

USN

LCR20MCO85

20MCA41

Fourth Semester MCA Degree Examination, July/August 2022
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 various scalar types and operators available in PHP. (10 Marks)
b. Explain the various looping statements in PHP. Write any PHP code to explain 'for' loop. (10 Marks)

OR

- 2 a. Outline file handling in PHP. (10 Marks)
b. Illustrate steps to join web form with database in PHP. (06 Marks)
c. Write a short note on Cookies in PHP. (04 Marks)

Module-2

- 3 a. Illustrate with syntax any eight built-in-methods for arrays in Ruby. (08 Marks)
b. Explain codeblocks and iterators with sample code in Ruby. (06 Marks)
c. Summarize the following string methods i) Capitalize ii) Strip iii) Chop. (06 Marks)

OR

- 4 a. With a neat diagram, explain directory structure for Rails application. (10 Marks)
b. Illustrate the concept of classes in Ruby with an example. (10 Marks)

Module-3

- 5 a. What is Ajax? Explain with an diagram, how it is different from traditional web applications. (10 Marks)
b. With a neat diagram, explain Ajax web application model. (10 Marks)

OR

- 6 a. With an example illustrate sending data to server using methods of GET and POST for XMLHttpRequest object. (10 Marks)
b. List and explain Ajax principles. (06 Marks)
c. Differentiate between synchronous and Asynchronous requests. (04 Marks)

Module-4

- 7 a. Explain how to handle multiple XMLHttpRequest objects in the same page. (08 Marks)
b. Explain the pattern of predictive fetch using Ajax. (08 Marks)
c. Write a note on submission throttling. (04 Marks)

OR

- 8 a. With example, explain pattern of periodic refresh and fall back pattern in Ajax. (10 Marks)
b. Illustrate how to implement the Ajax pattern of multistage download. (10 Marks)

Module-5

- 9 a. What is Bootstrap? Explain the Bootstrap life structure with a neat diagram and give an example of basic HTML template using Bootstrap. (10 Marks)
b. With a sample code explain Fluid Grid system and container layouts. (10 Marks)

OR

- 10 a. Explain with an example, the Bootstrap Button classes for different styles of Button in Bootstrap CSS. (10 Marks)
b. Explain the following HTML concepts with Bootstrap CSS, i) Headings ii) Emphasis classes. (10 Marks)

... will be treated as malpractice.

1 a) Explain various scalar types and operators available in PHP.

Data Types

PHP provides four scalar types namely *Boolean*, *integer*, *double* and *string* and two compound types namely *array* and *object* and two special types namely *resource* and *NULL*.

PHP has a single integer type, named *integer*. This type is same as *long* type in C. The size of an integer type is generally the size of word in the machine. In most of the machines that size will be 32 bits.

PHP's *double* type corresponds to the *double* type in C and its successors. Double literals can contain a decimal point, an exponent or both. An exponent is represented using E or e followed by a signed integer literal. Digits before and after the decimal point are optional. So, both .12 and 12. are valid double literals.

String is a collection of characters. There is no special type for characters in PHP. A character is considered as a string with length 1. String literals are represented with single quotes or double quotes. In a string literal enclosed in single quotes, escape sequences and variables are not recognized and no substitutions occurs. Such substitution is known as *interpolation*. In string literals enclosed in double quotes, escape sequence and variables are recognized and corresponding action is taken.

The only two possible values for a *Boolean* type are *TRUE* and *FALSE* both of which are case-insensitive. Integer value 0 is equal to Boolean *FALSE* and anything other than 0 is

equal to *TRUE*. An empty string and string "0" are equal to Boolean *FALSE* and remaining other strings are equal to *TRUE*. Only double value equal to Boolean *FALSE* is 0.0.

PHP Operators:

Operators are used in expressions to perform operations on operands. There are several operators supported by PHP which are categorized into following categories:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

Arithmetic Operators

PHP arithmetic operators are used along with numbers to perform operations like addition, subtraction, multiplication etc. Below is a list of arithmetic operators:

Operator	Name	Example	Description
+	Addition	$\$x + \y	Sum of x and y
-	Subtraction	$\$x - \y	Difference of x and y
*	Multiplication	$\$x * \y	Product of x and y
/	Division	$\$x / \y	Quotient of x divided by y
%	Modulus	$\$x \% \y	Remainder of x divided by y
**	Exponentiation	$\$x ** \y	Result of x raised to the power of y

Assignment Operators

PHP assignment operators are used in assignment expressions to store the value of expression in to a variable. Below is a list of assignment operators:

Assignment	Same as	Description
$x = y$	$x = y$	Assigning value of y to x
$x += y$	$x = x + y$	Adding x and y and store the result in x
$x -= y$	$x = x - y$	Subtracting y from x and store the result in x
$x *= y$	$x = x * y$	Multiplying x and y and store the result in x
$x /= y$	$x = x / y$	Dividing x by y and store the quotient in x
$x \% = y$	$x = x \% y$	Dividing x by y and store the remainder in x

Increment/Decrement Operators

The increment/decrement operators are used to increment the value of variable by 1 or decrement the value of variable by 1. The increment operator is ++ and decrement operator is --.

Relational or Comparison Operators

PHP comparison operators are used to compare two values and are frequently seen in Boolean expressions. Below is a list of comparison operators:

Operator	Name	Example	Description
==	Equal	\$x == \$y	Returns true if x and y are equal
===	Identical	\$x === \$y	Returns true if x and y are equal and of same type
!=	Not equal	\$x != \$y	Returns true if x and y are not equal
!==	Not identical	\$x !== \$y	Returns true if x and y are not equal and of same type
<	Less than	\$x < \$y	Returns true if x is less than y
<=	Less than or equal to	\$x <= \$y	Returns true if x is less than or equal to y
>	Greater than	\$x > \$y	Returns true if x is greater than y
>=	Greater than or equal to	\$x >= \$y	Returns true if x is greater than or equal to y
<>	Not equal	\$x <> \$y	Returns true if x and y are not equal

Logical Operators

PHP logical operators are used to find the Boolean value of multiple conditional expressions. Below is a list of logical operators:

Operator	Name	Example	Description
and	And	\$x and \$y	Returns true when both x and y are true
or	Or	\$x or \$y	Returns true when either x or y or both of them are true
xor	Xor	\$x xor \$y	Returns true when either x or y is true
&&	And	\$x && \$y	Returns true when both x and y are true
	Or	\$x \$y	Returns true when either x or y or both of them are true
!	Not	!\$x	Returns true when x is false and vice versa

String Operators

PHP provides two operators which are used with strings only. They are listed below:

Operator	Name	Example	Description
.	Concatenation	\$str1.\$str2	str1 and str2 are concatenated
.=	Concatenation Assignment	\$str1.= \$str2	str2 is appended to str1

Array Operators

Below is a list of operators which are used with arrays:

Operator	Name	Example	Description
==	Equality	\$x==\$y	Returns true if x and y have the same key-value pairs
===	Identity	\$x=== \$y	Returns true if x and y have the same key-value pairs in same order and are of same type
!=	Inequality	\$x!=\$y	Returns true if x and y are not equal
!==	Non-Identity	\$x!== \$y	Returns true if x and y are not identical
<>	Inequality	\$x<>\$y	Returns true if x and y are not equal
+	Union	\$x+\$y	Returns union of x and y

1b) Explain various looping statements in php. Write any PHP code to explain 'for' loop.

Iteration or Loop Statements

The iteration statements in PHP allows PHP processor to iterate over or repeat a set of statements for a finite or infinite times. Iteration statements supported by PHP are while, do-while, for and foreach.

Syntax of *while* loop is given below:

```
while(condition / expression)
{
    statements(s);
}
```

Syntax of *do-while* loop is given below:

```
do
{
    statement(s);
}
while(condition / expression);
```

Syntax of *for* loop is given below:

```
for(initialization; condition / expression; increment/decrement)
{
    statement(s);
}
```

The *foreach* loop is used to iterate over array elements and its syntax is given below:

```
//For normal arrays
foreach(array as variable_name)
{
    statement(s);
}
```

or

```
//For associative arrays
foreach(array as key => value)
{
    statement(s);
}
```

Jump Statements

The jump statements available in PHP are *break* and *continue*. The break statement is used to break the control from a loop, take it to the next statement after the loop and continue is used to skip the control from current line to the next iteration of the loop.

```

<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1 //EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- powers.php
      An example to illustrate loops and arithmetic
-->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> powers.php </title>
  </head>
  <body>
    <table border = "border">
      <caption> Powers table </caption>
      <tr>
        <th> Number </th>
        <th> Square Root </th>
        <th> Square </th>
        <th> Cube </th>
        <th> Quad </th>
      </tr>
      <?php
        for ($number = 1; $number <=10; $number++) {
          $root = sqrt($number);
          $square = pow($number, 2);
          $cube = pow($number, 3);
          $quad = pow($number, 4);
          print("<tr align = 'center'> <td> $number </td>");
          print("<td> $root </td> <td> $square </td>");
          print("<td> $cube </td> <td> $quad </td> </tr>");
        }

        ?>
      </table>
    </body>
  </html>

```

2a) Outline file handling in php

Because PHP is a server-side technology, it is able to create, read, and write files on the server system. In fact, PHP can deal with files residing on any server system on the Internet, using both HTTP and FTP protocols.

11.11.1 Opening and Closing Files

The first step in some file operations is to open it, a process that prepares the file for use and associates a program variable with the file for future reference. This program variable is called the *file variable*. The `fopen` function performs these operations. It takes two parameters: the filename, including the path to it if it is in a different directory, and a use indicator, which specifies the operation or operations that must be performed on the file. Both parameters are given as strings. The `fopen` function returns the reference to the file for the file variable. Every open file has an internal pointer that is used to indicate where the next file operation should take place within the file. We call this pointer the *file pointer*. Table 11.4 describes the possible values of the use indicator.

It is possible for the `fopen` function to fail—for example, if an attempt is made to open a file for reading but no such file exists. It would also fail if the file access permissions did not allow the requested use of the file. The `fopen` function returns `FALSE` if it fails. PHP includes the `die` function, which produces a message and stops the interpretation process. The `die` function is often used

"r"	Read only. The file pointer is initialized to the beginning of the file.
"r+"	Read and write an existing file. The file pointer is initialized to the beginning of the file; if a read operation precedes a write operation, the new data is written just after where the read operation left the file pointer.
"w"	Write only. Initializes the file pointer to the beginning of the file; creates the file if it does not exist.
"w+"	Read and write. Initializes the file pointer to the beginning of the file; creates the file if it does not exist. Always initializes the file pointer to the beginning of the file before the first write, destroying any existing data.
"a"	Write only. If the file exists, initializes the file pointer to the end of the file; if the file does not exist, creates it and initializes the file pointer to its beginning.
"a+"	Read and write a file, creating the file if necessary; new data is written to the end of the existing data.

with input and output operations, which sometimes fail. For example, the following statement attempts to open a file named `testdata.dat` for reading only, but calls `die` if the open operation fails:

```
$file_var = fopen("testdata.dat", "r") or  
            die ("Error - testdata.dat cannot be opened");
```

This form appears a bit odd, but is exactly what is needed. Because the `or` operator has lower precedence than a function call, `die` will only be called if `fopen` fails, in which case `fopen` returns `FALSE`.

The problem of `fopen` failing because the specified file does not exist can be avoided by determining whether the file exists with `file_exists` before calling `fopen`. The `file_exists` function takes a single parameter, the file's name. It returns `TRUE` if the file exists, `FALSE` otherwise.

A file is closed with the `fclose` function, which takes a file variable as its only parameter.

operation stops when either the end-of-file marker is read or the specified number of bytes has been read.

Large collections of data are often stored in database systems, so usually only smaller data sets are stored in files. Therefore, files are often read in their entirety with a single call to `fread`. If the whole file is to be read at once, the file's length is given as the second parameter to `fread`. The best way to get the correct file length is with the `filesize` function, so a call to `filesize` is often used as the second parameter to `fread`. The `filesize` function takes a single parameter, the name of the file (not the file variable). For example, to read the entire contents of the file `testdata.dat` as a string into the variable `$file_string`, the following statement could be used:

```
$file_string = fread($file_var,  
                    filesize("testdata.dat"));
```

One alternative to `fread` is `file`, which takes a filename as its parameter and returns an array of all of the lines of the file. (A line is a string of non-newline characters, followed by a newline.) One advantage of `file` is that the file open and close operations are not necessary. For example, the following statement places the lines of `testdata.dat` into an array named `@file_lines`:

```
$file_lines = file("testdata.dat");
```

PHP has another file input function that does not require calling `fopen`, `file_get_contents`, which takes the file's name as its parameter. This function reads the entire contents of the file. For example, consider the following call:

```
#file_string = file_get_contents("testdata.dat");
```

A single line of a file can be read with `fgets`, which takes two parameters: the file variable and a limit on the length of the line to be read. Consider the following statement:

```
$line = fgets($file_var, 100);
```

This statement reads characters from `testdata.dat` until it finds a newline character, encounters the end-of-file marker, or has read 99 characters. Note that the maximum number of characters `fgets` reads is one fewer than the limit given as its second parameter.

A single character can be read from a file with `fgetc`, whose only parameter is the file variable. When reading a file by lines or by characters, the read operation must be controlled by the detection of the end of the file. This can be done with the `feof` function, which takes a file variable as its only parameter. It returns a Boolean value: `TRUE` if the last read character of the file was the end-of-file character, `FALSE` otherwise.

11.11.3 Writing to a File

The `fwrite`⁸ function takes two parameters: a file variable and the string to be written to the file. It is possible to include a third parameter, which would be used to specify the number of bytes to be written. This parameter is rarely needed. The `fwrite` function returns the number of bytes written. The following is an example of a call to `fwrite`:

```
$bytes_written = fwrite($file_var, $out_data);
```

This statement writes the string value in `$out_data` to the file referenced with `$file_var` and places the number of bytes written in `$bytes_written`. Of course, this will work only if the file has been opened for writing.

The `file_put_contents` function is the counterpart of `file_get_contents`—it writes the value of its second parameter, a string, to the file specified in its first parameter. For example, consider the following call:

```
file_put_contents("savedata.dat", $str);
```

11.11.4 Locking Files

If it is possible for more than one script to access a file at the same time, the potential interference of those accesses can be prevented with a file lock. The lock prevents any other access to the file while the lock is set. Scripts that use such files lock them before accessing them and unlock them when the access is completed. File locking is done in PHP with the `flock` function, which should sound familiar to UNIX programmers. The `flock` function takes two parameters: the file variable of the file and an integer that specifies the particular operation. A value of 1 specifies that the file can be read by others while the lock is set, a value of 2 allows no other access, and a value of 3 unlocks the file.

2b) Illustrate steps to join web form with database in PHP.

13.6.2 Connecting to MySQL and Selecting a Database

The PHP function `mysql_connect` connects a script to a MySQL server. This function takes three parameters, all of which are optional. The first is the host that is running MySQL; the default is `localhost` (the machine on which the script is running). The second parameter is the username for MySQL; the default is the username in which the PHP process runs. The third parameter is the password for the database; the default is blank (works if the database does not require a password). For example, if the default parameters were acceptable, we could use the following:

```
$db = mysql_connect();
```

Of course, the connect operation could fail, in which case the value returned would be `false` (rather than a reference to the database). Therefore, the call to `mysql_connect` usually is used in conjunction with `die`.

The connection to a database is terminated with the `mysql_close` function. This function is not necessary when using MySQL through a PHP script because the connection will be closed implicitly when the script terminates.

When running MySQL from the command line, a database must be selected as the current, or focused, database. This is also necessary when using MySQL through PHP; it is accomplished with the `mysql_select_db` function, as shown in the following:

```
mysql_select_db("cars");
```

13.6.3 Requesting MySQL Operations

MySQL operations are requested through the `mysql_query` function. Typically, the operation, in the form of a string literal, is assigned to a variable. Then `mysql_query` is called with the variable as its parameter. For example:

```
$query = "SELECT * from Corvettes";  
$result = mysql_query($query);
```

The return value from `mysql_query` is used to identify, internally, the data that resulted from the operation. In most cases, the first thing to do with the result is to determine the number of rows. This is obtained with the `mysql_num_rows` function, which is given the result value returned by `mysql_query`, as shown in the following:

```
$num_rows = mysql_num_rows($result);
```

The number of fields in a result row can be determined with `mysql_num_fields`, as shown in the following:

```
$num_fields = mysql_num_fields($result);
```

The rows of the result can be retrieved into several different forms. We will use `mysql_fetch_array`, which returns an array of the next row. Then the field values can be obtained by subscripting the return array with the column names. For example, if the result of a query had columns for `State_id` and `State`, we could display the results with the following code:

```
$num_rows = mysql_num_rows($result);  
  
for ($row_num = 1; $row_num <= $num_rows; $row_num++) {  
    $row = mysql_fetch_array($result);  
    print "<p> Result row number" . $row_num .  
        ". State_id: ";  
    print htmlspecialchars($row["State_id"]);  
    print " State: ";  
    print htmlspecialchars($row["State"]);  
    print "</p>";  
}
```

The situation in which the column names are not known is considered in Section 13.6.4, which includes a complete example of accessing a database through PHP and MySQL.

2c) Write short note on Cookies in PHP.

- Session is a time span during which browser interacts with a particular server.
- Session begins when a browser becomes connected to a server
- It ends when the browser ceases to be connected to that server because either it becomes connected to a different server or it is terminated.
- The HTTP protocol is essentially stateless—it includes no means to store information about a session that is available to a subsequent session. However using Cookies and session HTTP can be made statefull.

Cookies

- 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"]
```

3a) Illustrate with syntax any eight built in methods for arrays in Ruby.

Built-in methods

- Shift , unshift and pop , push

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]

Concat

```
>> 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]
```

```
>> 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]
```

Reverse

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

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

Include?

```
=> [2, 4, 8, 16]
>> list.include?(4)
=> true
>> list.include?(10)
=> false
```

Sort

```
>> list = [16, 8, 4, 2]
=> [16, 8, 4, 2]
>> list.sort
=> [2, 4, 8, 16]
>> list2 = ["jo", "fred", "mike", "larry"]
=> ["jo", "fred", "mike", "larry"]
>> list2.sort
=> ["fred", "jo", "larry", "mike"]
```

```
>> set1 = [2, 4, 6, 8]
=> [2, 4, 6, 8]
>> set2 = [4, 6, 8, 10]
=> [4, 6, 8, 10]
>> set1 & set2
=> [4, 6, 8]
>> set1 - set2
=> [2]
>> set1 | set 2
=> [2, 4, 6, 8, 10]
```

3b) Explain code blocks and iterators with sample code in Ruby.

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

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

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

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

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

```
>> list = [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]
```

```
>> def get_name
```

```
>>   puts "Your name:"
```

```
>>   name = gets
```

```
>>   yield(name)
```

```
>> end
```

```
=> nil
```

```
>> get_name {|name| puts "Hello, " + name}
```

```
Your name:
```

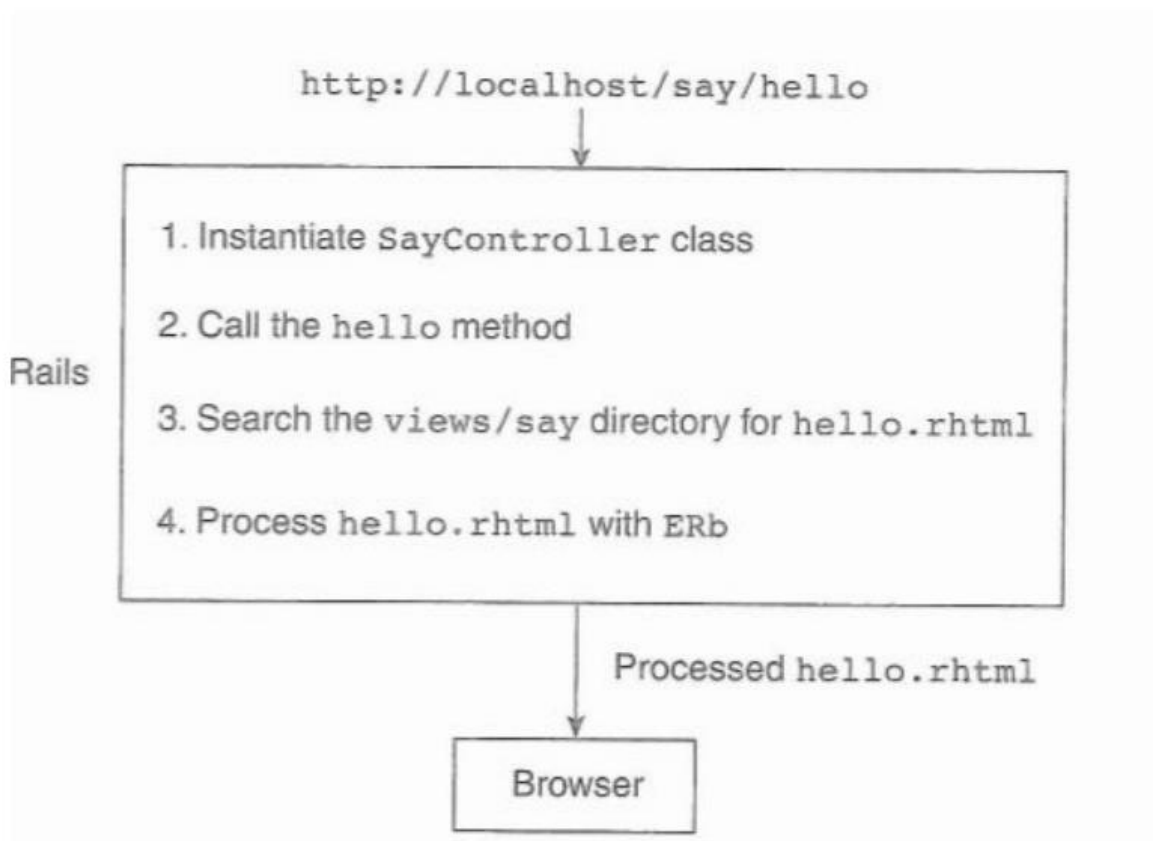
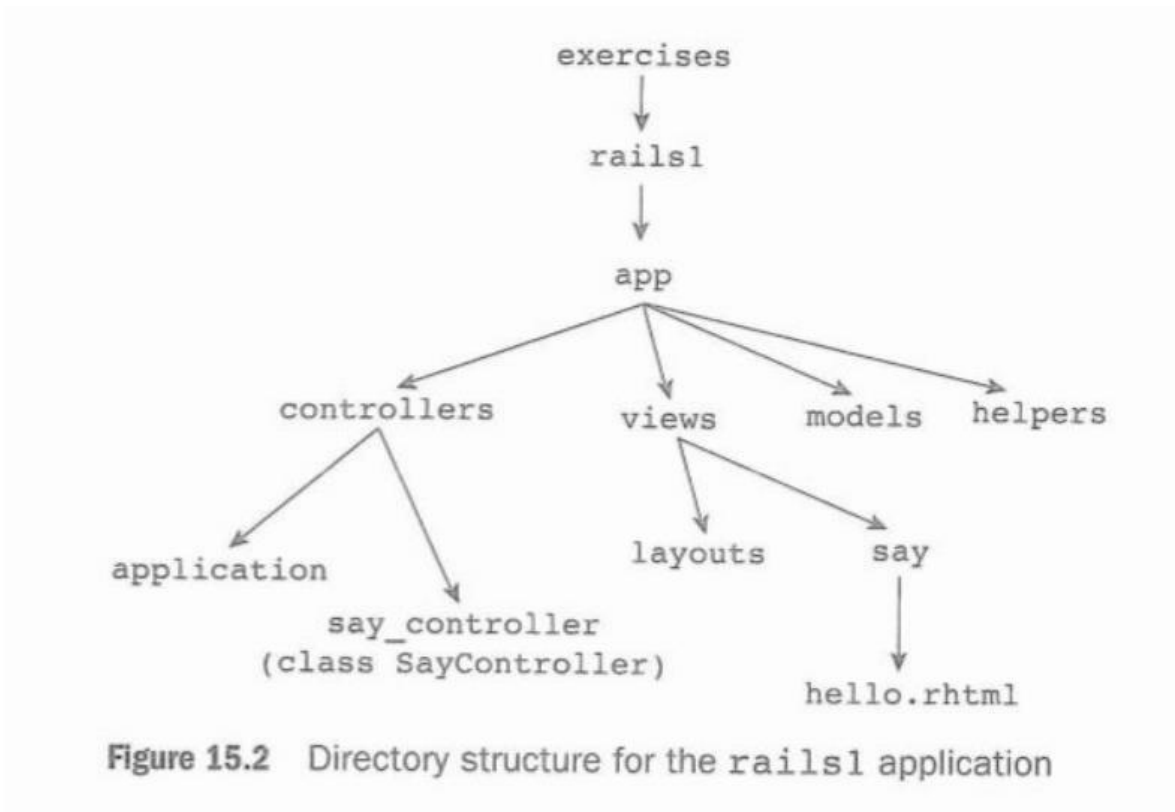
```
Freddie
```

```
Hello, Freddie
```

3c) Summarize the following string methods 1) Capitalize 2) Strip 3) chop

Method	Action
capitalize	Convert the first letter to uppercase and the rest of the letters to lowercase
chop	Removes the last character
strip	Removes the spaces on both ends

4a) with a neat diagram explain directory structure of Rails application



4 b) Illustrate the concept of Classes in Ruby with any example.

```
class Stack2_class

  def initialize(len = 100)
    @stack_ref = Array.new(len)
    @max_len = len
    @top_index = -1
  end

  # push method
  def push(number)
    if @top_index == @max_len
      puts "Error in push - stack is full"
    else
      @top_index += 1
      @stack_ref[@top_index] = number
    end
  end

  def pop()
    if @top_index == -1
      puts "Error in pop - stack is empty"
    else
      @top_index -= 1
    end
  end
end

end
```

```
mystack = Stack2_class.new(50)
mystack.push(42)
mystack.push(29)
puts "Top element is (should be 29): #{mystack.top}"
puts "Second from the top is (should be 42): #{mystack.top2}"
mystack.pop
mystack.pop
mystack.pop # Produces an error message - empty stack
```

5a) What is Ajax? Explain with a diagram, how it is different from traditional web applications.

AJAX

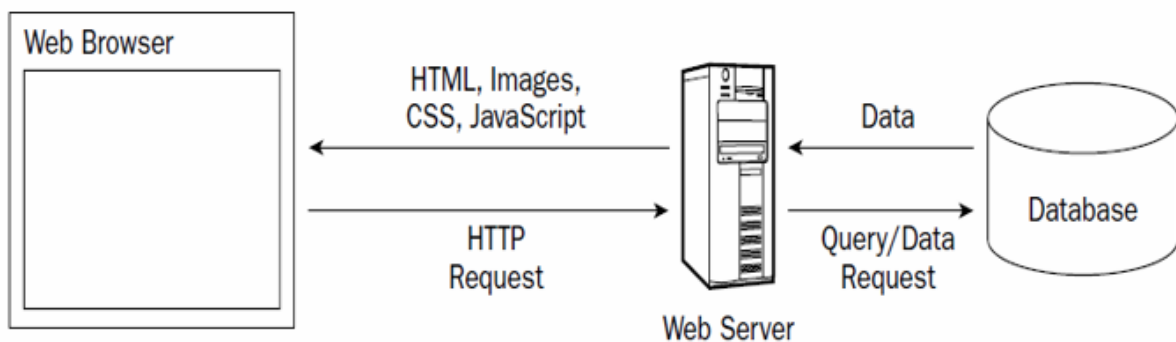
AJAX stands for – **Asynchronous JavaScript and XML**

Ajax is nothing more than an **approach to web interaction**. This approach involves transmitting only a **small amount of information** to and from the server in order to give the user the most responsive experience possible.

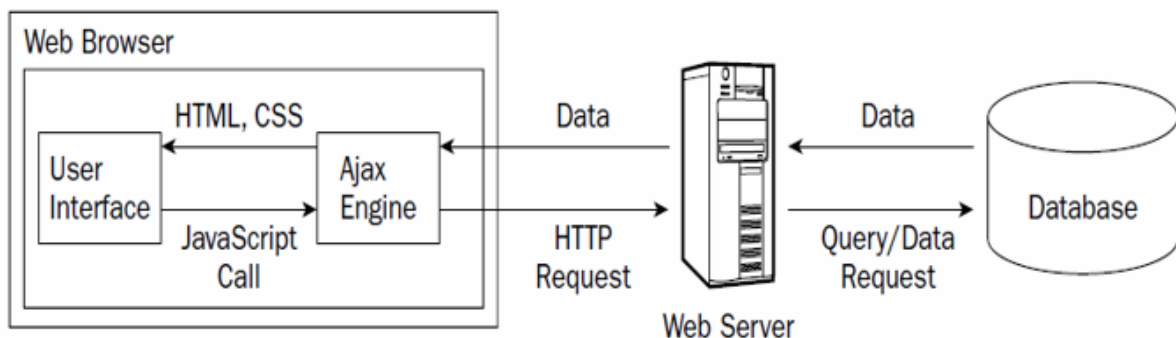
Ajax is a **set of web development techniques** using many web technologies on the client side to create **asynchronous web applications**. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behaviour of the existing page

Ajax is not a technology, but rather a programming concept.*

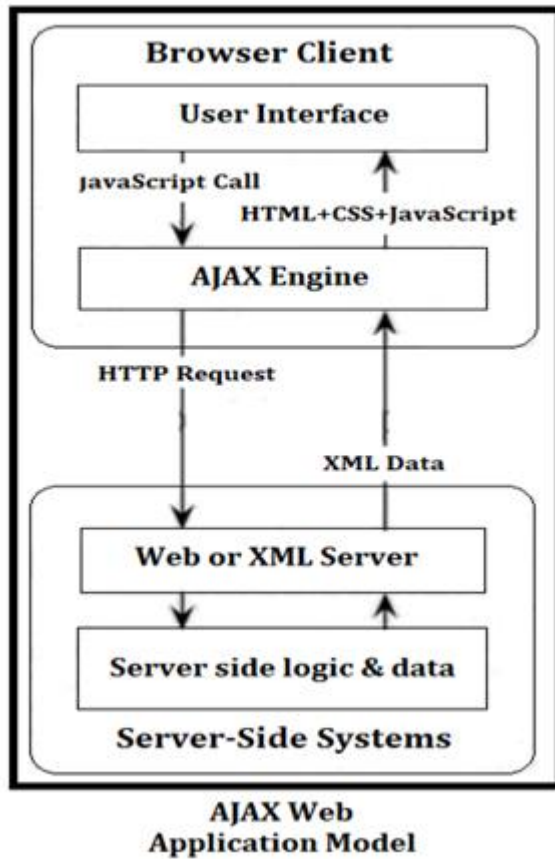
Traditional Web Application Model



Ajax Web Application Model



5b) Ajax web Application Model



6a) Write an example illustrate sending data to server using methods of GET and POST for XMLHttpRequest object.

```
<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");
      }
    </script>
  </head>
</html>
```

```

function getData(dataSource, divID){
    if(XMLHttpRequestObject) {
        var obj = document.getElementById(divID);
        XMLHttpRequestObject.open("GET", dataSource);
        XMLHttpRequestObject.onreadystatechange = function()
        {
            if (XMLHttpRequestObject.readyState == 4 &&
                XMLHttpRequestObject.status == 200) {
                obj.innerHTML =
XMLHttpRequestObject.responseText;
            }
        }
        XMLHttpRequestObject.send(null);
    }
}
</script>
</head>
<body>
    <H1>An Ajax example</H1>
    <form>
        <input type = "button" value = "Fetch the message"
            onclick = "getData('data.txt', 'targetDiv')">
    </form>
    <div id="targetDiv">
        <p>The fetched message will appear here.</p>
    </div>
</body>
</html>

```

Program Using POST

```
<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>

```

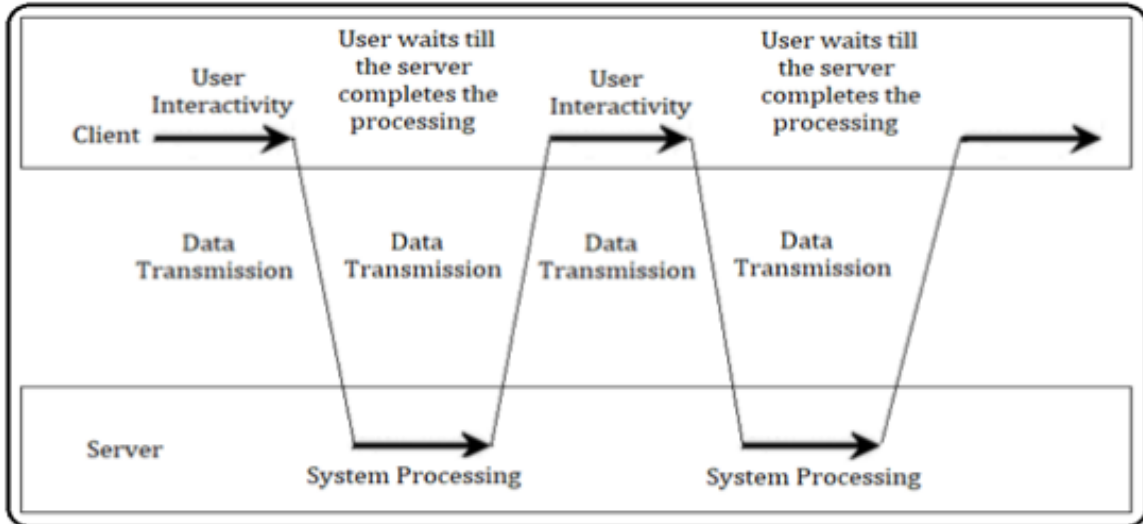
```
<H1>An Ajax example</H1>
<form>
  <input type = "button" value = "Fetch the first message"
    onclick = "getData('dataresponder.php','targetDiv',1)">
  <input type = "button" value = "Fetch the second message"
    onclick = "getData('dataresponder.php','targetDiv',2)">
</form>
<div id="targetDiv">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

6b) AJAX Principles:

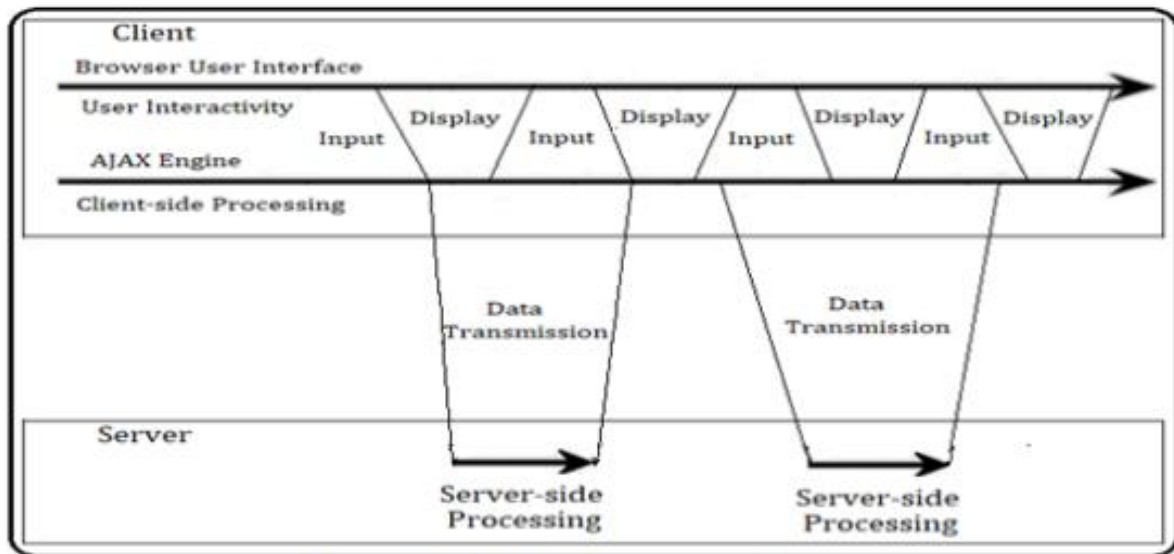
- Minimal traffic:** Ajax applications should send and receive as little information as possible to and from the server. In short, Ajax can minimize the amount of traffic between the client and the server. Making sure that your Ajax application doesn't send and receive unnecessary information adds to 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.

6c) Difference between Synchronous and asynchronous requests.



Synchronous Mode of Communication



Asynchronous Mode of Communication

7a) Explain how to handle multiple XMLHttpRequest objects in the same page.

```
<html>
<head>
<title>Sending Data to the Server</title>
<script language = "javascript">
```



```

var XMLHttpRequestObject1 = false;
if (window.XMLHttpRequest) {
XMLHttpRequestObject1 = new XMLHttpRequest();
} else if (window.ActiveXObject) {
XMLHttpRequestObject1 = new
ActiveXObject("Microsoft.XMLHTTP");
}
function getData1(dataSource, divID)
{
if(XMLHttpRequestObject1) {
var obj = document.getElementById(divID);
XMLHttpRequestObject1.open("GET", dataSource);
XMLHttpRequestObject1.onreadystatechange = function()
{
if (XMLHttpRequestObject1.readyState == 4 &&
XMLHttpRequestObject1.status == 200) {
obj.innerHTML = XMLHttpRequestObject1.responseText;
}
}
XMLHttpRequestObject1.send(null);
}
}
var XMLHttpRequestObject2 = false;
if (window.XMLHttpRequest) {
XMLHttpRequestObject2 = new XMLHttpRequest();
} else if (window.ActiveXObject) {
XMLHttpRequestObject2 = new
ActiveXObject("Microsoft.XMLHTTP");
}
function getData2(dataSource, divID)
{

```

```

if(XMLHttpRequestObject2) {
var obj = document.getElementById(divID);
XMLHttpRequestObject2.open("GET", dataSource);
XMLHttpRequestObject2.onreadystatechange = function()
{
if (XMLHttpRequestObject2.readyState == 4 &&
XMLHttpRequestObject2.status == 200) {
obj.innerHTML = XMLHttpRequestObject2.responseText;
}
}
XMLHttpRequestObject2.send(null);
}
}
</script>
</head>
<body>
<h1>Sending Data to the Server</h1>
<form>
<input type = "button" value = "Fetch message 1"
onclick = "getData1('dataresponder1.php?data=1', 'targetDiv')">
<input type = "button" value = "Fetch message 2"
onclick = "getData2('dataresponder1.php?data=2', 'targetDiv')">
</form>
<div id="targetDiv">
<p>The fetched message will appear here.</p>
</div>
</body>
</html>

```

7b) Explain the pattern of predictive fetch using Ajax.

- 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 user's intentions.
- Suppose user is reading an online article that is separated into three pages. It is logical to assume that if user is interested in reading the first page, then user 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.
- **Example:** This approach is taken by many web-based e-mail systems, including *Gmail* and *AOL Webmail*; During the writing of an e-mail, its general e-mail is for someone whom user knows, so it's logical to assume that the person is already in user's address book. To help out user, it may be wise to pre-load user's address book in the background and offer suggestions.

7c) Write a note on submission throttling.

- Submission throttling solves the issue of when to send user data to the server.
- In a traditional web site or web application, each click makes a request back to the server so that the server is always aware of what the client is doing.
- In the Ajax model, the user interacts with the site or application without additional requests being generated for each click.
- In a traditional web solution data send back to the server every time when user action occurs. Thus, when the user types a letter, that letter is sent to the server immediately. The process is then repeated for each letter typed.
- The problem with this approach is that it has the possibility to create a large number of requests in a short amount of time, which may not only cause problems for the server but may cause the user interface to slow down as each request is being made and processed.
- The Submission Throttling design pattern is an alternative approach to solve this problematic issue.
- Using Submission Throttling, you buffer the data to be sent to the server on the client and then send the data at predetermined times.
 - Submission Throttling typically begins either when the web site or application first loads or because of a specific user action.
 - Then, a client-side function is called to begin the buffering of data.
 - Every so often, the user's status is checked to see if user is idle or not. If the user is still active, data continues to be collected.
 - When the user is idle, means not performing any action, it's time to decide whether to send the data. This determination varies such as,

data is sent only when it reaches a certain size, or data is sent every time when the user is idle.

- After the data is sent, the application typically continues to gather data until either a server response or some user action signals to stop the data collection.
 - **Example: Google Suggest** feature does this brilliantly. It doesn't send a request after each character is typed. Instead, it waits for a certain amount of time and sends all the text currently in the text box. The delay from typing to sending has been fine-tuned to the point that it doesn't seem like much of a delay at all. Submission Throttling, in part, gives Google Suggest its speed.
-

8a) with example, explain pattern of periodic refresh and fall back pattern in Ajax.

Fallback Patterns

- We pre-suppose that everything goes according to plan on the server-side: the request is received, the necessary changes are made, and the appropriate response is sent to the client. But what happens if there's an error on the server? Or worse yet, what if the request never makes it to the server?
- When developing Ajax applications, it is imperative that you plan ahead for these problems and describe how your application should work if one of these should occur.

Cancel Pending Requests

- If an error occurs on the server, meaning a status of something other than 200 is returned, you need to decide what to do. Chances are that if a file is not found (404) or an internal server error occurred (500), trying again in a few minutes isn't going to help since both of these require an administrator to fix the problem.
- The simplest way to deal with this situation is to simply cancel all pending requests. You can set a flag somewhere in your code that says, "don't send any more requests."
- This solution has maximum impact on the **Periodic Refresh Pattern**.

Try Again

- Another option when dealing with errors is to silently keep trying for either a specified amount of time or a particular number of tries.
- Once again, unless the Ajax functionality is key to the user's experience, there is no need to notify user about the failure. It is best to handle the problems behind the scenes until it can be resolved.
- In general, the Try Again pattern should be used only when the request is intended to occur only once, as in a **Multi-Stage Download**.

8b) Illustrate how to implement the Ajax pattern of multistage download.

While establishing asynchronous communication, a set of patterns are followed to make the process more efficient. One such design pattern is Multistage Downloading.

Until now, some action of the user (like click) has triggered the process of fetching the response. This includes a small wait period from the user's side until it is fetched. But in certain cases this can be annoying. The goal is to automatically fetch data without the user having to ask for it.

Multistage Downloading is necessary when a huge amount of data needs to be downloaded. The order in which this download occurs is very crucial. The most relevant and light data must arrive first so that the user can get started. The heaviest ones should probably arrive later. The advantage of this approach is that, if the user navigates away from the page fairly quickly, then the heavy downloads are avoided.

This process should include indicators (like progress bars) to inform the user that more data is arriving. It should also have place holders which can accept this arriving data easily.

9a) What is Bootstrap? Explain the Bootstrap life structure with a neat diagram and give an example of basic HTML template using Bootstrap.

What Is Bootstrap?

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

In the earlier days of Twitter, engineers used almost any library they were familiar with to meet front-end requirements. Inconsistencies among the individual applications made it difficult to scale and maintain them. Bootstrap began as an answer to these challenges and quickly accelerated during Twitter's first Hackweek. By the end of Hackweek, we had reached a stable version that engineers could use across the company. — Mark Otto

Since Bootstrap launched in August 2011, it has taken off in popularity. It has evolved from being an entirely CSS-driven project to include a host of JavaScript plugins and icons that go hand in hand with forms and buttons. At its base, it allows for responsive web design and features a robust 12-column, 940px-wide grid. One of the highlights is the build tool on Bootstrap's website, where you can customize the build to suit your needs, choosing which CSS and JavaScript features you want to include on your site. All of this allows frontend web development to be catapulted forward, building on a stable foundation of forward-looking design and development. Getting started with Bootstrap is as simple as dropping some CSS and JavaScript into the root of your site.

Bootstrap File Structure bootstrap

```
bootstrap/  
├─ css/  
│   └─ bootstrap.css  
│   └─ bootstrap.min.css  
├─ js/  
│   └─ bootstrap.js  
│   └─ bootstrap.min.js  
└─ img/  
    └─ glyphsicons-halflings.png  
    └─ glyphsicons-halflings-white.png
```

The Bootstrap download includes three folders: css, js, and img. For simplicity, add these to the root of your project. Minified versions of the CSS and JavaScript are also included. It is not necessary to include both the uncompressed and the minified versions.

Basic HTML Template

Normally, a web project looks something like this:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Bootstrap 101 Template</title>  
  </head>  
  <body>  
    <h1>Hello, world!</h1>  
  </body>  
</html>
```

With Bootstrap, we include the link to the CSS stylesheet and the JavaScript:

```
<!DOCTYPE html>
```

```

<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Hello, world! </h1>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>

```

9b) With a sample code explain Fluid grid system and container layouts.

Fluid Grid System

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>

```

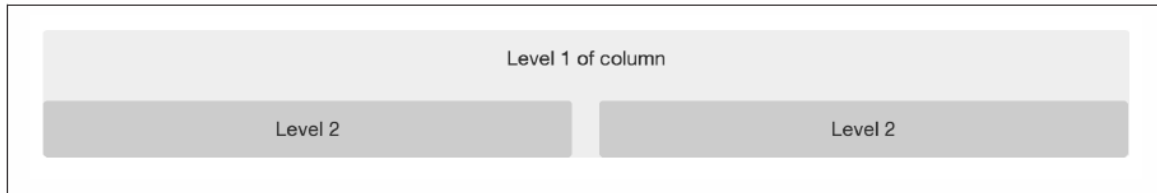
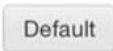









Figure 1-4. Nesting fluid grid

Container Layouts

To add a fixed-width, centered layout to your page, simply wrap the content in `<div class="container">...</div>`. If you would like to use a fluid layout but want to wrap everything in a container, use the following: `<div class="container-fluid">...</div>`. Using a fluid layout is great when you are building applications, administration screens, and other related projects.

10a) Explain with an example, the Bootstrap button classes for different styles of Button in Bootstrap CSS.

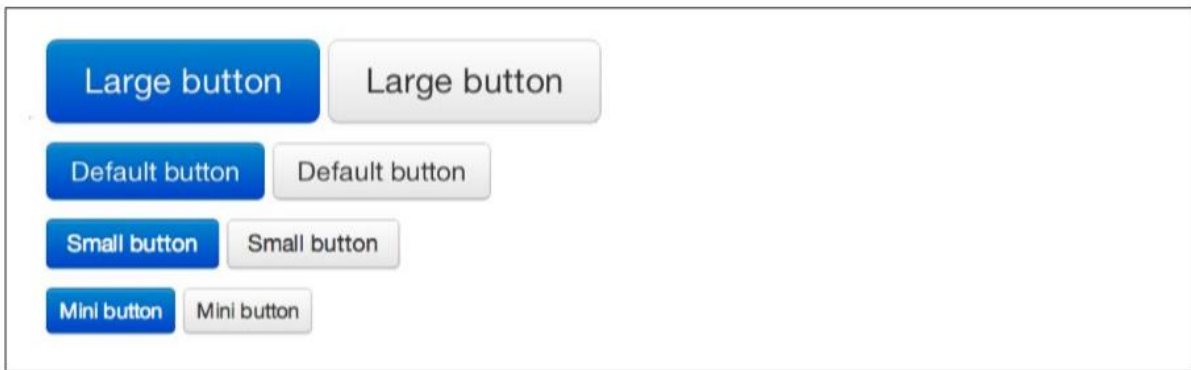
Buttons	Class	Description
	btn	Standard gray button with gradient
	btn btn-primary	Provides extra visual weight and identifies the primary action in a set of buttons (blue)
	btn btn-info	Used as an alternative to the default styles (light blue)
	btn-success	Indicates a successful or positive action (green)
	btn btn-warning	Indicates caution should be taken with this action (orange)
	btn btn-danger	Indicates a dangerous or potentially negative action (red)
	btn btn-inverse	Alternate dark-gray button, not tied to a semantic action or use
	btn btn-link	De-emphasizes a button by making it look like a link while maintaining button behavior


```

<p>
  <button class="btn btn-large btn-primary" type="button">Large button</button>
  <button class="btn btn-large" type="button">Large button</button>
</p>
<p>
  <button class="btn btn-primary" type="button">Default button</button>
  <button class="btn" type="button">Default button</button>
</p>
<p>
  <button class="btn btn-small btn-primary" type="button">Small button</button>

  <button class="btn btn-small" type="button">Small button</button>
</p>
<p>
  <button class="btn btn-mini btn-primary" type="button">Mini button</button>
  <button class="btn btn-mini" type="button">Mini button</button>
</p>

```



10b) Explain the following HTML concepts with Bootstrap CSS, 1) Headings 2) Emphasis classes.

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;
}

```

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 `` tag. This will add `font-weight:bold;` to the selected text.

Italics

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

Emphasis Classes

Along with `` and ``, 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>
```