

Lab 2. PHP Variables & HTML Input Form

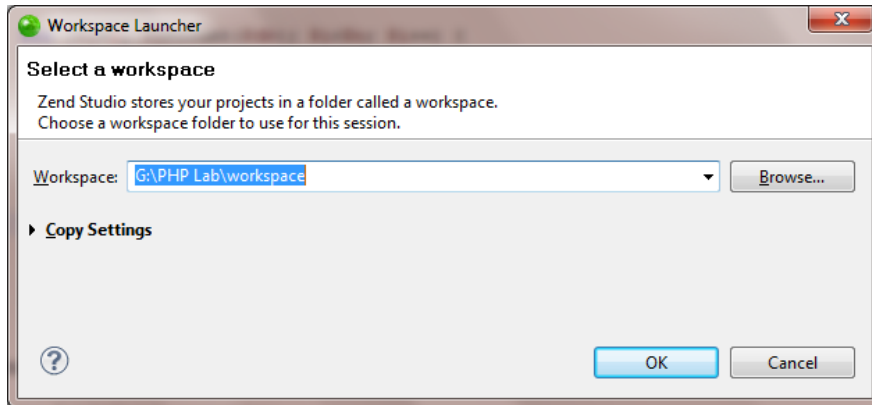
Prepared: TrangNTT

| | | |
|--------|--------------------------------------|----|
| Lab 2. | PHP Variables & HTML Input Form..... | 1 |
| 2.1. | Create a PHP project..... | 2 |
| 2.2. | Expression Example | 3 |
| 2.3. | Create a String test | 8 |
| 2.4. | Create a Basic form..... | 8 |
| 2.5. | Create a registration form | 9 |
| 2.6. | Create a confirm form | 9 |
| 2.7. | Change to GET method..... | 10 |
| 2.8. | Firefox and Add-ons..... | 10 |
| 2.9. | Exercise | 13 |

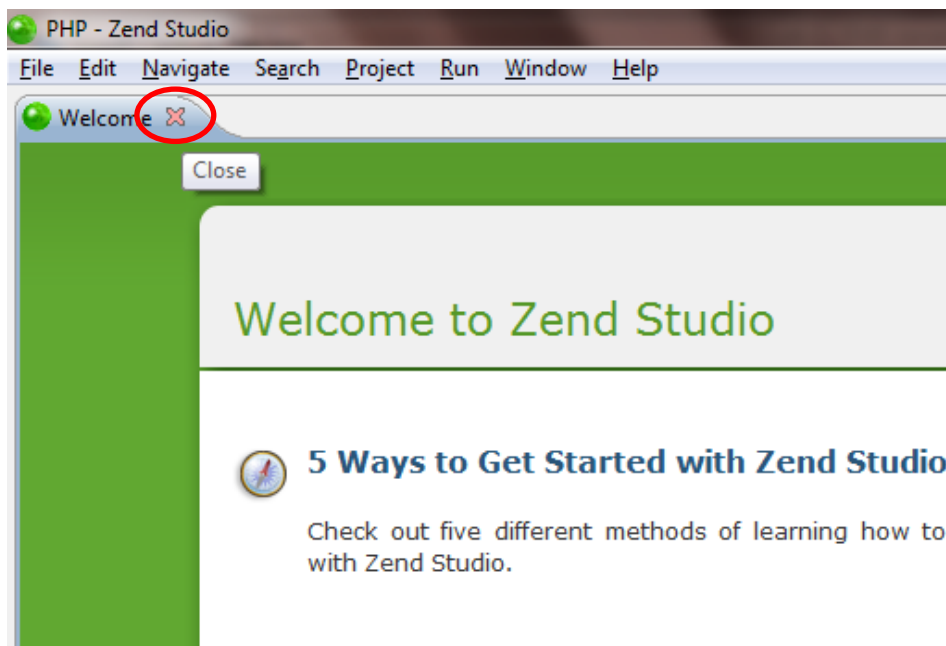
2.1. Create a PHP project

Step 1. Open Zend Studio 7.0

- Select Start → All Programs → Zend Studio – 7.0.0 → Zend Studio – 7.0.0
- Choose a workspace for the workspace directory. This should be a new workspace. The tool creates the folder and workspace for you. If there is not a **Workspace Launcher** dialog, select **File** → **Switch Workspace** → **Other**.




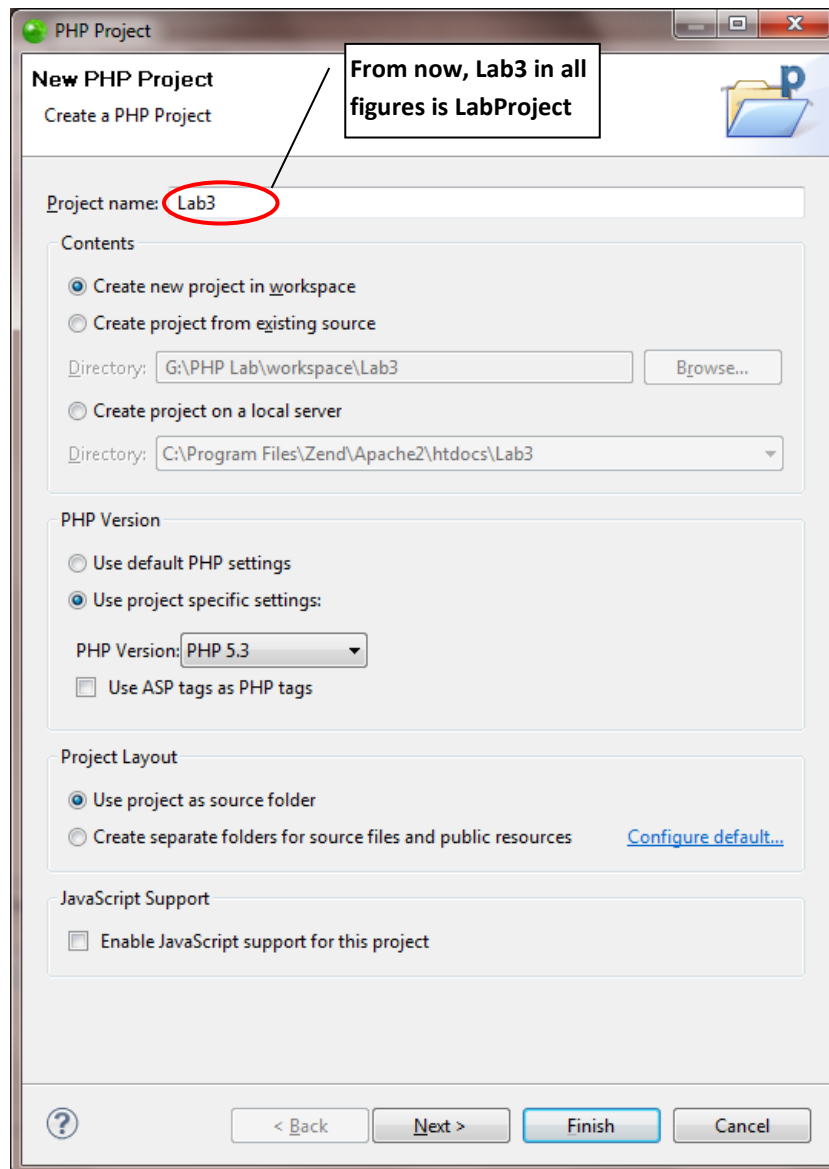
- When **Zend Studio** opens, sketch the **Welcome** page to get some topics there. Spend time at home to read these topics. Close the **Welcome** page.



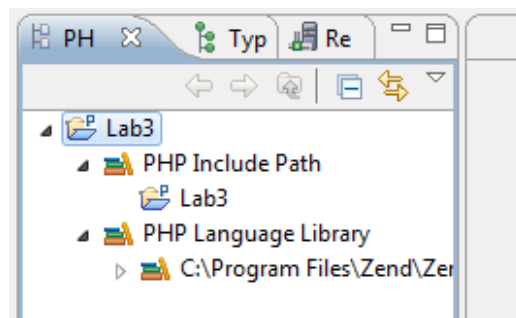
- By default, the PHP Perspective is opened when a new workspace is created in Zend Studio. The title bar indicates the currently active perspective.

Step 2. Create a PHP project named “LabProject”

- Click Create a PHP project by clicking the arrow beside the New button  from the main toolbar, or selecting **File** → **New** and choose **PHP Project**. The **PHP Project** dialog appears.
- In the **Project name** field type **LabProject** (*All the below figures are for the project name “Lab3” – don’t care about it. Consider all “Lab3” in all below figures are “LabProject”*).
- Choose **Create new project in workspace** to ensure that the project you will create is located in the your selected workspace.
- Change the version of PHP to **Use project specific settings**. Choose **PHP 5.3** in **PHP Version** dropdown list. Click **Finish**.



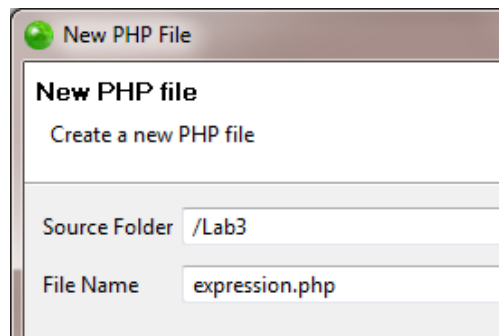
- In the PHP Explorer view, locate the **LabProject** project. Expand the project to display the **PHP Include Path** and **PHP Language Library**. The **PHP Include Path** is a set of locations that is used for finding resources referenced by include/require statements. The library that is responsible for built-in PHP functions is called **PHP Language Library**.



2.2. Expression Example

Step 1. Create a php file named “Expression Example”

- From the main menu, click **File** → **New** → **PHP File**. The **New PHP File** dialog appears.
- Enter the File Name is “**Expression.php**”. Click **Finish**.



- Enter the following source code in the editor of expression.php

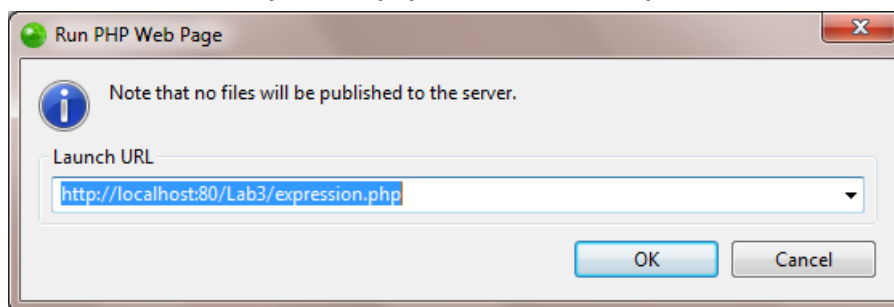
```

1 <HTML>
2   <HEAD> <TITLE>Expression Example </TITLE> </HEAD>
3   <BODY>
4       <?PHP
5           $gr1 = 50;
6           $gr2 = 100;
7           $gr3 = 75;
8           $aver = ($gr1 + $gr2 + $gr3) / 3;
9           print "The average is $aver";
10          ?>
11   </BODY>
12 </HTML>
13

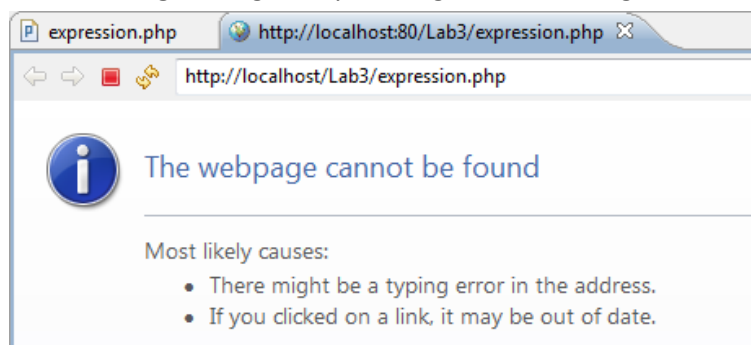
```

Step 2. Run the Expression Example

- Right click on the editor or the **expression.php** file on the **PHP Explorer**, choose **Run As** → **PHP Web page**.



- Confirm the **Run PHP Web Page** dialog then you will get the following result:




- To run this project, you need to include the ExampleProject directory to the path of web server.
- Copy these lines in the end of the httpd.conf in "C:\Program Files\Zend\Apache2\conf"

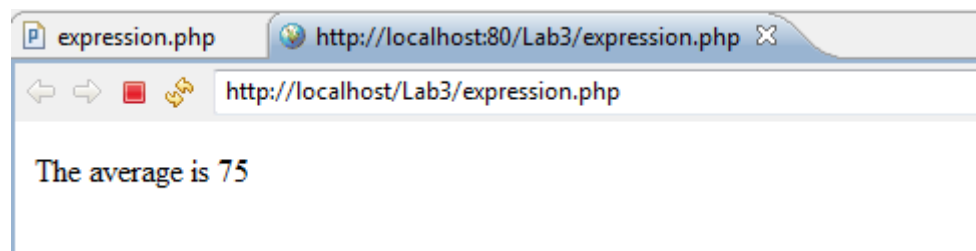
```

<Location /LabProject>
    Order deny,allow
    Allow from all
</Location>

```

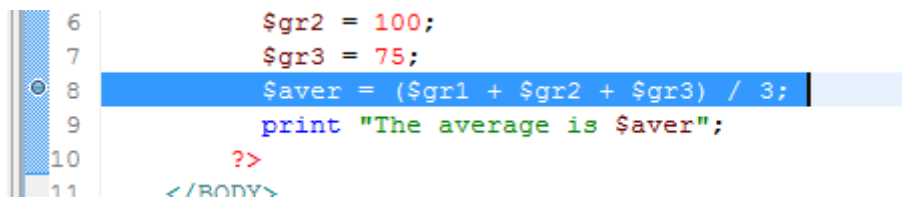
```
Alias /LabProject "G:\PHP Lab\workspace\LabProject"
```


- Stop and Start Server using Apache Server Monitor.
- Run the expression.php again by clicking the button  in the main toolbar or press Ctrl + F11 and see the following result

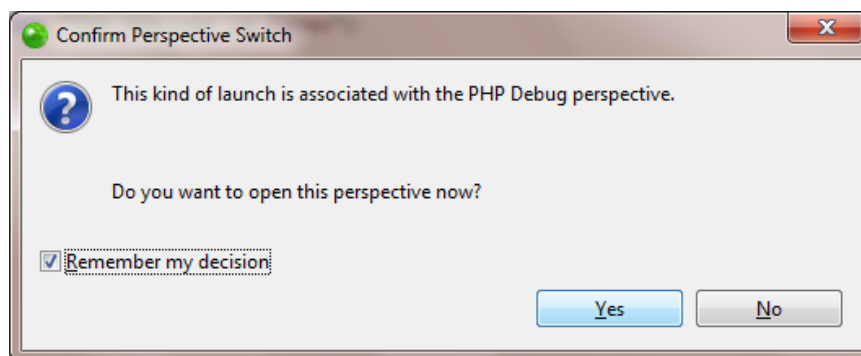


Step 3. Debug the Expression Example

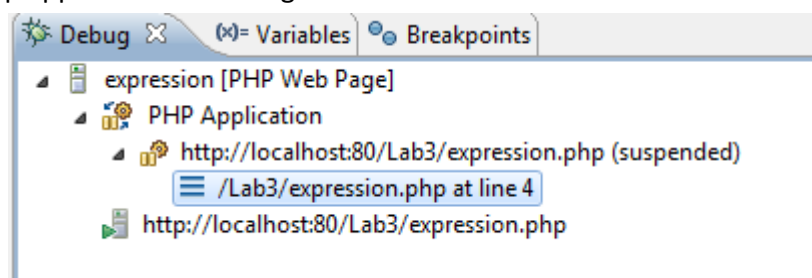
- Double click to line 8 which calculate the average to put a break point at this line.



- Right click at the expression.php or click the button  in the main toolbar or press F11 to start debug the expression.php.
- When prompted to switch to the **PHP Debug** perspective, select the **Remember my decision** check box and click **Yes**.



- The **Debug** view displays all threads executing in the runtime environment. At this point, only the expression.php application is running.



- The **PHP Editor** displays the code of the currently selected thread. You can set breakpoints and step through code using this editor.

```

1 <HTML>
2   <HEAD> <TITLE>Expression Example </TITLE> </HEAD>
3   <BODY>
4     <?PHP
5         $gr1 = 50;
6         $gr2 = 100;
7         $gr3 = 75;
8         $aver = ($gr1 + $gr2 + $gr3) / 3;
9         print "The average is $aver";
10    ?>
11 </BODY>
12 </HTML>
13

```

- Use the **Variables** view to the right of the **Debug** view to check all variables in scope.

| Name | Value |
|---------------|------------|
| ◆ \$_POST | Array [0] |
| ▶ ◆ \$_GET | Array [11] |
| ▶ ◆ \$_COOKIE | Array [0] |
| ▶ ◆ \$_FILES | Array [0] |
| ◆ \$gr1 | (int) 50 |
| ◆ \$gr2 | (int) 100 |
| ◆ \$gr3 | (int) 75 |

- For variables other than primitive types and String objects, object fields can be examined by clicking the arrow beside the variable. Expand **\$_GET** variable (HTTP GET variables - An associative array of variables passed to the current script via the URL parameters) to see its value.

| Name | Value |
|-----------------------|---|
| ◆ \$_POST | Array [0] |
| ◆ \$_GET | Array [11] |
| ◆ debug_host | (string:22) 116.99.16.30,127.0.0.1 |
| ◆ debug_fastfile | (string:1) 1 |
| ◆ start_debug | (string:1) 1 |
| ◆ debug_port | (string:5) 10137 |
| ◆ use_remote | (string:1) 1 |
| ◆ original_url | (string:39) http://localhost:80/Lab3/expression.php |
| ◆ send_sess_end | (string:1) 1 |
| ◆ debug_stop | (string:1) 1 |
| ◆ debug_start_session | (string:1) 1 |
| ◆ debug_no_cache | (string:13) 1251917442096 |
| ◆ debug_session_id | (string:4) 1010 |
| ◆ \$_COOKIE | Array [0] |
| ◆ \$_FILES | Array [0] |

Array [11]

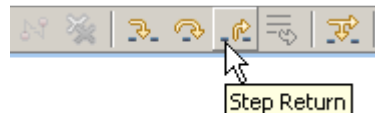
- Using the four stepping modes and other commands, you can execute PHP files one line of code at a time. This allows you to examine, adjust variable values, and monitor the program execution path. Stepping through code is an essential practice for checking logic errors in programs.
 - The **Step Into** button evaluates the current line of code one level deeper into the call stack. The **F5** function key is equivalent to the **Step Into** button.



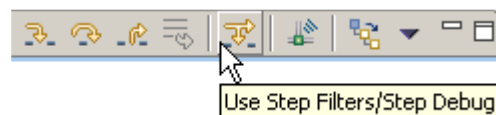
- The **Step Over** button executes until the next line of code in the same level of the call stack. This is a useful function if you are not concerned with the methods invoked by the current line of code. The **F6** function key is equivalent to the **Step Over** button.



- The **Step Return** button continues execution until control is returned to the next level higher in the call stack. Issuing a **Step Return** command in the main method will run the program until completion. The **F7** function key is equivalent to the **Step Return** button



- The **Step Debug** button allows stepping through code that contains debug information (that is, the source code) while skipping code that does not. The currently selected line executes and execution continues until reaching the next line with debug information. There is no equivalent function key for the **Step Debug** command.



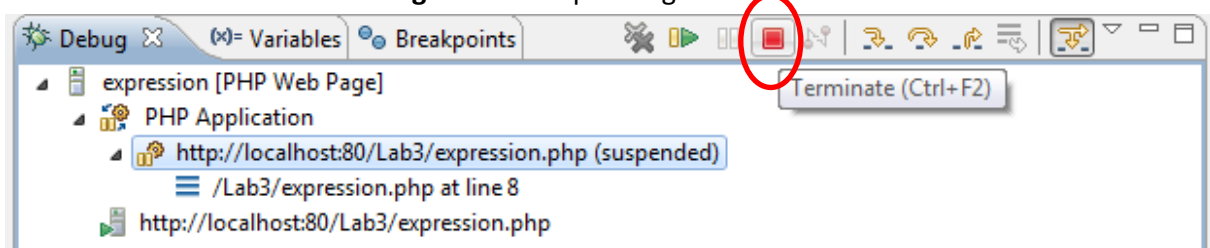
- The **Run to Line** command allows execution to continue until reaching the currently highlighted line in the editor. This command is a useful alternative to continuously issuing the **Step Over** command or inserting a breakpoint to the code. It can be accessed by selecting **Run -> Run to Line** from the workbench menu or pressing **Ctrl-R**.
- Go to line 8 by pressing F7. Move the mouse to \$gr1, \$gr2 and \$gr3 in line 5, 6, 7 to see the value.

```

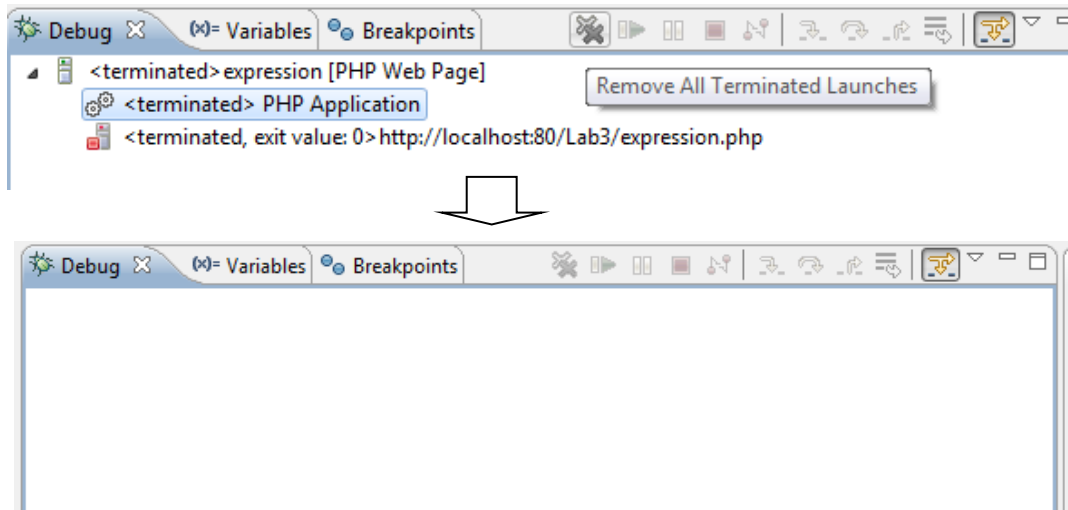
expression.php  http://localhost:80/Lab3/expression.php
1  <HTML>
2      <HEAD> <TITLE>Expression Example </TITLE>
3      <BODY>
4          <?PHP
5              $gr1 = 50;
6              $gr1 = (int) 50;
7
8          $aver = ($gr1 + $gr2 + $gr3) / 3;
9          print "The average is $aver";
10
11      </BODY>

```

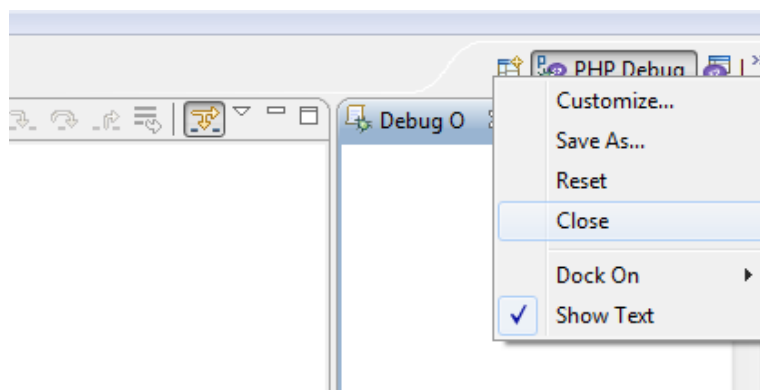
- If the code has a bug which has to fix, for example, the way to calculate the average is changed to $(\text{grade1} \times 6 + \text{grade2} \times 2 + \text{grade3} \times 2) / 10$; we can terminate the application by selecting <http://localhost:80/LabProject/expression.php> (suspended) in the **Debug** view and clicking the **Terminate** button on the **Debug** title bar or pressing **Ctrl+F2**.



- Click the **Remove All Terminated Launches** button to clear the **Debug** view



- Close the **Debug** perspective by right-clicking on the Debug icon in the perspective switcher toolbar and selecting **Close**.



- Edit the expression.php as the expectation and debug again to observe the result.

2.3. Create a String test

Do the same to create and run stringtest.php which includes all examples in the lecture.

2.4. Create a Basic form

Do the same to create and run basicform.php.

```

expression.php  http://localhost:80/Lab3/form1.php  basicform.php
1 <HTML>
2   <HEAD> <TITLE> A Simple Form </TITLE> </HEAD>
3   <BODY>
4       <FORM ACTION="http://fit.hut.edu.vn/~trangntt/courses/wp"
5           METHOD=POST >
6           Click submit to start our initial PHP program.
7       <BR>
8       <input type="text" name="Name">
9       <BR>
10      <INPUT TYPE="SUBMIT" VALUE="Click To Submit">
11      <INPUT TYPE="RESET" VALUE="Erase and Restart">
12  </FORM>
13  </BODY>
14 </HTML>
15

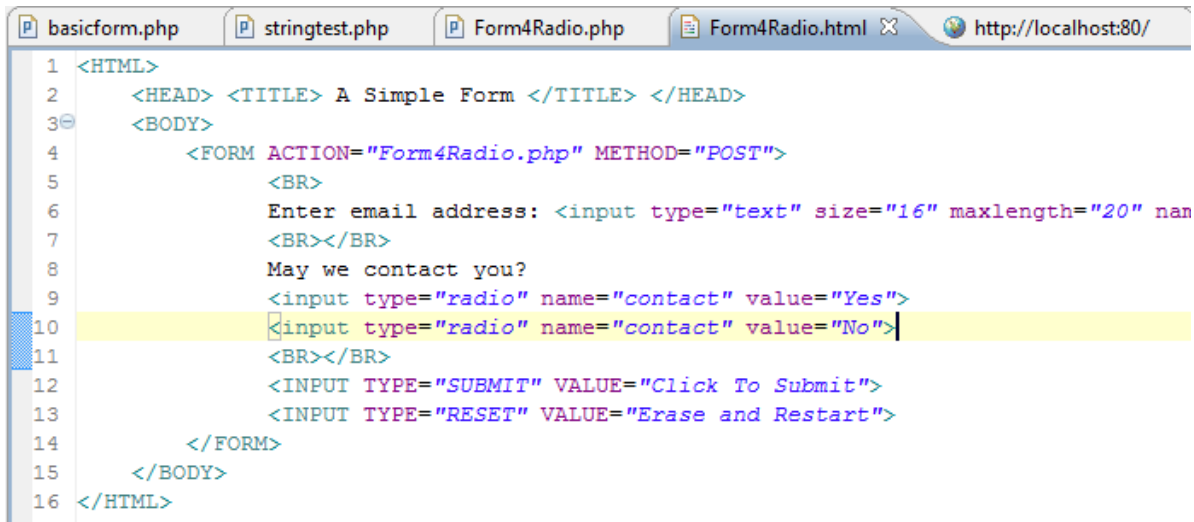
```


2.5. Create a registration form

Copy the basicform.php to registrationform.php which includes all HTML inputs in the lecture (refer to yahoo mail or gmail sign up).

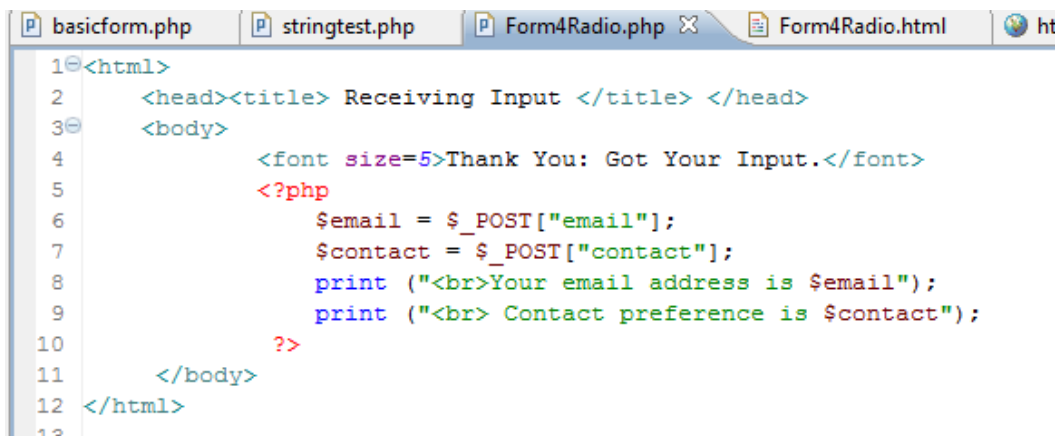
2.6. Create a confirm form

Step 1. Create Form4Radio.html



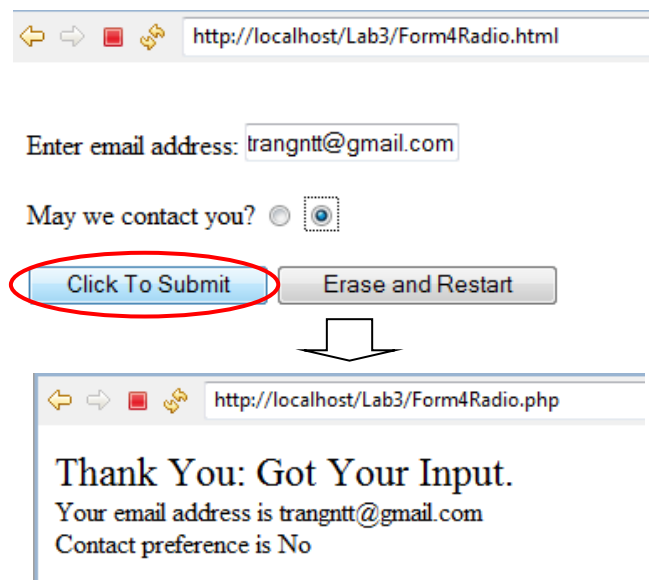
```
1 <HTML>
2   <HEAD> <TITLE> A Simple Form </TITLE> </HEAD>
3   <BODY>
4     <FORM ACTION="Form4Radio.php" METHOD="POST">
5       <BR>
6       Enter email address: <input type="text" size="16" maxlength="20" name="email">
7       <BR></BR>
8       May we contact you?
9       <input type="radio" name="contact" value="Yes">
10      <input type="radio" name="contact" value="No">
11      <BR></BR>
12      <INPUT TYPE="SUBMIT" VALUE="Click To Submit">
13      <INPUT TYPE="RESET" VALUE="Erase and Restart">
14    </FORM>
15  </BODY>
16 </HTML>
```

Step 2. Create Form4Radio.php



```
1 <html>
2   <head><title> Receiving Input </title> </head>
3   <body>
4     <font size=5>Thank You: Got Your Input.</font>
5     <?php
6       $email = $_POST["email"];
7       $contact = $_POST["contact"];
8       print ("<br>Your email address is $email");
9       print ("<br> Contact preference is $contact");
10    <?>
11  </body>
12 </html>
```

Step 3. Run the Form4Radio.html



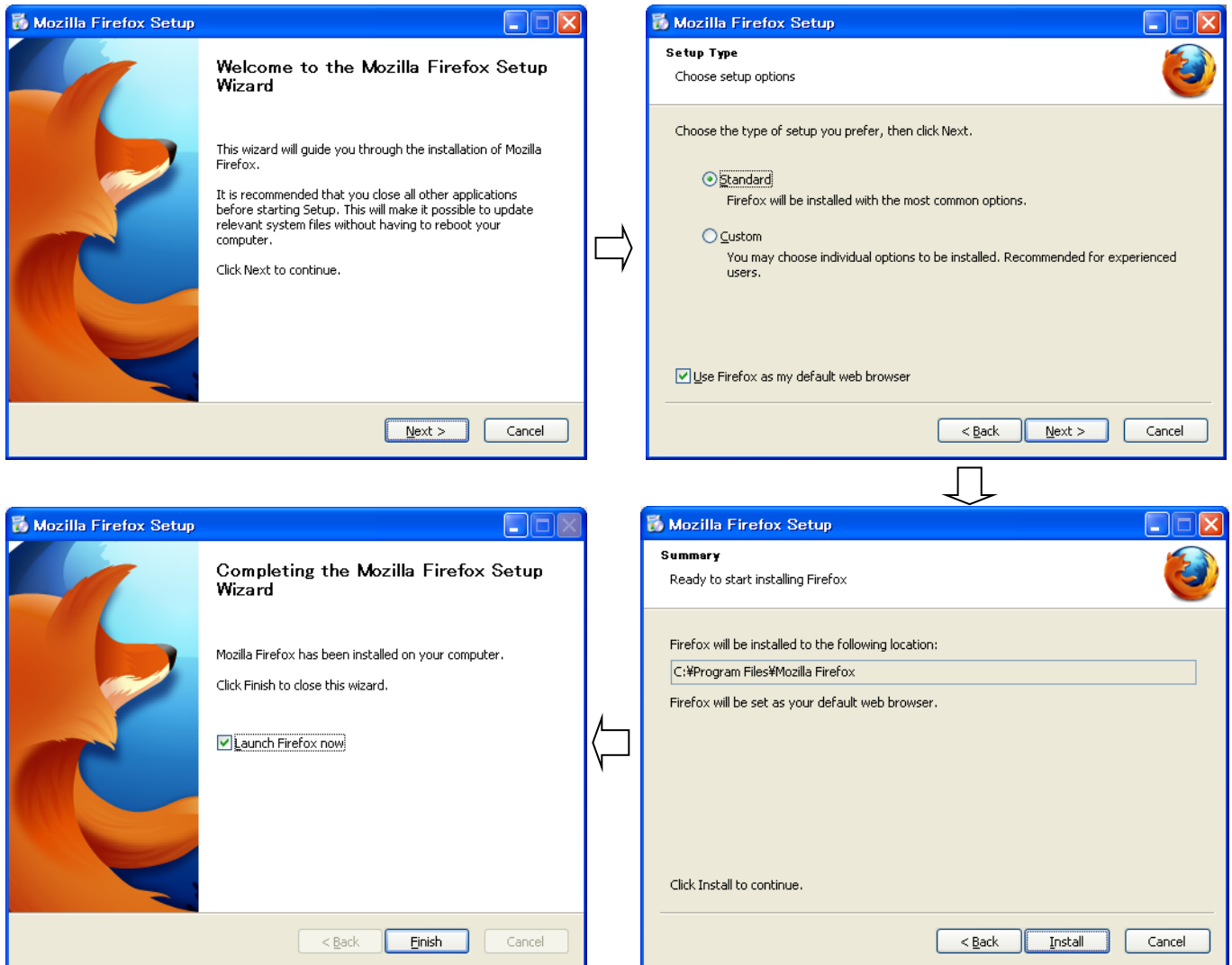
2.7. Change to GET method

Change the Form4Radio example from POST method to GET method. Observe the result and give comments.

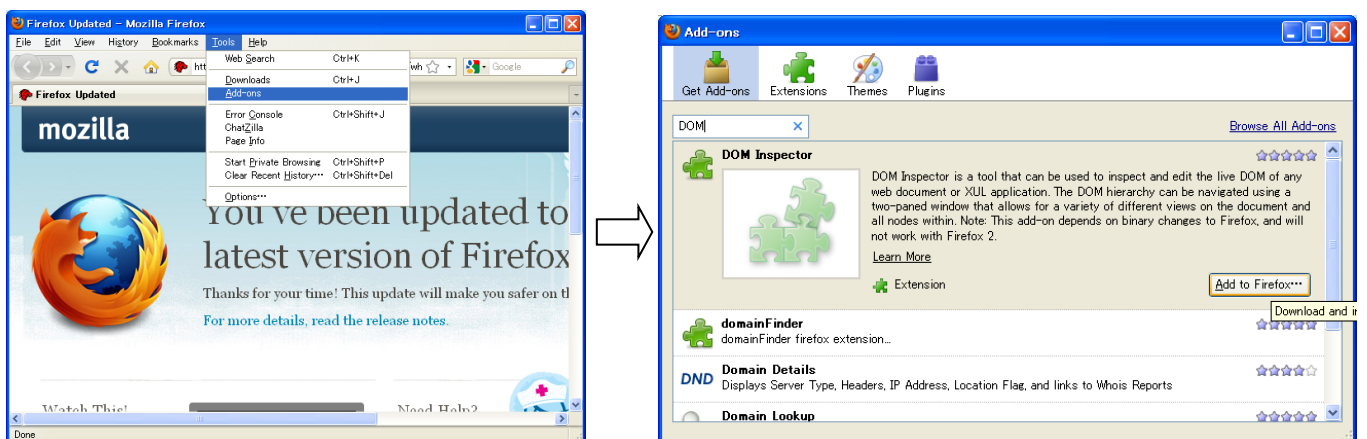
2.8. Firefox and Add-ons

Step 1. Install Firefox & Add-ons

- Install Firefox: <http://www.mozilla.com/en-US/firefox/>



- Install Add-ons: DOM Inspector, Firebug and Web Developer



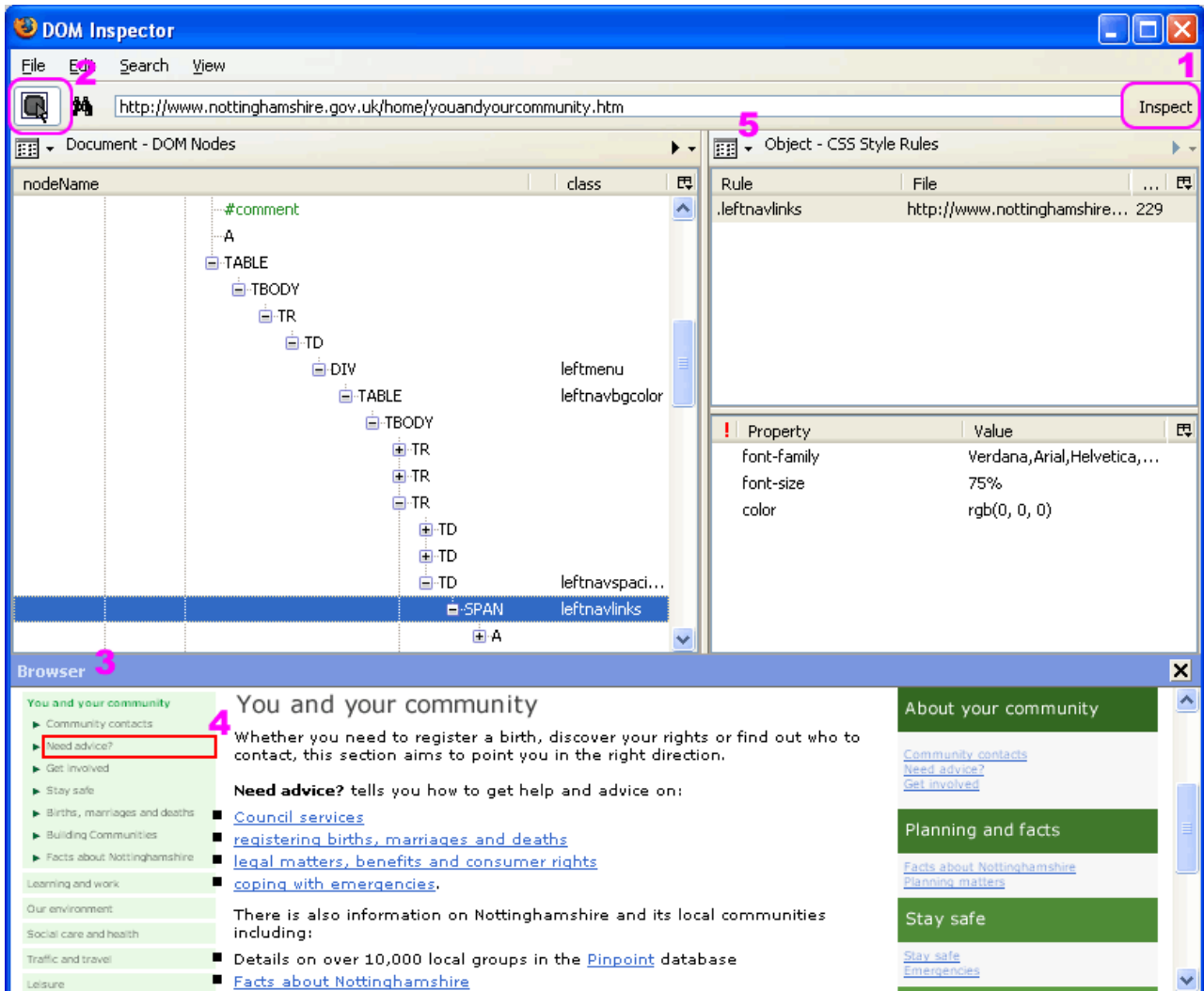
- Restart Firefox

Step 2. Using DOM Inspector

- Practise steps:
 - Open Form4Radio.html in Firefox
 - Check DOM and other information by using DOM Inspector
- Additional guide:

Understanding the way a web page has been constructed is normally a case of viewing the source HTML and trying to build a mental picture of its structure, based on the way the different elements are nested within each other. It takes some practice, but it can be done. Much easier is to use a tool like the DOM Inspector, which lets us look at each part of the page, piece-by-piece, using a structured tree approach.

As an example I'll take [the page](#) I talked about in [my last post](#). The first step is to browse to it in Firefox. Now launch the DOM Inspector from the Tools menu (Ctrl+Shift+I). You'll see a window something like this:

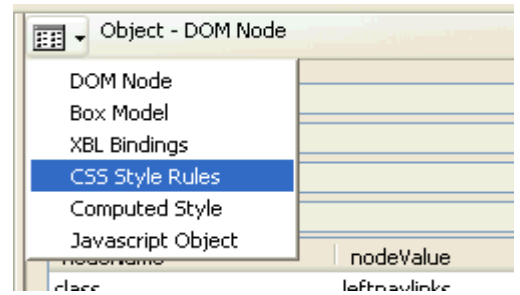


From the View menu enable the "browser" and make sure that Blink Selected Element is selected. The URL of the page you're interested in should already be the address bar at the top. Click on the Inspect button (highlighted by #1 in pink letters on the screengrab above) to the right of the address bar and it will load the page in the "browser" pane.

In order to find the part of the page you're interested within the document tree you can use two methods. The hardest is to drill-down through the tree, element by element, until that part of the page flashes (highlighted #4). Much easier is to click on the button (highlighted #2) that lets you simply click on said element to automatically find it in the tree. Click on one of the problems links with the small text and it will quickly that element, as shown above. In this case, the structured hierarchy of the link within the DOM is:

HTML > BODY > TABLE > TR > TD > TABLE > TR > TD > DIV > TABLE > TR > TD > SPAN > A

Armed with this information it is then fairly simple to work out which bits of the CSS stylesheet have an affect on the link element. How? Well, we can find out much more information about any given element node than simply its place in the document. Using the button highlighted at #5 we can choose to view different property-sets for the selected node. Switch to CSS Style Rules and you'll see all the styles applied to it. Work back through the tree and you'll see the other styles, some of which it inherits.



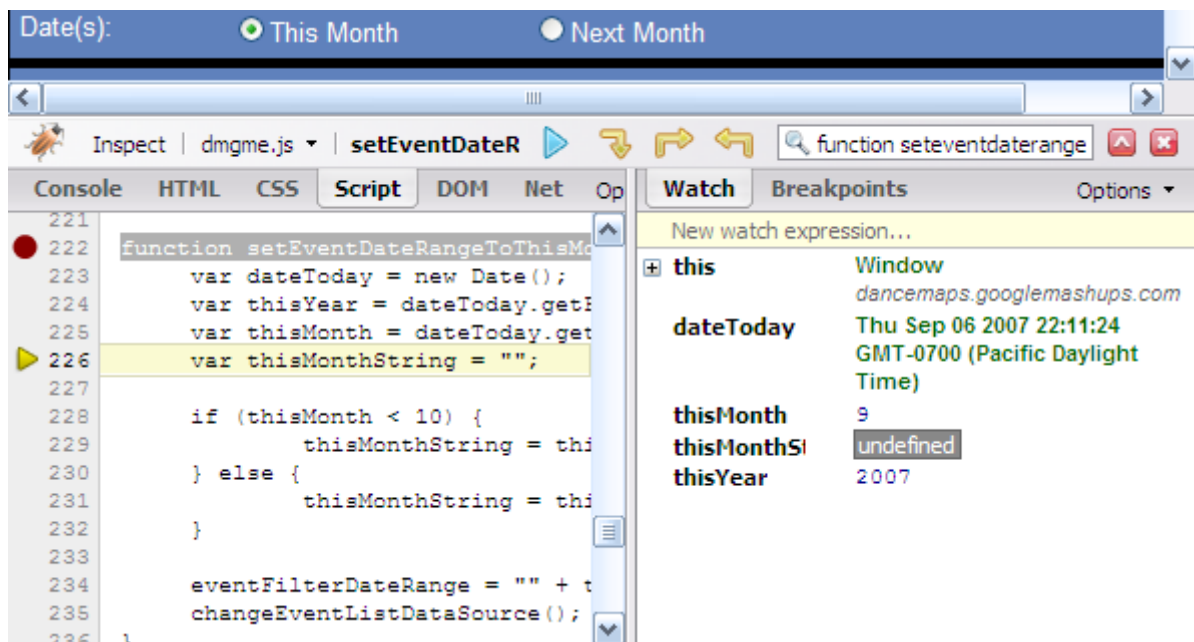
It's hard to sum up just how useful this tool can be. Hopefully this is a good example though. It may be a little rough around the edges but it's an extremely useful tool. Until recently I'd over-looked it and never bothered working out how to use it. Now I know how I'll probably be using it daily.

Here are some of the other uses and **benefits of the DOM Inspector**:

- Learn the structure of a HTML document.
- See how particular elements on a page are nested.
- Delete elements to see effect on page.
- Edit/change class names properties attributes of elements
- Add attributes to the tree (In theory! This doesn't seem to work too well).

Step 3. Using Firebug

- Firebug features:
 - Inspect custom stylesheets included by Google Mashup Editor
 - Modify in-memory stylesheets to see the changes reflected immediately
 - Place watches and breakpoints into running JavaScript
 - Execute arbitrary JavaScript in the context of your running application
 - Monitor Ajax calls, showing response times, posted content, and results
 - Profile JavaScript functions to help you identify bottlenecks and optimize your application.



- Practise steps:
 - Open Form4Radio.html in Firefox
 - Check DOM and other information like HTML sources in Firebug
 - Browse web pages that you are interested in and check their information as well
 - Edit their HTML sources on the fly

2.9. Exercise

- Design a form including all types of HTML inputs which requests the users to fill their name, class, university, hobby (checkboxes to list some common hobbies and others), ... (These fields are only for recommendation, you have to develop by yourselves other information which will help you get a higher score for this exercise).
- Then create a PHP file to process the action of the above HTML form. This PHP file will displays all the information that users entered.
- Print all screens which you use add-ons of Firefox including DOM Inspector, and Firebug for the above exercise and paste to a MS Word document with a brief comment for each screen.

For example:

Hello, <user name>

You are studying at <class>, <university>

Your hobby is

1. <hobby 1>

2. <hobby 2>

3. ...

...