POST))
{
   echo "<h2>Welcome back!</h2>";
   // Look at the hidden submitFlag variable to determine what to do
   $submitFlag = $_POST['hidSubmitFlag'];
   echo("DEBUG: hidSubmitFlag is: $submitFlag<br />");

   switch($submitFlag)
   {
      case "99": deleteRecord( );
      break;
      // more to come later
   }
}
else
{
   echo "<h2>Welcome to the Inventory Page</h2>";
   // First time visitor? If so, create the array
   $invenArray = array( );
   $invenArray[0][0] ="1111";
   $invenArray[0][1] ="Rose";
   $invenArray[0][2] ="1.95";
   $invenArray[0][3] ="100";

   $invenArray[1][0] ="2222";
   $invenArray[1][1] ="Dandelion Tree";
   $invenArray[1][2] ="2.95";
   $invenArray[1][3] ="200";

   $invenArray[2][0] ="3333";
   $invenArray[2][1] ="Crabgrass Bush";
   $invenArray[2][2] ="3.95";
   $invenArray[2][3] ="300";
}
```

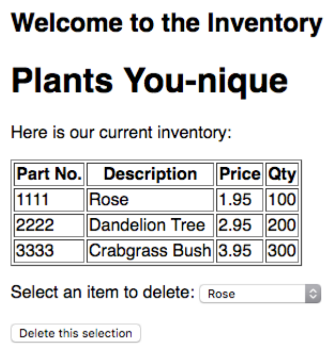
Notes on the Code:

- **Use array_key_exists( ) function to see if a specific value from the form exists.** If it does, this is a returning visitor. If not, then this is a brand new request (no form data was included with the request). PHP maintains an array, similar to $_SERVER[ ] that keeps track of all the POST variables included with the last request. array_key_exists( ) looks in this array, named $_POST[ ], to see if the form object named hidSubmitFlag is there. This is where the form object prefixes come in handy.
- **A returning user!** If there is a data in the $_POST[ ] for hidSubmitFlag then PHP displays an <h2> heading "Welcome back!"
- The value of hidSubmitFlag is saved in the variable $submitFlag.
- **Debugging with echo.** At this stage, we want to make sure we are getting the correct information from the client so display the value of $submitFlag. It should be 99. Use the word "DEBUG" in all caps so it is easy to notice in the browser.
- **Using a switch to determine which action to take.** There are going to several codes designating the different states (add, delete, and possible

incorrect codes) so use a switch to determine which one was sent with the request from the browser. A switch is like a series of if/else statements but much more elegant. It allows a variable to be checked for multiple options, each one triggering different code to run.

- **For New Requests.** If there is no trace of hidSubmitFlag, this is a new request with no form data included. The else section will run.
- A welcome message is displayed and the sample array data is created and sent back with the rest of the HTML code back to the browser.

Here is what the page will look like in both states:
A new visitor (no form data included with the request):



Welcome to the Inventory Page

Plants You-nique

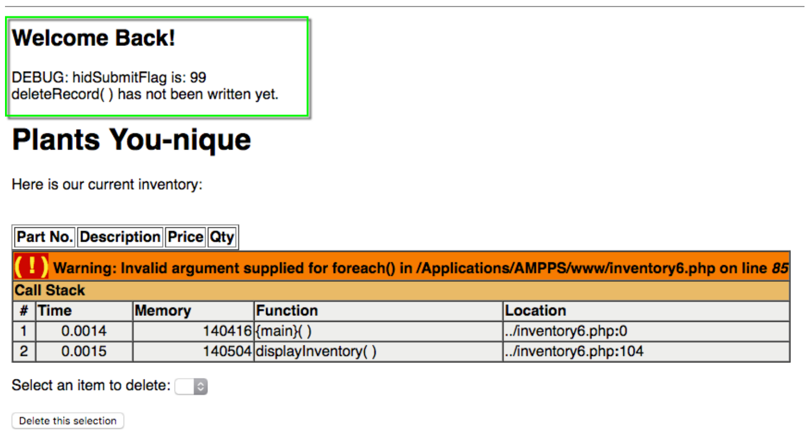
Here is our current inventory:

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Select an item to delete: Rose

Delete this selection

When the user clicks on the submit button the page will display like this:



Welcome Back!

DEBUG: hidSubmitFlag is: 99
deleteRecord( ) has not been written yet.

Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty

```
( ! ) Warning: Invalid argument supplied for foreach() in /Applications/AMPPS/www/inventory6.php on line 85
Call Stack
```

#	Time	Memory	Function	Location
1	0.0014	140416	{main}( )	../inventory6.php:0
2	0.0015	140504	displayInventory( )	../inventory6.php:104

Select an item to delete:

Delete this selection

Notice the different headings as well as the debug value displayed. You can also see that the deleteRecord( ) function hasn't been written yet, but we know the system is working because our stubbed code displayed.

What about the error?
When the first request is made, an array is built. Then, in the body of the page the function displayArray( ) is called.
However, when the submit button "Delete this selection" the server sees this a repeat user so it doesn't create a new array. If it did that, the only thing returning users would see would be the original sample data every time.

The Web is stateless

This "not remembering the array" is a good example of what it means when people say the Web is stateless. Servers are designed to take millions of requests and handle them. But, the server treats each request as unique and doesn't remember data from one request to the next.

We will solve this "problem" in the next section by introducing Session variables. These allow the server to remember data, such as our array, from request to request.

A Developer Trick with Localhost

When using Google Chrome and Firefox if you refresh the page (CTRL R) the localhost server will treat the page as a returning visitor.
But, if you click on the URL, selecting the web address at the top of the browser, and hit ENTER, a brand-new request is sent to the localhost server which is treated as a new visitor.

Using these two techniques you can quickly test for a new page or an updated page.

Using Sessions

Being stateless makes servers fast and efficient. They receive requests, process the page, and respond with a file back to the server. It takes very little overhead.

But, sometimes developers are willing to give up this efficiency in order to track on-going actions from the user. An online shopping cart is a good example. Each time the user selects an item for the shopping cart, we want the server to remember it along with all the other items they may be purchasing.

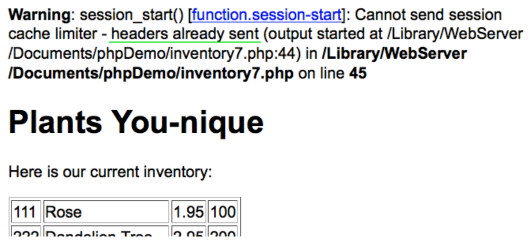
Tell the server to use sessions.

At the very top of the page, add the code shown in bold:

```
<?php
   // Tell server that you will be tracking session variables
   session_start( );
?>
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
```

This has to be at the very top of the file before anything else is seen by the browser. Type it in immediately before the `<!DOCTYPE html>` statement.

If you add this line after the browser starts building the page in memory you will see an error similar to this. Notice the phrase underlined in green, "headers already sent".



Warning: session_start() [function.session-start]: Cannot send session cache limiter - headers already sent (output started at /Library/WebServer /Documents/phpDemo/inventory7.php:44) in **/Library/WebServer /Documents/phpDemo/inventory7.php** on line **45**

Plants You-nique

Here is our current inventory:

111	Rose	1.95	100

Telling the server what to remember

You have already used the server arrays $_SERVER[ ] and $_POST[ ]. Now we are going to use the $_SESSION[ ] server array.
Type in the following line of code and comment, right after the array is created:

```
else
{
   echo("DEBUG: Welcome to the Inventory Page<br />");
   // First time visitor? If so, create the array
   $invenArray = array( );
   $invenArray[0][0] ="1111";
   $invenArray[0][1] ="Rose";
   $invenArray[0][2] ="1.95";
   $invenArray[0][3] ="100";

   $invenArray[1][0] ="2222";
   $invenArray[1][1] ="Dandelion Tree";
   $invenArray[1][2] ="2.95";
   $invenArray[1][3] ="200";

   $invenArray[2][0] ="3333";
   $invenArray[2][1] ="Crabgrass Bush";
   $invenArray[2][2] ="3.95";
   $invenArray[2][3] ="300";

   // Save this array as a serialized session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
}
```

Notes on the Code:

- **serialize( )** - Only String data can be stored in a session variable, not Arrays. So we need to serialize( ) the array, making it into one long String.
- urlencode( ) - We also want to preserve all the non-alphanumeric characters (<, >, quotes, commas, percent signs, etc.) as hexadecimal numbers and convert all the space characters into +. This is especially important if the information includes a URL. Later, urldecode( ) will be used to restore these non-alphanumeric characters.
- **$_SESSION['serializedArray']** - Stores the serialized data in the $_SESSION[ ] array out on the server.

Using the Server's Memory

When the user returns to the server, requesting to view the page again, we can easily restore this session variable.

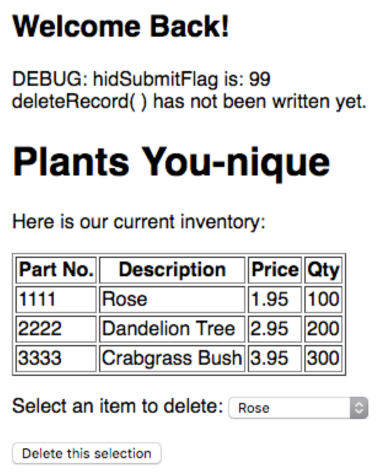
Right before the switch statement in the if clause, add the following code and comment, shown in bold text:

```
// Get the array that was stored as a session variable
$invenArray = unserialize(urldecode($_SESSION['serializedArray']));
switch($submitFlag)
{
   case "99": deleteRecord( );
```

So, when the server sees that this is a return visitor, the serializedArray is extracted from the server's $_SESSION[ ] array, unencoded, and unserialized (converted back into an array) and assigned to the $invenArray variable.

Now, the array is available for the displayInventory( ) function.

The page, after clicking on the submit button should look something similar to this:



Welcome Back!

DEBUG: hidSubmitFlag is: 99
deleteRecord( ) has not been written yet.

Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Select an item to delete: Rose

In the next section, we will write the delete function.

Delete records

Using unset( ) to delete a record from an array.

Type in the following code inside the deleteRecord( ) function, replacing the stubbed in code you originally used:

```
function deleteRecord( )
{
   global $invenArray;
   global $deleteMe;

   // Get the selected index from the lstItem
   $deleteMe = $_POST['lstItem'];
   // Remove the selected index from the array
   unset($invenArray[$deleteMe]);
   // Save the updated array in its session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
   echo "<h2>Record deleted</h2>";
} // end of deleteRecord( )
```

Notes on the Code:

- **global** - We need to tell PHP which global variables this function will use.
- **$_POST['lstItem']** - When the user clicks the submit button to delete, the request is sent to the server along with all the data from the form. Here is the form element we are using:

```
<form action="<?php $self ?>"
      method="POST"
      name="frmDelete">
```

Because the method="POST", all the information from the form is sent in a package along with the request. The server opens up this package and saves all the values from the form objects in its $_POST[ ] array. (If we had used method="GET" then the server would have stored the form values in the $_GET[ ] array.)

Each value from the form is identified by its name=" " attribute. Here is the HTML code from the form:

```
   <select name="lstItem" size="1">
      <?php
```

```
         // Populate the list box using data from the array
         foreach($invenArray as $index => $lstRecord)
         {
            // Make the value the index and the text displayed the description from
the array
            // The index will be used by deleteRecord( )
            echo "<option value='" . $index . "'>" . $lstRecord[1] . "</option>\n";
         }
      ?>
   </select>
```

Each option has a value of the index from the array. This is the value that is sent to the server and stored in $_POST[ ] along with the name of the form object, lstItem. This value, or index of the array, is stored in the variable $deleteMe.

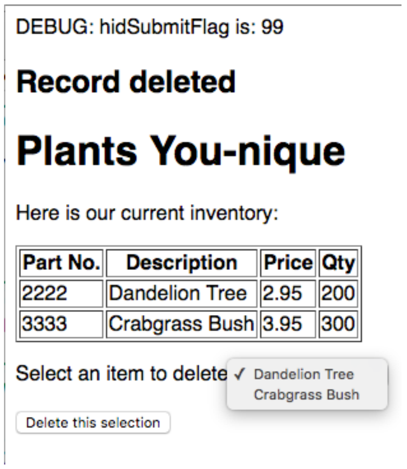
- **unset($invenArray[$deleteMe])** - deletes the index from the array $invenArray[$deleteMe]
- **$_SESSION['serializedArray']** - Each time the array is changed it has to be saved as a session variable so the most recent version is always available on the server. Just like before it is serialized, urlencoded( ) and then assigned to the $_SESSION[ ] array on the server.

Why not always use session variables?

Maintaining session variables is resource intensive. Imagine 500,000 people requesting pages from a server. If session variables are used, each one will require extra memory to store the session variables. Saving and retrieving session variables also takes extra cycles on the server, slowing down performance.

To keep overhead to a minimum, most sessions are only kept active for 20 minutes. If a user hasn't sent a request in 20 minutes, the session is abandoned, leaving those resources for another user. In some systems though, such as Blackboard, BrightSpace, Moodle and other LMS, people expect to be on the server much longer and resources have to be set aside to accommodate the user needs.

With the deleteRecord( ) function in place you should now be able to delete items from the list. Notice that as soon as you click the delete button, the table display changes as well as the listbox. Here is the page after "Rose" was selected from the listbox and deleted:



DEBUG: hidSubmitFlag is: 99

Record deleted

Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Select an item to delete

Add records

If records are deleted, the user will also want to add new records.

Create an "Add Item Form

Type in the following code to add in a form to collect new data to be added. You can type this in right after the inventory is displayed, and before the frmDelete form.

```
<p>
Here is our current inventory:<br />
<?php displayInventory( ); ?>
</p>

<form action="<?php $self ?>"
      method="POST"
      name="frmAdd">

      <input type="text" name="txtPartNo" id="txtPartNo" value="999" size="5" />
      <br /><br />

      <input type="text" name="txtDescr" id="txtDescr" value="Test Descr" />
      <br /><br />

      <input type="text" name="txtPrice" id="txtPrice" value="123.45" />
      <br /><br />

      <input type="text" name="txtQty" id="txtQty" value="123" size="5" />
      <br /><br />
      <!-- This field is used to determine if the page has been viewed already
           Code 01 = Add
      -->
      <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='01' />
      <input name="btnSubmit" type="submit" value="Add this information" />
   </fieldset>
</form>
```

Notes on the Code:

- **<form>** - The only changed made to the <form> element is the name of the form
- **name=" " and id=" "** - Each input box is given a name=" " (for the server) and an id=" " for CSS and the browser. Notice that these can be the same. Notice the prefix "txt" is used to designate that these are textboxes.
- **<br /><br />** - Each field is separated using line breaks so the form will

stack, especially useful on mobile devices. But, later, you can add CSS code to change the appearance of the form.

- **<input type='hidden'** uses the same name and id as the hidden field in the frmDelete but a new value='01' is used. This value will be used to determine if the user clicked the submit button in the frmDelete or the frmAdd forms.
- **Sample data saves time** - Each text field has sample data for the value=" " attributes. This will save you LOTS of time. As you test the page all you have to do is click the "Add this information" button, and the sample data will be submitted to the server. (Compare that to the time it would take you to key in all those text fields every time you run the page...) Later, when the page goes live, these values can be quickly removed.

Handle the data from frmAdd on the server

Type in this code inside the addRecord( ) function, replacing the stubbed in code you already have there.

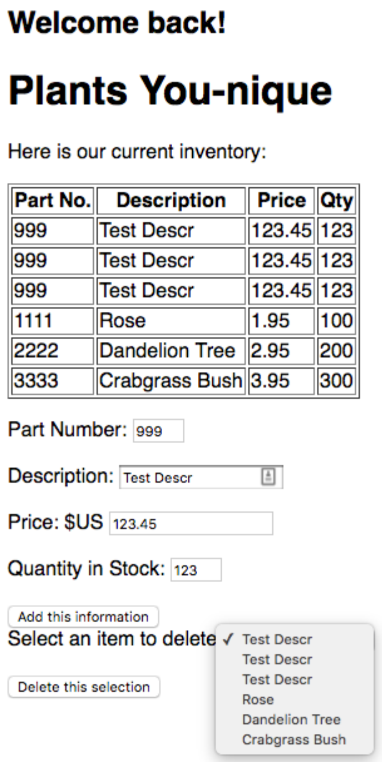
```
function addRecord( )
{
   global $invenArray;
   // Add the new information into the array
   // items stacked for readability
   $invenArray[ ] = array($_POST['txtPartNo'],
                          $_POST['txtDescr'],
                          $_POST['txtPrice'],
                          $_POST['txtQty']);
   // The sort will be on the first column (part number) so use this to re-order the
displays
   sort($invenArray);
   // Save the updated array in its session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
} // end of addRecord( )
```

Notes on the Code:

- **global** is used to access the global array.
- **Values stored in the $_POST[ ] array.** A new record is then added to the array using the data from the $_POST[ ] array. This is where the form prefixes are useful. You can quickly see in your code that we are pulling in

values from txt fields, text boxes.
An array of columns inside an array of rows - Remember that the 2D array is really an array of rows, each containing a mini-array of columns. So the array of rows $invenArray[ ] adds a new array of columns using = `array($POST[ 'txtPartNo'], $_POST['txtDescr'], $_POST['txtPrice']` etc. The code is stacked for readability but to PHP it is all a single line.
sort($invenArray) - automatically sorts the array on the first field which is the part number. Note, if you wanted to sort by another field you could use the usort( ) function.
$_SESSION['serializedArray'] - Finally, save the updated array as the session variable named serializedArray.

When you view the page on your localhost and click the "Add this information" button a few times, the results should look like this:



Welcome back!

Plants You-nique

Here is our current inventory:

Part No.	Description	Price	Qty
999	Test Descr	123.45	123
999	Test Descr	123.45	123
999	Test Descr	123.45	123
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Part Number: 999

Description: Test Descr

Price: $US 123.45

Quantity in Stock: 123

Notice that there are several new records automatically displaying in the array as well as showing up on the listbox.

Now the user can add new items as well as delete any item.

Decision Making with the Switch

You now have two forms, one for adding information and one to delete records.
How will the program know which one the user is using?

This is where the hidden fields come in. Each form has its own hidden field named hidSubmitFlag. The hidSubmit field for frmAdd has the value of "01" and the hidSubmit field for frmDelete has the value of "99". (These are arbitrary numbers the developer came up with. Any value could be used, even the words "addMe" and "deleteMe".)

When the user clicks the submit button, all the form information along with the hidden field values are sent to the server as part of the page request.

When the server sees that this is a return request, because there is data in the $_POST[ ] array, then it can use a switch to determine which process to run.

Add this code, highlighted with bold text, to the existing switch statement. Note the second DEBUG echo statement that used gettype( ) to determine the data type from the form. (Forms always return String values, even if they are numbers.)

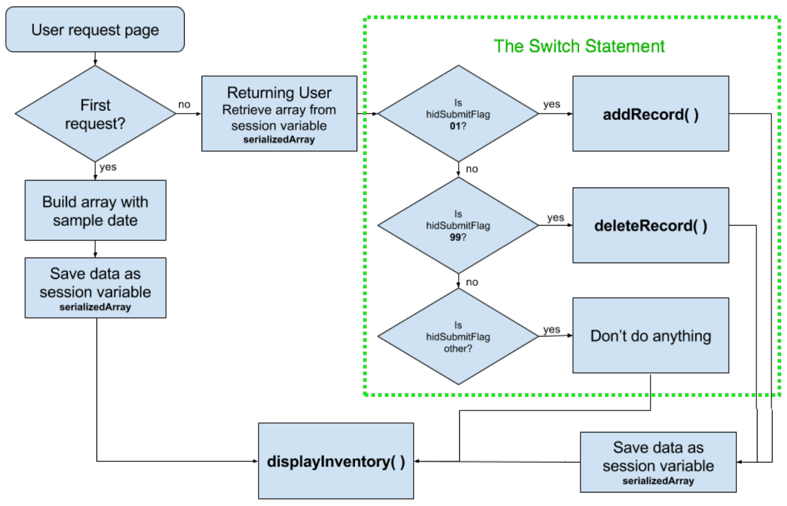
```
if(array_key_exists('hidSubmitFlag', $_POST))
{
   echo "<h2>Welcome back!</h2>";
   // Look at the hidden submitFlag variable to determine what to do
   $submitFlag = $_POST['hidSubmitFlag'];
   echo "DEBUG: hidSubmitFlag is: $submitFlag<br />";
   echo "DEBUG: hidSubmitFlag is type of: " . gettype($submitFlag) . "<br />";

   // Get the array that was stored as a session variable
   $invenArray = unserialize(urldecode($_SESSION['serializedArray']));
   switch($submitFlag)
   {
      case "01": addRecord( );
      break;

      case "99": deleteRecord( );
```

```
        break;

        default: displayInventory($invenArray);
    }
}
else
```

Notes on the Code:

- A switch allows the program to follow multiple paths based on a single input. Each "path" is designated by the keyword: case and a value. To end a "path" the keyword: break is used. This tells the program to break from the switch.
- The values from a form is always type String, so you have to put quotes around each case.
- Each case statement ends with a colon ":"
- The keyword: default should always be included and used to handle any data that isn't handled by the case statements.
- The switch statement is shown here in the flow chart, highlighted inside the green box.

Program flow for inventory.php



Format Forms for Accessibility

It is very important that your apps be accessible to everyone, including people who cannot see the screen. You can do this by including accessible features used by JAWS and other screen reading software.

For each form object, you need to associate the text labels with the actual input field using the <label> element.
In addition you can group items on the page using the <fieldset> and <legend> elements. <fieldset> will put a nice frame around a group of form objects. <legend> is used to label this frame.

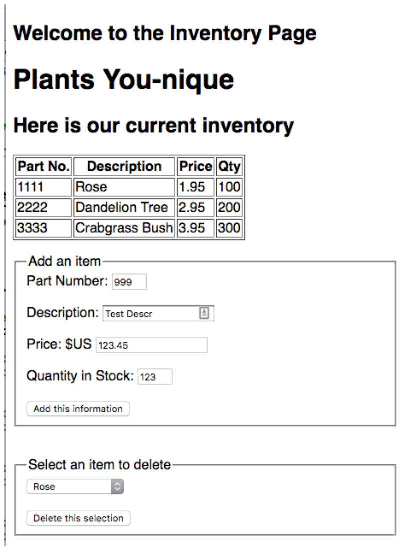
For both the frmAdd and frmDelete, add in the code shown in bold text:

```
<form action="<?php $self ?>"
      method="POST"
      name="frmAdd">

   <fieldset id="fieldsetAdd">
   <legend>Add an item</legend>

      <label for="txtPartNo">Part Number:</label>
      <input type="text" name="txtPartNo" id="txtPartNo" value="999" size="5" />
      <br /><br />

      <label for="txtDescr">Description:</label>
      <input type="text" name="txtDescr" id="txtDescr" value="Test Descr" />
      <br /><br />

      <label for="txtPrice">Price: $US</label>
      <input type="text" name="txtPrice" id="txtPrice" value="123.45" />
      <br /><br />

      <label for="txtQty">Quantity in Stock:</label>
      <input type="text" name="txtQty" id="txtQty" value="123" size="5" />
      <br /><br />
      <!-- This field is used to determine if the page has been viewed already
           Code 01 = Add
      -->
      <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='01' />
      <input name="btnSubmit" type="submit" value="Add this information" />
   </fieldset>
</form>

<form action="<?php $self ?>"
      method="POST"
      name="frmDelete">

   <fieldset id="fieldsetDelete">
   <legend>Select an item to delete:</legend>
   <select name="lstItem" size="1">
      <?php
         // Populate the list box using data from the array
         foreach($invenArray as $index => $lstRecord)
         {
            // Make the value the index and the text displayed the description from
the array
            // The index will be used by deleteRecord( )
            echo "<option value='" . $index . "'>" . $lstRecord[1] . "</option>\n";
         }
      ?>
   </select>
   <!-- This field is used to determine if the page has been viewed already
        Code 99 = Delete
   -->
   <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='99' />
   <br /><br />
   <input name="btnSubmit" type="submit" value="Delete this selection" />
   </fieldset>
</form>
```

Notes on the Code:

- The listbox does not need <label> because each item has its own built in label using <option>.
- The for=" " attribute uses the id=" " of the form object. For example <label for="txtPartNo"> refers to <input type="text" id="txtPartNo" /> Remember that name=" " is used by the server and id=" " is used by the browser.

Here is what the page should look like with these additions:



Welcome to the Inventory Page

Plants You-nique

Here is our current inventory

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Add an item

Part Number: 999

Description: Test Descr

Price: $US 123.45

Quantity in Stock: 123

Add this information

Select an item to delete

Rose

Delete this selection

Use CSS to Improve the UX

The User eXperience (UX) for any app is very critical in today's world. By adding a graphic image and a little bit of CSS we can make the page much friendly and easier to use.

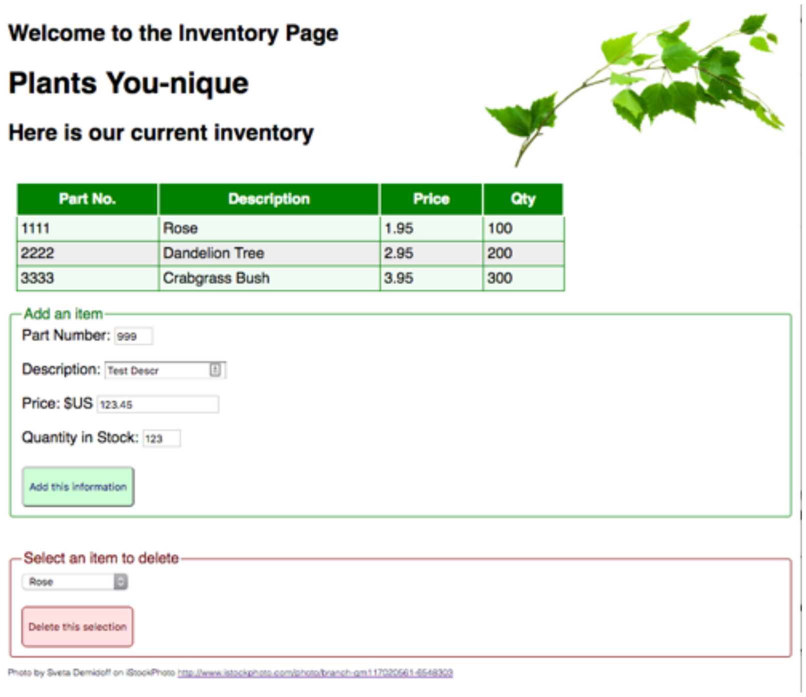
Here is the CSS code you can put in an external CSS file, or in the <style> element inside the <head> section of the page. (If you type this into an external .css file, don't include the <title> and <style> HTML elements!)

```
<title>My Inventory</title>
<style>
fieldset#fieldsetAdd {
border:                  #008000 solid 1px;  /* forest green */
border-radius:           3px;
}

fieldset#fieldsetAdd legend {
color:                   #008000;
}

fieldset#fieldsetAdd input[type="submit"] {
color:                  #000080;             /* dark blue */
background:             #cdffd8;             /* light green */
margin:                 .25em 0 0 0;
height:                 4em;
display:                block;
border-radius:          5px;
}

fieldset#fieldsetDelete {
border:                  #800000 solid 1px;  /* dark red */
border-radius:           3px;
}

fieldset#fieldsetDelete legend {
color:                   #800000;
}

fieldset#fieldsetDelete input[type="submit"] {
color:                  #800000;             /* dark red */
background:             #ffe0e3;             /* light red */
border:                 #800000 solid 1px;
height:                 4em;
display:                block;
border-radius:          5px;
float:                  left;
}
```

```
table {
width:                  70%;
margin:                 .5em;
border-collapse:        collapse;
}

table,
td {
border:                 1px solid #008000;
padding:                .25em;
}

th {
background:             #008000;
color:                  white;
border:                 2px white solid;
padding:                .5em;
}

tr:nth-child(even) {
background:             #f0fcf3;        /* very light green */
}

tr:nth-child(odd) {
background:             #eee;           /* very light gray */
}

img {
float:                  right;
width:                  40%;
margin:                 -4em 0 1em 0;
}

</style>
</head>
```

For the graphic, just add an <img /> element right above the <h1>. You will have to find your own image. The one shown in the examples was purchased from iStockPhoto and their licensing doesn't allow it to be shared. Note that there is only one <img> on the page so it was styled in the <style> section.

Here is what the finished page could look like: Your final product will vary according to how you code the CSS.



Welcome to the Inventory Page

Plants You-nique

Here is our current inventory

Part No.	Description	Price	Qty
1111	Rose	1.95	100
2222	Dandelion Tree	2.95	200
3333	Crabgrass Bush	3.95	300

Add an item

Part Number: 999

Description: Test Descr

Price: $US 123.45

Quantity in Stock: 123

Select an item to delete

Rose

Photo by Sveta Demidoff on iStockPhoto http://www.istockphoto.com/photo/branch-gm117020561-6548303

What did you learn?

This lab showed you how to:

- Set up a working PHP page and build on it, one step at a time.
- Set up and use a two-dimensional array.
- Use function stubs for faster development.
- Display a table of data using PHP and foreach loops.
- Create a dynamic drop-down list from an array.
- Have a PHP page call itself - First time user or returning user?
- Use Sessions to "remember" user information.
- Use multiple forms on one page.
- Delete records from an array using unset( )
- Adding records to an array.
- Formatting forms for greater accessibility
- Using CSS to improve the UX - User eXperience

Complete Source Code

You can download the complete source code here: http://WebExplorations.com/book/php/arraySessionFormCode.zip

Note: The branch image could not be shared due to iStockPhoto's license agreement. You'll have to find your own.

Source code for inventory.php

```
<?php
   // Tell server that you will be tracking session variables
   session_start( );
?>
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
<!-- inventory.php -  Keep track of inventory
  Peter Johnson - peterk@webExplorations.com
  Written:     08-08-11
  Revised: PKJ 04-29-16 updated course presentation
  -->
 <title>My Inventory</title>

<style>
fieldset#fieldsetAdd {
border:                  #008000 solid 1px;  /* forest green */
border-radius:           3px;
}

fieldset#fieldsetAdd legend {
color:                   #008000;
}

fieldset#fieldsetAdd input[type="submit"] {
color:                  #000080;             /* dark blue */
background:             #cdffd8;             /* light green */
margin:                 .25em 0 0 0;
height:                 4em;
display:                block;
border-radius:          5px;
}

fieldset#fieldsetDelete {
border:                  #800000 solid 1px;  /* dark red */
border-radius:           3px;
}
```

```
fieldset#fieldsetDelete legend {
color:                   #800000;
}

fieldset#fieldsetDelete input[type="submit"] {
color:                  #800000;             /* dark red */
background:             #ffe0e3;             /* light red */
border:                 #800000 solid 1px;
height:                 4em;
display:                block;
border-radius:          5px;
float:                  left;
}

table {
width:                  70%;
margin:                 .5em;
border-collapse:        collapse;
}

table,
td {
border:                 1px solid #008000;
padding:                .25em;
}

th {
background:             #008000;
color:                  white;
border:                 2px white solid;
padding:                .5em;
}

tr:nth-child(even) {
background:             #f0fcf3;       /* very light green */
}

tr:nth-child(odd) {
background:             #eee;          /* very light gray */
}

img {
float:                  right;
width:                  40%;
margin:                 -4em 0 1em 0;
}

</style>
</head>
<body>

<?php
// The filename of the currently executing script to be used
// as the action=" " attribute of the form element.
```

```
$self = $_SERVER['PHP_SELF'];

// Check to see if this is the first time viewing the page
// The hidSubmitFlag will not exist if this is the first time
if(array_key_exists('hidSubmitFlag', $_POST))
{
   echo "<h2>Welcome back!</h2>";
   // Look at the hidden submitFlag variable to determine what to do
   $submitFlag = $_POST['hidSubmitFlag'];
   // echo "DEBUG: hidSubmitFlag is: $submitFlag<br />";
   //echo "DEBUG: hidSubmitFlag is type of: " . gettype($submitFlag) . "<br />";

   // Get the array that was stored as a session variable
   $invenArray = unserialize(urldecode($_SESSION['serializedArray']));
   switch($submitFlag)
   {
      case "01": addRecord( );
      break;

      case "99": deleteRecord( );
      break;

      default: displayInventory($invenArray);
   }
}
else
{
   echo "<h2>Welcome to the Inventory Page</h2>";
   // First time visitor? If so, create the array
   $invenArray = array( );
   $invenArray[0][0] ="1111";
   $invenArray[0][1] ="Rose";
   $invenArray[0][2] ="1.95";
   $invenArray[0][3] ="100";

   $invenArray[1][0] ="2222";
   $invenArray[1][1] ="Dandelion Tree";
   $invenArray[1][2] ="2.95";
   $invenArray[1][3] ="200";

   $invenArray[2][0] ="3333";
   $invenArray[2][1] ="Crabgrass Bush";
   $invenArray[2][2] ="3.95";
   $invenArray[2][3] ="300";

   // Save this array as a serialized session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
}

// echo "DEBUG: invenArray:";
// print_r($invenArray);
// echo "<br /><br />";

/* =========================================
          Functions are alphabetical
   ========================================= */
function addRecord( )
{
   global $invenArray;
   // Add the new information into the array
   // items stacked for readability
   $invenArray[ ] = array($_POST['txtPartNo'],
                          $_POST['txtDescr'],
                          $_POST['txtPrice'],
                          $_POST['txtQty']);
   // The sort will be on the first column (part number) so use this to re-order the
displays
   sort($invenArray);
   // Save the updated array in its session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
} // end of addRecord( )


function deleteRecord( )
{
   global $invenArray;
   global $deleteMe;

   // Get the selected index from the lstItem
   $deleteMe = $_POST['lstItem'];
   // echo "DEBUG: \$deleteMe is: " . $_POST['lstItem'];

   // Remove the selected index from the array
   unset($invenArray[$deleteMe]);
   // echo "<h2>Record deleted</h2>";
   // Save the updated array in its session variable
   $_SESSION['serializedArray'] = urlencode(serialize($invenArray));
} // end of deleteRecord( )


function displayInventory( )
{
   global $invenArray;
   echo "<table border='1'>";

   // display the header
   echo "<tr>";
   echo "<th>Part No.</th>";
   echo "<th>Description</th>";
   echo "<th>Price</th>";
   echo "<th>Qty</th>";
   echo "</tr>";

   // Walk through each record or row
   foreach($invenArray as $record)
   {
      echo "<tr>";
      // for each column in the row
      foreach($record as $value)
      {
         echo "<td>$value</td>";
      }
      echo "</tr>";
   }
   // stop the table
   echo "</table>";
} // end of displayInventory( )
?>


<img src="graphic/branch.jpg" alt="photo of tree branch with leaves" />

<h1>Plants You-nique</h1>
<h2>Here is our current inventory</h2>
<?php displayInventory( ); ?>
</p>

<form action="<?php $self ?>"
      method="POST"
      name="frmAdd">

   <fieldset id="fieldsetAdd">
   <legend>Add an item</legend>

      <label for="txtPartNo">Part Number:</label>
      <input type="text" name="txtPartNo" id="txtPartNo" value="999" size="5" />
      <br /><br />

      <label for="txtDescr">Description:</label>
      <input type="text" name="txtDescr" id="txtDescr" value="Test Descr" />
      <br /><br />

      <label for="txtPrice">Price: $US</label>
      <input type="text" name="txtPrice" id="txtPrice" value="123.45" />
      <br /><br />

      <label for="txtQty">Quantity in Stock:</label>
      <input type="text" name="txtQty" id="txtQty" value="123" size="5" />
      <br /><br />
      <!-- This field is used to determine if the page has been viewed already
           Code 01 = Add
      -->
      <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='01' />
      <input name="btnSubmit" type="submit" value="Add this information" />
   </fieldset>
</form>
<br /><br />
<form action="<?php $self ?>"
      method="POST"
      name="frmDelete">

   <fieldset id="fieldsetDelete">
   <legend>Select an item to delete</legend>
      <select name="lstItem" size="1">
         <?php
         // Populate the list box using data from the array
         foreach($invenArray as $index => $lstRecord)
         {
            // Make the value the index and the text displayed the description from 
the array
            // The index will be used by deleteRecord( )
            echo "<option value='" . $index . "'>" . $lstRecord[1] . "</option>\n";
         }
         ?>
      </select>
      <!-- This field is used to determine if the page has been viewed already
           Code 99 = Delete
      -->
      <input type='hidden' name='hidSubmitFlag' id='hidSubmitFlag' value='99' />
      <br /><br />
      <input name="btnSubmit" type="submit" value="Delete this selection" />
   </fieldset>
</form>
<p style="font-size:10px;font-weight:200;">
   Photo by Sveta Demidoff on iStockPhoto <a href="http://www.istockphoto.com/photo/
branch-gm117020561-6548303" target="_blank">http://www.istockphoto.com/photo/branch-
gm117020561-6548303</a>
</p>
</body>
</html>
```

Credits

Arrays, Sessions, and Forms - Written by Peter K. Johnson, Web Explorations, LLC

web explorations

Last update: 07/05/17