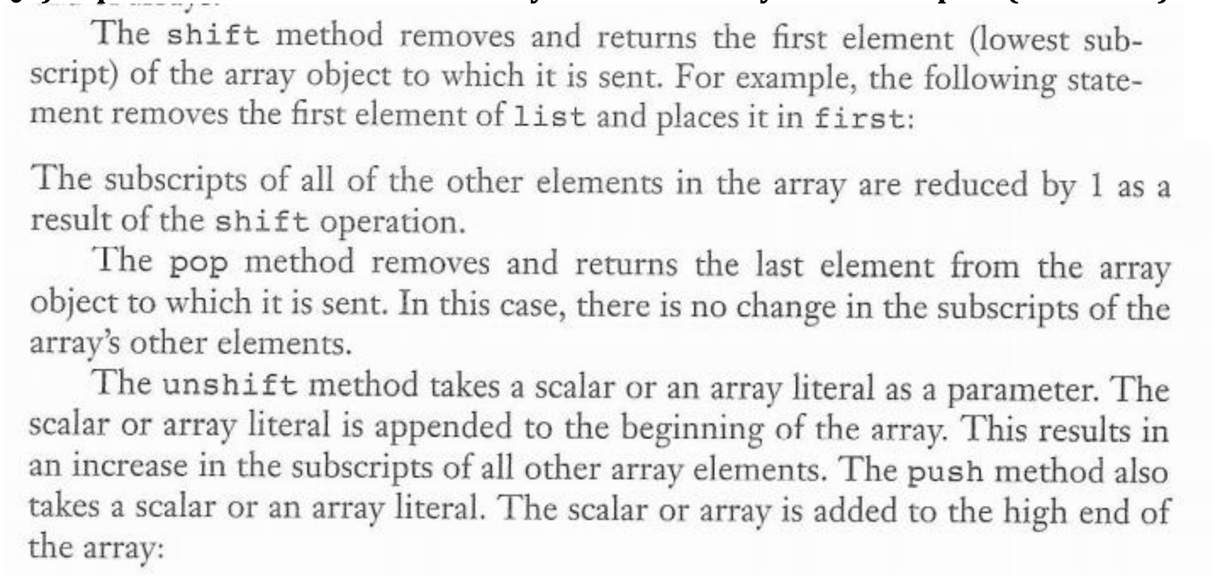
CMR
INSTITUTE OF
TECHNOLOGY

USN

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 2 Answer Key– May. 2021									
Sub:	Advanced Web Programming				Sub Code:	18MCA42	Branch:	MCA	
Date:	19/05/2021	Duration:	90 min's	Max Marks:	50	Sem	IV		

Q1) Explain built-in methods of array and list in ruby with examples (10 marks)



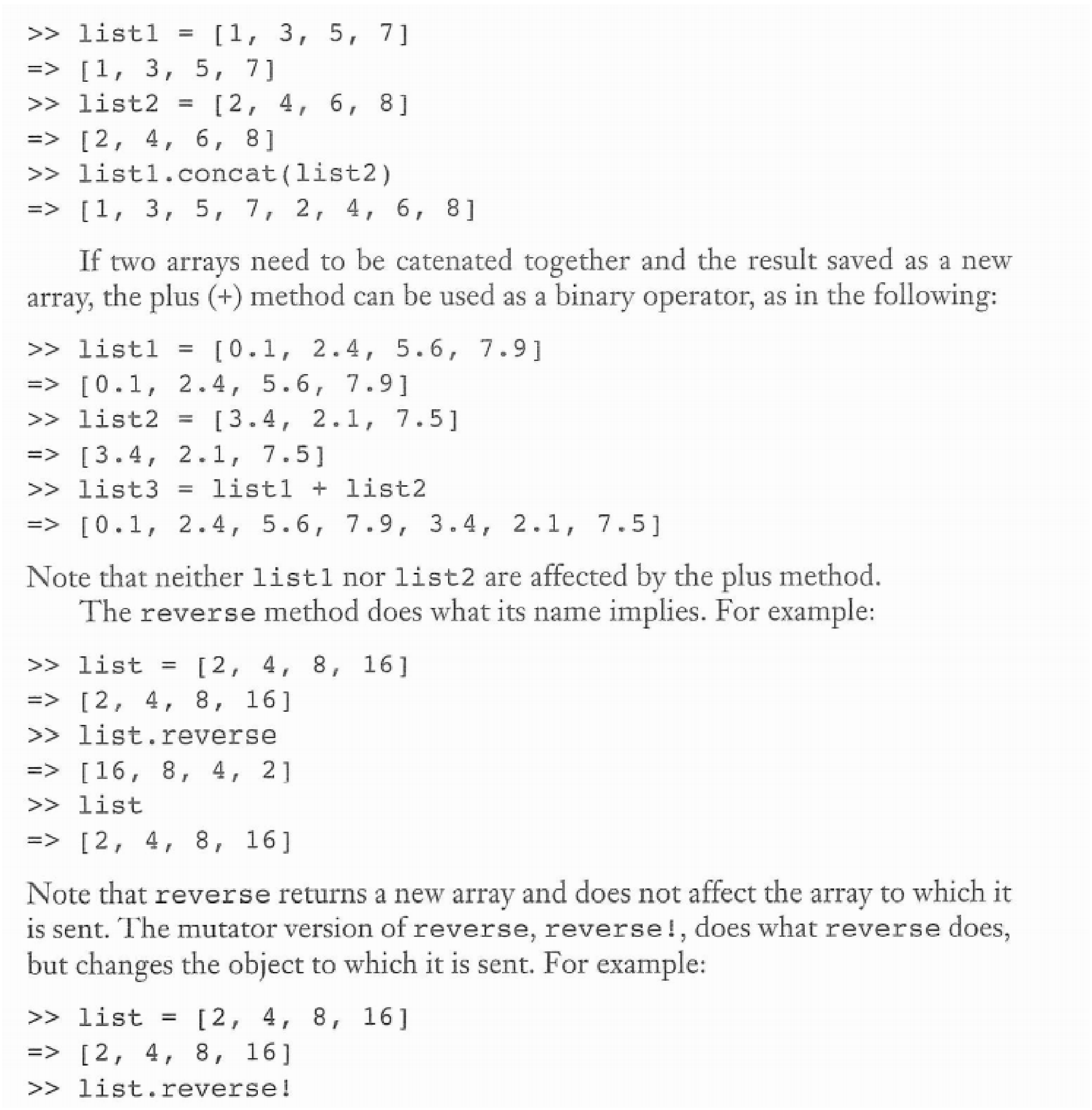
The shift method removes and returns the first element (lowest subscript) of the array object to which it is sent. For example, the following statement removes the first element of list and places it in first:

The subscripts of all of the other elements in the array are reduced by 1 as a result of the shift operation.

The pop method removes and returns the last element from the array object to which it is sent. In this case, there is no change in the subscripts of the array's other elements.

The unshift method takes a scalar or an array literal as a parameter. The scalar or array literal is appended to the beginning of the array. This results in an increase in the subscripts of all other array elements. The push method also takes a scalar or an array literal. The scalar or array is added to the high end of the array:

```
list=[2, 4, 17, 3]

list.shift
o/p   [4, 17, 3]

list.pop
o/p   [2, 4, 17]

list.unshift(8)
o/p   [8, 4, 17, 3]

list.push(8,5)
o/p   [2, 4, 17, 8,5]
```



```
>> list1 = [1, 3, 5, 7]
=> [1, 3, 5, 7]
>> list2 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
>> list1.concat(list2)
=> [1, 3, 5, 7, 2, 4, 6, 8]
```

If two arrays need to be catenated together and the result saved as a new array, the plus (+) method can be used as a binary operator, as in the following:

```
>> list1 = [0.1, 2.4, 5.6, 7.9]
=> [0.1, 2.4, 5.6, 7.9]
>> list2 = [3.4, 2.1, 7.5]
=> [3.4, 2.1, 7.5]
>> list3 = list1 + list2
=> [0.1, 2.4, 5.6, 7.9, 3.4, 2.1, 7.5]
```

Note that neither `list1` nor `list2` are affected by the plus method.

The `reverse` method does what its name implies. For example:

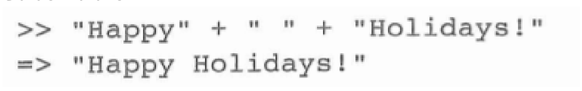
```
>> list = [2, 4, 8, 16]
=> [2, 4, 8, 16]
>> list.reverse
=> [16, 8, 4, 2]
>> list
=> [2, 4, 8, 16]
```

Note that `reverse` returns a new array and does not affect the array to which it is sent. The mutator version of `reverse`, `reverse!`, does what `reverse` does, but changes the object to which it is sent. For example:

```
>> list = [2, 4, 8, 16]
=> [2, 4, 8, 16]
>> list.reverse!
```

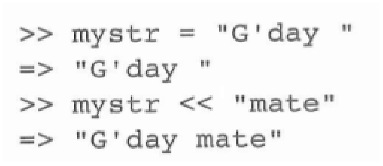
Q2) Explain built-in methods of string in ruby with examples (10 marks)

Catenation



```
>> "Happy" + " " + "Holidays!"
=> "Happy Holidays!"
```

Append



```
>> mystr = "G'day "
=> "G'day "
>> mystr << "mate"
=> "G'day mate"
```

mystr += “mate”

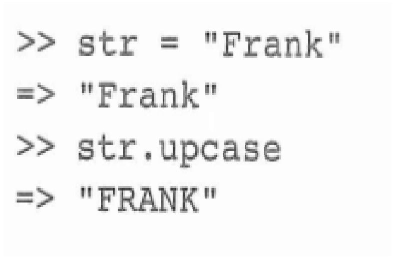
```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> yourstr
=> "Wow!"
```

```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> mystr = "What?"
=> "What?"
>> yourstr
=> "Wow!"
```

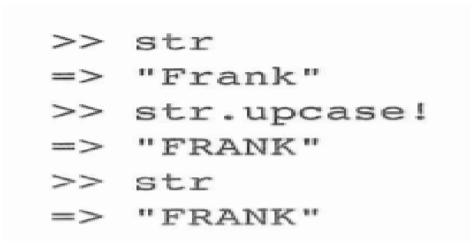
```
>> mystr = "Wow!"
=> "Wow!"
>> yourstr = mystr
=> "Wow!"
>> mystr.replace("Golly!")
=> "Golly!"
>> mystr
=> "Golly!"
>> yourstr
=> "Golly!"
```

Method	Action
`capitalize`	Convert the first letter to uppercase and the rest of the letters to lowercase
`chop`	Removes the last character
`chomp`	Removes a newline from the right end, if there is one
`upcase`	Converts all of the lowercase letters in the object to uppercase
`downcase`	Converts all of the uppercase letters in the object to lowercase
`strip`	Removes the spaces on both ends
`lstrip`	Removes the spaces on the left end
`rstrip`	Removes the spaces on the right end
`reverse`	Reverses the characters of the string
`swapcase`	Convert all uppercase letters to lowercase and all lowercase letters to uppercase

Bang or mutator methods

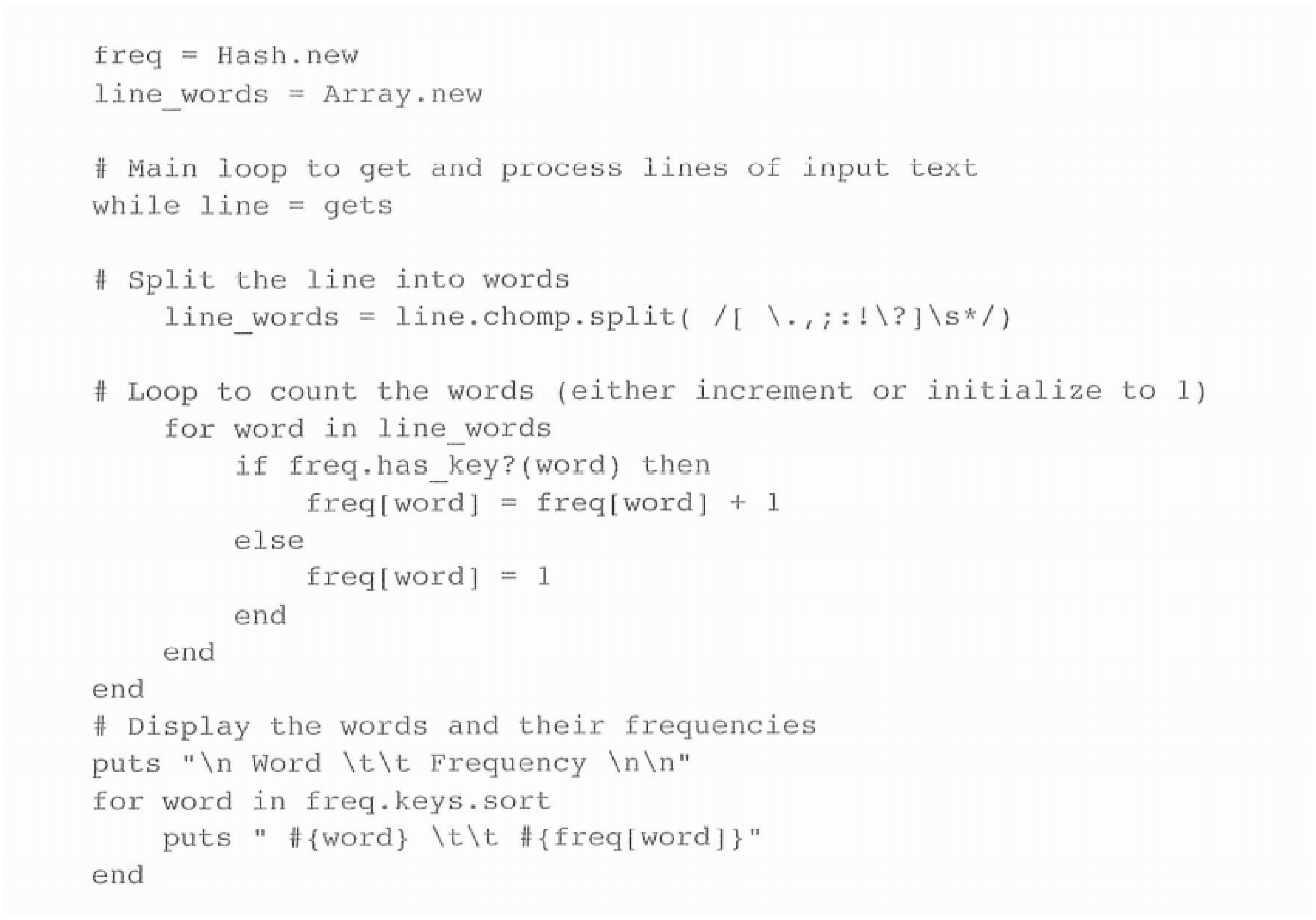


```
>> str = "Frank"
=> "Frank"
>> str.upcase
=> "FRANK"
```



```
>> str
=> "Frank"
>> str.upcase!
=> "FRANK"
>> str
=> "FRANK"
```

Q3) Write a ruby program to count words and their frequencies in the given string (10 marks)



```
freq = Hash.new
line_words = Array.new

# Main loop to get and process lines of input text
while line = gets

# Split the line into words
    line_words = line.chomp.split( /[ \.,;:!\?]\s*/)

# Loop to count the words (either increment or initialize to 1)
    for word in line_words
        if freq.has_key?(word) then
            freq[word] = freq[word] + 1
        else
            freq[word] = 1
        end
    end
end
# Display the words and their frequencies
puts "\n Word \t\t Frequency \n\n"
for word in freq.keys.sort
    puts " #{word} \t\t #{freq[word]}"
end
```

Q4) Give an example how dynamic documents are generated in ruby on rails (10 marks)

As an example of a dynamic document, we construct a new application that gives a greeting, but also displays the current date and time, including the number of seconds since midnight (just so some computation would be included). This application is named `rails2` and the controller is named `time`. This application will illustrate how Ruby code that is embedded in a template file can accesse instance variables that are created and assigned values in the action method of the controller.

Ruby code is embedded in a template file by placing it between the `<%` and `%>` markers. If the Ruby code produces a result and the result is to be inserted into the template document, an equal sign (=) is attached to the opening marker. For example:

```
<p> The number of seconds in a day is: <%= 60 * 60 * 24 %>
</p>
```

After interpretation, this is as follows:

```
<p> The number of seconds in a day is: 86400 </p>
```

The date can be obtained by calling Ruby's `Time.now` method. This method returns the current day of the week, month, day of the month, time, time zone,[2] and year, as a string. So, we can put the date in the response template with:

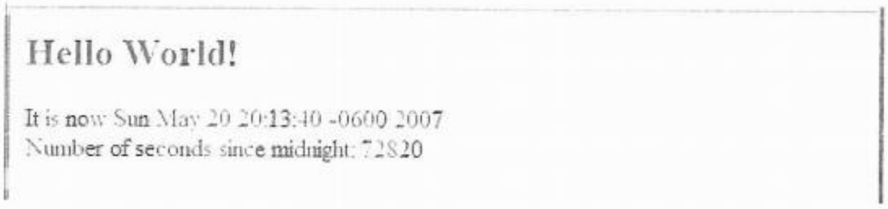
```
<p> It is now <%= Time.now %> </p>
```

The value returned by `Time.now` can be parsed with the methods of the `Time` class. For example, the `hour` method returns the hour of the day, the `min` method returns the minutes of the hour, and the `sec` method returns the seconds of the minute. These methods can be used to compute the number of seconds since midnight. Putting these together results in the following template file:

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- timer.rhtml - Response document for rails2 -
     Hello World + time
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> rails2 example </title>
  </head>
  <body>
    <h2> Hello World! </h2>
    <p>
      It is now <%= t = Time.now %> <br />
      Number of seconds since midnight:
      <%= t.hour * 3600 + t.min * 60 + t.sec %>
    </p>
  </body>
</html>
```

In this case, the template file resides in the `time` subdirectory (`time` is the name of the controller of this application) of the `views` subdirectory of the `app` subdirectory of the `rails2` directory.

Figure 15.5 shows the display of the `rails2` application.

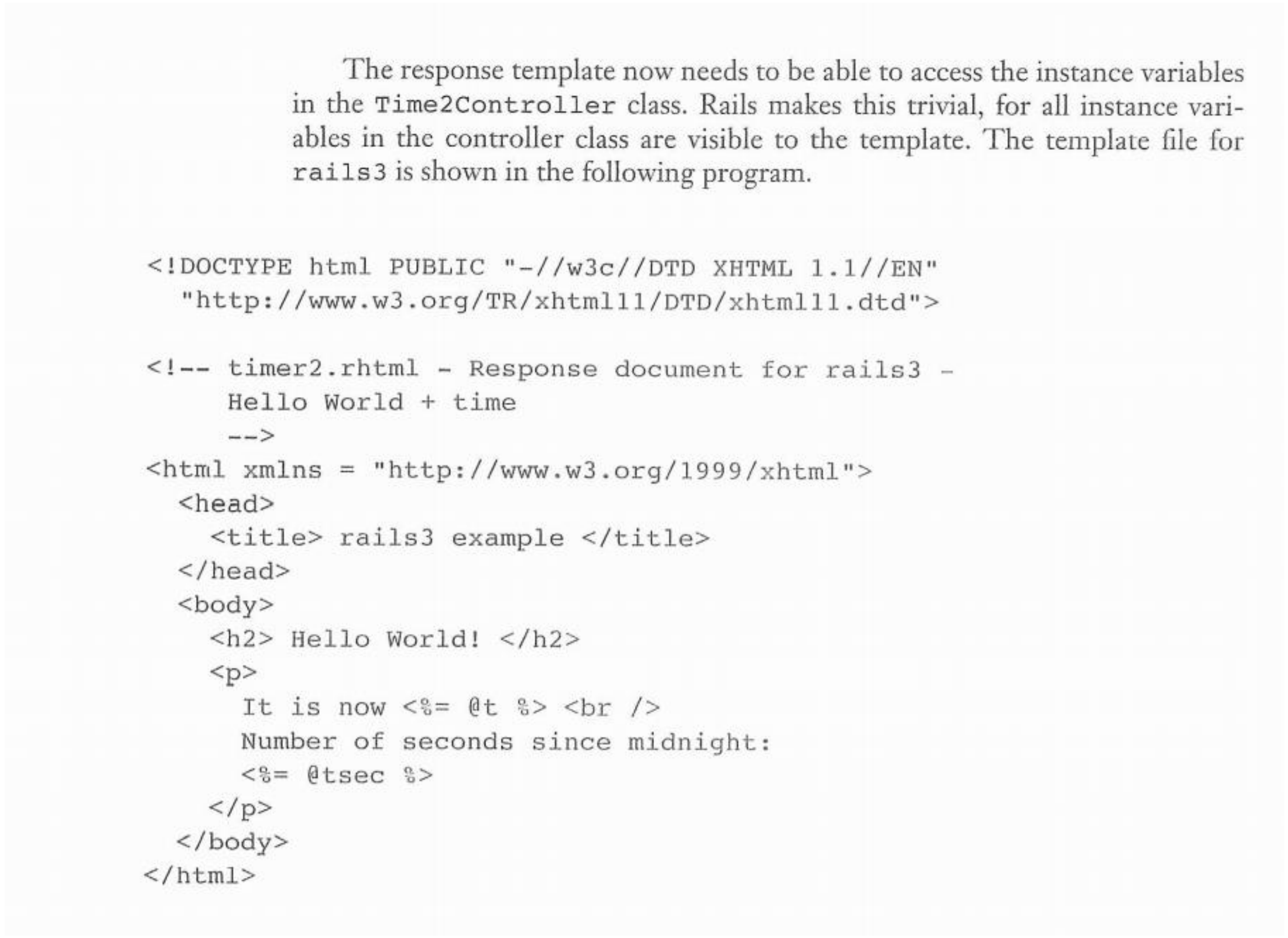


```
Hello World!

It is now Sun May 20 20:13:40 -0600 2007
Number of seconds since midnight: 72820
```

Figure 15.5 The output of `rails2 (time/timer)`

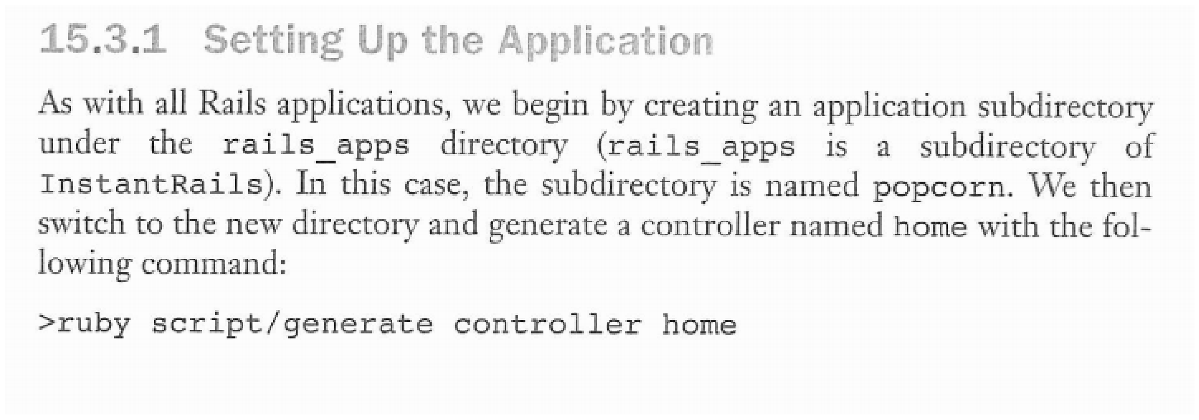
It would be better to place the Ruby code of `rails2` in the controller, because that would separate the program code from the markup. In this case, it does not amount to much code, but we will show it as a new application named `rails3`, with the controller named `time2` with an action method named `timer2`, just to illustrate how it would appear. The controller class would be as follows:

```
class Time2Controller < ApplicationController
  def timer2
    @t = Time.now
    @tsec = @t.hour * 3600 + @t.min * 60 + @t.sec
  end
end
```


The response template now needs to be able to access the instance variables in the `Time2Controller` class. Rails makes this trivial, for all instance variables in the controller class are visible to the template. The template file for `rails3` is shown in the following program.

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- timer2.rhtml - Response document for rails3 -
     Hello World + time
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> rails3 example </title>
  </head>
  <body>
    <h2> Hello World! </h2>
    <p>
      It is now <%= @t %> <br />
      Number of seconds since midnight:
      <%= @tsec %>
    </p>
  </body>
</html>
```

Q5) Explain form handling in rails. (10 marks)


15.3.1 Setting Up the Application

As with all Rails applications, we begin by creating an application subdirectory under the `rails_apps` directory (`rails_apps` is a subdirectory of `InstantRails`). In this case, the subdirectory is named popcorn. We then switch to the new directory and generate a controller named home with the following command:

```
>ruby script/generate controller home
```

We then open that controller, whose file is named `home_controller.rb`. (It resides in the `controllers` directory.) We add an empty action method named `the_form` to this class.

The next step in developing this application is to add the `the_form.rhmtl` template to the `home` subdirectory of the `views` directory of our application. The contents of `the_form.rhtml` is exactly the same as the popcorn HTML files in Chapters 2 and 11, except that the opening form tag appears as follows:

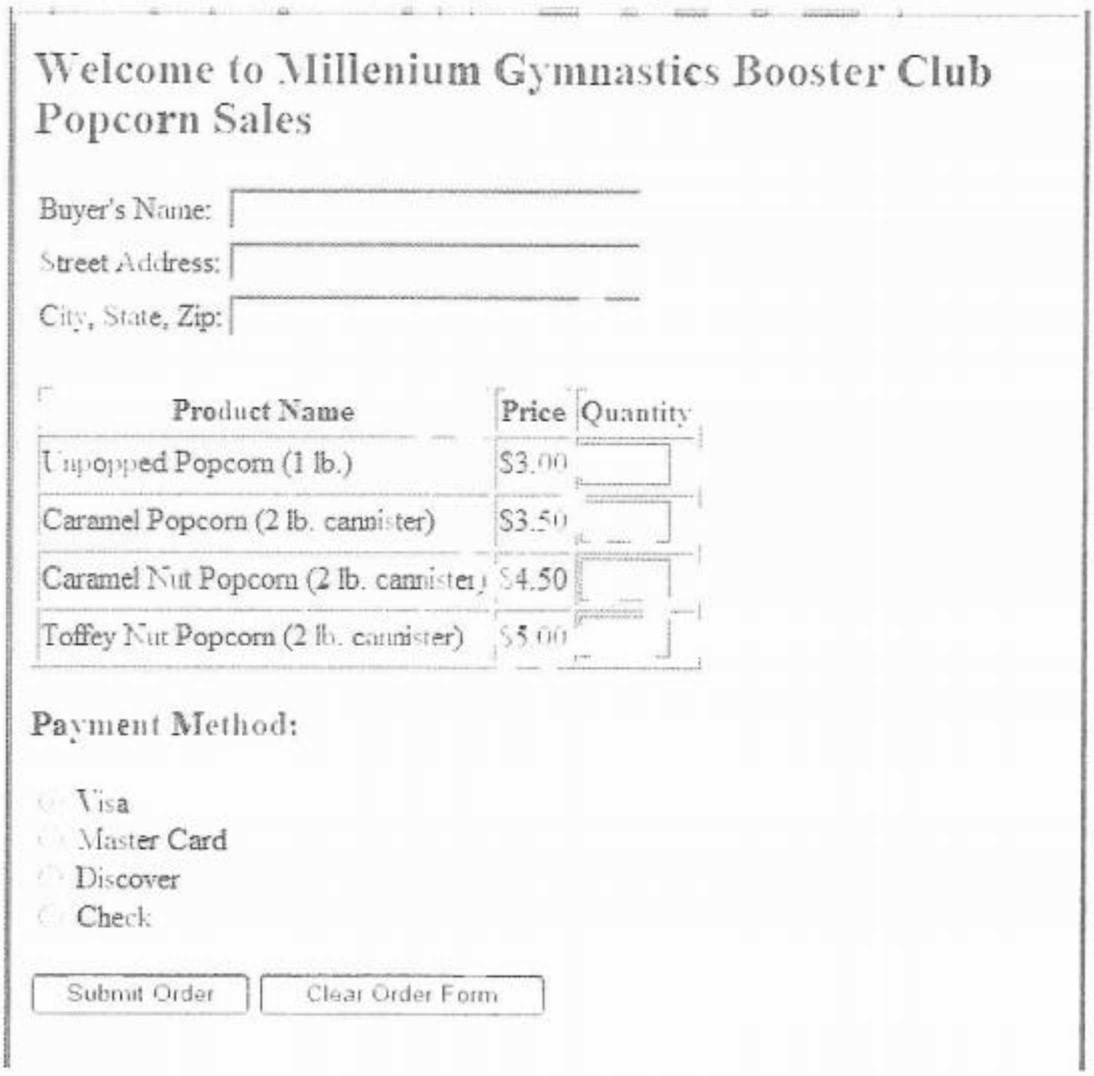
```
<form action = "result"  method = "post">
```

This specifies that the name of the action method in the application's controller, as well as the template for the result of submitting the form, is `result`. Notice that this tag specifies the POST HTTP method. Rails requires that POST be used.

Now, we point our browser to the following:

```
http://localhost/home/the_form
```

The resulting display is shown in Figure 15.6.



Welcome to Millenium Gymnastics Booster Club Popcorn Sales

Buyer's Name:
Street Address:
City, State, Zip:

Product Name	Price	Quantity
Unpopped Popcorn (1 lb.)	$3.00	
Caramel Popcorn (2 lb. cannister)	$3.50	
Caramel Nut Popcorn (2 lb. cannister)	$4.50	
Toffey Nut Popcorn (2 lb. cannister)	$5.00	

Payment Method:

- Visa
- Master Card
- Discover
- Check

Figure 15.6 The popcorn application initial display

15.3.2 The Controller and the View

The next step of the construction of the application is to build the action method in `home_controller.rb` to process the form data when the form is submitted. In the initial template file, `the_form.rhtml`, this method is named `result` in the `action` attribute of the form tag.The `result` method has two tasks, the first of which is to fetch the form data. This data is used to display back to the customer and to compute the results. The form data is made available to the controller class through the Rails-defined object, `params`. `params` is a hash-like object that contains all of the form data (as well as some other things). It is hash-like because it is a hash that can be indexed with either symbols or actual keys (a hash can be indexed only with keys). The common Rails convention is to index `params` with symbols. For example, to fetch the value of the form element whose name is `phone`, we would use the following:

```
@phone = params[:phone]
```

Of course, all form data is in string form. However, some of the values are integer numeric quantities, so they must be converted to integers with the `to_i` method of `String`. The form of the statements to fetch the form data is illustrated by the following statement:

```
@unpop = params[:unpop].to_i
```

Notice that the instance variable has the same name as the form element. In this case, the value is a quantity, which is converted to an integer. The other quantities on the form are those for the form elements named `caramel`, `caramelnut`, and `toffeynut`. In addition, the string values for `name`, `street`, `city`, and `payment` must be fetched.

The computations for the application are for the cost of each variety of popcorn, the total number of items ordered, and the total cost of the order. The unit prices are as follows:

unpopped corn	$3.00
caramel corn	$3.50
caramel nut corn	$4.50
toffey nut corn	$5.00

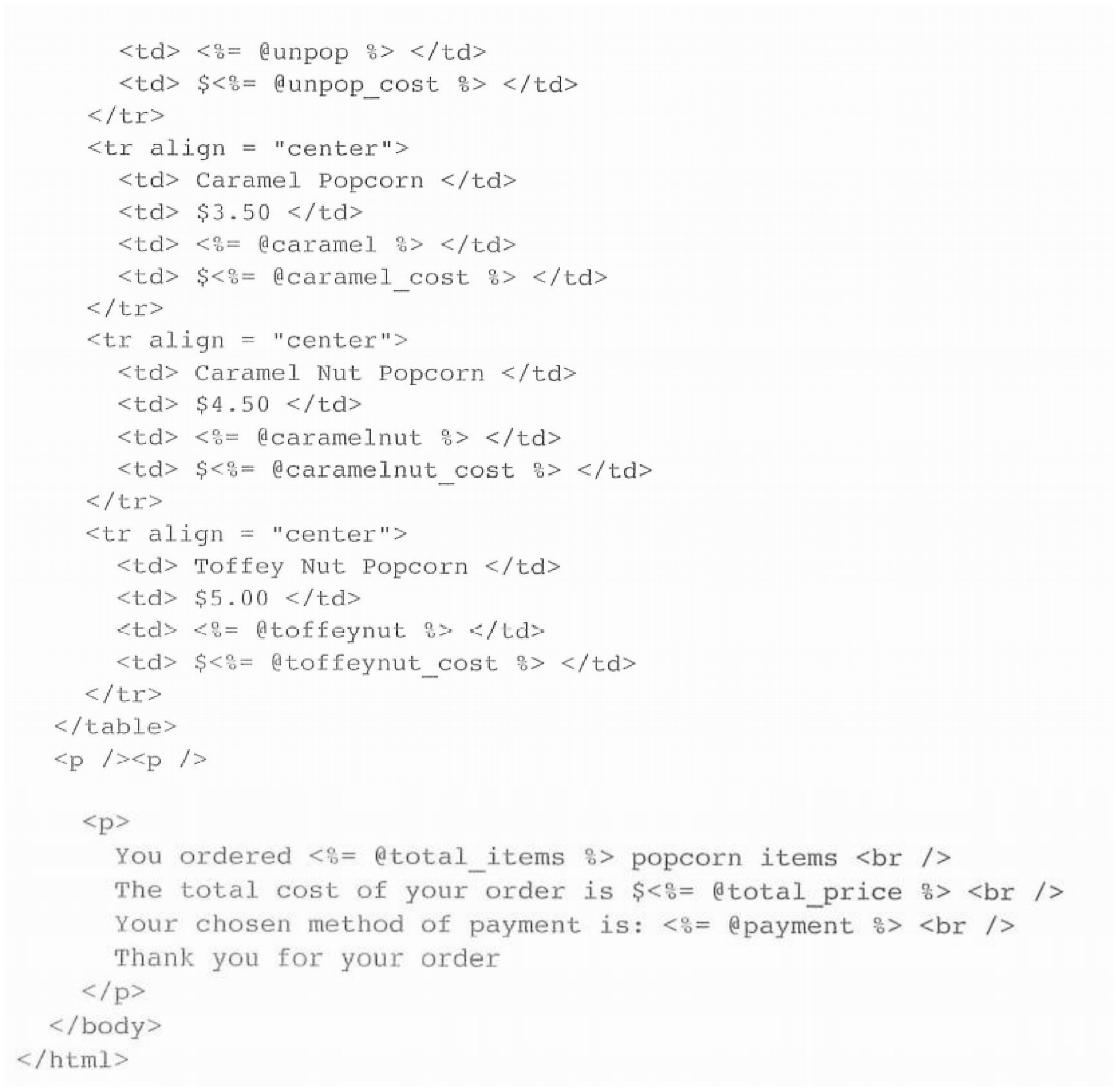
The computations are relatively simple. The only complication is that people prefer that when money amounts are displayed, there should be exactly two digits displayed to the right of the decimal point. Formatted numbers such as these can be created as strings in the controller for display in the template. The way to convert a floating-point value to a formatted string is with a variation of the old C language function `sprintf`. This function, which also is named `sprintf`, takes a string parameter that contains a format code, followed by the name of a variable to be converted. The string version is returned by the function. The format codes most commonly used are `f` and `d`. The form of a format code is a percent sign (`%`), followed by a field width, followed by the code letter (`f` or `d`). The field width for the `f` code appears in two parts, separated by a decimal point. For example, `%f7.2` means a total field width of `7` spaces, with `2` digits to the right of the decimal point, which is perfect for money. The `d` code field width is just a number of spaces, for example, `%5d`. So, to convert a floating-point value referenced by the variable `@total` to a string with two digits to the right of the decimal point, the following could be used:

```
@str = sprintf("%5.2f", @total)
```

The `sprintf` function is used in the controller for our `popcorn` application, which is shown in the following program:

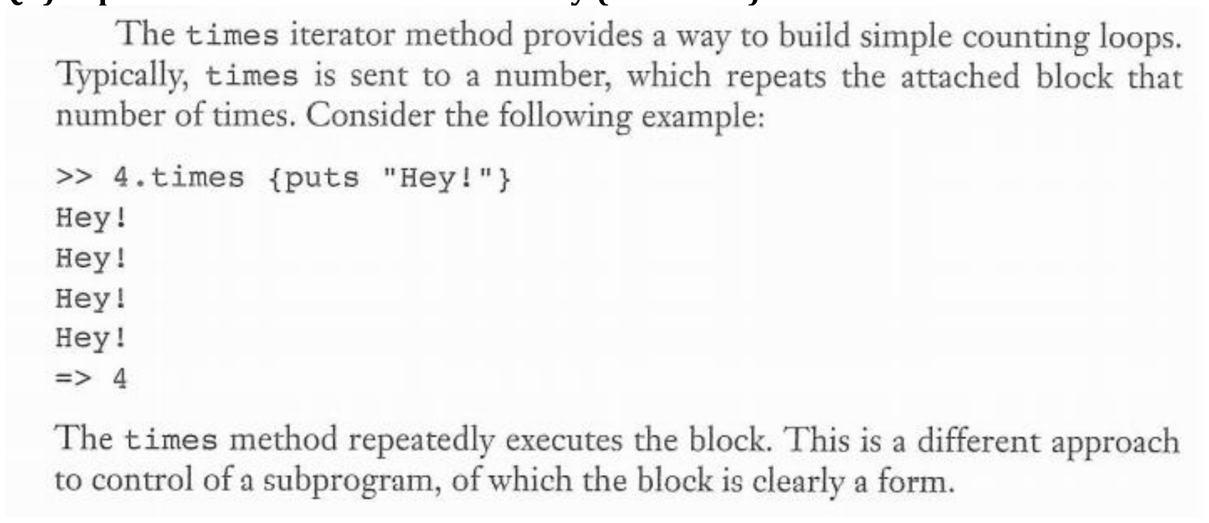
```
# home_controller.rb - for the popcorn application
class HomeController < ApplicationController
  def the_form
  end

# result method - fetch data and compute the cost
  def result

# Fetch the form values
    @unpop = params[:unpop].to_i
    @caramel = params[:caramel].to_i
    @caramelnut = params[:caramelnut].to_i
    @toffeynut = params[:toffeynut].to_i
    @name = params[:name]
    @street = params[:street]
    @city = params[:city]
    @payment = params[:payment]

# Compute the item costs and total cost
    @unpop_cost = 3.0 * @unpop
    @caramel_cost = 3.5 * @caramel
    @caramelnut_cost = 4.5 * @caramelnut
    @toffeynut_cost = 5.0 * @toffeynut
    @total_price = @unpop_cost + @caramel_cost +
                   @caramelnut_cost + @toffeynut_cost
    @total_items = @unpop + @caramel + @caramelnut + @toffeynut

# Now convert the dollar amounts to strings with 2 digits
#  to the right of the decimal point
    @total_price = sprintf("%5.2f", @total_price)
    @unpop_cost = sprintf("%5.2f", @unpop_cost)
    @caramel_cost = sprintf("%5.2f", @caramel_cost)
    @caramelnut_cost = sprintf("%5.2f", @caramelnut_cost)
    @toffeynut_cost = sprintf("%5.2f", @toffeynut_cost)

  end
end
```

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- result.rhtml - result view for the popcorn application
     -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title> result.rhtml </title>
  <head>
  <body>

<!-- Display the customer information -->
    <h4> Customer: </h4>
    <%= @name %> <br /> <%= @street %> <br />
    <%= @city %>
    <p /> <p />

<!-- Display a table of the order information -->
    <table border = "border">
      <caption> Order Information </caption>
      <tr>
        <th> Product </th>
        <th> Unit Price </th>
        <th> Quantity </th>
        <th> Item Cost </th>
      </tr>
```



```
      <td> <%= @unpop %> </td>
      <td> $<%= @unpop_cost %> </td>
    </tr>
    <tr align = "center">
      <td> Caramel Popcorn </td>
      <td> $3.50 </td>
      <td> <%= @caramel %> </td>
      <td> $<%= @caramel_cost %> </td>
    </tr>
    <tr align = "center">
      <td> Caramel Nut Popcorn </td>
      <td> $4.50 </td>
      <td> <%= @caramelnut %> </td>
      <td> $<%= @caramelnut_cost %> </td>
    </tr>
    <tr align = "center">
      <td> Toffey Nut Popcorn </td>
      <td> $5.00 </td>
      <td> <%= @toffeynut %> </td>
      <td> $<%= @toffeynut_cost %> </td>
    </tr>
  </table>
  <p /><p />

    <p>
      You ordered <%= @total_items %> popcorn items <br />
      The total cost of your order is $<%= @total_price %> <br />
      Your chosen method of payment is: <%= @payment %> <br />
      Thank you for your order
    </p>
  </body>
</html>
```

Q6) Explain code block iterators in ruby (10 marks)



The `times` iterator method provides a way to build simple counting loops. Typically, `times` is sent to a number, which repeats the attached block that number of times. Consider the following example:

```
>> 4.times {puts "Hey!"}
Hey!
Hey!
Hey!
Hey!
=> 4
```

The `times` method repeatedly executes the block. This is a different approach to control of a subprogram, of which the block is clearly a form.

The most commonly used iterator is `each`, which is often used to go through arrays and apply a block to each element. For this, it is convenient to allow blocks to have parameters. Blocks *can* have parameters, which appear at the beginning of the block, delimited by vertical bars (|). The following example, which uses a block parameter, illustrates the use of `each`:

```
>> list = [2, 4, 6, 8]
=> [2, 4, 6, 8]
>> list.each {|value| puts value}
2
4
6
8
=> [2, 4, 6, 8]
```

The `each` iterator works equally well on array literals, as in the following:

```
>> ["Joe", "Jo", "Joanne"].each {|name| puts name}
Joe
Jo
Joanne
=> ["Joe", "Jo", "Joanne"]
```

The `upto` iterator method is used like `times`, except that the last value of the counter is given as a parameter. For example:

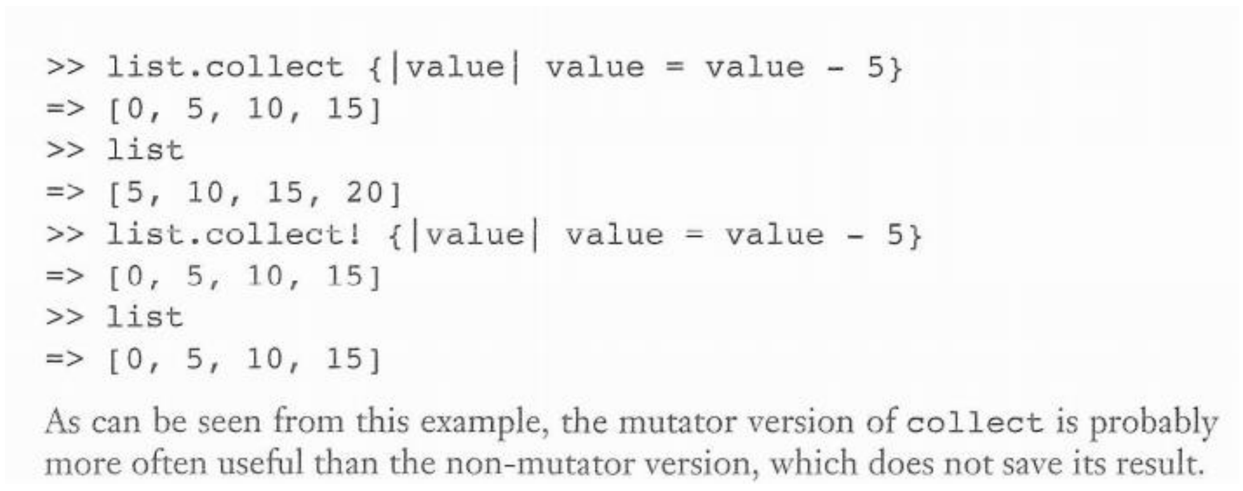
```
>> 5.upto(8) {|value| puts value}
5
6
7
8
=> 5
```

The `step` iterator method takes a terminal value and a step size as parameters and generates the values from that of the object to which it is sent and the terminal value. For example:

```
>> 0.step(6, 2) {|value| puts value}
0
2
4
6
=> 0
```

The `collect` iterator method takes the elements from an array, one at a time, like `each`, and puts the values generated by the given block into a new array. For example:

```
>> list = [5, 10, 15, 20]
=> [5, 10, 15, 20]
```



```
>> list.collect {|value| value = value - 5}
=> [0, 5, 10, 15]
>> list
=> [5, 10, 15, 20]
>> list.collect! {|value| value = value - 5}
=> [0, 5, 10, 15]
>> list
=> [0, 5, 10, 15]
```

As can be seen from this example, the mutator version of `collect` is probably more often useful than the non-mutator version, which does not save its result.

Q7) Explain default grid system in Bootstrap. (10 marks)

Default Grid System

The default Bootstrap grid (see Figure 1-1) system utilizes 12 columns, making for a 940px-wide container without responsive features enabled. With the responsive CSS file added, the grid adapts to be 724px or 1170px wide, depending on your viewport. Below 767px viewports, such as the ones on tablets and smaller devices, the columns become fluid and stack vertically. At the default width, each column is 60 pixels wide and offset 20 pixels to the left. An example of the 12 possible columns is in Figure 1-1

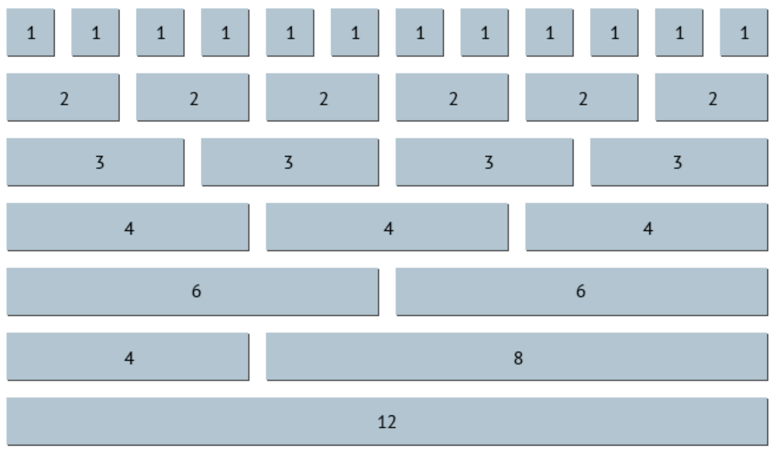


Figure 1-1. Default grid

Basic Grid HTML

To create a simple layout, create a container with a <div> that has a class of .row and add the appropriate amount of .span* columns. Since we have a 12-column grid, we just need the amount of .span* columns to equal 12. We could use a 3-6-3 layout, 4-8, 3-5-4, 2-8-2… we could go on and on, but I think you get the gist.
The following code shows .span8 and .span4, which adds up to 12:

```
<div class="row">
<div class="span8">...</div>
<div class="span4">...</div>
</div>
```

Offsetting Columns

You can move columns to the right using the .offset* class. Each class moves the span over that width. So an .offset2 would move a .span7 over two columns (see Figure 1-2):

```
<div class="row">
<div class="span2">...</div>
<div class="span7 offset2">...</div>
```

```
</div>
```

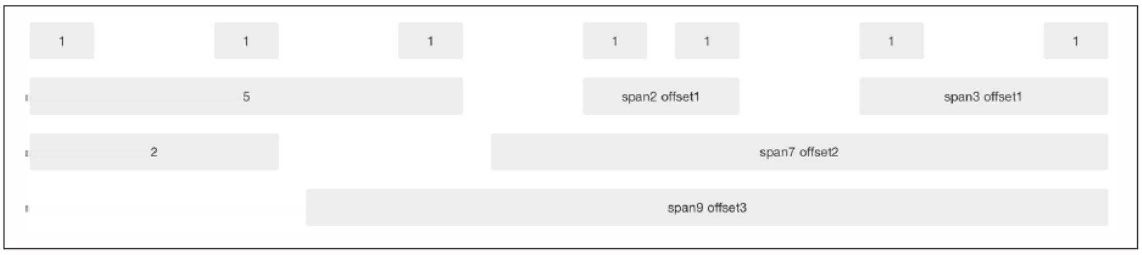


Figure 1-2. Offset grid

Nesting Columns
To nest your content with the default grid, inside of a .span*, simply add a new .row with enough .span* that it equals the number of spans of the parent container (see Figure 1-3):

```
<div class="row">
  <div class="span9">
    Level 1 of column
    <div class="row">
      <div class="span6">Level 2</div>
      <div class="span3">Level 2</div>
    </div>
  </div>
</div>
```

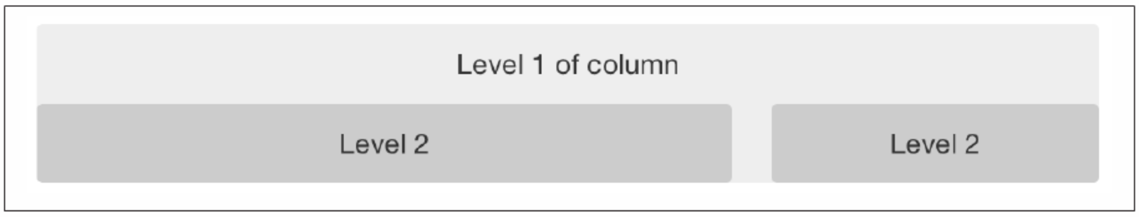


Figure 1-3. Nesting grid

Q8) Discuss typography in bootstrap with examples (10 marks)

Starting with typography, Bootstrap uses Helvetica Neue, Helvetica, Arial, and sans-serif in its default font stack. These are all standard fonts and are included as defaults on all major computers. If by chance these fonts don't exist, they fall back to `sans-serif` (the catchall) to tell the browser to use the default font for the browser. All body copy has the `font-size` set at 14 pixels, with the `line-height` set at 20 pixels. The `<p>` tag has a `margin-bottom` of 10 pixels, or half the `line-height`.

Headings

All six standard heading levels have been styled in Bootstrap (see Figure 2-1), with the `<h1>` at 36 pixels tall, and the `<h6>` down to 12 pixels (for reference, default body text is 14 pixels tall). In addition, to add an inline subheading to any of the headings, simply add `<small>` around any of the elements and you will get smaller text in a lighter color. In the case of the `<h1>`, the small text is 24 pixels tall, normal font weight (i.e., not bold), and gray instead of black:

```
h1 small {
    font-size:24px;
        font-weight:normal;
        line-height:1;
        color:#999;
        }
```

Lead Body Copy

To add some emphasis to a paragraph, add `class="lead"` (see Figure 2-2). This will give you larger font size, lighter weight, and a taller line height. This is generally used for the first few paragraphs in a section, but it can really be used anywhere:

```
<p class="lead">Bacon ipsum dolor sit amet tri-tip pork loin ball tip frankfurter
swine boudin meatloaf shoulder short ribs cow drumstick beef jowl.
Meatball chicken sausage tail, kielbasa strip steak turducken venison prosciutto.
Chuck filet mignon tri-tip ribeye, flank brisket leberkas. Swine
turducken turkey shank, hamburger beef ribs bresaola pastrami venison rump.</p>
```

Emphasis

In addition to using the `<small>` tag within headings, as discussed above, you can also use it with body copy. When `<small>` is applied to body text, the font shrinks to 85% of its original size.

Bold

To add emphasis to text, simply wrap it in a `<strong>` tag. This will add `font-weight:bold;` to the selected text.

Italics

For italics, wrap your content in the `<em>` tag. The term "em" derives from the word "emphasis" and is meant to add stress to your text.

Emphasis Classes

Along with <strong> and <em>, Bootstrap offers a few other classes that can be used to provide emphasis (see Figure 2-3). These could be applied to paragraphs or spans:

```
<p class="muted">This content is muted</p>
<p class="text-warning">This content carries a warning class</p>
<p class="text-error">This content carries an error class</p>
<p class="text-info">This content carries an info class</p>
<p class="text-success">This content carries a success class</p>
<p>This content has <em>emphasis</em>, and can be <strong>bold</strong></p>
```

Abbreviations

The HTML <abbr> element provides markup for abbreviations or acronyms, like WWW or HTTP (see Figure 2-4). By marking up abbreviations, you can give useful information to browsers, spell checkers, translation systems, or search engines. Bootstrap styles <abbr> elements with a light dotted border along the bottom and reveals the full text on hover (as long as you add that text to the <abbr> title attribute):

```
<abbr title="Real Simple Syndication">RSS</abbr>
```

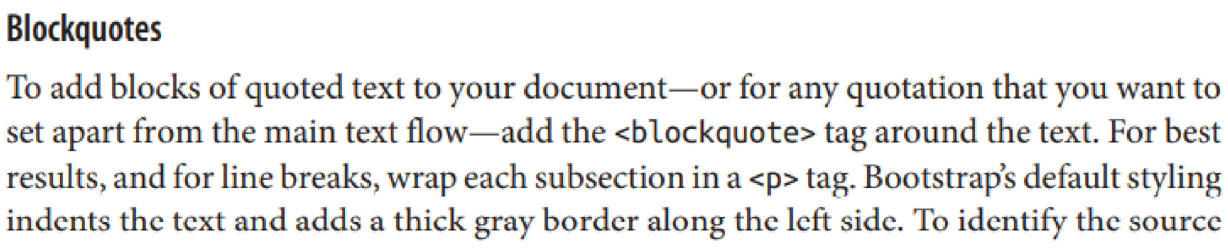
Add .initialism to an <abbr> for a slightly smaller font size (see Figure 2-5):

```
<abbr title="rolling on the floor, laughing out loud">That joke had me ROTFLOL
</abbr>
```

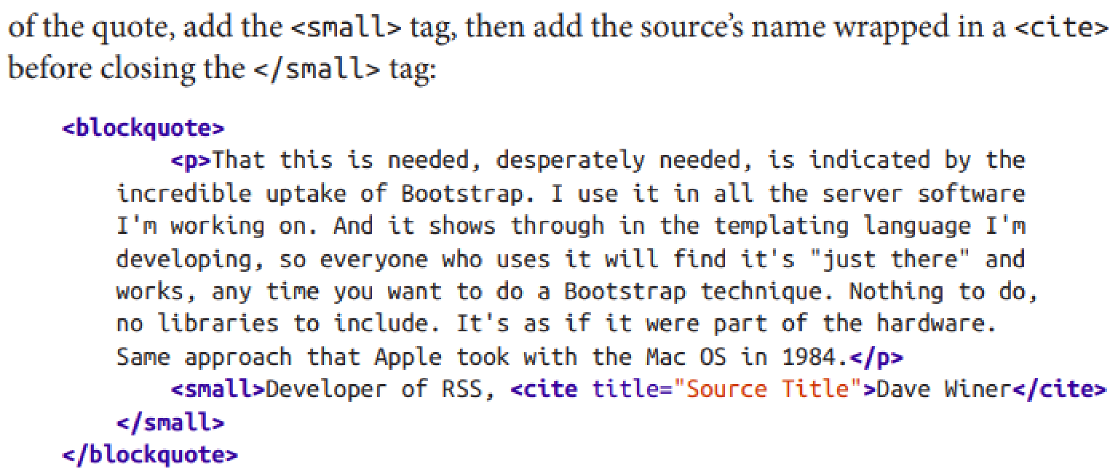
Addresses

Adding <address> elements to your page can help screen readers and search engines locate any physical addresses and phone numbers in the text (see Figure 2-6). It can also be used to mark up email addresses. Since the <address> defaults to display:block; you'll need to use <br> tags to add line breaks to the enclosed address text (e.g., to split the street address and city onto separate lines):

```
<address>
  <strong>O'Reilly Media, Inc.</strong><br>
  1005 Gravenstein HWY North<br>
  Sebastopol, CA 95472<br>
  <abbr title="Phone">P:</abbr> <a href="tel:+17078277000">(707) 827-7000</a>
</address>

<address>
  <strong>Jake Spurlock</strong><br>
  <a href="mailto:#">flast@oreilly.com</a>
</address>
```


Blockquotes

To add blocks of quoted text to your document—or for any quotation that you want to set apart from the main text flow—add the <blockquote> tag around the text. For best results, and for line breaks, wrap each subsection in a <p> tag. Bootstrap's default styling indents the text and adds a thick gray border along the left side. To identify the source


of the quote, add the <small> tag, then add the source's name wrapped in a <cite> tag before closing the </small> tag:

```
<blockquote>
        <p>That this is needed, desperately needed, is indicated by the
    incredible uptake of Bootstrap. I use it in all the server software
    I'm working on. And it shows through in the templating language I'm
    developing, so everyone who uses it will find it's "just there" and
    works, any time you want to do a Bootstrap technique. Nothing to do,
    no libraries to include. It's as if it were part of the hardware.
    Same approach that Apple took with the Mac OS in 1984.</p>
        <small>Developer of RSS, <cite title="Source Title">Dave Winer</cite>
    </small>
</blockquote>
```

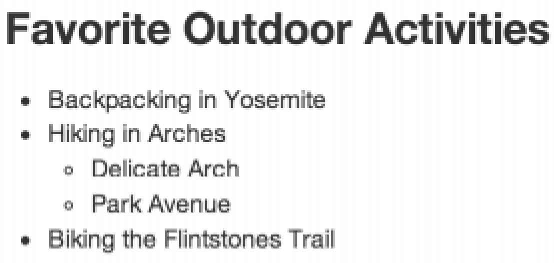
Q9) Define Bootstrap. Explain what support and styling bootstrap offers for the three main list types (ordered, unordered, and definition lists) (10 marks)

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation and other interface components.
Bootstrap is an open source product from Mark Otto and Jacob Thornton who, when it was initially released, were both employees at Twitter. There was a need to standardize the frontend toolsets of engineers across the company

Unordered list

If you have an ordered list that you would like to remove the bullets from, add class="unstyled" to the opening <ul> tag

```
<h3>Favorite Outdoor Activities</h3>
<ul>
      <li>Backpacking in Yosemite</li>
      <li>Hiking in Arches
     <ul>
           <li>Delicate Arch</li>
           <li>Park Avenue</li>
     </ul>
     </li>
     <li>Biking the Flintstones Trail</li>
</ul>
```


Favorite Outdoor Activities

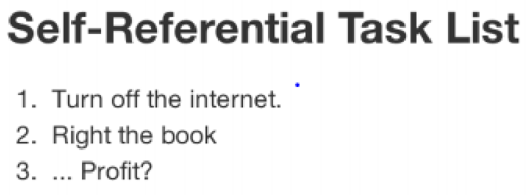
- Backpacking in Yosemite
- Hiking in Arches
  - Delicate Arch
  - Park Avenue
- Biking the Flintstones Trail

Ordered list

An ordered list is a list that falls in some sort of sequential order and is prefaced by

numbers rather than bullets. This is handy when you want to build a
list of numbered items like a task list, guide items, or even a list of comments on a blog post:

```
<h3>Self-Referential Task List</h3>
<ol>
 <li>Turn off the internet.</li>
 <li>Write the book.</li>
 <li>... Profit?</li>
</ol>
```

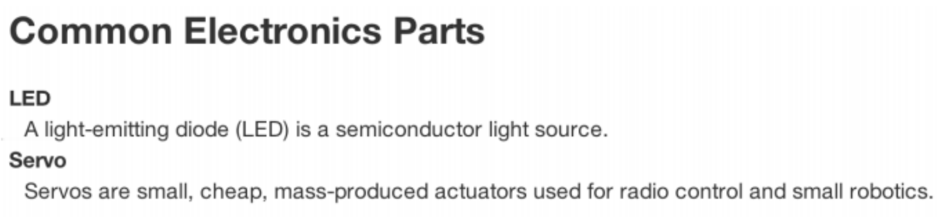

Self-Referential Task List

1. Turn off the internet.
2. Right the book
3. ... Profit?

Definition list

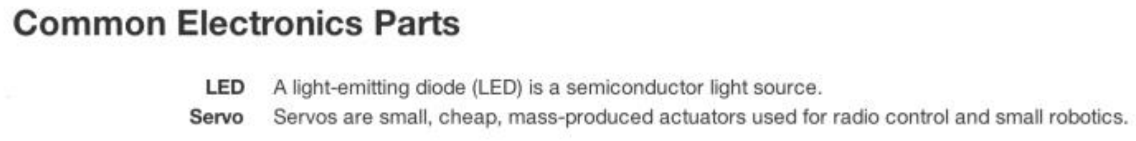
The third type of list you get with Bootstrap is the definition list. The definition list differs from the ordered and unordered list in that instead of just having a block-level <li> element, each list item can consist of both the <dt> and the <dd> elements. <dt> stands for "definition term," and like a dictionary, this is the term (or phrase) that is being defined. Subsequently, the <dd> is the definition of the <dt>.
A lot of times in markup, you will see people using headings inside an unordered list. This works, but may not be the most semantic way to mark up the text. A better method would be creating a <dl> and then styling the <dt> and <dd> as you would the heading and the text. That being said, Bootstrap offers some clean default styles
and an option for a side-by-side layout of each definition:

```
<h3>Common Electronics Parts</h3>
<dl>
 <dt>LED</dt>
 <dd>A light-emitting diode (LED) is a semiconductor light source.</dd>
 <dt>Servo</dt>
 <dd>Servos are small, cheap, mass-produced actuators used for radio
 control and small robotics.</dd>
</dl>
```


Common Electronics Parts

LED
A light-emitting diode (LED) is a semiconductor light source.
Servo
Servos are small, cheap, mass-produced actuators used for radio control and small robotics.

To change the <dl> to a horizontal layout, with the <dt> on the left side and the <dd> on the right, simply add class="dl-horizontal" to the opening tag


Common Electronics Parts

LED	A light-emitting diode (LED) is a semiconductor light source.
Servo	Servos are small, cheap, mass-produced actuators used for radio control and small robotics.

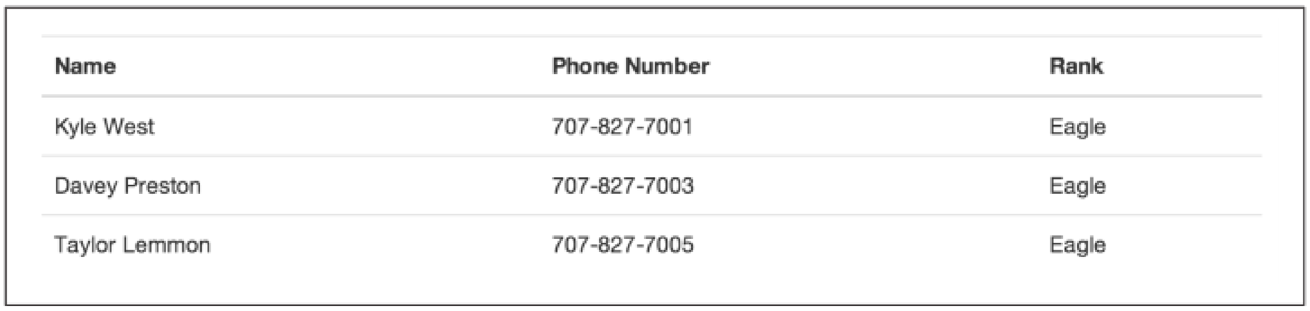
Q10) Discuss how tables are handled in bootstrap. (10 marks)

Table 2-1. Table elements supported by Bootstrap

Tag	Description
`<table>`	Wrapping element for displaying data in a tabular format
`<thead>`	Container element for table header rows (`<tr>`) to label table columns
`<tbody>`	Container element for table rows (`<tr>`) in the body of the table
`<tr>`	Container element for a set of table cells (`<td>` or `<th>`) that appears on a single row
`<td>`	Default table cell
`<th>`	Special table cell for column (or row, depending on scope and placement) labels. Must be used within a `<thead>`
`<caption>`	Description or summary of what the table holds, especially useful for screen readers

If you want a nice, basic table style with just some light padding and horizontal dividers, add the base class of `.table` to any table (see Figure 2-13). The basic layout has a top border on all of the `<td>` elements:

```
<table class="table">
  <caption>...</caption>
  <thead>
    <tr>
      <th>...</th>
      <th>...</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>...</td>
      <td>...</td>
    </tr>
  </tbody>
</table>
```



Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

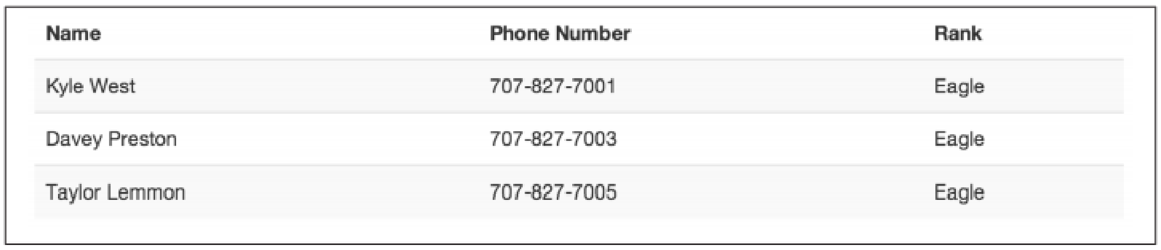
Figure 2-13. Basic table class

Optional Table Classes

Along with the base table markup and the `.table` class, there are a few additional classes that you can use to style the markup. These four classes are: `.table-striped`, `.table-bordered`, `.table-hover`, and `.table-condensed`.

Striped table

By adding the `.table-striped` class, you will get stripes on rows within the `<tbody>` (see Figure 2-14). This is done via the CSS `:nth-child` selector, which is not available on Internet Explorer 7–8.

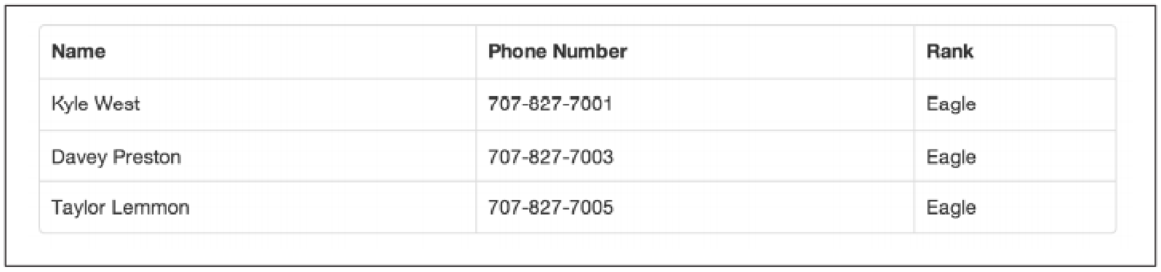


Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Figure 2-14. Striped table class

Bordered table

If you add the `.table-bordered` class, you will get borders surrounding every element and rounded corners around the entire table, as shown in Figure 2-15.

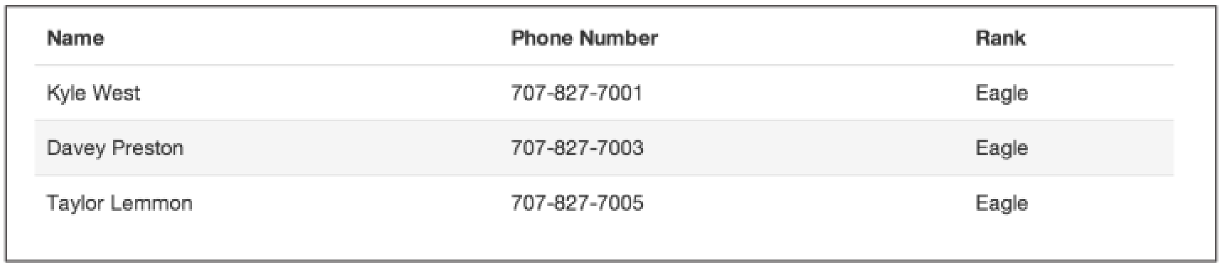


Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Figure 2-15. Bordered table class

Hover table

Figure 2-16 shows the `.table-hover` class. A light gray background will be added to rows while the cursor hovers over them.

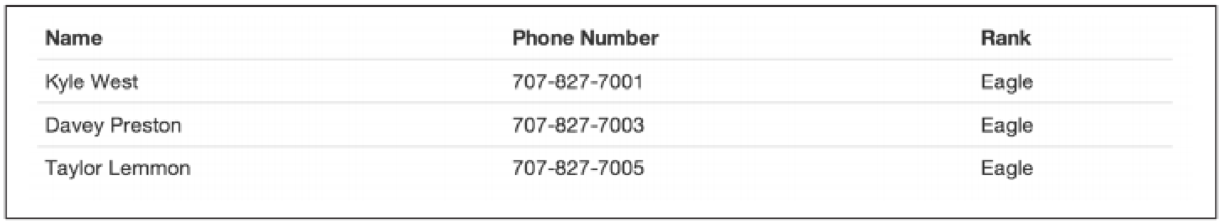


Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Figure 2-16. Hover table class

Condensed table

If you add the `.table-condensed` class, as shown in Figure 2-17, row padding is cut in half to condense the table. This is useful if you want denser information.



Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

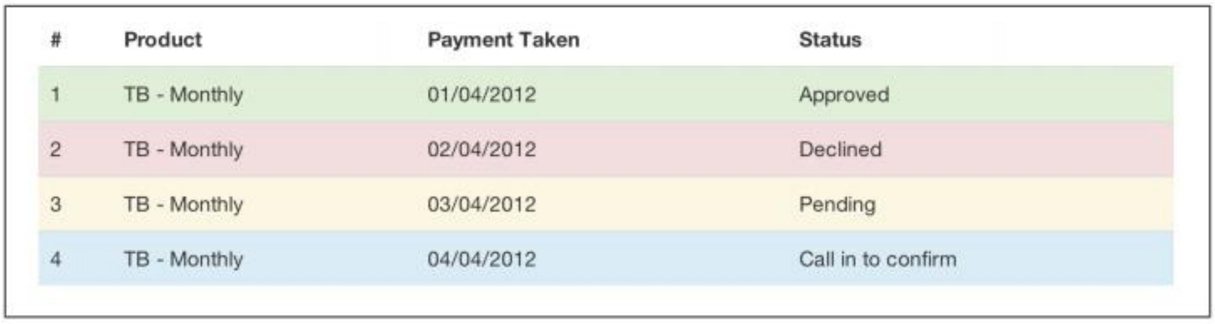
Figure 2-17. Condensed table class

Table Row Classes

The classes shown in Table 2-2 will allow you to change the background color of your rows (see Figure 2-18).

Table 2-2. Optional table row classes

Class	Description	Background color
`.success`	Indicates a successful or positive action.	Green
`.error`	Indicates a dangerous or potentially negative action.	Red
`.warning`	Indicates a warning that might need attention.	Yellow
`.info`	Used as an alternative to the default styles.	Blue



#	Product	Payment Taken	Status
1	TB - Monthly	01/04/2012	Approved
2	TB - Monthly	02/04/2012	Declined
3	TB - Monthly	03/04/2012	Pending
4	TB - Monthly	04/04/2012	Call in to confirm

Figure 2-18. Table row classes