Chapter 5: Regular Expressions, Frames and Rollovers (14 marks)

Regular Expression –
language of regular expression, finding non matching characters, entering a range of characters, matching digits and non digits, matching punctuations and symbols, matching words, replacing a the text using regular expressions. returning the matched character. regular expression object properties.

Frames —
create a frame, invisible borders of frame, calling a child windows, changing a content and focus of a child window, writing to a child window, accessing elements of another child window.

Rollover —
creating rollover, text rollover, Multiple actions for rollover, more efficient rollover.

Frames

Introduction

- Used to display more than one HTML documents inside one browser window.
- Each frame consists of one html document and frames are independent to each other.
- To divide browser window into different frames, ***<frameset>*** tag is used.
- While using ***<frameset>*** tag, ***<body>*** tag can not be used.

<frameset> attributes

- **rows**: The rows attribute is used to create horizontal frames in web browser. This attribute is used to define no of rows and its size inside the frameset tag.
- **cols**: The cols attribute is used to create vertical frames in web browser. This attribute is basically used to define the no of columns and its size inside the frameset tag.
- **border**: This attribute of frameset tag defines the width of border of each frames in pixels. Zero value is used for no border.
- **frameborder**: This attribute of frameset tag is used to specify whether the three-dimensional border should be displayed between the frames or not for this use two values 0 and 1, where 0 defines for no border and value 1 signifies for yes there will be border.
- **framespacing**: This attribute of frameset tag is used to specify the amount of spacing between the frames in a frameset.

<frame> attributes

- **name:** This attribute is used to give names to the frame. It differentiate one frame from another.
- **src:** This attribute in frame tag is basically used to define the source file that should be loaded into the frame. The value of src can be any url.
- **marginwidth:** This attribute in frame tag is used to specify width of the spaces in pixels between the border and contents of left and right frame.
- **marginheight:** This attribute in frame tag is used to specify height of the spaces in pixels between the border and contents of top and bottom frame.

Creating frames

```
<html >
<head>
 <title>Create a Frame</title>
</head>
<frameset rows="50%,50%">
 <frame src="WebPage1.html" name="topPage" />
 <frame src="WebPage2.html" name="bottomPage" />
</frameset>
</html>
```

In example code to create frame, we have used frameset tag to divide browser window into two rows, each row represent one frame of equal width as we have used 50%,50%. Observe that we have not used *<body>* tag in this html file, that is not required while using *<frameset>* tag.

Consider following code of line

```
<frameset rows = "30%, *, 30%">
```

there are three rows, first and third of size 30% each and remaining (40%) height will be assigned to second frame.

src property is used to define document that we want to load in frame. WebPage1.html will be loaded in first frame and WebPage2.html will be loaded in second frame.

Invisible borders

The border can be hidden by setting the frameborder and border attributes of the <frameset> tag to zero (0). Any value other than 0 that is assigned to the frameborder and border attributes causes the browser to display the border.
border attribute defines the width of border, if it set to 0 and ***frameborder*** set to 1 means border is there with 0 pixel width, so that it will not be visible.

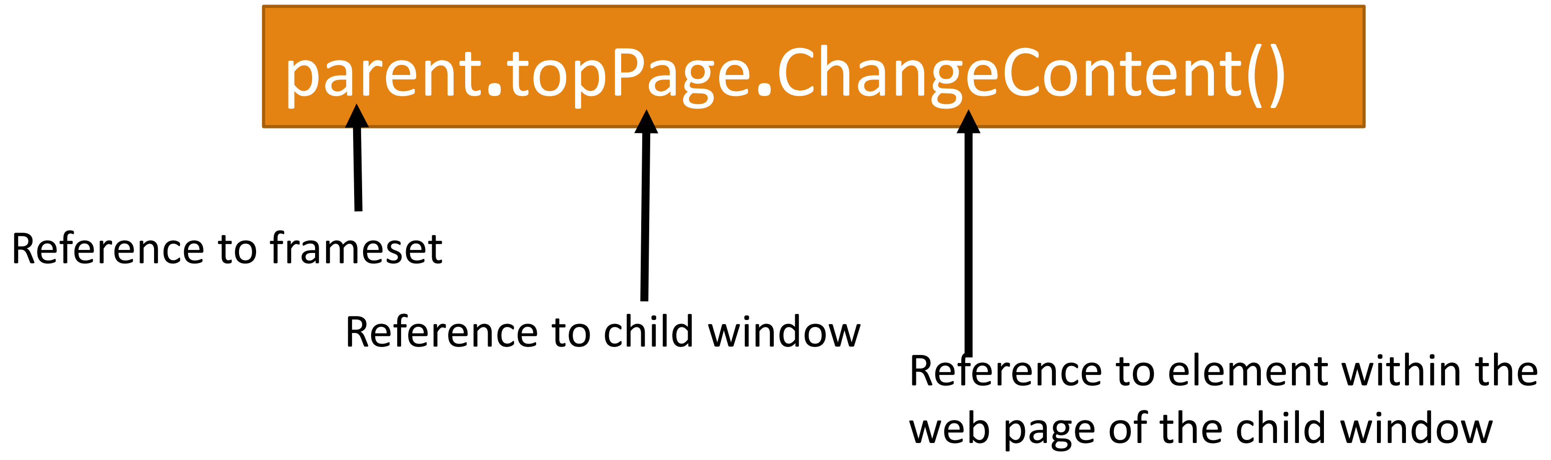
```
<html>

<frameset rows="50%,50%" frameborder="1" border="10">

 <frame src="header.html" name="topPage"/>

 <frame src="footer.html" name="bottomPage"/>

</frameset>

</html>
```

calling a child window

Here we will perform a simple task of calling a JavaScript function that is defined in another child window, *but how to do it?* Answer is

You can refer to another child window by referencing the frameset, which is the parent window, and then by referencing the name of the child window, followed by whatever element within the web page of the child window that you want to access.

In following code of line *topPage* is a name of first child window of which element (*i.e. ChangeContent() function*) we are going to access from second child window (*bottomPage*). Detailed example is also given.



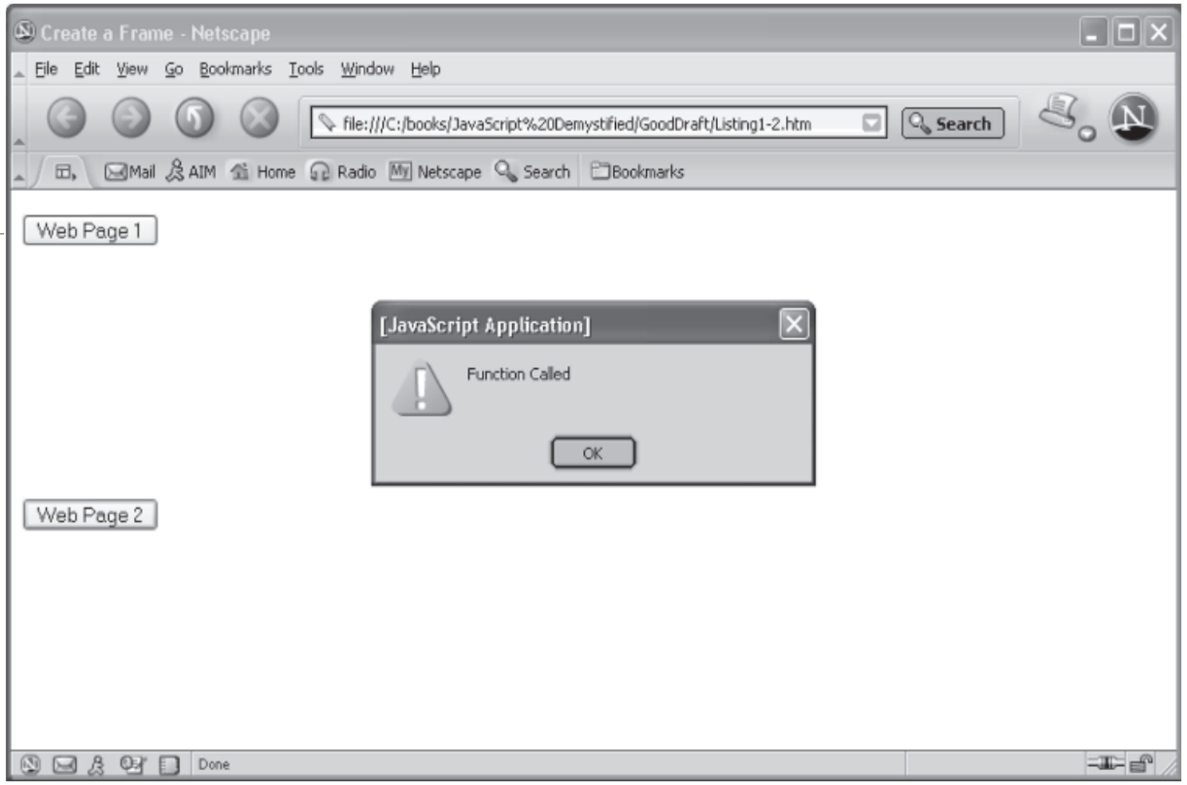
```
parent.topPage.ChangeContent()
```

Reference to frameset
Reference to child window
Reference to element within the web page of the child window

main_wnidow.html

```
<html>
<frameset rows="50%,50%" frameborder="1" border="0">
 <frame src="first.html" name="topPage"/>
 <frame src="second.html" name="bottomPage"/>
</frameset>
</html>
```

main_wnidow.html output in Browser Window


Create a Frame - Netscape
File Edit View Go Bookmarks Tools Window Help
file:///C:/books/JavaScript%20Demystified/GoodDraft/Listing1-2.htm
Search
Mail AIM Home Radio Netscape Search Bookmarks
Web Page 1
[JavaScript Application]
OK
Web Page 2
Done

Function Called

first.html

```
<html>
<body>
<INPUT name="WebPage1" value="Web Page 1" type="button">
</body>
<script>
            function ChangeContent() {
                        alert("Function Called") }
</script>
</html>
```

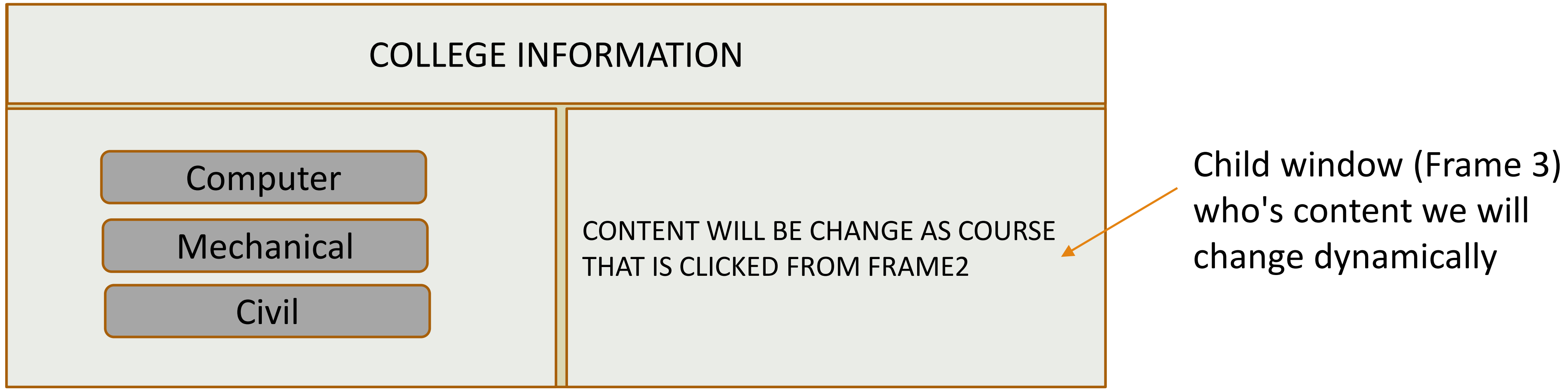
second.html

```
<html>
<body>
<INPUT name="WebPage2" value="Web Page 2" type="button" onclick="parent.topPage.ChangeContent()" />
</body>
</html>
```

Changing a content of a child window

Consider a Browser window with three frames

- First frame is to display College Information
- Second Frame is to Display list of available courses by using input type button.
- Third Frame is to display Information of Course that is clicked from list of courses displayed in frame2.



COLLEGE INFORMATION
Computer
Mechanical
Civil
CONTENT WILL BE CHANGE AS COURSE THAT IS CLICKED FROM FRAME2
Child window (Frame 3) who's content we will change dynamically

To change content of a child window we have to access it by using name and then use ***location.href*** to change the source document to be display in that window. Consider following line of code

```
function ChangeContent() {
 parent.crs_content.location.href= content
 }
```

Here ***content*** is js variable which defines html docs that we want to display in a frame having name as ***crs_content***. Refer detailed example on next page.

college.html

```
<html>
<frameset rows="20%,80%" frameborder="1" border="1">
    <frame src="college_content.html" name="clg_info"/>
    <frameset cols="50%,50%" frameborder="1" border="1">
        <frame src="course_list.html" name="crs_list"/>
        <frame src="co.html" name="crs_content"/>
    </frameset>
</frameset>
</html>
```

College_content.html

```
<html>
<body>
<center>
 <h1>AI ARKP</h1>
 <h4>Plot # 2 & 3, near Thana naka, Khanda
        Gaon, New Panvel, Navi Mumbai </h4>
</center>
</body>
</html>
```

crs_content.html

```
<html>
    <body>
        <input type="button" value="Computer" onclick="change_content(this.value)" class="btn"><br>
        <input type="button" value="Mechanical" onclick="change_content(this.value)" class="btn"><br>
        <input type="button" value="Civil" onclick="change_content(this.value)" class="btn"><br>
    </body>
    <script>
        function change_content(crs)
        {   if(crs == 'Computer')
                {       content = 'co.html'     }
            else if(crs == 'Mechanical')
                {       content = 'mech.html'   }
            else
                {       content = 'ce.html'     }
            parent.crs_content.location.href = content
        }
    </script>
</html>
```

co.html

```
<html>
    <body>
        <h1>COMPUTER ENG</h1>
        <h4>INTAKE : 120</h4>
        <h4>TOTAL NO OF STUDENTS ADMITTED : 120</h4>
    </body>
</html>
```

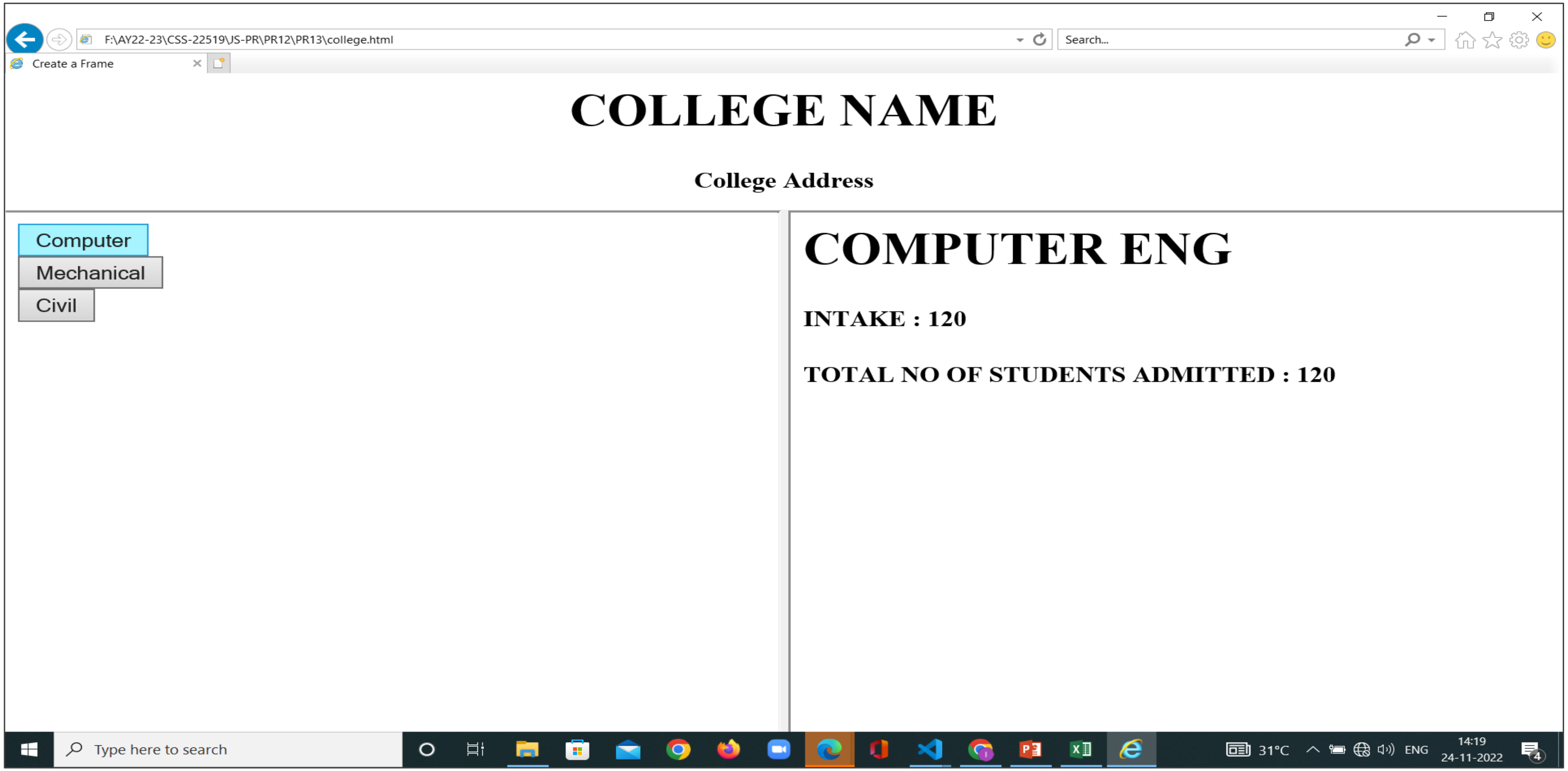
me.html

```
<html>
    <body>
        <h1>MECHANICAL ENG</h1>
        <h4>INTAKE : 90</h4>
        <h4>TOTAL NO OF STUDENTS ADMITTED : 85</h4>
    </body>
</html>
```

ce.html

```
<html>
    <body>
        <h1>COMPUTER ENG</h1>
        <h4>INTAKE : 60</h4>
        <h4>TOTAL NO OF STUDENTS ADMITTED : 50</h4>
    </body>
</html>
```

Out Put



F:\AY22-23\CSS-22519\JS-PR\PR12\PR13\college.html

Create a Frame

COLLEGE NAME

College Address

COMPUTER ENG

INTAKE : 120

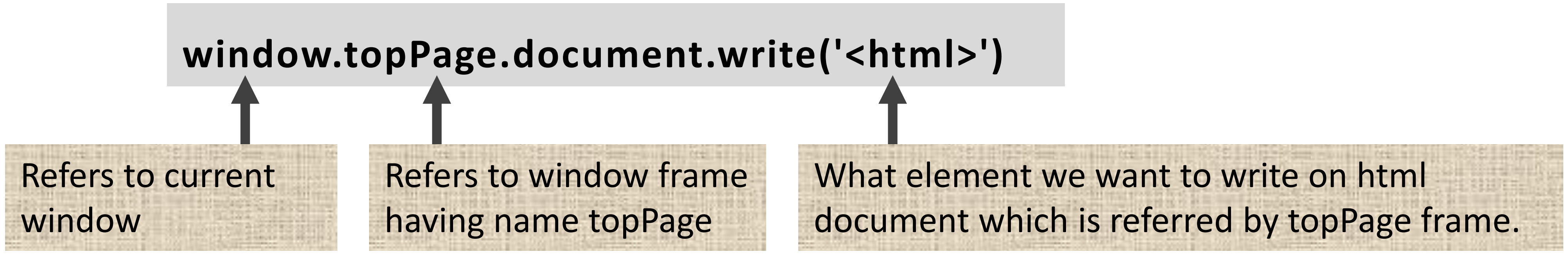
TOTAL NO OF STUDENTS ADMITTED : 120

Writing to a child window

We can dynamically create the content when you define the frameset by directly writing to the child window from a JavaScript.

The JavaScript must be defined in the HTML file that defines the frameset and called when the frameset is loaded.

To write we will use following statement



```
window.topPage.document.write('<html>')
```

Refers to current window
Refers to window frame having name topPage
What element we want to write on html document which is referred by topPage frame.

This is illustrated in the next example, where the JavaScript function writes the content for the ***topPage*** child window, assuming the child is from the same domain.

main_wnidow.html

```
<html>
<head>
                <script>
                                function ChangeContent()
                                {
                                window.topPage.document.write('<html>')
                                window.topPage.document.write('<body>')
                                window.topPage.document.write('<h1>COMPUTER ENGINEERING DEPT</h1>')
                                window.topPage.document.write('<h4>INTAKE : 120</h4>')
                                window.topPage.document.write('<h4>NO OF STUDENTs ADMITTED : 270</h4>')
                                window.topPage.document.write('</body>')
                                window.topPage.document.write('</html>')
                                }
                </script>
<frameset rows="50%,50%" frameborder="1" border="0" onload="ChangeContent()">
                <frame src="first.html" name="topPage"/>
                 <frame src="second.html" name="bottomPage"/>
</frameset>
</html>
```

first.html

```
<html>
<body>
<h1>FIRST PAGE</h1>
</body>
</html>
```

second.html

```
<html>
<body>
</body>
</html>
```

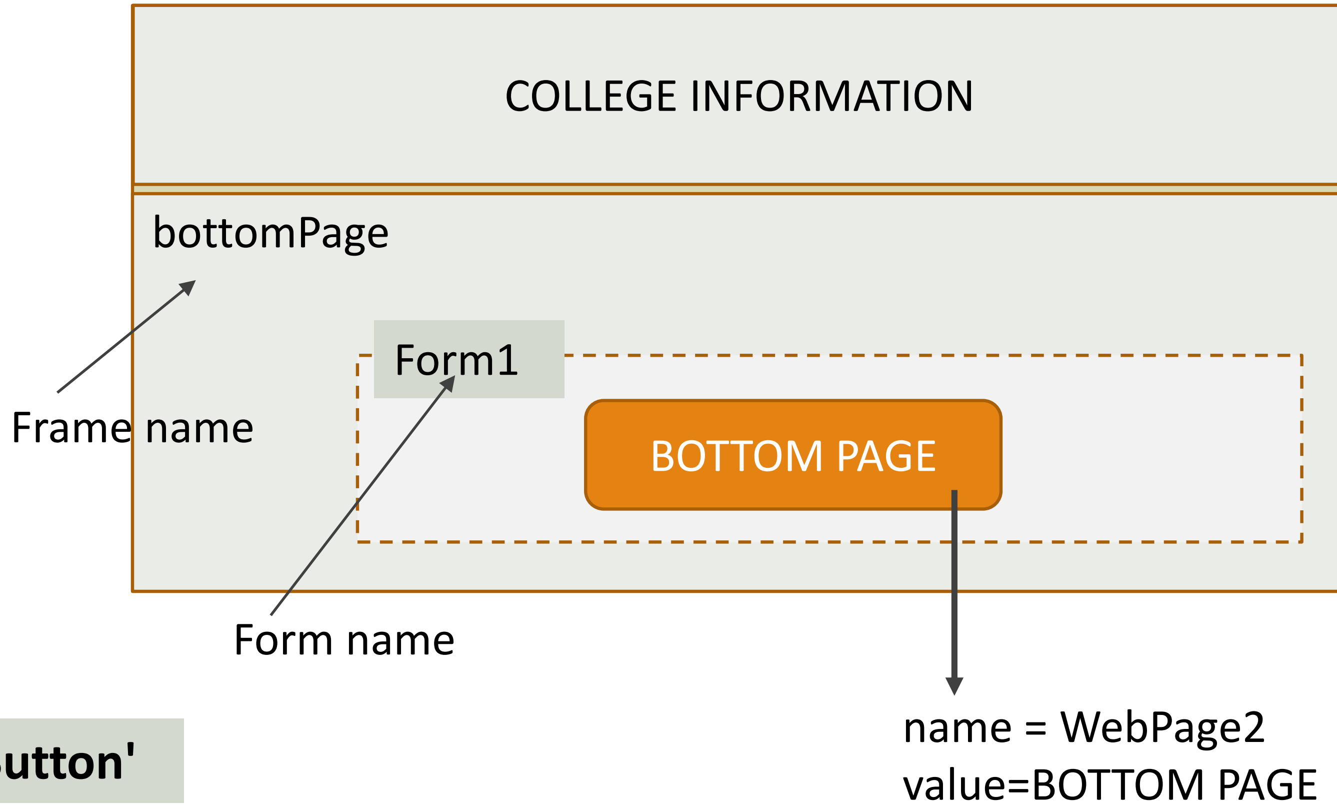
Accessing elements of another child window

We can access and change the value of elements of another child window by directly referencing the element from within your JavaScript.

Let's see how this works. Suppose that a button named WebPage2 is on Form1, located on the web page that is displayed in the ***bottomPage*** frame of the frameset.

The objective is to change the label of the ***Web Page2*** button. You'll need to specify the full path and then assign text to the ***value*** attribute of ***WebPage2***, as shown here:

```
parent.bottomPage.Form1.WebPage1.value='NewButton'
```



Rollover

→Rollover means a **webpage changes when the user moves mouse over an object** on the page.

→There are two ways to create rollover, using plain HTML or using a mixture of JavaScript and HTML.

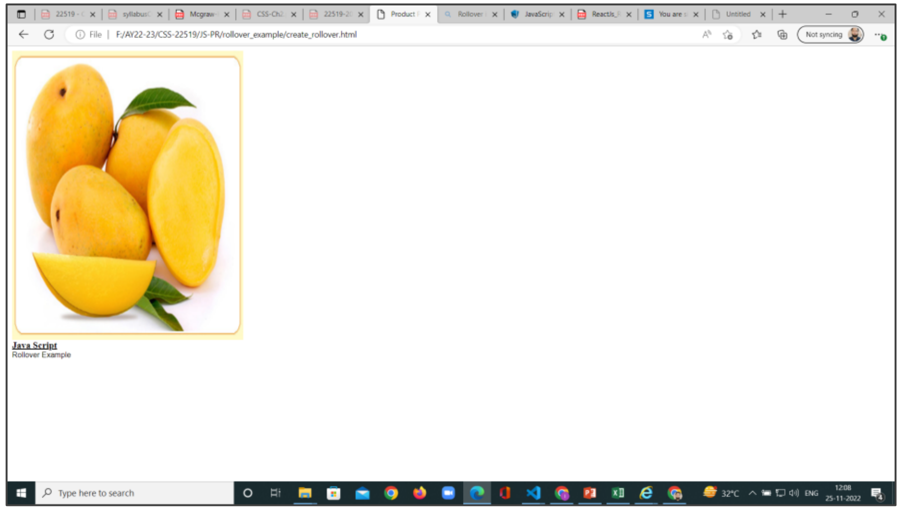
→The keyword that is used to create rollover is the ***<onmousover>*** event.

→The easiest way to determine browser compatibility is to test the *document.images* object in an if statement, Basically, the *document.images* object is not null if the browser supports rollovers.

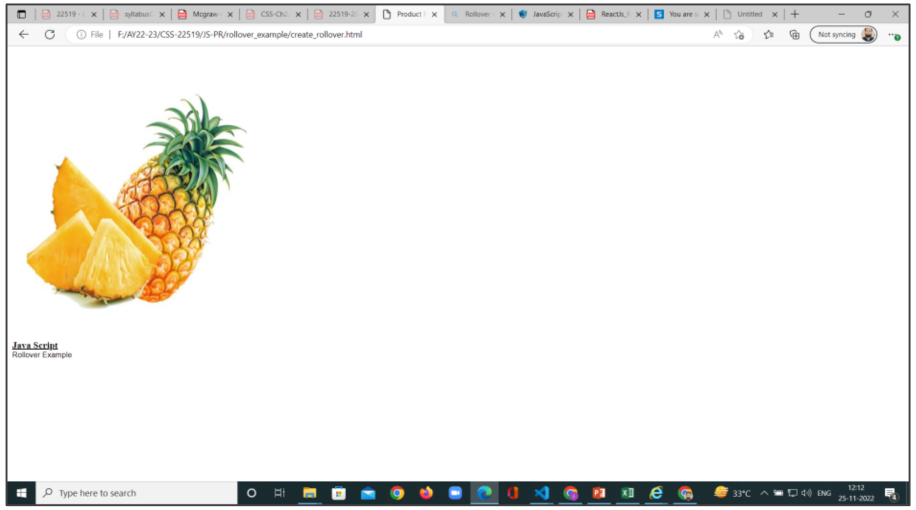
Creating Rollover

Let's say that we want to change the image on the product page whenever the visitor moves the mouse cursor over the image. The ***<img>*** tag defines the image object. The value assigned to the ***src*** attribute of the ***<img>*** tag identifies the image itself. Whenever the ***onmouseover*** event occurs, we need to change the value of the ***src*** attribute to identify the new image. Here's how this is done:

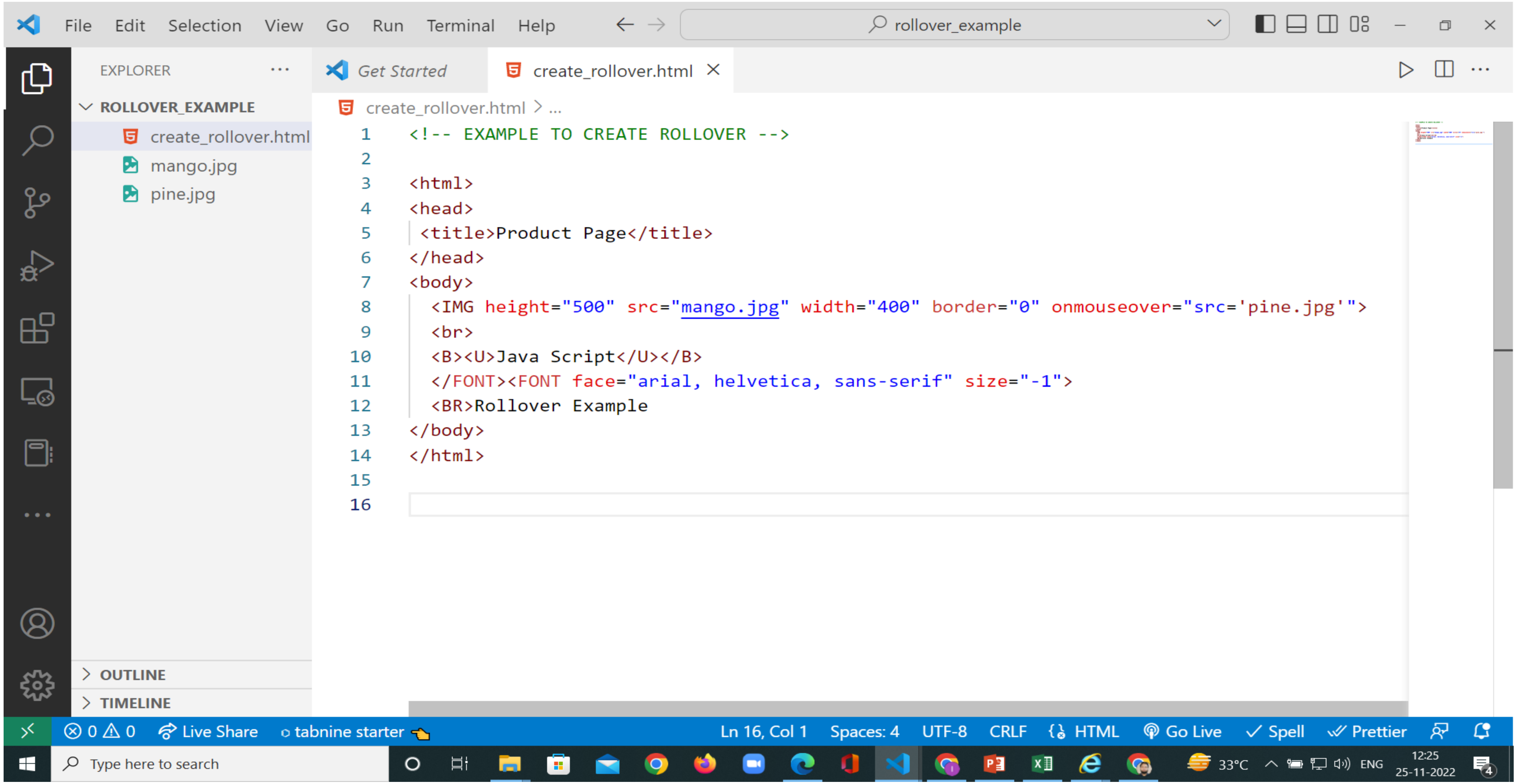
```
<img height="92" src=“mango.jpg" width="70“  border="0" onmouseover="src='pine.jpeg'">
```



Before mouseover



After mouseover



```
<!-- EXAMPLE TO CREATE ROLLOVER -->

<html>
<head>
 <title>Product Page</title>
</head>
<body>
  <IMG height="500" src="mango.jpg" width="400" border="0" onmouseover="src='pine.jpg'">
  <br>
  <B><U>Java Script</U></B>
  </FONT><FONT face="arial, helvetica, sans-serif" size="-1">
  <BR>Rollover Example
</body>
</html>

```

Text Rollover

Example 1:

In example, we create a rollover effect that can change the color of its text using the style attribute.

```
<p onmouseover="this.style.color='red'"
       onmouseout="this.style.color='blue'">
       Move the mouse over this text to change its color to red.
       Move the mouse away to change the text color to blue.
</p>
```

NOTE:

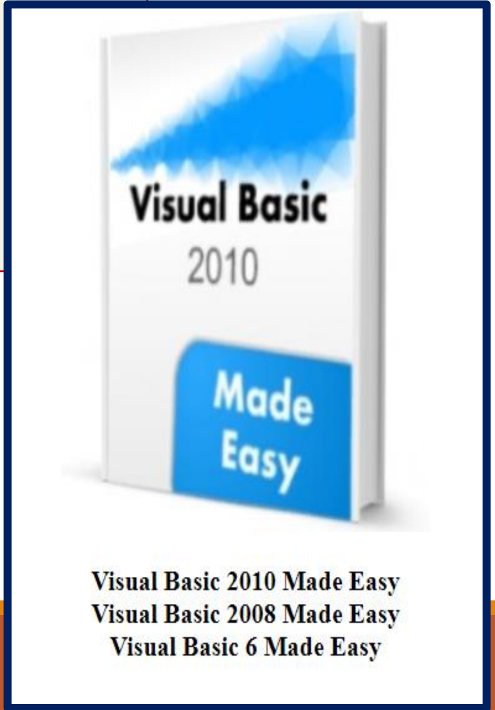
Observe the* onmouseout *event which is used to Rollback the color to blue again.

Text Rollover

Example 2:

In this example, we have create an array MyBooks to store the images of three book covers. Next, we create a ShowCover(book) to display the book cover images on the page. Finally, we call the ShowCover function using the onmouseover event.

```
<html>
<body>
<a><img src="vb2010book.jpg" name="book"></a>
<br>
<a onmouseover="document.book.src='vb2010book.jpg'"><b>Visual Basic 2010 Made Easy</b></a>
<br>
<a onmouseover="document.book.src='vb2008book.jpg'"><b>Visual Basic 2008 Made Easy</b></a>
<br>
<a onmouseover="document.book.src='vb6book.jpg'"><b>Visual Basic 6 Made Easy</b></a>
<br>
</body>
</html>
```



Visual Basic 2010 Made Easy
Visual Basic 2008 Made Easy
Visual Basic 6 Made Easy

NOTE:
Though HTML can be used to create rollovers , it can only performs simple actions. If you wish to create more powerful rollovers, you need to use JavaScript. To create rollovers in JavaScript, we need to create a JavaScript function.

Multiple Actions for a Rollover

As you probably realize, you don't need JavaScript to use rollovers with your application, because you can react to an onmouseover event by directly assigning an action to the onmouseover attribute of an HTML tag. This direct method enables you to perform one action in response to an onmouseover event. However, you may find that you want more than one action to occur in response to an onmouseover event. To do this, you'll need to create a JavaScript function that is called by the onmouseover attribute when an onmouseover event happens.

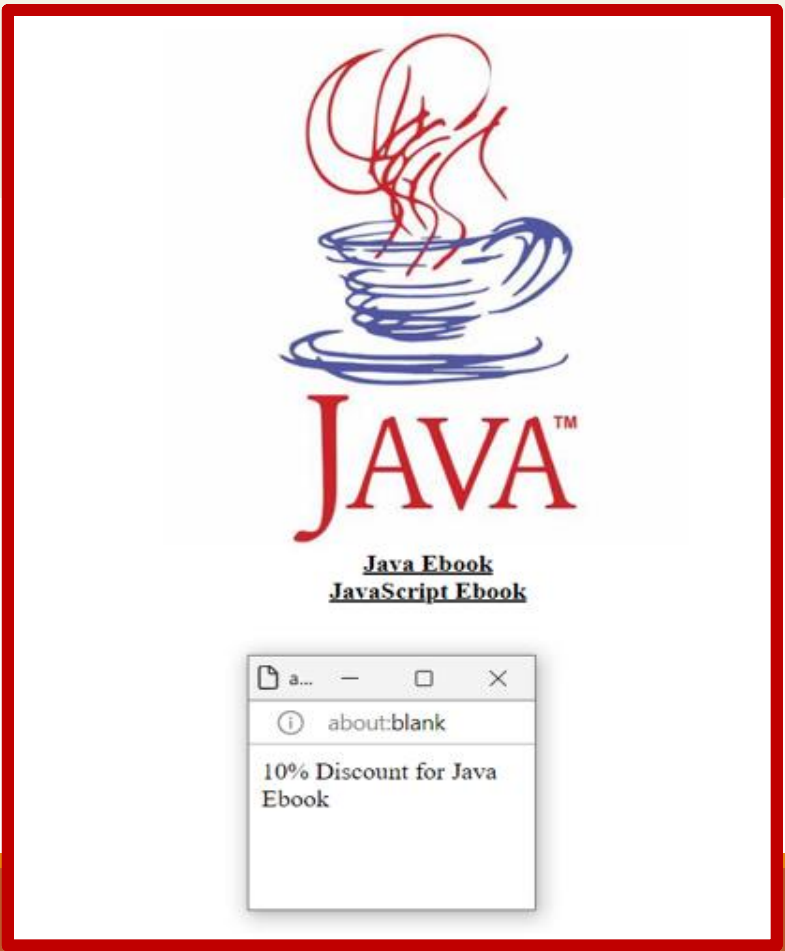
Let's suppose a visitor rolls the cursor over a book title, as in the previous example.

Instead of simply changing the image to reflect the cover of the book, you could also display an advertisement for the book in a new window, encouraging the visitor to purchase the book.

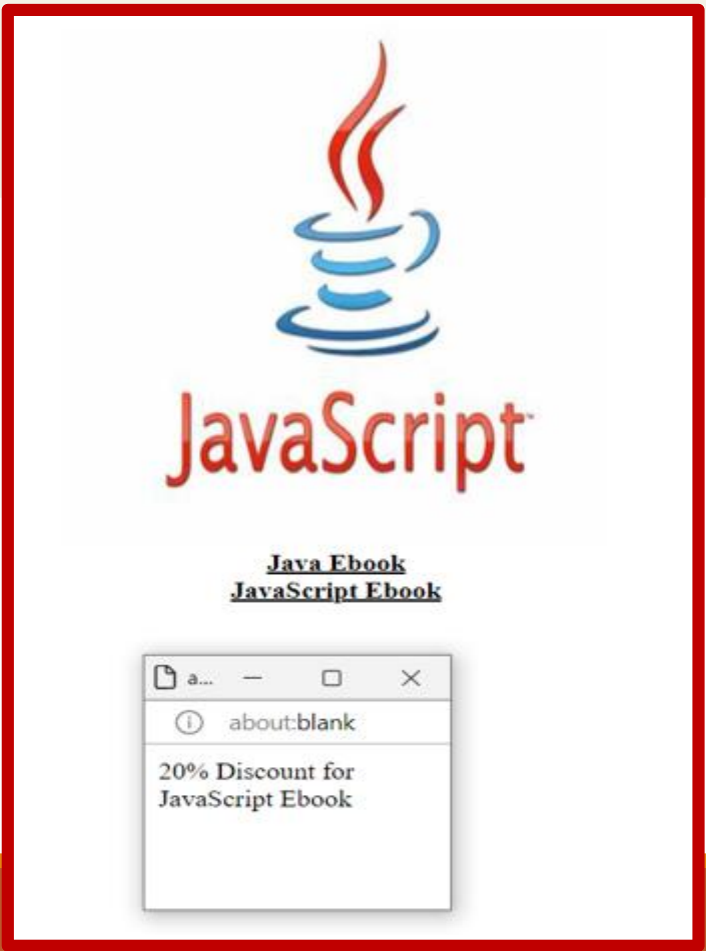
EXAMPLE for Multiple Actions for a Rollover

```
<html>
<head>
 <title>Open Window</title>
 <script>
 function OpenNewWindow(book)
       {
       if (book== 1)
              {
       document.cover.src='java.jfif'
       MyWindow = window.open('', 'myAdWin','height=50,width=150,left=650,top=500')
       MyWindow.document.write('10% Discount for Java EBook')
              }
       if (book== 2)
              {
       document.cover.src='js.jfif'
       MyWindow = window.open('', 'myAdWin','height=50,width=150,left=650,top=500')
       MyWindow.document.write('20% Discount for JavaScript Ebook')
              }
       }
  </script>
</head>
```

```
<body>
        <center>
        <a> <IMG height="350" src="java.jfif" width="320" border="0" name="cover"></a>
        <br> <a onmouseover="OpenNewWindow(1)" onmouseout="MyWindow.close()">
        <B><U>Java Ebook </U></B></a>
        <br> <A onmouseover="OpenNewWindow(2)" onmouseout="MyWindow.close()">
        <B><U>JavaScript Ebook</U></B>
        </a>
        </center>
</body>
</html>
```


Java Ebook
JavaScript Ebook

```
a...
about:blank
10% Discount for Java
Ebook
```


Java Ebook
JavaScript Ebook

```
a...
about:blank
20% Discount for
JavaScript Ebook
```

CHAPTER 10

Regular Expressions

Don't you hate it when someone enters the wrong information into a form displayed on your web page? Although you cannot prevent this from happening, you can write a JavaScript that validates information on the form before the form is processed by the CGI application running on the web server.

You learned how to use methods of the string object to validate text in Chapter 6. While this was useful for performing basic validation of a form, the string object lacks the power to perform sophisticated validation and formatting that is found in commercial JavaScript applications.

JavaScript professionals supercharge their JavaScript by using regular expressions to validate and format text. In this chapter, you'll learn how to master regular expressions and use them to manipulate information in amazing ways.

What Is a Regular Expression?

The concept of a regular expression is a little tricky to understand, but once you get the gist of it, you'll add this powerful tool into your JavaScript arsenal. You'll recall from Chapter 2 that an expression uses operators to tell the browser how to manipulate values, such as adding two numbers (10 + 5). This is called a *mathematical expression* because the values being manipulated are numbers.

A regular expression is very similar to a mathematical expression, except a regular expression tells the browser how to manipulate text rather than numbers by using special symbols as operators, which you'll learn about in this chapter.

For example, the browser might be told to determine whether a specific character exists in one or more lines of text. Likewise, the browser might be told to replace all occurrences of a word with another word. This and more can be accomplished by writing a regular expression.

Let's take a look at a simple example of how to create and use a regular expression in a JavaScript that tells the browser to determine whether the letter *b* or the letter *t* is in the name *Bob* and display an appropriate message in an alert dialog box when a button is clicked on the form.

```
<!DOCTYPE html PUBLIC
         "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Simple Regular Expression</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function RegExpression() {
            var name='Bob'
            re = /[bt]/
            if (re.test(name))
            {
                alert('Found')
             }
             else
             {
                alert('Not Found')
             }
        }
       -->
      </script>
</head>
   <body>
```

```
      <FORM action="http://www.jimkeogh.com" method="post">
         <P>
<INPUT name="Run Reg Expression" value=" Run Reg Expression "
             type="button" onclick=" RegExpression()"/>
         </P>
      </FORM>
   </body>
</html>
```

The regular expression is located in the `RegExpression()` function definition in the `<head>` tag of the web page. No doubt this looks strange to you, so let's dissect the regular expression letter by letter.

Unlike a mathematical expression, a regular expression begins and ends with a slash (/). You place the special symbols that make up the regular expressions between slashes. You'll notice that a pair of square brackets (`[]`) appears following the first slash. This tells the browser to search the text for characters that appear within the brackets. In this expression, two characters are within the square brackets: a *b* and a *t*, which tells the browser to determine whether the text includes a *b* or a *t,* or both. That's the regular expression.

The regular expression is assigned to the `re` variable. Notice that we don't use quotation marks, which would tell the browser that the special symbols of the regular expression is part of a string, which it isn't.

The `test()` method is called and passed the variable name that contains the string *Bob*. The `test()` method is one of several methods of the regular expression object. You'll learn about the other methods later in this chapter. The browser evaluates *Bob* using the regular expression. A true is returned if either a *b* or a *t* or both are found in the name *Bob*; otherwise a false is returned. Depending on this result, the appropriate alert dialog box is displayed on the screen.

The Language of a Regular Expression

Admittedly, a regular expression looks like gobbledygook to the untrained eye, but a regular expression is a complex instruction that the browser has no trouble understanding. By learning the language of a regular expression, you'll be able to make a browser jump through hoops by manipulating any text that is entered into a form on your web page.

The words of the regular expression language are called *special characters* and act similarly to an operator in a mathematical expression. An operator, as you'll recall from Chapter 2, tells the browser to perform an operation on operands, which are values. Special characters tell the browser to perform an operation on text.

Table 10-1 contains special characters that are used to create a regular expression. We'll take a closer look at a number of these to show how they are used in a regular expression. In the previous example, you saw how to ask the browser whether the text contains either the character *b* or the character *t* or both by using the following regular expression:

```
/[bt]/
```

You can place any number of characters, numbers, or punctuation or symbols within the brackets, and the browser will determine whether they exist in the text.

However, one symbol may pose a problem: suppose you want to determine whether the text contains the bracket (`[`) symbol? This can be troublesome since the `[` is a special character in a regular expression and will confuse the browser. The browser assumes the `[` is enclosing an operation to perform, so it won't search the text for the `[` character. If you want to search for a symbol that is also a special character, you must precede the symbol with a backslash (\), which is known as an *escape character*. The backslash tells the browser to ignore the special meaning of the symbol. Here's what you'd need to write to search for the `[` symbol in text:

```
/[\[]/
```

At first, this might look strange, but it should begin to make sense as you read each character the way the browser reads it. Here's how the browser reads this regular expression:

1. The / character tells the browser that this is the beginning of a regular expression.
2. The `[` character tells the browser to search the text for the following character(s).
3. The \ tells the browser to ignore the special meaning of the next character.
4. The `[` character is the character that the browser will search for in the text.
5. The `]` character tells the browser that there are no more characters to search for.
6. The / character tells the browser that this is the end of the regular expression.

TIP *Whenever a regular expression becomes confusing to understand, you can read each character in the expression the way the browser reads it and any confusion will be cleared up.*

Special Character	Description
\	Tells the browser to ignore the special meaning of the following character
^	Beginning of a string or negation operator, depending on where it appears in the regular expression
$	End of a string
*	Zero or more times
+	One or more times
?	Zero or one time; also referred to as the *optional* qualifier
.	Any character except a newline character (\n)
\b	Word boundary
\B	Nonword boundary
\d	Any digit, 0–9
\D	Any nondigit
\f	Form feed
g	Search the first and subsequent occurrences of the character(s)
i	Search without matching the case of the character
\n	Newline; also called a line feed
\r	Carriage return
\s	Any single whitespace character
\S	Any single non-whitespace character
\t	Tab
\v	Vertical tab
\w	Any letter, number, or underscore
\W	Any character other than a letter, number, or underscore
\x*nn*	The ASCII character defined by the hexadecimal number *nn*
\o>*nn*	The ASCII character defined by the octal number *nn*
\cx	The control character *x*
[abcde]	A character set that matches any one of the enclosed characters
[^abcde]	A character that does not match any of the enclosed characters
[a-e]	A character that matches any character in this range of characters; the hyphen indicates a range
[\b]	The backspace character
{*n*}	Exactly *n* occurrences of the previous subpattern or character set

Table 10-1 Special Characters Used to Create a Regular Expression

Special Character	Description
`{n,}`	At least *n* occurrences of the previous subpattern or character set
`{n,m}`	At least *n* but no more than *m* occurrences of the previous subpattern or character set
`(x)`	A grouping or subpattern, which is also stored for later use
`x|y`	Either *x* or *y*

Table 10-1 Special Characters Used to Create a Regular Expression *(continued)*

Finding Nonmatching Characters

Sometimes a JavaScript application prohibits certain characters from appearing within text entered into a form, such as a hyphen (-); otherwise, the character might inhibit processing of the form by the CGI program running on the web server. You can direct the browser to search for illegal character(s) by specifying the illegal character(s) within brackets and by placing the caret (^) as the first character in the bracket. Let's see how this works in the following example:

```
/[^\-]/
```

In this case, the browser is asked to determine whether the text *does not* contain the hyphen.

The caret asks the browser to determine whether the following character(s) *do not* appear in the text. Table 10-1 shows that the hyphen inside a character set is used to define a range of characters (also discussed in the next section). To find the hyphen in text, you need to escape the hyphen with the backslash, like so `\-`.

NOTE *It is important that you know exactly what you're telling the browser to do so that you can properly interpret the browser's response to your regular expression.*

Suppose you wrote the following regular expression and the browser didn't find the hyphen in the text. The browser responds with a false—this is because you are telling the browser to determine whether the hyphen appears in the text. If the hyphen appears, the browser would respond with a true.

```
/[\-]/
```

However, by placing a caret in the regular expression, as shown next, the browser responds with a true if the hyphen *is not* found in the text. This is because you are telling the browser to determine whether the hyphen *does not* appear in the text.

```
/[^\-]/
```

Entering a Range of Characters

You don't need to enter every character that you want the browser to match or not match in the text if those characters are in a series of characters, such as *f* through *l*. Instead of including each and every character within brackets, you can use the first and last character in the series, separated by a hyphen.

Let's say that you need to tell the browser to match any or all of the characters *f*, *g*, *h*, *i*, *j*, *k*, or *l* in the text. You could write the following regular expression:

```
/[fghijkl]/
```

Alternatively, you could write the following regular expression, which tells the browser to match any letter(s) that appears in the series *f* through and including *l*:

```
/[f-l]/
```

Likewise, you can tell the browser *not* to match any characters in a range of characters using the same kind of regular expression, except you place the caret in front of the first character, as shown here:

```
/[^f-l]/
```

In this case, the browser would return true if none of the characters *f* through *l* were found.

Matching Digits and Nondigits

Limiting an entry either to digits or nondigits is a common task for many JavaScript applications. For example, a telephone number entered by a user should be a series of digits, and a first name should be nondigits. Nondigits appearing in a phone number indicate the user entered an invalid phone number. Likewise, a first name that contains digits is likely an invalid first name.

You can have the browser check to see whether the text has digits or nondigits by writing a regular expression. The regular expression must contain either `\d` or `\D`, depending on whether you want the browser to search the text for digits (`\d`) or nondigits (`\D`).

The `\d` symbol, as shown in the following example, tells the browser to determine whether the text contains digits. The browser returns a true if at least one digit appears in the text. You'd use this regular expression to determine whether a first name has any digits, for example. If it does, the browser returns a true and your JavaScript notifies the user that an invalid first name was entered into the form.

```
/\d/
```

The `\D` symbol is used to tell the browser to search for any nondigit in the text. This is illustrated next. The browser returns a true if a nondigit is found. This is the regular expression you would use to validate a telephone number, assuming the user was asked to enter digits only. If the browser finds a nondigit, the telephone number is invalid and you can notify the user who entered the information into the form.

```
/\D/
```

NOTE *You probably noticed that the letters* d *and* D *are preceded by a backslash. The backslash tells the browser that these shouldn't be treated as characters and instead should be treated as special characters.*

Matching Punctuation and Symbols

You can have the browser determine whether text contains or doesn't contain letters, punctuation, or symbols, such as the @ sign in an e-mail address, by using the `\w` and `\W` special symbols in a regular expression.

The `\w` special symbol tells the browser to determine whether the text contains a letter, number, or an underscore, and the `\W` special symbol reverses this request by telling the browser to determine whether the text contains a character *other than* a letter, number, or underscore.

Let's say that you were expecting a person to enter the name of a product that has a combination of letters and numbers. You can use the following regular expression to determine whether the product name that was entered into the form on your web page contains a symbol:

```
/\W/
```

Using `\W` is equivalent to using `[a-zA-Z0-9_]`.

NOTE *Notice that no space (whitespace character) appears between the* `9` *and the underscore in* `[a-zA-Z0-9_]`*. A common error is to insert a space such as* `[a-z A-Z 0-9 _]`*. This matches the whitespace character, too.*

Matching Words

You might want the browser to search for a particular word within the text. A *word* is defined by a *word boundary*—that is, the space between two words. You define a word boundary within a regular expression by using the `\b` special symbol.

Think of the `\b` special symbol as a space between two words. You need to use two `\b` special symbols in a regular expression if you want the browser to search for a word: the first `\b` represents the space at the beginning of the word and the second represents the space at the end of the word.

Let's say you want to determine whether the name *Bob* appears in the text. Since you don't want the browser to match just text that contains the series of letters *B-o-b,* such as *Bobby*, you'll need to use the word boundary to define *Bob* as a word and not simply a series of letters. Here's how you'd write this regular expression:

```
/\bBob\b/
```

Note *Be sure to use the lowercase* `\b`*, because the uppercase* `\B` *signifies that there is no word boundary. Using* `\B` *means any series of the letters* B-o-b *is considered a match, including Bobby.*

Replace Text Using a Regular Expression

In this chapter, you've learned how to construct a regular expression that the browser uses to determine whether letters, numbers, or symbols appear or do not appear in text by passing the regular expression to the `test()` method. While testing text is necessary for some JavaScript applications, you can also use a regular expression to replace portions of the text by using the `replace()` method.

The `replace()` method requires two parameters: a regular expression and the replacement text. Here's how the `replace()` method works. First, you create a regular expression that identifies the portion of the text that you want replaced. Then you determine the replacement text. Pass both of these to the `replace()` method, and the browser follows the direction given in the regular expression to locate the text. If the text is found, the browser replaces it with the new text that you provided.

The next example tells the browser to replace *Bob* with *Mary* in the text. The regular expression specifies the word *Bob*. The `replace()` method of the string object is then called to use the regular expression to search for *Bob* within the text and then replace *Bob* with *Mary*.

However, the original string isn't modified. The modified string is returned by the `replace()` method. You could assign the modified string to the variable containing the original string if you don't need the original string anymore.

A common problem is to replace all occurrences of one or more characters of a string. You do this by creating a regular expression and calling the `replace()` method; however, you'll need to place the `g` special character at the end of the regular expression, which tells the browser to replace all occurrences of the regular expression in the string. This is shown here:

```
/\bBob\b/g

re = /\bBob\b/
text = 'Hello, Bob and welcome to our web site.'
text = text.replace(re, 'Mary')
```

Replacing Like Values

You've probably come across this situation: A company name is entered inconsistently in text. The first letter of the name might be capitalized sometimes, while at other times it appears in lowercase. A nickname might be used occasionally rather than the formal name.

A regular expression can be written to search for variations of a name and replace it with a standardized name. To do this, the regular expression must contain literal characters and wildcard characters. A *literal character* is a letter, number, or symbol that must match exactly within the text. A *wildcard* is a special symbol that tells the browser to accept one or multiple unspecified characters as a match.

Let's say that the text contains the words *Bob* and *Bobby* and you want to replace them with the word *Robert*. Since both *Bob* and *Bobby* have the letters *B-o-b*, it makes sense to specify *Bob* as a literal character for the browser to match. You'll then need to use a wildcard to tell the browser to match other characters that follow *Bob* in the text.

Two types of wildcards can be used: a period (`.`) and an asterisk (`*`). The period tells the browser to match any single character, while the asterisk indicates zero or more occurrences of whatever precedes it. For example, the following matches *Bob* but not *Bobby*, because a single wildcard character is used:

```
/Bob./
```

However, this regular expression matches both *Bob* and *Bobby* because the multiple character wildcard is used:

```
/Bob.*/
```

NOTE *Be careful when using wildcards, because the browser might return matches that you didn't expect when you wrote the regular expression. For example, the regular expression* `/Bob.*/` *also matches the following, and you don't want any of these to change:*

Bobbysoxer
Bobbing
Bobsled

The next example replaces *Bob* and similar spellings with *Robert*. You'll notice that two new special symbols are used in this regular expression: `g` and `i`. The `g` special symbol tells the browser to search for all occurrences of *Bob* throughout the text. Without the `g`, the browser changes only the first occurrence of *Bob*. The `i` special symbol tells the browser to ignore the case of the characters. That is, *bob* and *Bob* are both a match. If you don't use the `i` special symbol, the browser will ignore the case of the characters. That is, *bob* and *Bob* are both a match. If you don't use the `i` special symbol, the browser will match only the case that you specify in the regular expression. In this example, the browser would have matched only *Bob* if we had excluded the `i` special character.

```
re = /\b\iBob(by)?\b/g
text = 'Hello, Bob. Welcome Bobby to our web site.'
text.replace(re, 'Robert')
```

Return the Matched Characters

Sometimes your JavaScript application requires you to retrieve characters that match a regular expression rather than simply testing whether or not those characters exist in the text. You can have the browser return characters that match the pattern in your regular expression by calling the `exec()` method of the regular expression object.

Here's how to use the `exec()` method. First, create a regular expression that specifies the pattern that you want to match within the text. Characters that match this pattern will be returned to your JavaScript. Next, pass the `exec()` method the text for which you want to search. The `exec()` method returns an array. The first array element contains the text that matches your regular expression.

For example, suppose you want to return a person's first name. You know the name is *Bob* or some variation of it, such as *Bobby*, but you are unsure of how the name appears in the text. As you've seen previously in this chapter, the following regular expression matches *Bob* and any word that begins with *B-o-b*.

```
/\bBob.*\b/
```

We'll need to do the following:

1. Create the regular expression object and assign it the regular expression:

   ```
   re = /\bBob.*\b /
   ```

2. Call the `exec()` method, passing it the text and assigning the return value to an array variable. Remember that you can pass a reference to the text instead of the entire text, as shown here:

   ```
   re = /\bBob.*\b /
   MyArray = re.exec('Hello, my name is Bobby.')
   ```

3. We then display the value of the first array element:

   ```
   re = /\bBob.*\b /
   MyArray = re.exec('Hello, my name is Bobby.')
   alert('Welcome, ' + MyArray[0])
   ```

The Telephone Number Match

Validating a telephone number is a common task faced by JavaScript developers. The next example shows how you can use a regular expression to do this. Let's begin by creating the string that contains the telephone number. In a real JavaScript application, the telephone number is entered into a field on a form. You use the value attribute of the field to access the telephone number.

```
phone = '212-555-1212'
```

Next, create the regular expression, as shown here:

```
re = /^[\(]?(\d{3})[\)]?[ -\.]?(\d{3})[ -\.]?(\d{4})$/
```

No doubt the regular expression looks a little confusing, so let's break it down into parts to help you understand what is happening here:

/	Start a regular expression.
^	Start at the beginning of the string.
[\(]	Match the open parenthesis.
?(\d{3})	Match any digit, 0–9, exactly three occurrences. The parentheses tell the browser to store this as a subpattern and will be assigned to an element of the array that is returned by the `exec()` method.
[\)]	Match the close parenthesis.
?[-\.]	Match a hyphen.
?(\d{4})	Match any digit 0–9 exactly four occurrences. The parentheses tell the browser to store this as a subpattern and will be assigned to an element of the array that is returned by the `exec()` method.
$	Match the end of the string.
/	The end of the regular expression.

Now that we've built the regular expression, let's use it in the following JavaScript:

```
if(re.test(phone))
{
   MyArray = re.exec(phone)
   alert('Area code: ' + MyArray[1] + '\nExchange: ' +
           MyArray[2] + '\nNumber: ' + MyArray[3])
}
```

Before validating the telephone number, we must be sure that the *(phone)* string exists by passing the string *(phone)* to the `test()` method. If the string isn't empty (NULL), then the `test()` method returns a true and statements within the if statement are executed; otherwise, we don't need to validate the telephone number.

The string containing the telephone number is passed to the `exec()` method, where the regular expression is applied to the string. The `exec()` method returns an array. The first element of the array is the entire string that matches the regular expression. Subsequent elements of the array contain substrings of the string that match groups defined in the regular expression.

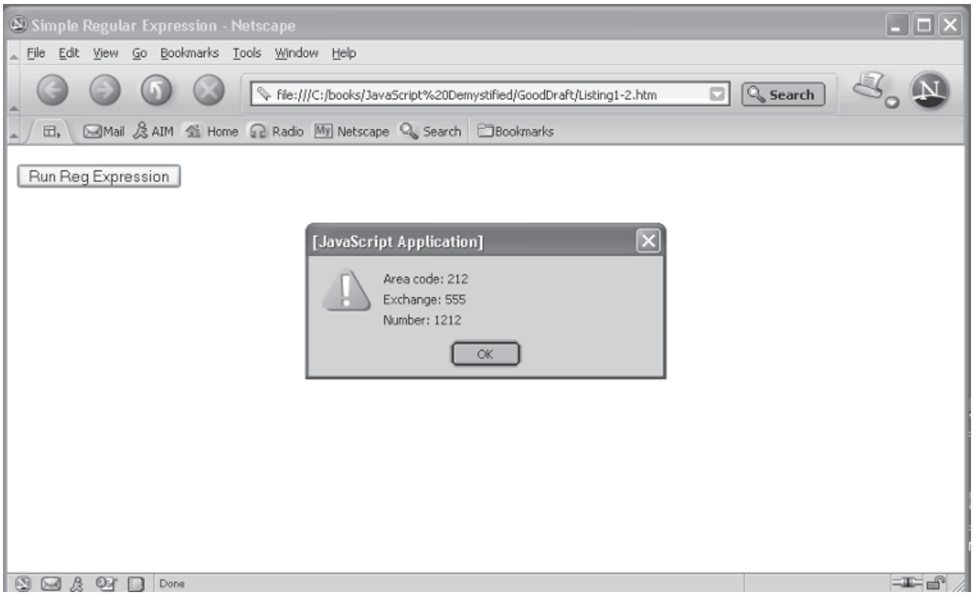
Three groups are defined in our regular expression, and each are contained within parentheses: the first group is the area code, the second group is the three-digit exchange, and the third group is the last four digits of the telephone number. The substring that matches each one of these groups is automatically assigned to the second and subsequent elements of the array in the order in which the groups are defined in the regular expression.

This means that `MyArray[1]` is assigned the substring containing the area code (that is, the first group defined in the regular expression). `MyArray[2]`is assigned the substring containing the exchange, and `MyArray[3]` is assigned the substring containing the last four digits of the telephone number.

Once you have isolated each substring of the telephone number string, you can continue the validation process to make sure that the telephone number is correct. Steps in this process depend on the nature of your JavaScript application.

The next example shows the complete JavaScript. This JavaScript separates the telephone number into area code, exchange, and number and displays each separately, regardless of the format characters used in the string (Figure 10-1). The same results are displayed even if you entered the following forms of the telephone number in the string:

2125551212
(212) 555-1212
212-555-1212
212.555.1212
(201)555-1212
212555-1212


Simple Regular Expression - Netscape
File Edit View Go Bookmarks Tools Window Help
file:///C:/books/JavaScript%20Demystified/GoodDraft/Listing1-2.htm
Search
Mail AIM Home Radio Netscape Search Bookmarks
Run Reg Expression
[JavaScript Application]
OK
Done

```
Area code: 212
Exchange: 555
Number: 1212
```

Figure 10-1 This regular expression extracts components of the telephone number regardless of how the telephone number is formatted.

```
<!DOCTYPE html PUBLIC
         "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Simple Regular Expression</title>
      <script language="Javascript" type="text/javascript">
         <!--
         function RegExpression() {
            phone = '212-555-1212'
            re = /^[\(]?(\d{3})[\)]?[ -\.]?(\d{3})[ -\.]
                       ?(\d{4})$/
            if(re.test(phone))
            {
               MyArray = re.exec(phone)
               alert('Area code: ' + MyArray[1] + '\nExchange: ' +
                     MyArray[2] + '\nNumber: ' + MyArray[3])
            }
        }
        -->
      </script>
</head>
   <body>
      <FORM action="http://www.jimkeogh.com" method="post">
         <P>
<INPUT name="Run Reg Expression" value=" Run Reg Expression "
            type="button" onclick=" RegExpression()"/>
         </P>
     </FORM>
   </body>
</html>
```

Regular Expression Object Properties

In addition to methods, the regular expression object has properties that you can access from within your JavaScript by referencing the name of the regular expression object followed by the property name. This is the same technique that you used to access properties in previous chapters. Table 10-2 lists these properties.

For example, let's say that you want to access the last characters that were matched by the regular expression. As you'll notice in Table 10-2, the `lastMatch` property contains the last characters that were matched by the regular expression object. You reference this by using the following expression:

```
re.lastMatch
```

Regular Expression Object	Properties
`$1 (through $9)`	Parenthesized substring matches
`$_`	Same an `input`
`$*`	Same as `multiline`
`$&`	Same as `lastMatch`
`$+`	Same as `lastParen`
`` $` ``	Same as `leftContent`
`$'`	Same as `rightContext`
`constructor`	Specifies the function that creates an object's prototype
`global`	Search globally (`g` modifier in use)
`ignoreCase`	Search case-insensitive (`i` modifier in use)
`input`	The string to search if no string is passed
`lastIndex`	The index at which to start the next match
`lastMatch`	The last match characters
`lastParen`	The last parenthesized substring match
`leftContext`	The substring to the left of the most recent match
`multiline`	Whether strings are searched across multiple lines
`prototype`	Allows the addition of properties to all objects
`rightContext`	The substring to the right of the most recent match
`source`	The regular expression pattern itself

Table 10-2 Properties of the Regular Expression Object

Looking Ahead

A regular expression is similar to a mathematical expression in that both contain operators that tell the browser how to manipulate values. A mathematical expression instructs the browser how to manipulate numbers. A regular expression directs the browser to manipulate text by using special characters as operators.

A regular expression can handle practically all your needs for manipulating text. You can use a regular expression to search text, extract text, replace text, and to format text.

JavaScript has a regular expression object that can be assigned a regular expression. The regular expression object has methods and properties, as do other