

CBGS SCHEME

20MCA41

OSN [C R N] [M O] [0 4 5]

Fourth Semester MCA Degree Examination, June/July 2023 Advances in Web Technologies

Time: 3 hrs.

Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

- 1 a. Explain any six string functions in PHP (06 Marks)
- b. Explain Cookies in PHP. (04 Marks)
- c. Construct a PHP program to read student data from an XML file and store it into MYSQL. Retrieve and display the details using search options. (10 Marks)

OR

- 2 a. Explain Logical internal structure of arrays in PHP (06 Marks)
- b. Illustrate Session Tracking in PHP. (04 Marks)
- c. Build a PHP form to display text base, table, radio button, submit button and clear button using XML. (10 Marks)

Module-2

- 3 a. List and explain different string methods in Ruby (10 Marks)
- b. Develop a program for generating dynamic documents in Ruby on Rails. (10 Marks)

OR

- 4 a. Demonstrate layouts in Rails. (08 Marks)
- b. Develop a program in Ruby to read list of names from the keyboard, convert them all to upper case letters and place in an array and display in a sorted format. (12 Marks)

Module-3

- 5 a. Discuss the difference between Traditional web application and Ajax model. (06 Marks)
- b. Describe the different HTTP status code with their message. (04 Marks)
- c. Build a program to send the data to the server using GET method in Ajax. (10 Marks)

OR

- 6 a. Explain the technology behind Ajax. (05 Marks)
- b. Create a program to send data to the server using POST method in Ajax. (10 Marks)
- c. Explain the principles of Ajax. (05 Marks)

Module-4

- 7 a. Create a web page using array of XMLHttpRequest object. (08 Marks)
- b. Build a program to cancel pending request using fallback pattern. (08 Marks)
- c. Describe Predictive fetch pattern. (04 Marks)

OR

- 8 a. Create a program for New comment Notifier using periodic refresh. (08 Marks)
- b. Describe Periodic refresh pattern. (04 Marks)
- c. Build a program for Page preloading using predictive fetch. (08 Marks)

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and/or equations written eg. 42+8 = 50, will be treated as malpractice

Module-5

- 9 a. Explain Fluid Grid system, with an example. (05 Marks)
b. Create a table using bootstrap table classes. (10 Marks)
c. Explain Responsive design with example. (05 Marks)

OR

- 10 a. Create a form using Optional form layouts of Bootstrap. (10 Marks)
b. Explain Prepended appended Input controls with example. (10 Marks)

VTU-08-08-2023
3- CR - CR - CR - CR - CR - CR
3 01:35:37 PM
3- CR - CR - CR - CR - CR - CR

Q1a) Explain any six string functions in PHP

Function Name	Parameters	Description
strlen	One string	Returns number of characters in the string
strcmp	Two strings	Returns zero if both strings are equal, a -ve number if the first string occurs before second string or a +ve number if the first string occurs after the second string
strpos	Two strings	Returns position of second string in the first string or false if not found
substr	One string and one integer	Returns the substring from the specified string from the position specified as an integer. If a third integer value is specified, it represents the length of the substring to be retrieved
chop	One string	Returns the string with all white space characters removed from the end
trim	One string	Returns the string with all white space characters removed on both sides
ltrim	One string	Returns the string with all white space characters removed from the beginning
strtolower	One string	Returns the string with all the characters converted to lowercase
strtoupper	One string	Returns the string with all the characters converted to uppercase
strrev	One string	Returns the reverse of the given string
str_replace	Three strings	Returns the string in which a old substring is replaced by the new substring
str_word_count	One string	Returns the word count in the given string

Function	Input	Output
strlen	strlen("star")	4
strcmp	strcmp("twinkle","twinkle")	0
strcmp	strcmp("twinkle","star")	1
strcmp	strcmp("star","twinkle")	-1
strpos	strpos("twinkele twinkle little star","little")	17
substr	substr("little star",6)	star
chop	chop("!star!", "!")	!star
trim	trim(("!star!", "!")	star
ltrim	ltrim("!star!", "!")	star!
strtolower	strtolower("Twinkle")	twinkle
strtoupper	strtoupper("Twinkle")	TWINKLE
substr	substr("twinkle",1,4)	wink

Q1b) Explain Cookies in PHP

- A cookie is a small object of information that consists of a name and a textual value. A cookie is created by some software system on the server.
- The header part of an HTTP communication can include cookies. So, every request sent from a browser to a server, and every response from a server to a browser, can include one or more cookies.
- At the time it is created, a cookie is assigned a lifetime. When the time a cookie has existed reaches its associated lifetime, the cookie is deleted from the browser's host machine.
- Cookie is set in PHP with setcookie function
- First parameter is cookie's name given as a string. The second, if present, is the new value for the cookie, also a string. If the value is absent, setcookie undefines the cookie.
- The third parameter, when present, is the expiration time in seconds for the cookie, given as an integer.
- The default value for the expiration time is zero, which specifies that the cookie is destroyed at the end of the current session. When specified, the expiration time is often given as the number of seconds in the UNIX epoch, which began on January 1, 1970. The time function returns the current time in seconds. So, the cookie expiration time is given as the value returned from time plus some number.
- For example,
setcookie("voted", "true", time() + 86400);

This call creates a cookie named "voted" whose value is "true" and whose lifetime is one day (86,400 is the number of seconds in a day).

- To delete a cookie, use the setcookie() function with an expiration date in the past:
setcookie("voted", "true", time() - 86400);
- All cookies that arrive with a request are placed in the implicit \$ COOKIES array, which has the cookie names as keys and the cookie values as values.
- We can retrieve the value of the cookie using the global variable \$_COOKIE
\$_COOKIE[\$cookie_name]

Eg. \$_COOKIE["voted"]

Program

```
<html>

<head>

<title>Last Visit using Cookies</title>

</head>

<body bgcolor="#cCFFCC" text="#003300">

<h1> Web Programming Lab</h1>
```

```
<p> Welcome to Web Programming Lab </p>

<hr />

<p style="color:blue; font-style: italic">

<?php

date_default_timezone_set('Asia/Calcutta');

//Calculate 60 days in the future

//seconds * minutes * hours * days + current time

// set expiry date to two months from now

$inTwoMonths = 60 * 60 * 24 * 60 + time();

setcookie('lastVisit', date("G:i - m/d/y"), $inTwoMonths);

if(isset($_COOKIE['lastVisit']))

{

$visit = $_COOKIE['lastVisit'];

echo "Last Visited on : ".$visit;

}

else

echo "You've got some old cookies!";

?>

</p>

</body>
```

```
</html>
```

Q1c) Construct a PHP program to read student data from an XML file and store it into MYSQL. Retrieve and display the details using search option.

8.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<student_info>
<student>
<usn>1CR17MCA01</usn>
```

```
<name>Ajay</name>
</student>
<student>
<usn>1CR17MCA02</usn>
<name>Akshatha</name>
</student>
<student>
<usn>1CR17MCA58</usn>
<name>Piyush</name>
</student>
<student>
<usn>1CR17MCA59</usn>
<name>Taj</name>
</student>
</student_info>
```

8.php

```
<html>
  <body>
    <form name="form1" method="post" action="8.php">
      Enter Name <input type="text" name="sname">
      <input type="submit" name="submit" value="search">
    </form>
  </body>
</html>
```

```
<?php
$con = mysql_connect("localhost","root","");
if (!$con)
{
    die('Could not connect: ' . mysql_error());
}

mysql_select_db("web1", $con);
$lib = simplexml_load_file("7.xml");

$i = "delete from student";
$result = mysql_query($i);
foreach($lib as $stu)
{
    $usn= $stu->usn;
    $name=$stu->name;
    $i="insert into student(usn,name) values('$usn','$name)";
```

```

        mysql_query($i);
    }

    if(($_SERVER["REQUEST_METHOD"]=="POST")||($_SERVER["REQUEST_METH
    OD"]=="post"))
    {
        $stname = $_POST["stname"];

        $result = mysql_query("SELECT * from student where name LIKE '%" . $stname . "%'");
        echo "<table border='1'><tr><th>USN</th><th>Name</th></tr>";

        while($row = mysql_fetch_array($result))
        {
            echo "<tr><td>" . $row['usn'] . "</td><td>" . $row['name'] . "</td></tr>";
        }
        echo "</table>";
    }
    ?>

```

Output:-

The screenshot shows two browser windows. The top window displays the raw HTML output of the PHP script, which is an XML-like structure for student information:

```

<student_info>
  <student>
    <usn>1CR17MCA01</usn>
    <name>Ajay</name>
  </student>
  <student>
    <usn>1CR17MCA02</usn>
    <name>Akshatha</name>
  </student>
  <student>
    <usn>1CR17MCA58</usn>
    <name>Piyush</name>
  </student>
  <student>
    <usn>1CR17MCA59</usn>
    <name>Taj</name>
  </student>
</student_info>

```

The bottom window shows the rendered web page. It features a search form with the text "Enter Name" and a text input field containing "Ajay", followed by a "search" button. Below the form is a table with the following data:

USN	Name
1CR17MCA01	Ajay

Q2a) Explain logical internal structure of arrays in PHP

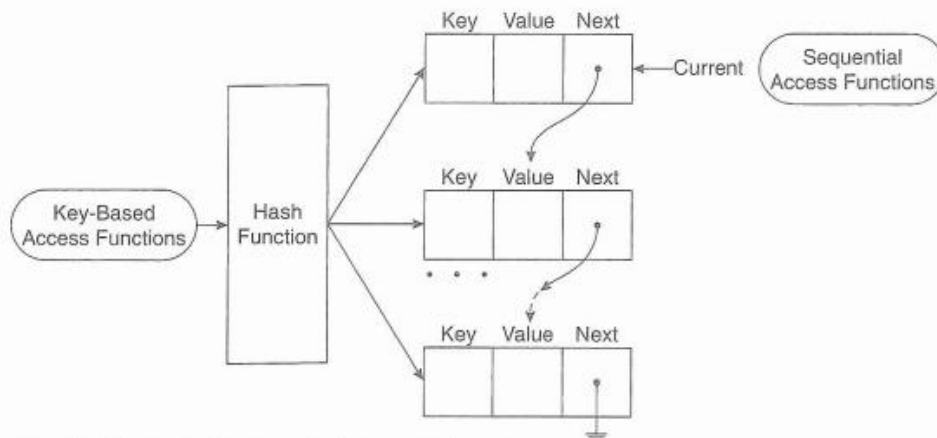


Figure 11.3 Logical internal structure of arrays

Internally, the elements of an array are stored in a linked list of cells, where each cell includes both the key and the value of the element. The cells themselves are stored in memory through a key hashing function so that they are randomly distributed in a reserved block of storage. Accesses to elements through

string keys are implemented through the hashing function. However, the elements all have links that connect them in the order in which they were created, which allows them to be accessed in that order if the keys are strings and in the order of their keys if the keys are numbers. Section 11.7.4 discusses the ways array elements can be accessed in order.

Figure 11.3 shows the internal logical structure of an array. Although arrays may not be implemented in this exact way, it shows how the two different access methods could be supported.

Q2b) Illustrate session tracking in PHP

In many cases, information about a session is needed only during the session. Also, the needed information about a client is nothing more than a unique identifier for the session, which is commonly used in shopping cart applications. For these cases, a different process, named session tracking, can be used.

- Rather than using one or more cookies, a single session array can be used to store information about the previous requests of a client during a session.
- In particular, session arrays often store a unique session ID for a session.
- One significant way that session arrays differ from cookies is that they can be stored on the server, whereas cookies are stored on the client.
- In PHP, a session ID is an internal value that identifies a session. Session IDs need not be known or handled in any way by PHP scripts.

- PHP is made aware that a script is interested in session tracking by calling the session start function, which takes no parameters. The first call to session start in a session causes a session ID to be created and recorded.
- On subsequent calls to session_start in the same session, the function retrieves the \$_SESSION array, which stores any session variables and their values that were registered in previously executed scripts in this session.
- Session key/value pairs are created or changed by assignments to the \$ SESSION array.
- They can be destroyed with the unset operator.
- Consider the following example:


```
session_start();
if (!isset($_SESSION["page_number"]))
    $_SESSION["page_number"] = 1;
$page_num = $_SESSION["page_number"];
print("You have now visited $page_num page(s) <br />");
$_SESSION["page_number"]++;
```
- If this is not the first document visited that calls session start and sets the page_number session variable, this script will produce the specified line with the last set value of \$_SESSION ["page_number") .
- If no document that was previously visited in this session set page_number, this script sets page_number to 1, produces the line ‘You have now visited 1 page(s)’, and increments page_number

Program

```
<html>
<head>
<title>Page Views </title>
</head>
<body bgcolor="#cCCFFCC" text="#003300">
<h1> Web Programming Lab</h1>
<p> Welcome to Web Programming Lab </p>
<hr />
<p style="color:blue; font-style: italic">
<?php
session_start();
//session_register("count");
$_SESSION["count"];
if(!isset($_SESSION["count"]))
{
$_SESSION["count"] = 0;
echo "Counter initialized... <br />";
}
else { $_SESSION["count"]++; }
echo "Number of Page Views : <b>$_SESSION[count]</b></p>";
?>
<p>Reload this page to increment</p>
</body>
```

</html>

C) Build a PHP form to display text base, table, radio button, submit button, and clear button using XML

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
        $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);

    echo "<h2>Your Input:</h2>";
    echo $name;
    echo "<br>";
    echo $email;
    echo "<br>";
    echo $website;
    echo "<br>";
    echo $comment;
    echo "<br>";
    echo $gender;
}
?>
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<table>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
    <tr>
        <td>Name: </td><td><input type="text" name="name" value=""></td>
    </tr>
    <tr>
        <td>E-mail: </td><td><input type="text" name="email" value=""></td>
    </tr>
    <tr>
        <td> Website: </td><td><input type="text" name="website" value=""></td>
```

```
</tr>
<tr>
  <td> Comment: </td><td><textarea name="comment" rows="5" cols="40"></textarea></td>
</tr>
<tr>
  <td> Gender: </td><td>
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female") echo
"checked";?> value="female">Female
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
    <input type="radio" name="gender" <?php if (isset($gender) && $gender=="other") echo "checked";?>
value="other">Other
  </td></tr>
  <tr><td colspan="2"><input type="submit" name="submit" value="Submit"> </td></tr>
</form>
</body>
</html>
```

Q3a) List and explain different string methods in Ruby

1. The String method for catenation is specified by plus (+), which can be used as a binary operator. This method creates a new string from its operands:

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

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

2. The << method appends a string to the right end of another string, which, of course, makes sense only if the left operand is a variable. Like +, the << method can be used as a binary operator.

```
>> mystr = "G'day,"
```

```
=> "G'day,"
```

```
>> mystr << "mate"
```

```
=> "G'day, mate"
```

3. The first assignment creates the specified string literal and sets the variable mystr to reference that memory location. If mystr is assigned to another variable, that variable will reference the same memory location as mystr. If a different string literal is assigned to mystr, Ruby will build a memory location with the value of the new string literal and mystr will reference that location. In order to change the content of same memory location of mystr replace method is used.

```
>> 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!"
```

4. The other most commonly used methods of Ruby are similar to those of other programming languages. Among these are the ones shown in Table below; all of them create new strings.

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

5. As stated previously, all of these methods produce new strings, rather than modify the given string in place. However, all of the methods also have versions that modify their objects in place. These methods are called bang or mutator methods and are specified by following their names with an exclamation point (!).

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

Q3b) develop a program for generating dynamic document in Ruby on rails

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 access 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>

```

```

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

```

Q4) Demonstrate layouts in Rails

First, the user needs to define a layout template, and after that, we have to let the controller know that we have created a template layout and can use it. Let's make the template first.

Create and add a new file named standard.html.erb to app/views/layouts. The controllers may know which file you are using as a template by the file's name, so using proper names is advisable.

To the new standard.html.erb file add the following code and save your changes:

```

<!DOCTYPE HTML >

<html >

  <head>

    <meta content = "text/html; charset = iso-8859-1" / http-equiv = "Content-Type" >

    <meta http-equiv = "Content-Language" content = "en-us" />

    <title>Library Info System</title>

    <%= stylesheet_link_tag "style" %>

```

```
</head>

<body id = "library">

  <div id = "container">

    <div id = "header">

      <h1>Library Info System</h1>

      <h3>Library powered by Ruby on Rails</h3>

    </div>

    <div id = "content">

      <%= yield -%>

    </div>

    <div id = "sidebar"></div>

  </div>

</body>

</html>
```

Explanation

The above code's elements were just standard HTML elements. But there are two different lines in the above code `stylesheet_link_tag` is a helper method that outputs a stylesheet `<link>`. In this example, we are linking `style.css` style sheets.

You must be wondering why we have written `yield` in the above code. The `yield` command has its advantage whenever there is a `yield` command, and then rails know that it should put HTML.erb for the method called here.

In the `book_controller.rb`, add the following code from the second line.

```
class BookController < ApplicationController

  layout 'standard'

  def list

    @books = Book.all

  end
```


The above code uses the layout available in the standard.html.erb file.

4b) Develop a program in ruby to read list of names from the keyboard, convert them all to uppercase letters and place in an array and display in a sorted format

```
print "Enter names seperated by space"

a = gets.chomp;

array = a.split(' ');

puts 'Names entered by you : ' +array.to_s

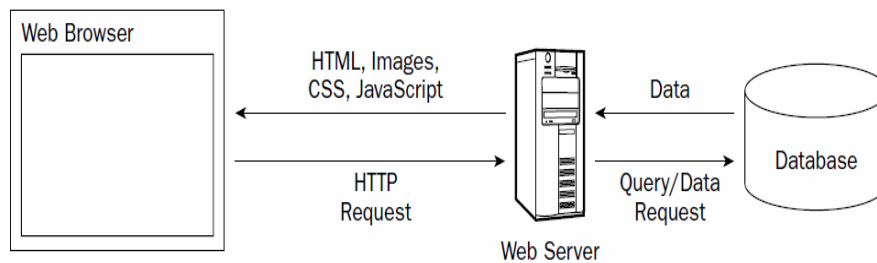
uparray=array.map(&:upcase);

sort_ar=uparray.sort

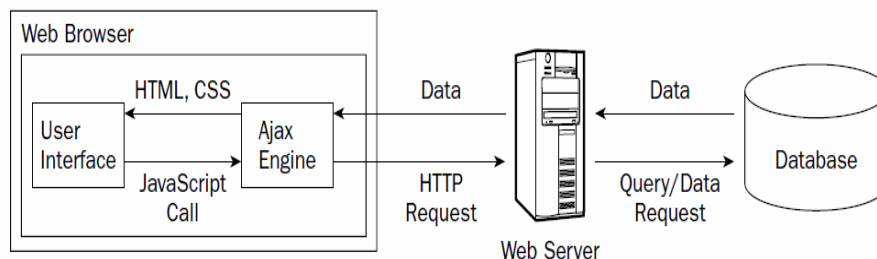
puts 'Names converted to uppercase and sorted : ' +sort_ar.to_s
```

Q5a) Discuss the difference between traditional web application and Ajax model

Traditional Web Application Model



Ajax Web Application Model



- ❑ Instead of the traditional web application model where the browser itself is responsible for initiating requests to, and processing requests from, the web server, the Ajax model provides an intermediate layer called an *Ajax engine*—to handle this communication.

- ❑ *An Ajax engine is really just a JavaScript* object or function that is called whenever information needs to be requested from the server.
- ❑ Instead of the traditional model of providing a link to another resource (such as another web page), each link makes a call to the Ajax engine, which schedules and executes the request. The request is done asynchronously, meaning that code execution doesn't wait for a response before continuing.
- ❑ The server—which traditionally would serve up HTML, images, CSS, or JavaScript—is configured to return data that the Ajax engine can use. This data can be plain text, XML, or any other data format that you may need. The only requirement is that the Ajax engine can understand and interpret the data
- ❑ When the Ajax engine receives the server response, it goes into action, often parsing the data and making several changes to the user interface based on the information it was provided. Because this process involves transferring less information than the traditional web application model, user interface updates are faster, and the user is able to do his or her work more quickly.

Q5b) Describe the different HTTP status code with their message

Inside the anonymous function, we need to check on the data that's been downloaded: Is the download complete? Are we ready to use that data? You can determine that with two properties of the XMLHttpRequest object: readyState and status.

The readyState property tells you how the data downloading is going. Here are the possible values for this property—a value of 4 is what you want to see, because that means the data has been fully downloaded:

0	Uninitialized
1	Loading
2	Loaded
3	Interactive
4	Complete

The status property is the property that contains the actual status of the download. This is actually the normal HTTP status code that you get when you try to download web pages. For example, if the data you're looking for wasn't found, you'll get a value of 404 in the status property. Here are some of the possible values—note that you'll want to see a value of 200 here, which means that the download completed normally:

200	OK
201	Created
204	No Content
205	Reset Content
206	Partial Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
411	Length Required

413	Requested Entity Too Large
414	Requested URL Too Long
415	Unsupported Media Type
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Q5 c) Build a program to send data to the server using GET method in Ajax

```
<html>
  <head>
    <title>An Ajax example</title>
    <script language = "javascript">
      var ajaxobj = false;
      if (window.XMLHttpRequest) {
        ajaxobj = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        ajaxobj = new ActiveXObject("Microsoft.XMLHTTP");
      }
      function getData(dataSource, divID){
        if(ajaxobj) {
          var obj = document.getElementById(divID);
          ajaxobj.open("GET", dataSource);
          ajaxobj.onreadystatechange = function()
```

```

        {
            if (ajaxobj.readyState == 4 &&
                ajaxobj.status == 200) {
                obj.innerHTML = ajaxobj.responseText;
            }
        }

        ajaxobj.send(null);
    }
}
</script>
</head>
<body>
    <H1>An Ajax example</H1>
    <form>
        <input type = "button" value = "Fetch the first message"
            onclick = "getData('dataresponder.php?data=1', 'targetDiv')">
        <input type = "button" value = "Fetch the second message"
            onclick = "getData('dataresponder.php?data=2', 'targetDiv')">
    </form>
    <div id="targetDiv">
        <p>The fetched message will appear here.</p>
    </div>
</body>
</html>

```

dataresponder.php

```

<?php
if ($_GET["data"] == "1") {
    echo "The server got a value of 1";
}
if ($_GET["data"] == "2") {
    echo "The server got a value of 2";
}
?>

```

Q6 a) Explain Technologies behind Ajax

- HTML/XHTML: Primary content representation languages
- CSS: Provides stylistic formatting to XHTML
- DOM: Dynamic updating of a loaded page
- XML: Data exchange format

- ❑ XSLT: Transforms XML into XHTML (styled by CSS)
- ❑ XMLHttpRequest: Primary communication broker
- ❑ JavaScript: Scripting language used to program an Ajax engine

Q6b) Create Program to send data to server using POST method in Ajax

```

<html>
  <head>
    <title>An Ajax example</title>
    <script language = "javascript">
      var XMLHttpRequestObject = false;
      if (window.XMLHttpRequest) {
        XMLHttpRequestObject = new XMLHttpRequest();
      } else if (window.ActiveXObject) {
        XMLHttpRequestObject = new
ActiveXObject("Microsoft.XMLHTTP");
      }
      function getData(dataSource, divID, data){
        if(XMLHttpRequestObject) {
          var obj = document.getElementById(divID);
          XMLHttpRequestObject.open("POST", dataSource);
          XMLHttpRequestObject.setRequestHeader('Content-
Type', 'application/x-www-form-urlencoded');
          XMLHttpRequestObject.onreadystatechange =
function()
          {
            if (XMLHttpRequestObject.readyState == 4 &&
XMLHttpRequestObject.status == 200) {
              obj.innerHTML =
XMLHttpRequestObject.responseText;
            }
          }
          XMLHttpRequestObject.send("data="+data);
        }
      }
    </script>
  </head>
  <body>
    <H1>An Ajax example</H1>
    <form>
      <input type = "button" value = "Fetch the first message"

```

```
        onclick = "getData('dataresponder.php','targetDiv',1)">
        <input type = "button" value = "Fetch the second message"
        onclick = "getData('dataresponder.php','targetDiv',2)">
    </form>
    <div id="targetDiv">
        <p>The fetched message will appear here.</p>
    </div>
</body>
</html>
```

dataresponder.php

```
<?php
if ($_GET["data"] == "1") {
    echo 'The server got a value of 1';
}
if ($_GET["data"] == "2") {
    echo 'The server got a value of 2';
}
?>
```

Q6c) Explain the principles of Ajax

- Minimal traffic:** Ajax applications should send and receive as little information as possible to and from the server. In short, Ajax can minimize the amount of traffic between the client and the server. Making sure that your Ajax application doesn't send and receive unnecessary information adds to its robustness.
- No surprises:** Ajax applications typically introduce different user interaction models than traditional web applications. As opposed to the web standard of click-and-wait, some Ajax applications use other user interface paradigms such as drag-and-drop or double-clicking. No matter what user interaction model you choose, be consistent so that the user knows what to do next.
- Established conventions:** Don't waste time inventing new user interaction models that your users will be unfamiliar with. Borrow heavily from traditional web applications and desktop applications, so there is a minimal learning curve.
- No distractions:** Avoid unnecessary and distracting page elements such as looping animations and blinking page sections. Such gimmicks distract the user from what he or she is trying to accomplish.
- Accessibility:** Consider who your primary and secondary users will be and how they most likely will access your Ajax application. Don't program yourself into a corner so that an unexpected

new audience will be completely locked out. Will your users be using older browsers or special software? Make sure you know ahead of time and plan for it.

- ❑ **Avoid entire page downloads:** All server communication after the initial page download should be managed by the Ajax engine. Don't ruin the user experience by downloading small amounts of data in one place but reloading the entire page in others.
- ❑ **User first:** Design the Ajax application with the users in mind before anything else. Try to make the common use cases easy to accomplish and don't be caught up with how you're going to fit in advertising or cool effects.

Q7a) create a web page using array of XMLHttpRequest request object

```
<html>
  <head>
    <title>An Ajax example</title>
    <script language = "javascript">

      var index = 0;

      var XMLHttpRequestObjects = new Array();

      function getData1(dataSource, divID)
      {
        if (window.XMLHttpRequest) {
          XMLHttpRequestObjects.push(new XMLHttpRequest());
        } else if (window.ActiveXObject) {
          XMLHttpRequestObjects.push(new
ActiveXObject("Microsoft.XMLHTTP"));
        }
        index = XMLHttpRequestObjects.length - 1;
        if(XMLHttpRequestObjects[index]) {
          XMLHttpRequestObjects[index].open("GET",
dataSource);

          var obj = document.getElementById(divID);

          XMLHttpRequestObjects[index].onreadystatechange = function()
          {
            if
(XMLHttpRequestObjects[index].readyState == 4 &&
XMLHttpRequestObjects[index].status == 200) {
              obj.innerHTML =
XMLHttpRequestObjects[index].responseText;
```

```

        }
    }
    XMLHttpRequestObjects[index].send(null);
}
}
function getData2(dataSource, divID)
{
    if (window.XMLHttpRequest) {
        XMLHttpRequestObjects.push(new XMLHttpRequest());
    } else if (window.ActiveXObject) {
        XMLHttpRequestObjects.push(new
ActiveXObject("Microsoft.XMLHTTP"));
    }
    index = XMLHttpRequestObjects.length - 1;
    if(XMLHttpRequestObjects[index]) {
        XMLHttpRequestObjects[index].open("GET",
dataSource);

        var obj = document.getElementById(divID);

        XMLHttpRequestObjects[index].onreadystatechange = function()
        {
            if
(XMLHttpRequestObjects[index].readyState == 4 &&
XMLHttpRequestObjects[index].status == 200) {
                obj.innerHTML =
XMLHttpRequestObjects[index].responseText;
            }
        }
        XMLHttpRequestObjects[index].send(null);
    }
}
</script>
</head>
<body>
    <H1>An Ajax example</H1>
    <form>
        <input type = "button" value = "Fetch the first message"
onclick = "getData1('dataresponder.php?data=1', 'targetDiv')">
        <input type = "button" value = "Fetch the second message"
onclick = "getData2('dataresponder.php?data=2', 'targetDiv')">
    </form>
    <div id="targetDiv">
        <p>The fetched message will appear here.</p>
    </div>

```



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

dataresponder.php

```
<?php
if ($_GET["data"] == "1") {
    echo 'The server got a value of 1';
}
if ($_GET["data"] == "2") {
    echo 'The server got a value of 2';
}
?>
```

Q7b) Build a program to cancel pending request using fallback pattern

```
var oXHR = null;
var iInterval = 1000;
var iLastCommentId = -1;
var divNotification = null;
var blnRequestsEnabled = true;
```

Now, the `blnRequestsEnabled` variable must be checked before any request is made. This can be accomplished by wrapping the body of the `checkComments()` function inside of an `if` statement:

```
function checkComments() {
    if (blnRequestsEnabled) {
        if (!oXHR) {
            oXHR = XMLHttpRequest.createRequest();
        } else if (oXHR.readyState != 0) {
            oXHR.abort();
        }

        oXHR.open("get", "CheckComments.php", true);
        oXHR.onreadystatechange = function () {
            if (oXHR.readyState == 4) {
                if (oXHR.status == 200 || oXHR.status == 304) {
                    var aData = oXHR.responseText.split("|");
                    if (aData[0] != iLastCommentId) {
                        iLastCommentId = aData[0];
                        if (iLastCommentId != -1) {
                            showNotification(aData[1], aData[2]);
                        }
                    }
                    setTimeout(checkComments, iInterval);
                }
            }
        };

        oXHR.send(null);
    }
}
```

```

function checkComments() {
    if (blnRequestsEnabled) {
        try {
            if (!oXHR) {
                oXHR = zXmlHttp.createRequest();
            } else if (oXHR.readyState != 0) {
                oXHR.abort();
            }

            oXHR.open("get", "CheckComments.php", true);
            oXHR.onreadystatechange = function () {
                if (oXHR.readyState == 4) {
                    if (oXHR.status == 200 || oXHR.status == 304) {

                        var aData = oXHR.responseText.split("||");
                        if (aData[0] != iLastCommentId) {

                            if (iLastCommentId != -1) {
                                showNotification(aData[1], aData[2]);
                            }

                            iLastCommentId = aData[0];
                        }

                        setTimeout(checkComments, iInterval);
                    } else {
                        blnRequestsEnabled = false;
                    }
                }
            };

            oXHR.send(null);
        } catch (oException) {
            blnRequestsEnabled = false;
        }
    }
}

```

Q7c) Describe predictive fetch pattern

In a traditional web solution, the application has no idea what is to come next. A page is presented with any number of links, each one leading to a different part of the site. This may be termed “fetch on demand,” where the user, through his or her actions, tells the server exactly what data should be retrieved. While this paradigm has defined the Web since its inception, it has the unfortunate side effect of forcing the start-and-stop model of user interaction upon the user. The Predictive Fetch pattern is a relatively simple idea that can be somewhat difficult to implement: the Ajax application guesses what the user is going to do next and retrieves the appropriate data. In a perfect world, it would be wonderful to always know what the user is going to do and make sure that the next data is readily available when needed.

In reality, however, determining future user action is just a guessing game depending on your intentions. There are simple use cases where predicting user actions is somewhat easier. Suppose that you are reading an online article that is separated into three pages. It is logical to assume that if you are interested in reading the first page, you’re also

interested in reading the second and third page. So, if the first page has been loaded for a few seconds (which can easily be determined by using a timeout), it is probably safe to download the second page in the background. Likewise, if the second page has been loaded for a few seconds, it is logical to assume that the reader will continue on to the third page. As this extra data is being loaded and cached on the client, the reader continues to read and barely even notices that the next page comes up almost instantaneously after clicking the Next Page link. The Google Maps is another real world example for predictive fetch pattern. It predicts the nearby places when we search a particular destination.

Q8a) create a program for new comment notifies using periodic refresh

```
<?php
header("Cache-control: No-Cache");
header("Expires: Fri, 30 Oct 1998 14:19:41 GMT");

//database information
```

```
$sDBServer = "your.database.server";
$sDBName = "your_db_name";
$sDBUsername = "your_db_username";
$sDBPassword = "your_db_password";

//create the SQL query string
$sSQL = "select CommentId,Name,LEFT(Message, 50) as ShortMessage from
        BlogComments order by Date desc limit 0,1";

$oLink = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
@mysql_select_db($sDBName) or die("-1|| || ");

if($oResult = mysql_query($sSQL) and mysql_num_rows($oResult) > 0) {
    $aValues = mysql_fetch_array($oResult,MYSQL_ASSOC);
    echo $aValues['CommentId']."||".$aValues['Name']."||".
        $aValues['ShortMessage'];
} else {
    echo "-1|| || ";
}

mysql_free_result($oResult);
mysql_close($oLink);
?>
```

```

function showNotification(sName, sMessage) {
    if (!divNotification) {
        divNotification = document.createElement("div");
        divNotification.className = "notification";
        document.body.appendChild(divNotification);
    }

    divNotification.innerHTML = "<strong>New Comment</strong><br />" + sName
        + " says: " + sMessage + "...<br /><a href=\"ViewComment.php?id="
            + iLastCommentId + "\">View</a>";
    divNotification.style.top = document.body.scrollTop + "px";
    divNotification.style.left = document.body.scrollLeft + "px";
    divNotification.style.display = "block";
    setTimeout(function () {
        divNotification.style.display = "none";
    }, 5000);
}

```

Q8b) Describe periodic refresh pattern

The Periodic Refresh design pattern describes the process of checking for new server information in specific intervals. This approach, also called polling, requires the browser to keep track of when another request to the server should take place. This pattern is used in a variety of different ways on the Web:

- ❑ ESPN uses Periodic Refresh to update its online scoreboards automatically. For example, the NFL Scoreboard, located at <http://sports.espn.go.com/nfl/scoreboard>, shows up-to-the-minute scores and drive charts for every NFL game being played at the time. Using XHR objects and a little bit of Flash, the page repeatedly updates itself with new information.
- ❑ Gmail (<http://gmail.google.com>) uses Periodic Refresh to notify users when new mail has been received. As you are reading an e-mail or performing other operations, Gmail repeatedly checks the server to see if new mail has arrived. This is done without notification unless there is new mail, at which point the number of new e-mails received is displayed in parentheses next to the Inbox menu item.
- ❑ XHTML Live Chat (www.plasticshore.com/projects/chat) uses Periodic Refresh to implement a chat room using simple web technologies. The chat room text is updated automatically every few seconds by checking the server for new information. If there is a new message, the page is updated to reflect it, thus creating a traditional chat room experience.

Q8c) Build a program for page preloading using predictive fetch

```

<?php
$page = 1;
$dataOnly = false;

```

```

    if (isset($_GET["page"])) {
        $page = (int) $_GET["page"];
    }

    if (isset($_GET["dataonly"]) && $_GET["dataonly"] == "true") {
        $dataOnly = true;
    }

    if (!$dataOnly) {
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Article Example</title>
        <script type="text/javascript" src="zxml.js"></script>
        <script type="text/javascript" src="Article.js"></script>
        <link rel="stylesheet" type="text/css" href="Article.css" />
    </head>
    <body>
        <h1>Article Title</h1>
        <div id="divLoadArea" style="display:none"></div>
<?php
    $output = "<p>Page ";

    for ($i=1; $i < 4; $i++) {
        $output .= "<a href='\"ArticleExample.php?page=$i\"' id='\"aPage$i\"'";
        if ($i==$page) {
            $output .= "class='\"current\"'";
        }
        $output .= ">$i</a> ";
    }
    echo $output;
}

```

```

    if ($page==1) {
?>
        <div id="divPage1"><!-- contents of page 1 --></div>
<?php
    } else if ($page == 2) {
?>
        <div id="divPage2"><!-- contents of page 2 --></div>
<?php
    } else if ($page == 3) {
?>
        <div id="divPage3"><!-- contents of page 3 --></div>
<?php
    }

    if (!$dataOnly) {
?>
        </body>
</html>
<?php
    }
?>

```

Q9a) Explain fluid grid system with example

The fluid grid system uses percentages instead of pixels for column widths. It has the same responsive capabilities as our fixed grid system, ensuring proper proportions for key screen resolutions and devices. You can make any row “fluid” by changing `.row` to `.row-fluid`. The column classes stay exactly the same, making it easy to flip between fixed and fluid grids. To offset, you operate in the same way as the fixed grid system—add `.offset*` to any column to shift by your desired number of columns:

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

<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span4 offset2">...</div>
</div>
```

Nesting a fluid grid is a little different. Since we are using percentages, each `.row` resets the column count to 12. For example, if you were inside a `.span8`, instead of two `.span4` elements to divide the content in half, you would use two `.span6` divs (see

Figure 1-4). This is the case for responsive content, as we want the content to fill 100% of the container:

```
<div class="row-fluid">
  <div class="span8">
    <div class="row">
      <div class="span6">...</div>
      <div class="span6">...</div>
    </div>
  </div>
</div>
```

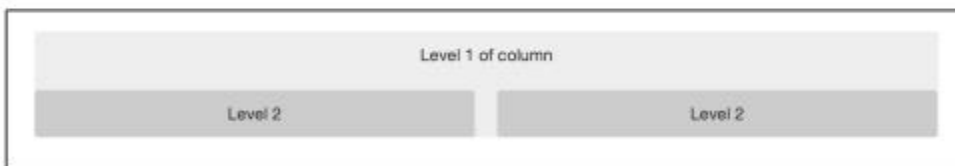


Figure 1-4. Nesting fluid grid

Q9b) Create a table using bootstrap table class

```
<table class="table table-bordered table-dark">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
```

```
<th scope="row">1</th>
<td>Mark</td>
<td>Otto</td>
<td>@mdo</td>
</tr>
<tr>
<th scope="row">2</th>
<td>Jacob</td>
<td>Thornton</td>
<td>@fat</td>
</tr>
<tr>
<th scope="row">3</th>
<td colspan="2">Larry the Bird</td>
<td>@twitter</td>
</tr>
</tbody>
</table>
```

Q9c) Explain responsive design with example

Responsive design is a method for taking all of the existing content that is on the page and optimizing it for the device that is viewing it. For example, the desktop not only gets the normal version of the website, but it might also get a widescreen layout, optimized for the larger displays that many people have attached to their computers. Tablets get an optimized layout, taking advantage of their portrait or landscape layouts. And then with phones, you can target their much narrower width. To target these different widths, Bootstrap uses CSS media queries to measure the width of the browser viewport and then, using conditionals, changes which parts of the stylesheets are loaded. Using the width of the browser viewport, Bootstrap can then optimize the content using a combination of ratios or widths, but it mostly relies on *min-width* and *max-width* properties.

At the core, Bootstrap supports five different layouts, each relying on CSS media queries. The largest layout has columns that are 70 pixels wide, contrasting with the 60 pixels of the normal layout. The tablet layout brings the columns to 42 pixels wide, and when narrower than that, each column goes fluid, meaning the columns are stacked vertically and each column is the full width of the device (see [Table 1-1](#)).

Table 1-1. Responsive media queries

Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and up	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

To add custom CSS based on the media query, you can either include all rules in one CSS file via the media queries below, or use entirely different CSS files:

```
/* Large desktop */
@media (min-width: 1200px) { ... }

/* Portrait tablet to landscape and desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }

/* Landscape phone to portrait tablet */
@media (max-width: 767px) { ... }

/* Landscape phones and down */
@media (max-width: 480px) { ... }
```

For a larger site, you might want to divide each media query into a separate CSS file. In the HTML file, you can call them with the `<link>` tag in the head of your document. This is useful for keeping file sizes smaller, but it does potentially increase the HTTP requests if the site is responsive. If you are using LESS to compile the CSS, you can have them all processed into one file:

```
<link rel="stylesheet" href="base.css" />
<link rel="stylesheet" media="(min-width: 1200px)" href="large.css" />
<link rel="stylesheet" media="(min-width: 768px) and (max-width: 979px)"
      href="tablet.css" />
<link rel="stylesheet" media="(max-width: 767px)" href="tablet.css" />
<link rel="stylesheet" media="(max-width: 480px)" href="phone.css" />
```


Q10a) Create a form using Optional form layouts of bootstrap

Search form

Add `.form-search` to the `<form>` tag, and then add `.search-query` to the `<input>` for an input box with rounded corners and an inline submit button (see [Figure 2-20](#)):

```
<form class="form-search">
  <input type="text" class="input-medium search-query">
  <button type="submit" class="btn">Search</button>
</form>
```

A screenshot of a search form. It consists of a single horizontal line containing a rounded rectangular text input field on the left and a rectangular button labeled "Search" on the right.

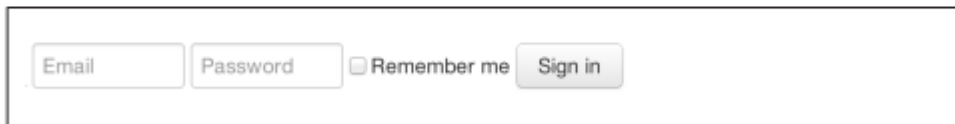
Figure 2-20. Search form

Inline form

To create a form where all of the elements are inline and labels are alongside, add the class `.form-inline` to the `<form>` tag (see [Figure 2-21](#)). To have the label and the input on the same line, use this inline form code:

```
<form class="form-inline">
  <input type="text" class="input-small" placeholder="Email">
  <input type="password" class="input-small" placeholder="Password">

  <label class="checkbox">
    <input type="checkbox" /> Remember me
  </label>
  <button type="submit" class="btn">Sign in</button>
</form>
```

A screenshot of an inline form. It contains four elements arranged horizontally: a text input field with the placeholder "Email", a password input field with the placeholder "Password", a checkbox followed by the text "Remember me", and a rectangular button labeled "Sign in".

Horizontal form

Bootstrap also comes with a prebaked horizontal form; this one stands apart from the others not only in the amount of markup, but also in the presentation of the form. Traditionally you'd use a table to get a form layout like the one shown in [Figure 2-22](#), but Bootstrap manages to do it without using tables. Even better, if you're using the responsive CSS, the horizontal form will automatically adapt to smaller layouts by stacking the controls vertically.

To create a form that uses the horizontal layout, do the following:

- Add a class of `.form-horizontal` to the parent `<form>` element.
- Wrap labels and controls in a `<div>` with class `.control-group`.
- Add a class of `.control-label` to the labels.
- Wrap any associated controls in a `<div>` with class `.controls` for proper alignment.



Figure 2-22. Horizontal form

```
<form class="form-horizontal">
  <div class="control-group">
    <label class="control-label" for="inputEmail">Email</label>

    <div class="controls">
      <input type="text" id="inputEmail" placeholder="Email">
    </div>
  </div>
  <div class="control-group">
    <label class="control-label" for="inputPassword">Password</label>
    <div class="controls">
      <input type="password" id="inputPassword" placeholder="Password">
    </div>
  </div>
  <div class="control-group">
    <div class="controls">
      <label class="checkbox">
        <input type="checkbox"> Remember me
      </label>
      <button type="submit" class="btn">Sign in</button>
    </div>
  </div>
</form>
```

Q10b) Explain pre appended input controls with example

Prepended and appended inputs

By adding prepended and appended content to an input field, you can add common elements to the user's input (see [Figure 2-28](#)). For example, you can add the dollar symbol, the @ for a Twitter username, or anything else that might be common for your application interface. To add extra content before the user input, wrap the prepended input in a `<div>` with class `.input-prepend`. To append input, use the class `.input-append`. Then, within that same `<div>`, place your extra content inside a `` with an `.add-on` class, and place the `` either before or after the `<input>` element:

```
<div class="input-prepend">
  <span class="add-on">@</span>
  <input class="span2" id="prependedInput" type="text" placeholder="Username">
</div>
<div class="input-append">
  <input class="span2" id="appendedInput" type="text">
  <span class="add-on">.</span>
</div>
```



The figure displays two examples of input fields. The first example is a text input field with a placeholder text "Username" and a small grey button with an "@" symbol to its left. The second example is a text input field with a small grey button with ".00" to its right.

Figure 2-28. Prepend and append

If you combine both of them, you simply need to add both the `.input-prepend` and `.input-append` classes to the parent `<div>` (see [Figure 2-29](#)):

```

<div class="input-prepend input-append">
  <span class="add-on">${</span>
  <input class="span2" id="appendedPrependedInput" type="text">
  <span class="add-on">.00</span>
</div>

```

Figure 2-29. Using both the append and prepend

Rather than using a ``, you can instead use `<button>` with a class of `.btn` to attach (surprise!) a button or two to the input (see Figure 2-30):

```

<div class="input-append">
  <input class="span2" id="appendedInputButtons" type="text">
  <button class="btn" type="button">Search</button>
  <button class="btn" type="button">Options</button>
</div>

```

Figure 2-30. Attach multiple buttons to an input

If you are appending a button to a search form, you will get the same nice rounded corners that you would expect (see Figure 2-31):

```

<form class="form-search">
  <div class="input-append">
    <input type="text" class="span2 search-query">
    <button type="submit" class="btn">Search</button>
  </div>
  <div class="input-prepend">
    <button type="submit" class="btn">Search</button>
    <input type="text" class="span2 search-query">
  </div>
</form>

```

Figure 2-31. Append button to search form