Lab 3 – ASP.NET Basics 1a

Contents

[1 Lab Objectives 1](#_Toc94959784)

[2 Create your Lab 3 Home Page 2](#_Toc94959785)

[3 View the Code-Behind File 4](#_Toc94959786)

[4 Add Some Controls 5](#_Toc94959787)

[5 Add another Control 8](#_Toc94959788)

[6 Add a Validator Control 9](#_Toc94959789)

[7 Exploring the Debugger 10](#_Toc94959790)

[8 A Question for You 14](#_Toc94959791)

[9 Add AJAX 15](#_Toc94959792)

[10 Package Assignment for Submission 16](#_Toc94959793)

# Lab Objectives

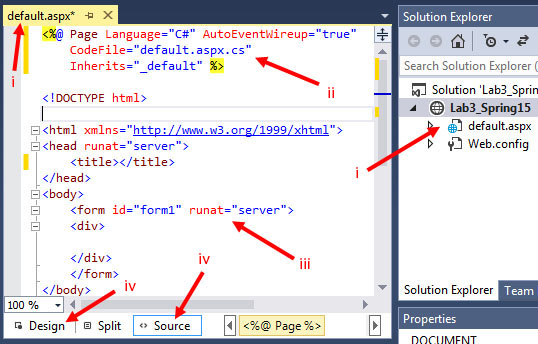
1. Build a simple web app that utilizes server-side processing.
2. Learn the architecture of an ASP.NET web app.
3. Know what a code-behind file is and how to utilize event-handlers.
4. Layout an app with basic server controls and validators
5. Become familiar with the page lifecycle.
6. Write simple event handlers.
7. Utilize AJAX in an app.
8. Use the debugger in VS

To make this document easier to read, it is recommended that you turn off spell checking and grammar checking in Word:

1. Choose: File, Option, Proofing
2. At the very bottom, check: “Hide spelling errors…” and “Hide grammar errors…”

# Create your Lab 3 Home Page

1. Create your *lab03* project (solution folder must be named *lab03\_lastName*).
2. **Add a Web Form.** A *web form* contains HTML and it contains ASP.NET *web controls*.
3. Right-click *lab03* in the SE, right-click and
4. Choose: *Add, Web Form*.
5. Supply the name: *default*
6. Press, *OK*. This is your Lab 3 home page.
7. Display the page in *Source* view. It will look similar to the figure on the right.



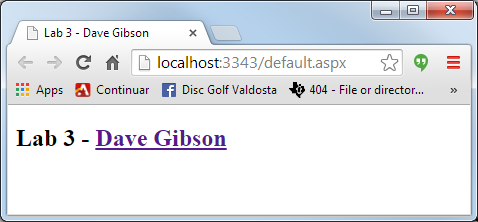
Note:

1. A *web form* has the file extension *.aspx*
2. The first line of a web form has is a *page directive*. A *page directive* specifies various properties for use by the compiler. The four properties shown:

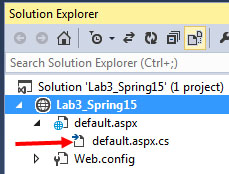
* *Language* – Specifies the C# language.
* *AutoEventWireup* – When *true*, automatically registers event handlers with events[[1]](#footnote-1).
* *CodeBehind* (the figure above is from a previous version of VS and used *CodeFile* as the name of the property) *– S*pecifies the name of the *code-behind* file that contains the code that accompanies this web form. By convention, the name is the name of the web form followed by the language, in this case “.cs” for the C# language. The *code-behind* file contains the server-side code to process this form.
* *Inherits* – The *Page* class instance *inherits* the members of the (somewhat awkwardly named) *\_default* class*.* We will see this class and code-behind file shortly. We will also see why the class name is *\_default* when the file name is *default*. In other words, what is the deal with that underscore?

1. The *body* always contains a single *form* with an important attribute and value, *runat=”server”*. This informs the *ASP.NET* engine to process this element on the server. We will consider this in subsequent labs and lecture.
2. The form is opened automatically in an editor that provides *Design* and *Source* modes.
3. Add the HTML title: “Lab 3 – FirstName LastName”, substituting your name. For example:

<title>Lab 3 - Dave Gibson</title>

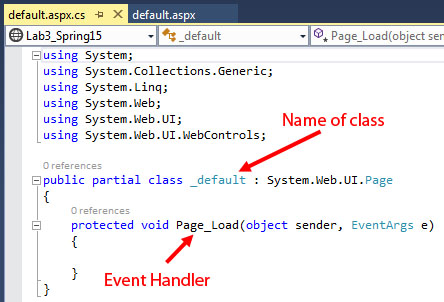
1. Write a level 2 header, inside the *<div>* tags, that reads: “Lab 3 – FirstName LastName”
2. View your page on local host: right-click in page, and choose: *View in Browser*. **Note: your page will not show a hyperlink as is shown on the right.**

# View the Code-Behind File



Note: I will frequently refer to a *web form* as a *page* as it is simpler to say and write. If I need to be more specific, I will refer to a *web form (.aspx)* or an *HTML page (.html).*

1. **Inspect the Page.**
2. In the SE, expand *default.aspx* and you will see there is another file named *default.aspx.cs*. This is the *code-behind* file. (You will also see another file, *default.aspx.designer.cs* which is not shown in the figure on the right. This file is auto-generated and generally, should not be modified directly. If you want to, open this file and inspect it.)
3. Open the code-behind file by double clicking on it (or right-click on the page and choose: *View Code*).



Note the following (read each item and scroll up to the figure to verify/coordinate the comment):

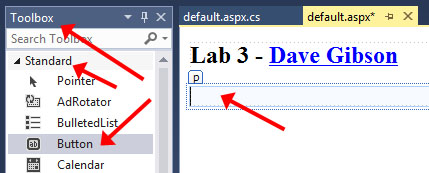
1. The name of the class is *\_default*. The “\_” is automatically added for a page named *default* or *Default* because the former is a reserved word in C#, the later is a reserved word in VB.NET (the two languages can be used together in the same page.). Any other page name that is not a reserved word is simply the name of the class. For instance, a *BookSearch* page would have a class named *BookSearch* as well.
2. A *Page\_Load* (empty) method is shown. We call this an *event handler*. When a client requests this page, the server always calls this method. Later in this lab we will place a line of code there and see how it works. In another lab we will explore in more detail how to use *Page\_Load.*
3. The *using* statements provide the same role as *import* in Java: access to the API classes.
4. The “:” in the class definition is the same as *extends* in Java. Thus, the superclass of *\_default* is *Page.*
5. The “partial” keyword in the definition of the class indicates that this is a *partial class*. The other part is in the *default.aspx.designer.cs* file. We have no real need to consider this; however, my notes talk a bit more about what a partial class is.

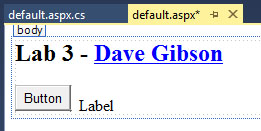
Note: VS uses the “next line” brace style. To change it to the more usual, same line, do the following:

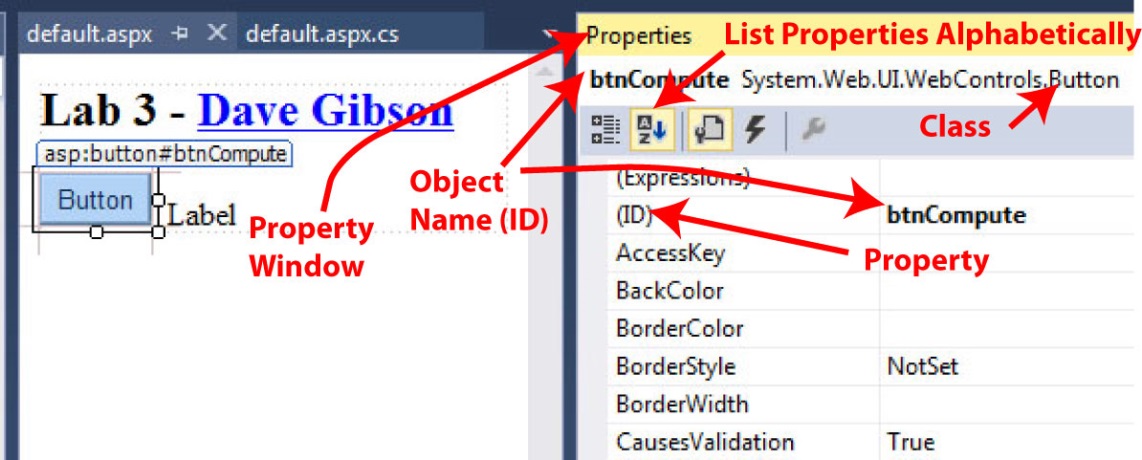
1. Go to Tools -> Options -> Text Editor -> C# -> Code Style -> Formatting-> New Lines
2. In the first section, “New line options for braces”, uncheck everything.

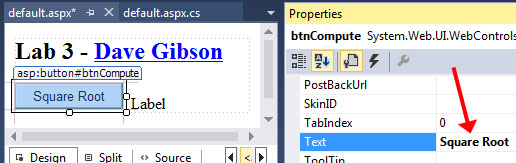
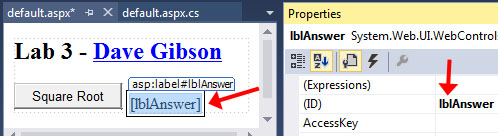
# Add Some Controls

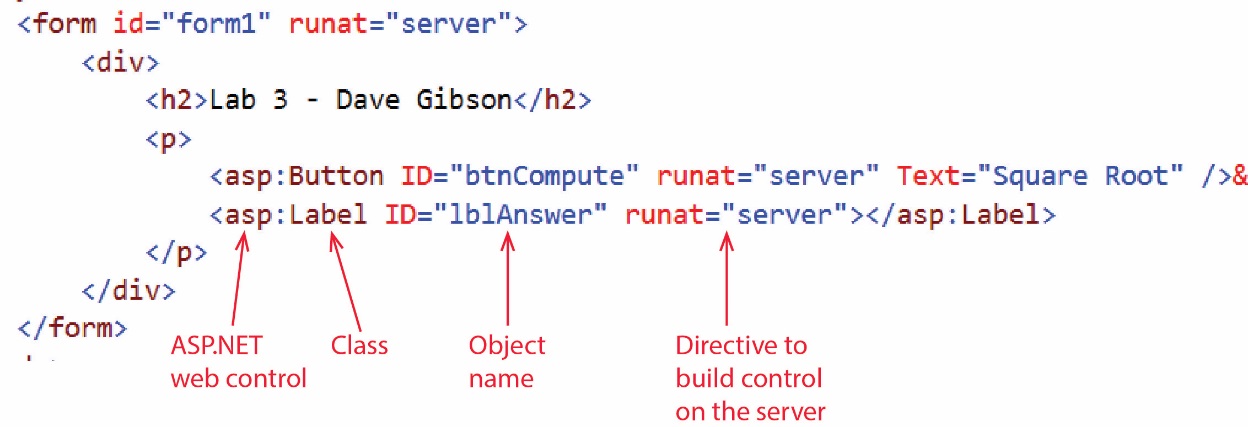
1. **Add a button and a Label**



1. Open your page (*default.aspx*)in *Design* view
2. Put your cursor at the end of your name and press Enter. This makes sure that subsequent additions to the page will occur *inside* the *form* tag. We will verify this later.
3. Find the *Toolbox* on the left of your window. Make sure the *Standard* tab is selected.
4. Drag a *Button* onto the page.
5. Drag a Label onto the design. Add a few spaces between the Button and Label. The result is shown on the right.
6. **(For the last time) Your page will not show a link as the one on the right does.**
7. **Change some Button properties**
8. Select the button and inspect the *Properties* window below the *SE* (If you don’t see it, right-click the Button and choose: *Properties*). In the figure on the right, I have hidden the SE to make the figure more compact. Notes the items indicated.



1. Change the “ID” property to: *btnCompute.* The button’s name is *btnCompute*, which is an object created from the *Button* class.
2. Change the “Text” property to: *Square Root*. This will change the text that is displayed on the face of the button.
3. ****Change the Label properties –**
4. Select the Label and make sure the Properties window is displayed.
5. Change the “ID” property to: *lblAnswer*.
6. Scroll down and see that the “Text” property is set to, “Label”, which is what is displayed when the page is viewed. Delete this text (empty). Later, we will write code that changes the “Text” property to a message when the button is pressed.
7. View the page in *Source* view. As shown below, you will see two *ASP.NET web controls* (*controls*) that represent the button and label. These tags are run on the server, which creates the button and label as pure HTML and sends them to the browser. We illustrate this later.

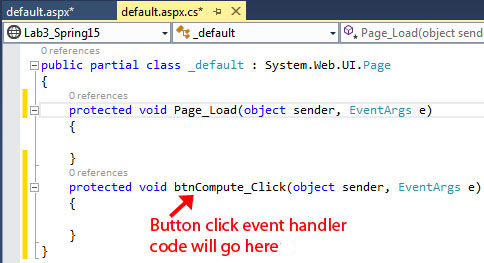
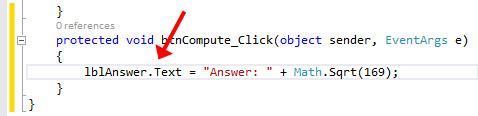
**

1. **Build and run your application**
2. Choose: *Build, Build Solution* (or Ctrl+Shift+B)*.* When complete, the message “Build succeeded” will appear in *Output* pane below the *Code* pane.

Note, When you right-click in page and choose: *View in Browser,* it automatically builds your page. However, if things start to act up, don’t hesitate to *Build* or *Rebuild* your page explicitly.

1. Display your page on localhost. There are 3 ways:

* Right-click in page and choose: *View in Browser*.
* Choose: *Debug, Start without Debugging*. This will display your page on localhost.
* Press: Ctrl+F5.

1. Note: you can press the Button but nothing happens. And, the label is not shown (because the Text property is empty).
2. **Add functionality to the Button**
3. Display the page in *Design* view.
4. Double-click the button. This will display the code-behind file and it will create a method named: *btnCompute\_Click* as shown in the figure on the right. This is an *event-handler* (for the button) and is called when the Button is pressed when the page is viewed.
5. Write code in the event-handler that will display the square root of 169 in the *Label*. As you type, you will notice that auto-complete will help you.

**lblAnswer.Text = "Answer = " + Math.Sqrt(169);**

Note the following:

* Microsoft’s convention is for the name of all public members of a class to begin with an upper-case letter, and following after that the usual camel-case.
* Later in the course we will consider the parameters for this event handler. Most (or all) event handlers we consider in this class have the same two parameters.
* The *.Text* property is being used as if it were an instance variable. In other words, we are not calling a *setText* method. Behind the scenes, there actually is a setter. Later, we will see that C# defines getters and setters in a really clever way: so that we don’t have to write *getText* and *setText;* we just use *Text:*

**lblAnswer.Text = "..."; // Text used as a setter**

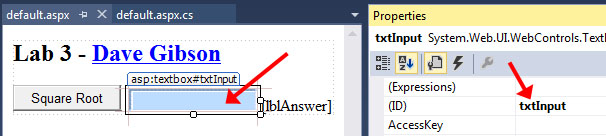
**String msg = lblAnswer.Text; // Text used as a getter**

1. Return to your page (*default.aspx*) and view it in the browser (right-click and choose: *View in Browser*). Nothing will happen if you try to run your code-behind file.

|  |  |
| --- | --- |
| Before data entry | After data entry |
|  |  |

# Add another Control

1. **Add a TextBox.**  We will have the user enter a number to take the square root of.
2. Open your page in *Design* view.
3. Use the Toolbox to add (drag) a *TextBox* to the page. Reposition it so that it is between the Button and the Label.
4. Use the Properties window to set the “ID” property for the TextBox to: *txtInput*



1. **Change the Button event handler.** We will change the button’s event-handler code to obtain the number the user typed in and compute the square root of that number.
2. Display the code-behind page (*default.aspx.cs*).
3. Replace the code in the event handler (*btnCompute\_Click*) with:

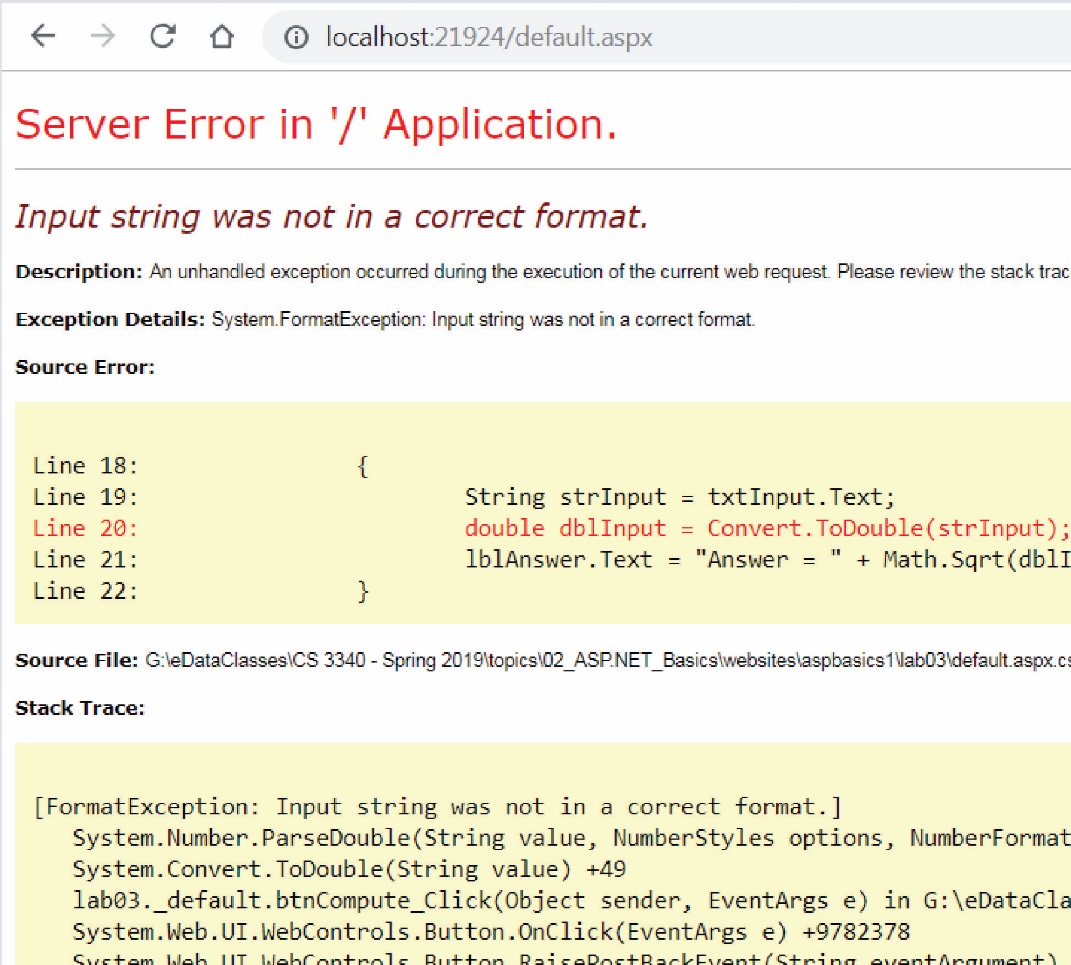
**String strInput = txtInput.Text;**

**double dblInput = Convert.ToDouble(strInput);**

**lblAnswer.Text = "Square Root = " + Math.Sqrt(dblInput);**

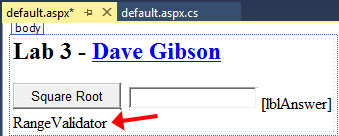
Study this code to understand what it is doing. Note that the *Convert* class contains static methods to convert almost anything into some other type.

1. **View your page in the browser.**

****

1. Test by supplying a number. The page should work as expected.
2. Test by supplying a string of characters. An exception is thrown when the run-time tries to convert these characters to a double. The output is shown on the right and is very similar to the output you receive when an exception is thrown in Java.

# Add a Validator Control

1. **Add input validation**
2. Open your page in *Design* view.
3. Open the Toolbox, open the Validation tab. Drag a *RangeValidator* onto the page
4. Select the *RangeValidator* in *Design* view and then change the properties below in the Properties Window:

|  |  |
| --- | --- |
| **Property** | **Value** |
| *ControlToValidate* | txtInput |
| *ErrorMessage* | Input must be positive |
| *ForeColor* | Red |
| *MaximumValue* | Double.MaxValue |
| *MinimumValue* | 0 |

1. **Modify *web.config.***
2. Run your page and type in a value. It will immediately throw an exception**.** To fix this, we have to add a node to *Web.config* to tell it to use *jQuery*. This file contains global information about your site (your project, *lab03* in this case).
3. In the SE, open your *Web.config* file. Copy the *appSettings* node below:

<appSettings>

<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />

</appSettings>

And paste it above (or below) the *system.web* node. Your (partial) *web.config* will look like this:

<configuration>

<appSettings>

<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />

</appSettings>

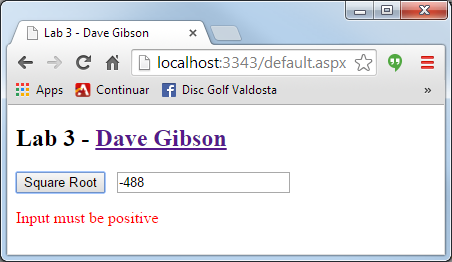
<system.web>

<compilation debug="true" targetFramework="4.5" />

<httpRuntime targetFramework="4.5" />

</system.web>

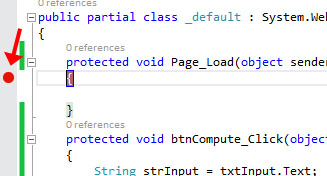
</configuration>

1. View your page. Test with a negative number (or a string).

# Exploring the Debugger

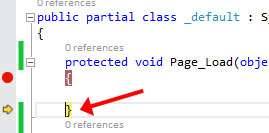
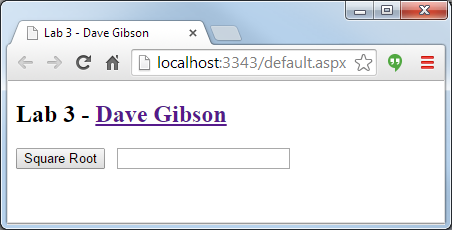
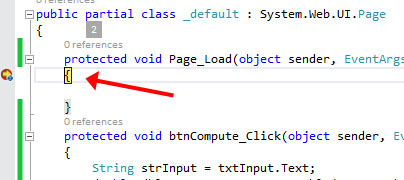
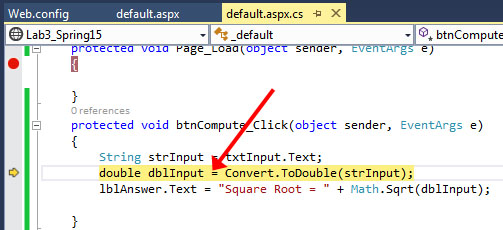
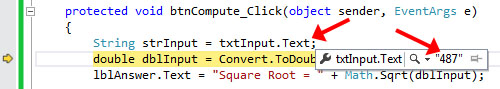
The Debugger is used to step through your code, line-by-line. You start by specifying a *breakpoint*(s). Then, you run the program in debug mode and the code executes until it gets to the breakpoint where it stops execution and waits input from you as to how to proceed. At that point, you can: (a) inspect the value of variables and controls, (b) step line-by-line through the code, (c) advance to the next breakpoint, (d) step into or over method calls, *etc*. **LEARN TO USE THE DEBUGGER!**

1. **Add a Breakpoint**

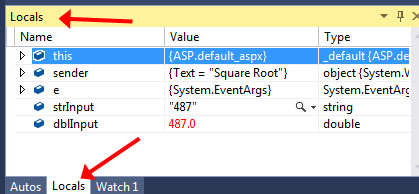
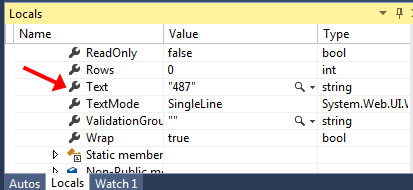
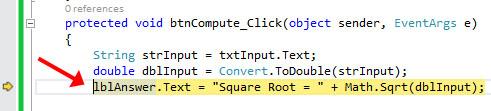
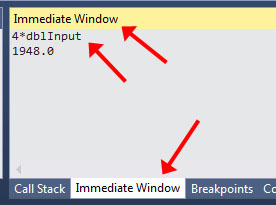


1. Open the code-behind page (*default.aspx.cs*).
2. Find the *Page\_Load* event handler. There is no code in the method; however, we could place code here.
3. Click in the far-left margin of the opening brace ({). A red circle will appear. This is a *Breakpoint*. When we run the program in Debug mode, execution will stop here. Then, we can step through the program.
4. **Run program in Debug mode.** If something goes wrong with the steps below, or you want to redo them, you need to stop the debugger, and then start the step(s) again. To stop the debugger, choose: *Debug, Stop Debugging*
5. Choose: *Debug, Start Debugging* (or press F5). It may take a few seconds. The browser will be displayed but there will be nothing on the page. You may be automatically directed back to VS as shown on the right below. If you aren’t, just switch back there. The yellow highlight means the program execution has been halted just before this line is executed.

|  |  |
| --- | --- |
| Browser | Visual Studio |
|  | E:\Data-Classes\CS 3340 - Web Programming\Topics\02_ASP.NET_Basics\Labs\pics\ww4.jpg |

1. **Execute the Page\_Load event.**
2. Choose: *Debug, Step Over* (or press F10). Notice the yellow highlight has moved indicating that the previous line of code has been executed (the brace!). Had there been code in Page\_Load, we would have stepped through it by continuing to press F10. **The purpose was to show you that the very first thing that happens on the server is that the Page\_Load event code is run.**
3. Step over again (*e.g.* press F10). Now, the page is loaded and the browser shows the full page.
4. **Type a number in.**
5. Type in a number and press the Button. The code executes and again stops at the Page\_Load event. When you pressed the button, the page did a “post-back” to the server. Notice that the Page\_Load event runs every time a post-back occurs.
6. Press F10 four (4) times. Notice that code finished *Page\_Load* and then entered *btnCompute\_Click.*
7. Hover your mouse over “Text” in the line with “txtInput.Text” (the line above the yellow highlighted line). Notice that the value you typed in is displayed (*e.g. 4*87 from the example above).
8. Hover your mouse over “strInput” and you will see a similar result.
9. **Use the Locals Window**

The discussion below refers to the *Locals* window. You will probably not see it, but instead see the *Autos* window. I’ve forgotten the difference between the two. However, for these purposes, either window will be fine.

1. Find the *Locals* window (look for a *Locals* tab along the bottom left of your screen). The *Locals* window shows variables defined in the local scope. If you don’t see it, choose: *Debug, Windows, Locals.* This is only available when you are actively debugging a program. It should look similar to the figure on the right. It looks like this:
2. Expand the “this” reference. “this” is the instance (object) of your *Page*. Thus, “this” is an object which contains references to all the variables and controls we have defined in the page (class).
3. Scroll down to “txtInput” (almost to the end of the long list) and expand.
4. Then, scroll down to the “Text” property of *txtInput* and see the value you typed in.
5. Redisplay the code-behind file. Press F10 once.
6. **Use the Immediate Window**
7. Find the *Immediate Window* (look for an *Immediate Window* tab along the bottom left of your screen). The *Immediate Window* allows you to write lines of code and execute them. This is occasionally useful. If you don’t see it, choose: *Debug, Windows, Immediate Window.*

1. Type:

4 \* dblInput

Then, press Enter. Notice that *dblInput* is a local variable defined in the event-handler. The immediate window allows you to compute with variables that are in scope. You could also see the value in an array (or any variable), e.g. myArray[3] and Enter. It is probably hard to see why this is useful, but it is from time-to-time, as a debugging technique. Note: in previous versions you had to type a “?” in front of the expression. It does still work when you do this, and the documentation says to do it, but it works without (at least in this example).

1. **Finish the debugging session**
2. Press F10 two more times and the result is shown in the browser. If you press the Button again, the debugging continues.
3. End the debugging session. Do one of the following:

* Choose: Debug, Stop Debugging
* Press Shift + F5
* Press the red (maroon?) square icon on the toolbar.

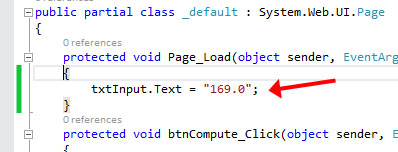
1. **Summary of Debugging Commands** – Read, no action required.

|  |  |  |
| --- | --- | --- |
| Description | Keyboard | Menu |
| Start debugging | F5 | Debug, Start Debugging |
| Continue, advance to next breakpoint | F5 | Debug, Continue |
| Step over a method call | F10 | Debug, Step Over |
| Step into a method | F11 | Debug, Step Into |
| Step out of a method | Shift+F11 | Debug, Step Out |
| Stop debugging | Shift+F5 | Debug, Stop Debugging |

1. **Other Debugging Techniques** – Read, no action required. You will need to display debug information as you work on your assignments. Here are several approaches:
2. Put a *TextBox* (with *TextMode* property set to *MultilLine*) at the bottom of each page you develop (so it is out of the way of the page you are developing). Then, you can display debug output there, *e.g.* txtDebug.Text +=“price=” + price + “\n”;. **I recommend this approach for every page you develop for this class. It will be fine to leave this on homework you submit, just make sure it is clearly below everything else.**
3. (Not working as described, Spring 19 – Ignore) You can use: *System.Diagnostics.Debug.WriteLine(…)* to write to the *Output* window in VS. If you choose this approach: (a) include a *using System.Diagnostics;* so that you can just write: *Debug.WriteLine(…)* (b) Go to *Tools, Options, Debugging, OutputWindow* and turn off: Module Load Messages, Module Unload Messages, Process Exit Messages, Thread Exit Messages. Otherwise, the output you write will have other messages interspersed.
4. Conditional compilation is useful for debugging and testing and custom builds. I couldn’t find a good link to reference to, though.

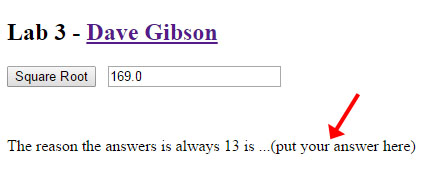
# A Question for You

1. **Figure out what is going on!**
2. Make sure the debugger is stopped.



1. Add this line to the Page\_Load event:

txtInput.Text = "169.0";

1. Run the program and simply press Enter. You should see the result: 13.
2. Next (the page should still be running), change the input value (to 4, say) and press Enter. The answer is still 13.
3. **Use the debugger (and your brain)** to figure out why the answer is always 13. When you find the answer, type it at the bottom of your page.
4. Comment out the line you added to Page\_Load. Thus, the page should properly calculate the square root of the input now.

# Add AJAX

1. **An AJAX Example**

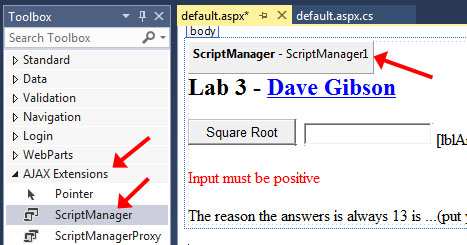
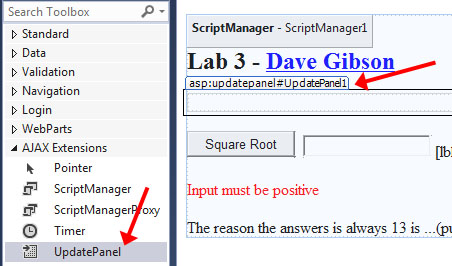
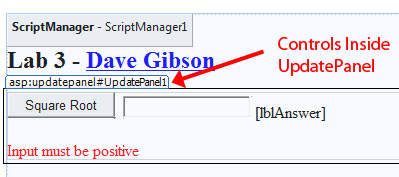
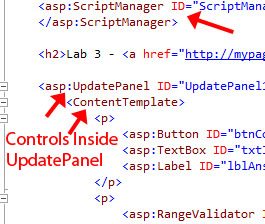
The page we have built so far communicates with the server in a *synchronous* manner. The user presses the button and *waits* for the server to respond. You can tell a page is using synchronous communication when the browser tab flashes after an action is taken.

1. Run your page and interact with it and observe the tab flicker.

*AJAX* is a technique for communicating with the server *asynchronously*. When the user makes an action, the page sends a message to the server in the background, but continues to run normally in the browser. An example is Google Maps and thousands (millions?) of other sites.

1. Move or zoom the map and observe how smooth the interaction is and notice that the tab doesn’t flicker: <http://goo.gl/maps/ih2wZ>
2. **AJAXify your page**

We will add code to make your button communicate asynchronously with the server.

1. Display your page in Design mode. Open the Toolbox and expand the *AJAX Extensions* heading
2. Drag a *ScriptManager* to the beginning of your page (make sure it is inside the *form* tag). Note: *ScriptManager* is a web control that has no visual presence when the page is run. It’s job is in the background, enabling asynchronous communication.
3. Drag an *UpdatePanel* to just before the “Square Root” button.
4. Cut/Paste or drag each of the components (button, textbox, label, validator) into the *UpdatePanel*.
5. Change to Source mode and the markup should look like this (I’ve formatted the code so yours will look different):
6. **View your page in a browser**. Make sure the tab doesn’t flicker. If it does, then check to see that the markup is correct (as shown above). Make sure the <ContentTemplate> is present. If you cut/paste in Source mode, it will not put this required tag.

# Package Assignment for Submission

1. Close VS and zip your *lab03\_lastName* solution folder and submit on Blazeview in the *Lab 03* dropbox.

If you need further directions, follow step 10 from Lab 1, exactly, substituting *lab03* for *lab01*.

|  |
| --- |
| To avoid a deduction of points verify the following:   1. The name of your zip file is: *lab03\_lastName* (your last name, of course). 2. Inside the zip file, verify that your *lab03\_lastName* solution folder is indeed named: *lab03\_lastName* 3. Inside your *lab03\_lastName* solution folder there should be:  * A *lab03\_lastName.sln* file * The page(s) you developed * Some other files & folders. |

1. <https://support.microsoft.com/en-us/help/324151/how-to-use-the-autoeventwireup-attribute-in-an-asp-net-web-form-by-usi> [↑](#footnote-ref-1)