IS 242 Web Application Development 1

LECTURE 14: C# PROGRAMMING BASICS (PART 1)

Outlines of today's lecture

- In this lecture we will explore the following C# programming fundamentals:

- Control Events
- Event Subroutines
- Variables and variable declaration

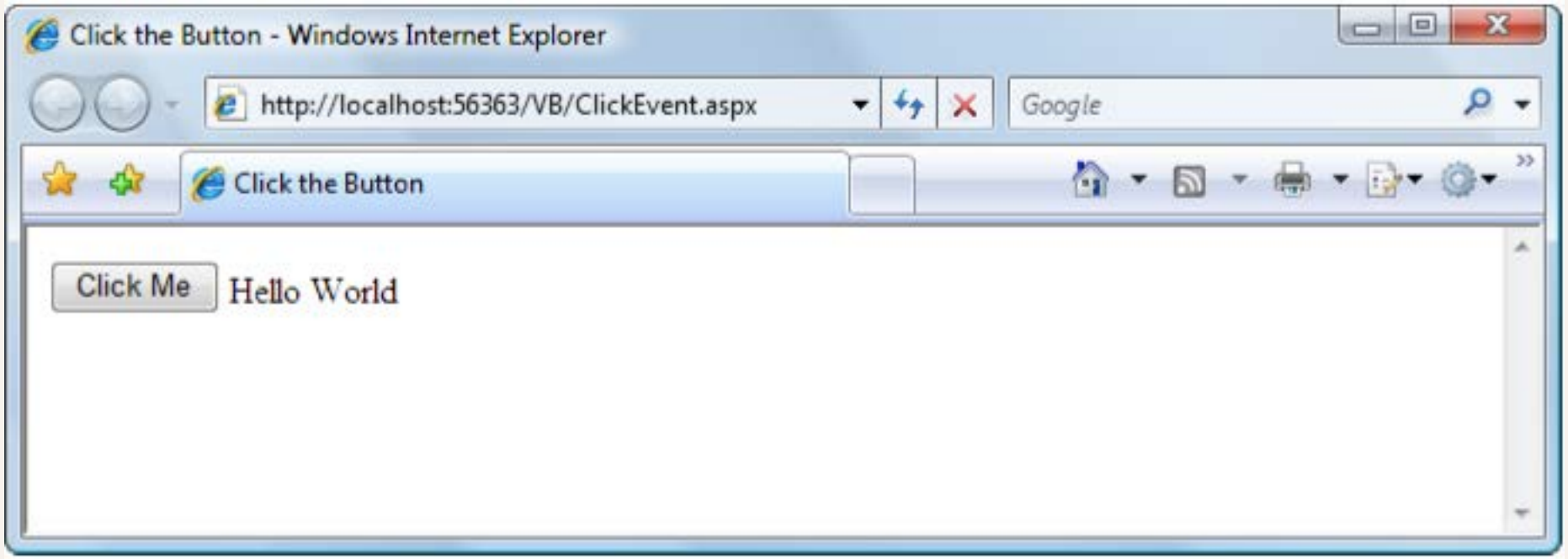
Programming Basics

- One of the building blocks of an ASP.NET page is the application logic: the actual programming code that allows the page to function.

- To get anywhere with ASP.NET, you need to grasp the concept of **events**.

- Example:
- Clicking a button on an ASP.NET page.
- That button (or, more specifically, the ASP.NET Button control) raises an event (in this case, it will be the Click event)
- A method called an **event handler** executes automatically when an event is raised

Control Events and Subroutines



Control Events and Subroutines (Cont.)

- An event (sometimes more than one) is raised, and handler code is called, in response to a specific action on a particular control

```
<%@ Page Language="C#" %>
⋮
<script runat="server">
public void button_Click(Object s, EventArgs e)
{messageLabel.Text = "Hello World";}
 </script>
<form id="form1" runat="server">
<div>
<asp:Button ID="button" runat="server“ OnClick="button_Click" Text="Click Me" />
<asp:Label ID="messageLabel" runat="server" />
⋮
⋮
```

Control Events and Subroutines (Cont.)

- When the button's clicked, it raises the Click event, and ASP.NET checks the button's OnClick attribute to find the name of the handler subroutine for that event.

- In the previous code, we instruct ASP.NET to call the button_Click routine when the user clicks the button.

Control Events–Button Control Events

- There are many events that your controls can use, though some of them are found only on certain controls.
- Here's the complete set of attributes that the Button control supports for handling events:
  - **OnClick**
  - **OnCommand**
  - **OnLoad**
  - **OnInit**
  - **OnPreRender**
  - **OnDisposed**
  - **OnDataBinding**

The Structure of Subroutines

- When a control raises an event, the specified subroutine (if one is specified) is executed. Let's take a look at the structure of a typical subroutine that interacts with a web control:

C#

```
public void mySubName(Object s, EventArgs e)
{
  ⋮ subroutine code…
}
```

- Public (for a global subroutine that can be used anywhere within the entire page) and Private (for subroutines that are available for the specific class only).

- void: This keyword defines that the subroutine doesn't return a result.

- Object s: It is a reference to the control that fired the event. Each Control has a particular type, such as Label or TextBox, Here, we're putting that Object in a variable named s.

- EventArgs e: This, the second parameter, contains certain information that's specific to the event that was raised.

Page Events

- The idea is the same as for control events, except that here, it's the page as a whole that generates the events.

- You've already used one of these events: the Page_Load event, which is fired when the page loads for the first time.

- Note that we don't need to associate handlers for page events as we did for control events; instead, we just place our handler code inside a subroutine with a preset name.

Page Event Subroutines

- Page_Init

  called when the page is about to be initialized with its basic settings

- Page_Load

  Called once the browser request has been processed, and all the controls in the page have their updated values

- Page_PreRender

  called once all objects have reacted to the browser request and any resulting events, but before any response has been sent to the browser

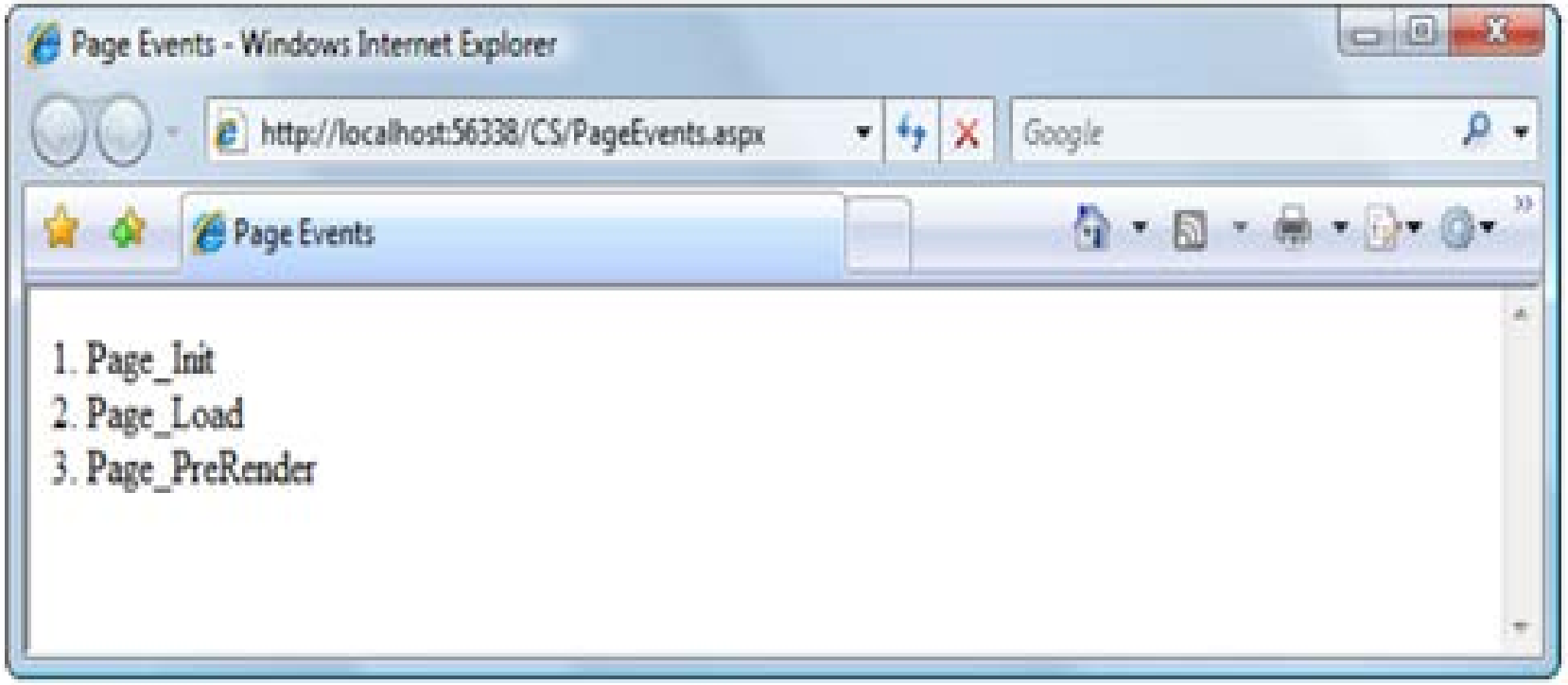
- Page_UnLoad

  called when the page is no longer needed by the server, and is ready to be discarded

Page Event Subroutines–Example

C# LearningASP\CS\PageEve

```
<%@ Page Language="C#" %>
⋮
<script runat="server">
  void Page_Init(Object s, EventArgs e)
  {
    messageLabel.Text = "1. Page_Init <br/>";
  }
  void Page_Load(Object s, EventArgs e)
  {
    messageLabel.Text += "2. Page_Load <br/>";
  }
  void Page_PreRender(Object s, EventArgs e)
  {
    messageLabel.Text += "3. Page_PreRender <br/>";
  }
  void Page_UnLoad(Object s, EventArgs e)
  {
    messageLabel.Text += "4. Page_UnLoad <br/>";
  }
</script>
⋮
```


Page Events - Windows Internet Explorer
http://localhost:56338/CS/PageEvents.aspx
Google
Page Events
1. Page_Init
2. Page_Load
3. Page_PreRender

Variables and Variable Declaration

- There are many different kinds of data types, including strings, integers (whole numbers), and floating point numbers (fractions or decimals).
- Before you can use a variable in C#, you must specify the types of data it can contain using keywords such as String, Integer, and Decimal, like this:

```
string name;
int age;
```

- These lines declare the types of data we want our variables to store, and are therefore known as **variable declarations**.
- Sometimes, we want to set an initial value for variables that we declare; we can do this using a process known as **initialization**, which simply involves declaring a variable and setting its initial value:

```
string carType = "BMW";
```

Variables and Variable Declaration (Cont.)

The table below lists the most useful data types available in C#.

C#	Description
`int`	whole numbers in the range -2,147,483,648 to 2,147,483,647
`decimal`	numbers up to 28 decimal places; this command is used most often when dealing with costs of items
`string`	any text value
`char`	a single character (letter, number, or symbol)
`bool`	true or false
`object`	a generic type that can be used to refer to objects of any type

Variables and Variable Declaration (Cont.)

- C# is strongly typed language.
- To convert String to integer

```
int intX;
string strY = "35";
IntX = Convert.ToInt32(strY) + 6;
```