

# **Advanced Web Programming** **(20MCA42)**

**Prepared by**  
**Prof. Jayashree J**

**Department of Master of Computer Applications**

## Department of MCA

**Subject: Advances in Web Technologies**

**Subject Code:20MCA42**

### **MODULE-1**

#### **Introduction to PHP**

Origins and uses of PHP, Overview of PHP, General syntactic characteristics, Primitives, operations and expressions, Output, Control statements, Arrays, Functions, Pattern matching.

#### **Building Web applications with PHP**

Form handling, Files, Tracking users, cookies, sessions, Using databases, Handling XML.

### **MODULE-2**

#### **Introduction to Ruby and Introduction to Rails**

Origins and uses of Ruby, Scalar types and their operations ,Simple input and output, Control statements, Arrays, Hashes, Methods, Classes, Codeblocks and iterates, Pattern matching.

**Overview of Rails**, Document requests, Processing forms, Layouts. Rails applications with Databases.

### **MODULE-3**

**Rich Internet Applications With Ajax:** Limitations of Classic Web application model, AJAX principles, Technologies behind AJAX, Examples of usage of AJAX; Asynchronous communication and AJAX application model.

#### **Ajax with XMLHttpRequest object: Part 1**

Creating Ajax Applications: An example, Analysis of example ajax.html, Creating the JavaScript, Creating and opening the XMLHttpRequest object, Data download, Displaying the fetched data, Connecting to the server, Adding Server-side programming, Sending data to the server using GET and POST.

### **MODULE-4**

#### **Ajax with XMLHttpRequest object: Part 2**

Handling multiple XMLHttpRequest objects in the same page, Using two XMLHttpRequest objects, Using an array of XMLHttpRequest objects,AJAX Patterns – Predictive Fetch, Multi-stage download, Periodic Refresh and Fallback patterns, Submission throttling.

### **MODULE-5**

**Introduction to Bootstrap.** What Is Bootstrap? Bootstrap File Structure, Basic HTML Template, Global Styles, Default Grid System, Basic Grid HTML, Offsetting Columns, Nesting Columns, Fluid Grid System, Container Layouts, Responsive Design. Typography, Emphasis Classes, Lists, Code, Tables, Optional Table Classes, Table Row Classes, Forms, Buttons, Images, Icons.

## Department of MCA

### Index

**Subject: Advances in Web Technologies**

**Subject Code:20MCA42**

Module	Contents	Page No.
<b>MODULE-1</b>	<b>Introduction to PHP and Building Web applications with PHP</b>	
	Origins and uses of PHP	2
	Overview of PHP	2
	Primitives, operations and expressions	4
	Output	7
	Control statements	8
	Array	9
	Functions	14
	Pattern matching.	16
	Form handling	17
	Files	18
	Tracking users	20
	Using databases	24
	Handling XML	27
<b>MODULE-2</b>	<b>Introduction to Ruby and Introduction to Rails</b>	
	Introduction	32
	Scalar types and Their Operations	32
	Simple Input and Output	38
	Control statements	38
	Fundamentals of Arrays	45
	Hashes	50
	Methods	51
	Class Method	55
	Code Blocks and Iterators	56

	Pattern Matching	59
	Overview of Rails	61
	Document Request	62
	Dynamic Documents	66
	Processing forms	67
	Layouts.	68
	Rails applications with Databases	69
<b>MODULE-3</b>	<b>Rich Internet Applications With Ajax</b>	
	Introduction	71
	Building Rich Internet Applications with AJAX	72
	Limitations of Classic Web application model	73
	AJAX principles	73
	Technologies behind AJAX	74
	Examples of usage of AJAX; Asynchronous communication and AJAX application model.	75
	Synchronous Vs. Asynchronous Application	75
	Ajax with XMLHttpRequest object	76
	The XMLHttpRequest Object	78
	AJAX - Send a Request To a Server	80
	The onreadystatechange Property	82
<b>MODULE-4</b>	<b>Ajax with XMLHttpRequest object: Part 2</b>	
	Handling multiple XMLHttpRequest objects in the same page	87
	Using multiple XMLHttpRequest objects:	87
	Ajax Patterns	88
	Predictive Fetch Algorithm	88
	Submission Throttling	89
	Multi-Stage Download	90
	Fall Back Patterns	90
	Creating an XHR object	95
	Cache Control	

<b>MODULE-5</b>	<b>Introduction to Bootstrap</b>	
	What is Bootstrap?	96
	Installation	97
	File Structure	98
	Basic HTML Template	98
	Grid System	99
	Offsetting columns	100
	Fluid Grid System	101
	Typography	102
	Lists	104
	Codes	105
	Tables	106
	Optional Table Classes	107
	Forms	111
	Buttons	120
	Images	122
	Icons	123

**MODULE-1**

**Introduction to PHP and Building Web applications with PHP**

- Origins and uses of PHP
- Overview of PHP
- General syntactic characteristics
- Primitives, operations and expressions
- Output, Control statements
- Arrays, Functions, Pattern matching
- Form handling, Files, Tracking users
- cookies, sessions, Using databases
- Handling XML.

## **Origins and uses of PHP**

- PHP was developed by **Rasmus Lerdorf** a member of the Apache Group.
- Originally used for tracking visits to his online resume, he named the suite of scripts "Personal Home Page Tools, more frequently referenced as "PHP Tools."
- PHP is a server-side, scripting language executed on the server.
- PHP is a widely-used, open source scripting language.

### **Uses:**

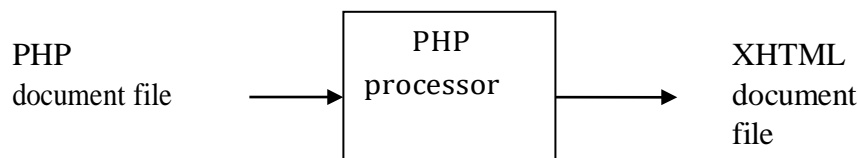
- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

### **Why PHP?**

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net)
- PHP is easy to learn and runs efficiently on the server side

## **Overview of PHP**

- When the browser requests an XHTML document that includes PHP script, the Web server determines that a document includes PHP script by the filename extensions such as .php, .php3, or .phtml and calls the PHP processor.
- PHP processor takes PHP document file as input file and produces as XHTML document file.



- PHP processor has two modes of operations:
  - Copy Mode
  - Interpret mode.





### Copy mode :

On the server side when the PHP processor finds XHTML code (which may include client side script) in the input file, it simply copies it to the output file.

### Interpret mode :

On the server side when it encounters a PHP script in the input file, it interprets it and sends any output of the script to the output file.

- The output from a PHP script must be XHTML or embedded client-side script.
- The output file is sent to the requesting browser. The client never sees the PHP script. If the user clicks view source while browser is displaying the document, only the XHTML will be shown.

### General syntactic characteristics

- PHP scripts can be embedded within the XHTML document by enclosing it between the `<?php` and `?>` tags.

A PHP script starts with **`<?php`** and ends with **`?>`**

Example:

```
<?php
// PHP Code goes here
?>
```

Comments in PHP : PHP supports several ways of commenting:

```
<!DOCTYPE html>
<html>
<body>
    <?php
        // This is a single-line comment
        # This is also a single-line comment
        /*
            This is a multiple-lines comment
            block that spans over multiple
            lines
        */
    </body>
</html>
```

## PHP's Reserved Words

and	else	global	require	virtual
break	elseif	if	return	xor
case	extends	include	static	while
class	false	list	switch	
continue	for	new	this	
default	foreach	not	true	
do	function	or	var	

## Primitives, operations and expressions

### PHP Variables :

- A variable starts with the \$ sign, followed by the name of the variable.
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
- Variable names are case-sensitive (\$age and \$AGE are two different variables)
- PHP automatically associates a data type to the variable, depending on its value.

**Example :** `isset($myVariable);`

Can be used to see, if \$myVariable is unbounded (false) or assigned a value (true)

### PHP Data types

Variables can store data of different types, and different data types can do different things. PHP supports the following data types:

- PHP has four *scalar* data types: – Boolean, integer, double and string.
- PHP has two *compound* data types: – array and object.
- PHP has two *special* types: – resource and NULL.

### Numeric Types (scalar type)

PHP has one **integer type**,

- Corresponds to C's long integer.
- It is usually 32 bits.

PHP's **double type** corresponds to C's double type.

- It can have a decimal point, exponent or both.
- These are valid doubles in PHP:

.345   345.   3.45E2   .34e-2

### String type (scalar type)

## Department of MCA

- PHP characters are single bytes(UNICODE is not supported). A single character of string length is

- String literals are defined with either single (') or double quotes (") delimiters.
- In single quoted string literals, escape characters such as \n, are not recognized as anything special and the values of embedded variables are not substituted.
- To have variable values and escape sequences used in a string, enclose them in double quotes.
- In double-quoted string literals, escape sequences are recognized and embedded variables are replaced by their current values.

Example :

<pre>\$sum=10; print 'the total \n is:\$sum'; print "the total is:\$sum";</pre>	<pre>Output: the total \n is :\$sum the total is :10</pre>
---	--

### Boolean Type (scalar type)

- Boolean values are either TRUE or FALSE, both of which are case insensitive.
- Other scalar types in boolean context:
- Integer values of 0 are interpreted as false; and others as true.
- Empty Strings or string "0" are interpreted false; and others are true.
- Doubles that are exactly 0.0 are false; others are true.

### Arithmetic Operators and Expressions

- PHP uses the usual operators (+, -, \*, /, %, ++, --)
- If both operands are integer,
  - + , - and \* will produce integer values.
- / will only produce an integer value if the quotient is integer; otherwise it is coerced to double.
- If either operand is double, the result is double.
- % expects integer operands and will coerce to integer if necessary.

Operator	Name	Example	Result
+	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 and \$y
%	Modulus	\$x % \$y	Remainder of \$x divided by \$y
**	Exponentiation	\$x ** \$y	Result of raising \$x to the \$y'th power

**Predefined Functions in PHP**

Function	Parameter Type	Returns
<b>floor</b>	Double	Largest integer $\leq$ parameter
<b>ceil</b>	Double	Smallest integer $\geq$ parameter
<b>round</b>	Double	Nearest integer
<b>srand</b>	Integer	Initializes a random number generator with the parameter
<b>rand</b>	Two numbers	A pseudorandom number $\text{firstParam} < \text{rand} <$
<b>abs</b>	Number	Absolute value of the parameter
<b>min</b>	One or more numbers	Smallest
<b>max</b>	One or more numbers	Largest

**String Operations:**

Strings can be treated like arrays to access individual characters by specifying its position in braces. String operators are shown below:

Operator	Name	Example	Result
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
.=	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

Examples :

<pre>&lt;?php \$txt1 = "Hello"; \$txt2 = "world!"; echo \$txt1 . \$txt2; ?&gt;</pre>	<pre>&lt;?php \$txt1 = "Hello"; \$txt2 = " world!"; \$txt1 .= \$txt2; echo \$txt1; ?&gt;</pre>	<pre>\$str = "apple"; print \$str{2};</pre>
Output: Hellow world!	Output: Hellow world!	Output: p

**Commonly Used PHP String Functions:**

Function	Parameter Type	Returns
<b>strlen</b>	A string	The # of characters in the string
<b>strcmp</b>	2 strings	= 0 if they're identical, <0 if string 1 precedes string 2; >0 if string 1 follows string 2
<b>strpos</b>	2 strings	The character position in the string-1 of the first character of string-2 if string-2 is contained within string-1; false if not
<b>strstr</b>	A string and an integer	Return A substring starting from the integer parameter. If a third parameter (an integer is specified), it specifies the substring's length.
<b>chop</b>	A string	The parameter with all trailing white space removed
<b>trim</b>	A string	The parameter with all white space removed from both ends.
<b>ltrim</b>	A string	The parameter with white space removed from the beginning
<b>strtolower</b>	A string	The parameter with all uppercase letters converted to lower case
<b>strtoupper</b>	A string	The parameter with all lowercase letters converted to upper case

### Scalar Type Conversion:

There are two types, those are,

- 1) Implicit Scalar Type Conversions
- 2) Explicit Scalar Type Conversions

#### 1) Implicit Scalar Type Conversions

- In implicit conversion, the context of the expressions determines what data type is expected or required.
- There are also numeric-string conversions, where a string with e or a period is converted to double and those without it are converted to integer. If no sign or digit, it fails and uses 0.
- Converting double to integer results in truncation.

#### 2) Explicit Scalar Type Conversions

Explicit type conversion can be done in 3 ways:

1. Casting involves putting the type name in parentheses:

Example : \$sum = 4.777;

(int) \$sum ;            // produces 4

2. An explicit conversion can be done using the functions intval, doubleval or strval

Example: intval(\$sum);            //produces        4 .

3. The settype function can convert the first parameter to the type indicated by the second parameter

Ex: settype(\$sum, "integer");    //produces 4

#### gettype:

- A program can determine the type of a variable in two ways:
- using the **gettype** function which returns the type as a string (which can also be "unknown") using one of the functions :
  - is\_int, is\_integer, is\_long ( which test for integers)
  - is\_double, is\_real and is\_float (which test for doubles)
  - is\_bool and is\_string .

## Output

With PHP, there are two basic ways to get output: **echo** and **print**.

- The **print** function is easiest way to produce unformatted output:  
**print "Apples are red <br />";**
- Double-quoted strings have their variable names replaced by their values:
- The echo statement can be used with or without parentheses: **echo** or **echo()**.

**Example: Echo “hello world”;**

## Control statements

### Relational Operators

PHP has 8 relational operators:

- The usual 6 (==, !=, >, >=, <, <=)  
    === (which is true if both operands have the same value and type)
- !== (which is the opposite of ===)

### Comparisons and Types

- If the operands do not have the same type, one is coerced into the type of the other.
- If one operand is a number and the other a string, the string is coerced and then compared to the other.
- If the string cannot be coerced, the number is coerced and then the comparison is done.
- If both operands are strings and can be converted to numbers, they are converted and then compared.  
    This can be avoided by using strcmp.

### Boolean Operators

- There are 6 Boolean operators (in order of precedence) : !, &&, ||, and, or, xor

### Selection Statements

- The control structures in PHP are very similar to C/C++/Java.
- The control expression (or condition) can be any type and is coerced to Boolean.
- The control statements include:
  - if and if-else
  - while and do..while
  - for and foreach

### if Statements :

The **if** statement in PHP is like that of C, although there is also an **elseif**. An **if** statement can include any number of **elseif** clauses. The condition can be of any type but it is coerced to Boolean.

Example:

```
if ($day == "Saturday" || $day == "Sunday")
    $today = "weekend";
Else
{
    $today = "weekday";
    $work = true;
}
```

## Loop Statements

PHP has while, for and do-while loops that works exactly like those in JavaScript, as well as a foreach statement.

**Example :**

```
$fact = 1;
$count = 1;
while ($count < $n)
{
    $count++;
    $fact *= $count;
}
```

## Switch Statements

- PHP's switch statement is the same as in JavaScript.
- The control expression and case expressions can be integer, double or string, with the case expressions coerced into the type of the control expression.
- Break is necessary to avoid falling through to the next case.

Example:

```
switch ( $borderSize )
{
    case "0": print "<table>"; break;
    case "1": print "<table border = \"1\">"; break;
    case "4": print "<table border = \"4\">"; break;
    case "8": print "<table border = \"8\">"; break;
    default: print "Error - invalid value:", " $borderSize <br />";
}
```

## Arrays

- In PHP, there are **two types** of arrays:
  - Indexed arrays
  - Associative arrays or Hashes
- Each element in the array consists of a key and value.
- If the array's logical structure is like indexed arrays, the keys can happen to be simply non-negative integers in ascending order.
- If the array's logical structure is like hashes, the keys are string and their order is based on a system-designed hashing function.



## Array Creation:

Arrays can be created in two ways:

1. By Assignment
2. By array() Construct

### 1. By Assignment:

Arrays in PHP can be created by simply assigning a value to its element . Examples:

```
$list1[0] = 17;           # if $list1 does not exist, it creates and initializes the array
$list2[] = 5;             # if Not previously defined this is the 0th element
$list2 [1] = "Today is my birthday!";
$list2 [] = 42            # stored in $list[2];
```

### 2. By array() Construct :

Arrays in PHP can be created by using the array() construct Examples:

```
$list = array(17, 24, 45, 91);
$list = array( 1 =>17, 2 => 24, 3 => 45, 4 =>91 );
$list = array( );
```

```
$list = array("Joe" => 42, "Mary" => 42, "Bif" => 17); # hashed array
```

```
$list =array("make"=>"Cessna", "model"=>C210,"year"=>1960,3=>"sold");      #mixed array
```

## Accessing Array Elements

- Individual array elements can be accessed by subscripting the key.
  - Brackets are used for both a numeric and a string key.
  - \$ages['Mary'] = 29;
- Multiple elements of an array can be assigned to different variables using the list construct:  
\$trees = array("oak", "pine", "binary");  
**list**( \$hardwood, \$softwood, \$data\_structure )= \$trees;

## Functions for Dealing with Arrays

There are many functions to deal with an array, some of the functions listed below

- count( )
- unset( )
- array\_keys( )
- array\_values( )
- array\_key\_exists( )
- is\_array( )
- in\_array( )
- sizeof( )
- implode( ) and explode( )

✓ **count( )** function: is used to return the length (the number of elements) of an array.

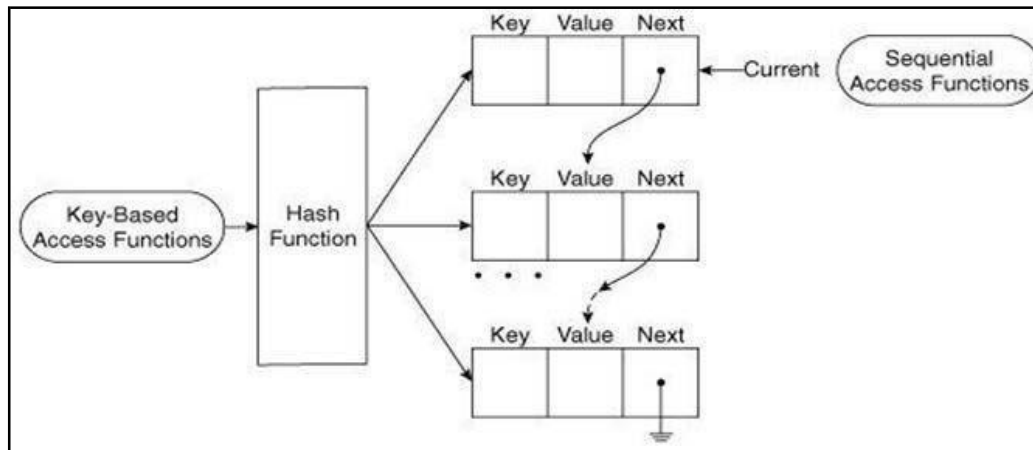
- ✓ **sizeof( )** function: returns the number of elements in an array.
- ✓ **unset( )** function: is used to delete entire array or a single element in the array.
- ✓ **array\_keys ( )** and **array\_values( )** function: is used for the keys and the values can be separately extracted from the array.
- ✓ **array\_key\_exists( )** function: The existence of an element in an array with a specific key can be determined using array\_key\_exists, which returns a Boolean.
- ✓ **is\_array( )** function: returns true if its parameter is an array.
- ✓ **in\_array( )** function: takes 2 parameters:
  - oan expression and an array.
  - oIf the expression is a value in the array, it returns true.
- ✓ **implode( )** and **explode( )** function: Strings can be converted to arrays and vice versa using explode and implode.
  - explode : allows a string to be separated into different array elements
  - implode : allows the elements of an array to be joined into a string

Examples	Outputs
<pre>&lt;?php \$scars = array("Volvo", "BMW", "Toyota"); echo count(\$scars); ?&gt;</pre>	3
<pre>&lt;?php \$list = array("Bob", "Fred", "Alan", "Bozo"); \$len = sizeof(\$list); Print "length = \$len"; ?&gt;</pre>	length = 4
<pre>&lt;?php \$list = array(2, 4, 6, 8); unset (\$list[2]); print_r( \$list); ?&gt;</pre>	2,4,8
<pre>&lt;?php  \$highs=array("Mon"=&gt;74,"Tue"=&gt;70,"Wed"=&gt;67); \$days=array_keys(\$highs); \$temps = array_values(\$highs); print " keys: "; print_r(\$days); print "Values : "; print_r(\$temps) ?&gt;</pre>	<p>keys: Array ( [0] =&gt; <b>Mon</b> [1] =&gt; <b>Tue</b> [2] =&gt; <b>Wed</b> )</p> <p>Values : Array ( [0] =&gt; <b>74</b> [1] =&gt; <b>70</b> [2] =&gt; <b>67</b> )</p>

<pre>&lt;?php     \$highs     array("Mon"=&gt;74,"Tue"=&gt;70,"Wed"=&gt;67);     if (array_key_exists("Tue", \$highs) )     { print "The key exist"; } ?&gt;</pre>	<p>The key exist</p>
<pre>&lt;?php     \$mit=array(45,78,12,34);     \$bms=25;     if(is_array(\$bms) )     { print "true"; }     else     { print "false"; } ?&gt;</pre>	<p>false</p>
<pre>&lt;?php     \$people = array("vijay", "rakesh", "venu",     "pratheek");     if (in_array("venu", \$people) )     { echo "Match found";     } else     { echo "Match not found"; } ?&gt;</pre>	<p>Match found</p>
<pre>&lt;?php     \$str = "mit mca dept";     \$words = explode(" ", \$str);     Print_r( \$words);     \$str = implode(" ", \$words);     Print_r(\$str); ?&gt;</pre>	<p>Array ( [0] =&gt; mit [1] =&gt; mca [2] =&gt; dept ) mit mca dept</p>

### Logical Internal Structure of Arrays

- ✓ Internally, the elements of an array are stored in a *linked list* of cells, Where each cell includes both key and the value of the element.
- ✓ The cell 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.
- ✓ The following figure shows the internal logical structure of an array. It shows how the two different access methods could be supported.



### Sequential Access to Array Elements

- `current( )`
  - `next( )`
  - `each( )`
  - `prev( )`
  - `key( )`
  - `array_push( )` and `array_pop( )`
  - `foreach( )`
- ✓ **`current( )`** : Every array has an internal pointer that references one element of the array, which is initially pointing to the first element. It can be accessed using the `current` function.
- ✓ **`next( )`**: `next` function moves the next pointer to the next element in the array, if it is already pointing to the last element, it returns `false`.
- ✓ **`each( )`** : it returns 2-element array, consisting of the key and "current" element's value.
  - It only returns `false` if the current pointer has gone beyond the end of the array.
  - The keys of these values are "key" and "value"
- ✓ **`prev( )`**: the function `prev` returns the value of the previous element in the array.
- ✓ **`key( )`**: the function `key` returns the key of the current element.
- ✓ **`array_push( )` and `array_pop( )`** : allow the programmer to implement a stack.
- ✓ **`foreach( )`**: allows each element in the array to be processed. There are two forms:
  - `foreach (array as scalar_var) loop body`
  - `foreach (array as key => value) loop body`

Examples	Outputs
<pre>&lt;?php \$cities = array ("Bangalore", "Chennai"); \$city = current (\$cities); print "The first city is \$city &lt;br /&gt;"; ?&gt;</pre>	<p>The first city is Bangalore</p>

<pre>&lt;?php     \$city = current (\$cities);     print "\$city &lt;br /&gt;";     while (\$city = next (\$cities))         print "\$city &lt;br /&gt;"; ?&gt;</pre>	Bangalore Chennai
<pre>&lt;?php     \$salaries=array("a"=&gt;4250,"b"=&gt;5120,         "c"=&gt;3790);     while (\$employee = each (\$salaries))     { \$name = \$employee ["key"];     \$sal = \$employee ["value"];         print ("Name salary is \$sal: &lt;br/&gt;");     } ?&gt;</pre>	a salary is 4250 b salary is 5120 c salary is 3790
<pre>&lt;?php     \$cities = array_push (\$cities,"Andhra");     print_r(\$cities); ?&gt;</pre>	Bangalore Chennai Andhra
<pre>&lt;?php     foreach (\$cities as \$temp)         print "\$temp"; ?&gt;</pre>	Bangalore Chennai Andhra
<pre>&lt;?php     \$lows = array("Mon" =&gt;32,"Tue"=&gt;36);     foreach (\$lows as \$day =&gt; \$temp)         print "Temperature on \$day was \$temp"; ?&gt;</pre>	Temperature on Mon was 32  Temperature on Tue was 36

## Functions

The general form of a function is:

```
function name ( [ parameters ] )
{
    ... ...
}
```

- The parameters are optional.
- A function can be placed anywhere in the document.
- Function overloading is not allowed.
- Functions can be nested but it is not a good idea.
- A return statement is only necessary when the function is returning a value.

### General Characteristics of a Function

- The main program passes actual parameters; the function receives formal parameters.
- The number of actual and formal parameters do not have to match.
  - Excess formal parameters are unbounded.

- Excess actual parameters are ignored.
- Parameters are passed by value. If the function needs to return more than one variable, a reference to the variable can be passed.

Pass by value	Pass by reference
<pre> &lt;?php     \$x=10;     \$y=20;     print "before swap x=". \$x ."y=" . \$y;     swap(\$x,\$y);      <b>Function</b> swap(\$x,\$y)     {         \$temp=\$x ;         \$x=\$y ;         \$y=\$temp;         print "after swap x=". \$x ."y="         . \$y;     } ?&gt; </pre>	<pre> &lt;?php     \$x=10;     \$y=20;     print "before swap x=". \$x ."y=" . \$y;     swap(\$x,\$y);     print "after swap x=". \$x ."y=" . \$y;      <b>Function</b> swap(&amp;\$x , &amp;\$y)     {         \$temp=\$x ;         \$x=\$y ;         \$y=\$temp;     } ?&gt; </pre>

## Scope of Variables

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

## Default Argument Value

The following example shows how to use a default parameter. If we call the function **setHeight()** without arguments it takes the default value as argument:

Ex:

```
<?php
function setHeight($minheight = 50)
{
    echo "The height is : $minheight <br> ";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```

## Pattern matching.

- PHP includes two different kinds of string pattern matching using regular expressions.
- They are:
  - preg\_match function
  - preg\_split function
- The **preg\_match** function takes two parameters.
- The first of which is the Perl-style regular expression as a string.
- The second parameter is the string to be search.
- The preg\_match( ) function searches string for pattern, returning true if pattern exists, and false otherwise.

Example:

<pre>&lt;?php \$college="Master of computer applications"; if(preg_match("/of/", \$college)) { print "true"; } else { print "false"; } ?&gt;</pre>	Output: true
--	--------------

- The **preg\_split** function takes two parameters.
- The first of which is a Perl-style pattern as a string.
- The second parameter is the string to be split.

**Ex:**

```
<?php
    $course="master%of%computer applications";
    $rns=preg_split("/%", $course);
    print_r($rns);
?>
```

#### OUTPUT

Array ( [0] => master [1] => of  
[2] => computer applications )

## Form handling

When PHP is used for form handling, the PHP scripts is embedded in an XHTMLdocument.

- Based on the request method, we have to use **implicit arrays** for form values, **\$\_POST** and **\$\_GET**.
- Both GET and POST create an array  
(e.g. array( key => value, key2 => value2, key3 => value3, ...)).
- This array holds key/value pairs, where keys are the names of the form elements and values are the input data from the user.
- These(\$\_GET and \$\_POST) are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- \$\_GET** is an array of variables passed to the current script via the **URL parameters**.
- \$\_POST** is an array of variables passed to the current script via the **HTTP POSTmethod**.

Form.html	Form.php
<pre>&lt;html&gt; &lt;body&gt;     &lt;form action="form.php" method="post"&gt;     Name:&lt;input type="text" name="name"&gt;&lt;br&gt;     E-mail:&lt;input type="text"     name="email"&gt;&lt;br&gt;         &lt;input type="submit" value="submit"&gt;     &lt;/form&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>&lt;html&gt; &lt;body&gt;     Welcome     &lt;?php echo \$_POST["name"]; ?&gt;&lt;br&gt;     Your email address is:     &lt;?php echo \$_POST["email"];     ?&gt; &lt;/body&gt; &lt;/html&gt;</pre>

- When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named **"form.php"**. The form data is sent with the HTTP POST method.

## Files

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

**All file operation in PHP is implemented as functions:**

- Opening and closing files
- Reading from a file
- Writing from a file
- Locking files

### 1. Opening and closing files

- fopen
- file\_exists



- fclose

### **fopen :**

- The *fopen* function is used for to open a file, it takes two parameters, the filename, including the path to it and a use indicator, which specifies the operation or operations that must be performed on the file. Both parameters are given as string.
- 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.
- The fopen function returns the reference to the file for the file variable.

Example: \$file\_var= **fopen**("count.dat","r") or die("Error – count.dat cannot be opened");

### File use indicator

Use indicator	Description
r	<b>Open a file for read only.</b> File pointer starts at the beginning of the file
r+	<b>Open a file for read/write.</b> File pointer starts at the beginning of the file
w	<b>Open a file for write only.</b> Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
w+	<b>Open a file for read/write.</b> 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	<b>Open a file for write only.</b> The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
a+	<b>Open a file for read/write.</b> Creating the file if necessary; new data is written to the end of the existing data.

### **file\_exists:**

The file\_exists function takes a single parameter, the file's name. It returns TRUE if the file exists, FALSE otherwise.

Example: **if( file\_exists("count.dat") )**

### **fclose :**

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

Example: **fclose(\$file\_var);**

## **2. Reading from a file**

- fread
- file
- file\_get\_contents
- fgets

### **fread:**

- The fread function reads part or all of a file and returns a string of what was read.
- This function takes two parameters, a file variable and the number of bytes to be read.
- The reading operation stops when the specified number of bytes has been read or when it reaches end-of-file.
- If the whole file is to be read at once. the file's length is given as the second parameter to fread.
- The filesize function takes a single parameter, the name of the file(not the file variable).  
Example: `$file_string = fread ($file_var, filesize("count.dat"));`

#### **file:**

- One alternative to fread is file, which takes a filename as its parameter and returns an array of all lines of the file.
- Advantage of file is that the file open and close operations are not necessary.  
Example: `$file_lines = file("count.dat");`

#### **file\_get\_contents :**

- This function reads the entire contents of the file, which takes the file's name its parameter.
- It does not require to call fopen function.  
Example: `$file_string1 = file_get_contents ( "count.dat" );`

#### **fgets**

- 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.  
Example: `$line = fgets($file_var, 100);`
- The above statement reads characters from count.dat until it finds a newline character, encounters the end-of-file marker, or has read 99 characters.

### **3. Writing from a file**

#### **fwrite:**

- The fwrite function takes two parameters, a file variable and the string to be written to the file.
- The fwrite function returns the number of bytes written.  
Example: `$bytes_written = fwrite ($file_var,$new_file);`
- The above statements writes the string value in \$new\_file to the file referenced with \$file\_var and places the number of bytes written in \$bytes\_written.

### **4. Locking files flock:**

- The flock is used for to lock and unlock the file for security purpose.
- Scripts that use lock before accessing the file and unlock them when the access is completed.
- The locking is done in PHP with the flock function, which is 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.

Example: `$file_lock = flock ( $file_var, 3);`

## Tracking users

Tracking Users Tracking users can be done in two ways:

1. Cookies and
2. Sessions

### Cookies

- A cookie is used to identify a user or tracking users.
- A cookie is a small text file that the server embeds on the **user's computer**.
- Each time the same computer requests a page with a browser, it will send the cookie too.
- A cookie is set in PHP with the **setcookie** function.
- This function takes one or more parameters.
- The *first parameter*, which is mandatory, is the **cookie's name** given as a string. All other parameters are optional.
- The *second*, if present, is the new **value** for the cookie, also a string.
- 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.
- 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.
- Cookies variables are set with the PHP global variable: **\$\_COOKIE**.

### Create Cookies with PHP

Example-1

```
<?php
    setcookie("mit","webclass" , time()+3600*24);           //3600 = 1hour
?>
```

Here, this call creates a cookie named mit whose value is webclass and lifetime is one day.

### Create/Retrieve a Cookie

- The following example creates a cookie named "mit" with the value "webclass ". The cookie will expire after 30 days (86400 \* 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).
- We then retrieve the value of the cookie "user" (using the global variable **\$\_COOKIE**). We also use the

isset() function to find out if the cookie is set

Example :

```
<?php
$cookie_name= "user";
$cookie_value= "JohnDoe";
setcookie($mit,$cookie_value,time()+(86400 * 30), "/"); //86400=1day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name]))
{
    echo "Cookie named '$mit' is not set!";
} else {
    echo "Cookie " . $cookie_name. " is set!<br>";
    echo "Value is: " . $_COOKIE[$mit];
}
?>

</body>
</html>
```

### Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example :

```
<?php
//
setcookie("user", "", time()- 3600);
?>
<html>
<body>

<?php

echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

### Sessions

- A session is a way to store information (in variable) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the user's computer.
- A session arrays often store a unique session ID for a session.
- Session files are usually stored in /tmp/ on the server, and named sess\_{session\_id}.
- A session is started with the **session\_start()** function.

- Session variables are set with the PHP global variable: `$_SESSION`.
- To remove all session variables use `session_unset( )`.
- To destroy the session use `session_destroy( )`.

<p><b><u>Favourites.html</u></b></p> <pre> &lt;?php     session_start( ); ?&gt; &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;     &lt;?php         // Set session variables         \$_SESSION["favcolor"] = "green";         \$_SESSION["favanimal"] = "dog";         echo "Session variables are set.";     ?&gt; &lt;/body&gt; &lt;/html&gt; </pre>	<p><b>Output:</b></p> <p>Session variables are set.</p>
--	---

### Get PHP Session Variable Values

- Session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`).
- Also notice that all session variable values are stored in the global `$_SESSION` variable:

<pre> &lt;?php     session_start(); ?&gt; &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;      &lt;?php         // Echo session variables that were set on previous example <u>Favourites.html</u>          echo "Favorite color is " . \$_SESSION["favcolor"] . " :&lt;br&gt;";         echo "Favorite animal is " . \$_SESSION["favanimal"] . " :"; ?&gt;      &lt;/body&gt; &lt;/html&gt; </pre>
<p>Output: Favorite color is : green Favorite animal is : dog</p>

- Another way to show all the session variable values for a user session is to run the following code

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
<body>
    <?php
        print_r($_SESSION);
    ?>
</body>
</html>
```

**Output:** Array ( [favcolor] => green [favanimal] => dog )

### Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset( )` and `session_destroy( )`.

```
<?php
    session_start();
?>
<!DOCTYPE html>
<html>
<body>
    <?php
        session_unset();           // remove all session variables
        session_destroy();         // destroy the session
        echo "All session variables are now removed, and the session is destroyed."
    ?>
</body>
</html>
```

**Output:** All session variables are now removed, and the session is destroyed.

### Difference between cookies and sessions

Cookies	Sessions
Cookies are client-side files that contain user information	Sessions are server-side files that contain user information
We have to set cookie max life time manually with php code with <code>setcookie</code>	Session Max life time is 1440 Seconds(24 Minutes) as defined in <code>php.ini</code> file
In php <code>\$_COOKIE</code> super global variable is used to manage cookie.	In php <code>\$_SESSION</code> super global variable is used to manage session.
You don't need to start Cookie as It is stored in your local machine.	Before using <code>\$_SESSION</code> , you have to write <code>session_start();</code>

Official MAX Cookie size is 4KB	You can store as much data as you like within in sessions. The only limits you can reach is the maximum memory a script can consume at one time, which by default is 128MB.
There is no function named unsetcookie( )	session_unset( ); is used to remove all session variables.

## Using databases

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.

What is mysql?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

### Connecting to Mysql Database:

PHP provides **mysql\_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax:

```
connection mysql_connect(server,user,passwd);
```

Sl No	Parameter and description
1	<b>Server:</b> Optional – The host name running database server. If not specified then default value is localhost:80.
2	<b>user</b> Optional – The username accessing the database. If not specified then default is the name of the user that owns the server process.
3	<b>passwd</b> Optional – The password of the user accessing the database. If not specified then default is an empty password

### Closing Database Connection

Its simplest function **mysql\_close** PHP provides to close a database connection. This function takes connection resource returned by **mysql\_connect** function. It returns TRUE on success or FALSE on

failure.

Syntax:

```
bool mysql_close ( resource $link_identifier );
```

### Creating a Database

To create and delete a database you should have admin privilege. Its very easy to create a new MySQL database. PHP uses **mysql\_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

Syntax:

```
bool mysql_query( sql, connection );
```

Sr.No	Parameter & Description
1	<b>sql</b> Required - SQL query to create a database
2	<b>connection</b> Optional - if not specified then last open connection by mysql_connect will be used.

### Selecting a Database

- Once you establish a connection with a database server then it is required to select a particular database where your all the tables are associated.
- This is required because there may be multiple databases residing on a single server and you can do work with a single database at a time.
- PHP provides function **mysql\_select\_db** to select a database. It returns TRUE on success or FALSE on failure.

Syntax:

```
bool mysql_select_db( db_name, connection );
```

SI No	Parameter and description
1	<b>db_name</b> Required - Database name to be selected
2	<b>connection</b> Optional - if not specified then last open connection by mysql_connect will be used.



## Creating Database Tables

To create tables in the new database you need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using `mysql_query()` function.

### Example:

```
CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP
)
```

### Insert data into the Database

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

### Syntax:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

### Example :

```
<?php

if(isset($_POST['t5'])) //code to Insert

{

$name=$_POST['t1'];
$address1=$_POST['t2'];
$address2=$_POST['t3'];
$email=$_POST['t4'];


$con=mysql_connect("localhost","root","");
if($con)
{
echo "db connected";
mysql_select_db("lab4");
$query1="insert into user2 values('$name','$address1','$address2', '$email')";
```

```

        mysql_query($query1);
    }

    else
    {
        echo "db not connected";

    }

}

else if(isset($_POST['t7'])) //code to search
{
    $con=mysql_connect("localhost","root","");

    if($con)
    {
        echo "db connected";
        mysql_select_db("lab4");
        $n=$_POST['t6'];

        $query1="select * from user2 where name='$n'";
        $r=mysql_query($query1);
        echo "<table border=1> <tr><th> Name</th> <th> Address1 </th> <th> Address2 </th> <th> email
            </th></tr>";
        while($s=mysql_fetch_Array($r))
        {
            echo "<tr> <td>".$s['name'].</td>";
            echo "<td>".$s['address1'].</td>";
            echo "<td>".$s['address2'].</td>";
            echo "<td>".$s['Email'].</td></tr>";
        }
    }
    else
    {
        echo "db not connected";

    }
}

```

## Handling XML.

### What is XML?

XML is a mark-up language to share the data across the web, XML is for both human read-able and machine read-able.

Types of XML:

- Tree based
- Event based

### XML Parse Extensions:

XML parse Extensions are works based on libxml. The following xml parsers are available in the php core.

- Simple XML parser
- DOM XML parser
- XML parser
- XML Reader

### Simple XML parser:

The Simple XML parser also called as tree based XML parser and it will parse the simple XML file. Simple XML parse will call `simplexml_load_file()` method to get access to the xml from specific path.

### DOM parser:

DOM Parser also called as a complex node parser, Which is used to parse highly complex XML file. It is used as interface to modify the XML file. DOM parser has encoded with UTF-8 character encoding.

### XML parse:

XML parsing is based on SAX parse. It is more faster the all above parsers. It will create the XML file and parse the XML. XML parser has encoded by ISO-8859-1, US-ASCII and UTF-8 character encoding.

### XML Reader:

XML Reader parse also called as Pull XML parse. It is used to read the XML file in a faster way. It works with high complex XML document with XML Validation.

### `simplexml_load_file()`

This function accepts file path as a first parameter and it is mandatory.

Syntax:

```
simplexml_load_file(($fileName,$class,$options,$ns,$is_prefix))
```

### Example:

#### XML File

```
<?xml version="1.0" encoding="UTF-8"?>

<student>
  <stud>
    <usn>01</usn>
    <name>abhi</name>
    <sem>4</sem>
```

```
<branch>mca</branch>
</stud>
<stud>
  <usn>02</usn>
  <name>nimish</name>
  <sem>4</sem>
  <branch>mca</branch>
</stud>
<stud>
  <usn>03</usn>
  <name>rajath</name>
  <sem>4</sem>
  <branch>mca</branch>
</stud>
<stud>
  <usn>05</usn>
  <name>nimish</name>
  <sem>4</sem>
  <branch>mca</branch>
</stud>
<stud>
  <usn>06</usn>
  <name>pavan</name>
  <sem>4</sem>
  <branch>mca</branch>
</stud>
<stud>
  <usn>06</usn>
  <name>vinay</name>
  <sem>4</sem>
  <branch>mca</branch>
</stud>
</student>
```

<!DOCTYPE html>

<!--

To change this license header, choose License Headers in Project Properties.

To change this template file, choose Tools | Templates  
and open the template in the editor.

-->

```

<html>
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <form method="post" action=" <?php $_SERVER['PHP_SELF'] ?>">
    NAME:<input type="text" name="t1">
    <input type="submit" name="t2" value="Retrive">
    <input type="submit" name="t3" value="Display">
  </form>
  <?php
include'dbcon.php';
$stud= simplexml_load_file("lab5.xml");
foreach($stud as $st)
{
  $usn=$st->usn;
  $name=$st->name;
  $sem=$st->sem;
  $branch=$st->branch;
  //echo $usn.$name.$sem.$branch;
  $query="insert into student values('$usn','$name','$sem','$branch')";
  $r=mysql_query($query);

}

if(isset($_POST['t2']))
{
  $name=$_POST['t1'];
  $query="select * from student where name='$name'";
  $res= mysql_query($query);
  if(mysql_num_rows($res)==0)
  {
    echo"name doesn't exist in text box";
  }
  else
  {
    echo"<body align='center'>";

```

```

        echo"<h2 align='center'>STUDENT DATA</h2>";
        echo"<table border='1'
align='center'><tr><td>usn</td><td>name</td><td>sem</td><td>branch</td></tr>";
        while($s=mysql_fetch_Array($res))
        {
            echo"<tr><td>".$s['usn']. "</td>";
            echo"<td>".$s['name']. "</td>";
            echo"<td>".$s['sem']. "</td>";
            echo"<td>".$s['branch']. "</td></tr>";
        }
        echo"</table>";
        echo"</body>";
    }
}
else if(isset($_POST['t3']))
{
    $query="select * from student";
    $res= mysql_query($query);
    echo"<body align='center'>";
    echo"<h2 align='center'>STUDENT DATA</h2>";
    echo"<table border='1'
align='center'><tr><td>usn</td><td>name</td><td>sem</td><td>branch</td></tr>";
    while($s=mysql_fetch_array($res))
    {
        echo"<tr><td>".$s['usn']. "</td>";
        echo"<td>".$s['name']. "</td>";
        echo"<td>".$s['sem']. "</td>";
        echo"<td>".$s['branch']. "</td></tr>";
    }
    echo"</table>";
    echo"</body>";
}
?>
</body>
</html>

```

## MODULE-2

### Introduction to Ruby and Introduction to Rails

#### Introduction

➤ Ruby is a pure object-oriented programming language. It was created in 1993 by Yukihiro Matsumoto of Japan.

➤ Ruby is a scripting language designed by Yukihiro Matsumoto, also known as Matz. It runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. Features of Ruby:

- Ruby is an open-source and is freely available on the Web, but it is subject to a license.
- Ruby is a general-purpose, interpreted programming language.
- Ruby is a true object-oriented programming language.
- Ruby is a server-side scripting language similar to Python and PERL.
- Ruby can be used to write Common Gateway Interface (CGI) scripts.
- Ruby can be embedded into Hypertext Markup Language (HTML).
- Ruby can be used for developing Internet and intranet applications.
- Ruby can be installed in Windows and POSIX environments.
- Ruby can easily be connected to DB2, MySQL, Oracle, and Sybase.
- Ruby has a rich set of built-in functions, which can be used directly into Ruby scripts.

#### Scalar types and Their Operations

Ruby has three categories of data types---**scalars, arrays, and hashes**. As stated early, everything in Ruby is an object- numeric literals, arrays, and even classes.

#### Numeric and String Literals

##### Numeric values:

- All numeric data types in Ruby are descendants of the **Numeric class**. The immediate child classes of numeric are **Float and Integer**. The Integer class has two classes **Fixnum and Bignum**.
- Underscore characters can appear embedded in integer literals. Ruby ignores such underscores.
- A numeric literal that has either an embedded decimal point or a following exponent value.

Example:

123	#Fixnum
-234	#Fixnum
6_799	# Fixnum (underscore is ignored)
123_456_789_123_456_789	# Bignum(underscore is ignored)



### Strings:

- A String object in Ruby holds and manipulates an arbitrary sequence of one or more bytes, typically representing characters that represent human language.
- The simplest string literals are enclosed in single quotes or double quotes.

Example: 'This is a simple Ruby string literal'

### Expression Substitution:

Expression substitution is a means of embedding the value of any Ruby expression into a string using `#{ and }`

Example:

```
#!/usr/bin/ruby
x, y, z = 12, 36, 72
puts "The value of x is #{ x }."
puts "The sum of x and y is #{ x + y }."
puts "The average was #{ (x + y + z)/3 }."
```

This will produce the following result –

```
The value of x is 12.
The sum of x and y is 48.
The average was 40.
```

### General Delimited Strings:

With general delimited strings, you can create strings inside a pair of matching though arbitrary delimiter characters, e.g., `!`, `(`, `{`, `<`, etc., preceded by a percent character (`%`). `Q`, `q`, and `x` have special meanings.

General delimited strings can be –

```
%{Ruby is fun.} equivalent to "Ruby is fun."
%Q{ Ruby is fun. } equivalent to " Ruby is fun. "
%q[Ruby is fun.] equivalent to a single-quoted string
```

### Variables and Assignment Statements

- Naming conventions in Ruby identify different categories of variables.
- **Identifiers** are started with a lowercase letter alphabet or an underscore, followed by any number of uppercase or lowercase letters, digits, or underscores.
- The letters in a variable name are case sensitive, meaning that `Test`, `test`, `test`, `Test` are all distinct names.

**Ruby global variable:** Global variables begin with `$`.

**Ruby local variable:** Local variables begin with `@`.

Ruby has **constants**, which are distinguished from variables by their names, which always begin with **uppercase letter**.

### Numeric operators

Ruby supports a rich set of operators, as you'd expect from a modern language.

#### Ruby Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then –

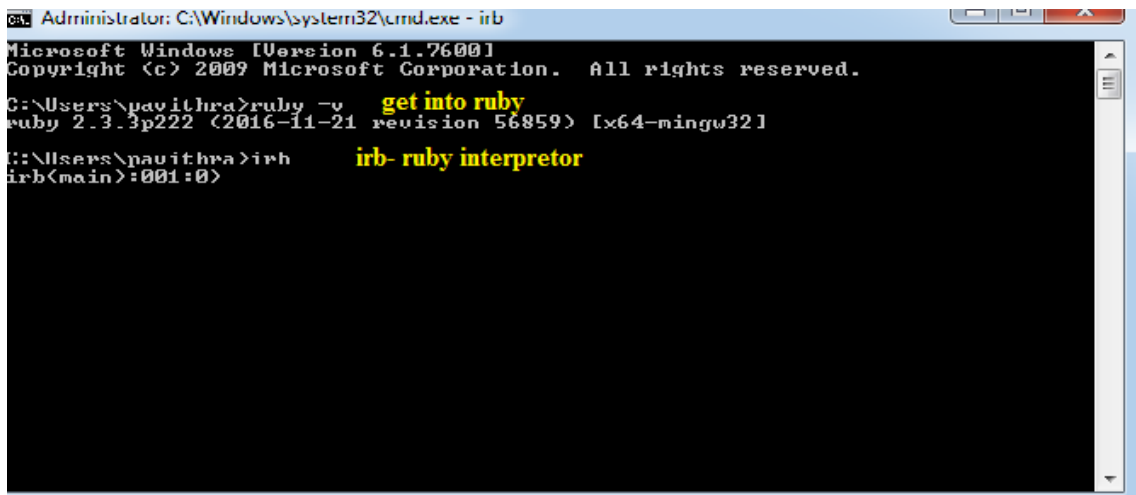
Operator	Description	Example
+	Addition – Adds values on either side of the operator.	a + b will give 30
–	Subtraction – Subtracts right hand operand from left hand operand.	a - b will give -10
*	Multiplication – Multiplies values on either side of the operator.	a * b will give 200
/	Division – Divides left hand operand by right hand operand.	b / a will give 2
%	Modulus – Divides left hand operand by right hand operand and returns remainder.	b % a will give 0
**	Exponent – Performs exponential (power) calculation on operators.	a**b will give 10 to the power 20

Ruby includes the Math module, which methods for basic trigonometric and transcendental functions. Among these are **cos** , **sin**, **log**, **sqrt**, and **tan**. The methods of the math module are referenced by prefixing their names with **Math.**, as in **Math.sin(x)**.

#### Ruby programs by using command prompt:

Download Ruby from the following website:

<http://www.ruby-lang.org>



### String methods

The string method class has more than 7 methods, a few which are described in this section.

#### (i) String Concatenation:

The string method for concatenation is specified by plus(+), which can be used as a binary operator or we can use **Concat** method for concatenation.

```
>> s1="hello"
=> "hello"
>> s2="world"
=> "world"
>> s1+s2
=> "helloworld"
>> s1.concat(s2)
=> "helloworld"
```

#### (ii) To append String

To append a string to the right end of another string, of course only makes sense if the left operand is a variable, use the **<<** method.

```
>> str1="good"
=> "good"
>> str1<< "day"
=> "good day"
```

#### (iii) String assignment:

```
>> str1="good"
=> "good"
>> str2=str1
=> "good"
```

The other most common used methods are similar to those of other programming languages. Some of them listed below:

capitalize	Convert the first letter to uppercase and rest of the letters to lowercase
chop	Removes the letter
chomp	Removes a newline from the right hand, if there is one
upcase	Converts all of the lower letters in string into uppercase
Downcase	Converts all of the uppercase letters in string into lowercase
reverse	Revers the characters of the string
swapcase	Convert all uppercase letters to lowercase and all lowercase to uppercase.

- Ruby strings can be indexed, somewhat as if they were arrays. The brackets of this method specify a getter method.

```
>> str1="hello mit"
=>"hello mit"
>>str1[3]
=>l
```

A multi-character substring of a string can be accessed by including two numbers in brackets, in which the first is the position of the first character of the string and the second is the number of characters in the substring.

```
>> str1="hellomit"
=>"hellomit"
>>str1[1,4]
=>"ello"
```

- Specific characters of a string can be set with the set method, `[]=`.

```
>> str1="hello"
=>"hello"
>>str1[1,3]="hii"
=>"hii"
>>str1
=>"hhiio"
```

- The usual way of comparing strings for equality is to use the `==` method as an operator.

```
>> "hello" == "hello"
=>true
>> "hello"=="hellow"
=>false
```

- A different sense of equality is tested with the equal? Or eq? Method, which determines the parameters is the same **object**.

```
>>"hello".equal?("equal")  
=>false
```

This produces false because, although the contents of the two strings literals are same, they are different objects.

- The **spaceship (<=>)** operators returns -1 if the second operand is greater than first, 0 if they are equal, and 1 if the first operand is greater than the second.

```
>>"apple"<=>"grape"  
=>-1  
>>"apple"<=>"apple"  
=>0  
>>"grape"<=>"apple"  
=>1
```

- The repetition operator is specified with an \*(asterisk). It takes a string as its left operand and an expression that evaluates to a number as its right operand.

```
>>"hello" *3  
=>"hello hello hello"
```

## Simple Input and Output

Among the most fundamental constructs in the most programming languages are the statements or functions that provide keyboard input and screen output.

### Screen Output

- Output is directed to the screen with the puts method. We prefer to treat it as an operator. The operand for **puts** is a string literal.
- If the value of a variable is to be part of a line of output, the `#{...}` notation can be used to insert it in a double-quoted string literal.

```
>> name="john"
=>"john"
>>puts"My name is: #{name}"
My name is john
```

### Screen input

The **gets** method gets a line of input from the keyboard. The retrieved line includes the newline character.

```
>> print "enter ur name"
=>name=gets()
john
>> puts "#{name}"
=>"john"
```

- If a number is to be input from the keyboard, the string from gets must be converted to an integer with **to\_i** method.

Example:

```
>> age=gets.to_i
25
=>25
```

- If a number is to be input from the keyboard, the string from gets must be converted to an floating-point value with **to\_f** method.

```
>> age=gets.to_f
25.55
=>25.55
```

## Control statements

Ruby has a complete collection of statements for controlling the execution flow through its programs.

### Control expressions

The expressions upon which statement control flow is based are Boolean expressions. They can be either of the constants true or false, variables, relational expressions, or compound expressions.

A control expression that is a simple variable is true if its value anything except nil.

### Ruby Comparison Operators

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
<=>	Combined comparison operator. Returns 0 if first operand equals second, 1 if first operand is greater than the second and -1 if first operand is less than the second.	(a <=> b) returns -1.
===	Used to test equality within a when clause of a case statement.	(1...10) === 5 returns true.
.eql?	True if the receiver and argument have both the same type and equal values.	1 == 1.0 returns true, but 1.eql?(1.0) is false.
equal?	True if the receiver and argument have the same object id.	if aObj is duplicate of bObj then aObj == bObj is true, a.eql?bObj is false but a.eql?aObj is true.

### Operator precedence and associativity

The following table lists all operators from highest precedence to lowest.

Operator	Description
::	Constant resolution operator
[ ] []=	Element reference, element set
**	Exponentiation (raise to the power)
! ~ + -	Not, complement, unary plus and minus (method names for the last two are +@ and -@)
* / %	Multiply, divide, and modulo
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
<=> == === != =~ !~	Equality and pattern match operators (!= and !~ may not be defined as methods)
&&	Logical 'AND'
	Logical 'OR'
.. ...	Range (inclusive and exclusive)
? :	Ternary if-then-else
= %= { /= -= +=  = &= >>= <<= *= &&=   = **=	Assignment
defined?	Check if specified symbol defined
Not	Logical negation
or and	Logical composition

### Selection and Loop statements

- Control statements require some syntactic container for sequences of statements whose execution they meant to control.
- A control construct is a control statement and the segment of code whose execution it controls.
  - Ruby offers conditional structures that are pretty common to modern languages. Here, we will explain all the conditional statements and modifiers available in Ruby.



### Ruby **if** statement

*if* expressions are used for conditional execution. The values *false* and *nil* are false, and everything else are true.

```
>>If a>10
=>b=a+2
End
```

### Ruby **if ...else** statement

Executes *code* if the *conditional* is true. If the *conditional* is not true, *code* specified in the else clause is executed.

Syntax:

```
if conditional [then]
  code...
[elsif conditional [then]
  code...]....
[else
  code...]
end
```

*if* expressions are used for conditional execution. The values *false* and *nil* are false, and everything else are true. Notice Ruby uses **elsif**, not else if nor **elsif**.  
Executes *code* if the *conditional* is true. If the *conditional* is not true, *code* specified in the else clause is executed.

**Example:**

```
x = 1
if x > 2
  puts "x is greater than 2" elsif
x <= 2 and x!=0
  puts "x is 1"
else
  puts "I can't guess the number" end
```

Output:

```
x is 1
```

### Ruby **unless** Statement

Ruby has an unless statement, which is the same as its if statement except that the inverse of the control expression is used.

Syntax:

```
unless conditional
  [then] code
[else
  code ]
```

Executes *code* if *conditional* is false. If the *conditional* is true, code specified in the else clause is executed.

Example:

```
x = 1 unless
x>=2
  puts "x is less than 2" else
  puts "x is greater than 2" end
```

### Ruby case Statement

Ruby includes two kinds of multiple selection constructs, both named **case**. One Ruby case construct, which is similar to a switch, has the following form:

Syntax:

```
case expression
[when expression [, expression ...] [then]
  code ]...
[else
  code ]
end
```

Example:

```
$age = 5
case $age
when 0 .. 2
  puts "baby"
when 3 .. 6
  puts "little child" when 7
.. 12
  puts "child"
when 13 .. 18
  puts "youth"
```

This will produce the following result –

```
little child
```

The second form of case constructs uses a Boolean Expression to choose a value to be produced by the construct. The general form as follows:

Case When Boolean expression the expression When Boolean expression the expression
--

.....

When Boolean expression the expression

Else      expression

End

### While statements

The Ruby while and for statements are similar to those of C and its descendants. The bodies of both are sequences of statements that end with end. The general form of the while statement as follows:

```
while control expressions
  loop body statement(s)
end
```

### Ruby until Statement

The until statement is similar to the while statement except that inverse of the value of the control expression is used.

Syntax:

```
until conditional [do]
  code
end
```

Example:

```
$i = 0
$num = 5

until $i > $num      do
  puts("Inside the loop i = #$i" )
  $i +=1;
end
```

This will produce the following result –

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
Inside the loop i = 5
```

### Ruby for Statement

Syntax:

---

Executes *code* once for each element in *expression*.

Example:

```
#!/usr/bin/ruby for i
in 0..5
  puts "Value of local variable is #{i}"
end
```

Here, we have defined the range 0..5. The statement for *i* in 0..5 will allow *i* to take values in the range from 0 to 5 (including 5). This will produce the following result –

```
value of local variable is 0 Value of
local variable is 1 Value of local
variable is 2 Value of local variable is
3 Value of local variable is 4 Value of
local variable is 5
```

There are two ways to control an infinite loop, the ***break*** and ***next*** statements. These statements can be made conditional by putting them in the then clause of an if construct.

### Ruby next Statement:

Syntax:

```
next
```

Jumps to the next iteration of the most internal loop. Terminates execution of a block if called within a block (with *yield* or call returning nil).

Example:

```
for i in 0..5 if i < 2
  then
    next
  end
  puts "Value of local variable is #{i}" end
```

This will produce the following result –

```
Value of local variable is 2 Value
of local variable is 3 Value of
local variable is 4 Value of local
variable is 5
```

```
for variable [, variable ...] in expression [do]
  code
end
```

## Ruby break Statement

Syntax:

```
break
```

Terminates the most internal loop. Terminates a method with an associated block if called within the block (with the method returning nil).

**Example:**

```
#!/usr/bin/ruby

for i in 0..5
  if i > 2 then
    break
  end
  puts "Value of local variable is #{i}"
end
```

This will produce the following result –

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
```

## Fundamentals of Arrays

Ruby arrays are ordered, integer-indexed collections of any object. Each element in an array is associated with and referred to by an index.

Ruby arrays can be created in two ways:

- First, an array can be created by sending **new message** to the predefined **Arrayclass**, including a parameter for the size of the array.

Syntax:

```
names = Array.new
```

We can set the size of an array at the time of creating array –

```
names = Array.new(20)
```

We can assign a value to each element in the array as follows –

```
names = Array.new(4, "mac")
puts "#{names}"
```

This will produce the following result –

```
["mac", "mac", "mac", "mac"]
```

- The second way is simply to assign a list literal is a list of literals delimited by brackets.

Example:

```
>> list = [2,"hello",'v', 33.022]
=>[2,"hello",'v', 33.022]
```

- All ruby array elements are use integers as subscripts, and the lower-bound subscript of every array is zero.
- Array elements are referenced through sub scripts delimited by brackets (`[]`), which is actually **getter method** that is allowed to be used as a unary operator. Likewise, `[]=` is a **setter method**.
- If an expression with a floating-point value is used as a subscript, the fractional part is truncated.

**Example:**

```
>> list = [2, 3, 5, 8]
=>[2, 3, 5, 8]
>>list[3]
=>8
>>list[3]=9
=>9
>>list
```

**For example:**

```
>>[2, 3, 5, 9]
```

- The length of an array is dynamic: elements can be added or removed from an array using the methods. The length of an array can be retrieved with the length method:

**Example:**

```
>> list.length
=>4
```

## The for-in statement

The for-in statement is used to process the elements of an array.

**Example:**

```
>> sum=0
=>0
>>List=[2, 3, 5, 8]
=> [2, 3, 5, 8]
>> for value in List
>> sum+=value
>> end
=> [2, 3, 5, 8]
>> sum
=> 18
```

### Built-in methods for arrays and Lists

There are different kind of built-in methods are there. Some of them listed below:

- Shift and un-shift method.
- Push and pop method.
- Concatenation method.
- Reverse and reverse! Method.
- Include? Method.
- Sort and sort! method.

#### Shift and Un-shift method:

**Shift** method removes and returns the **first element** of the array object to which it is sent. The un-shift method takes a scalar or an array literal as a parameter.

##### Example:

```
>> list = [2, 4, 6, 99]
=> [2, 4, 6, 99]
>> list.shift
=> 2
>> list
=> [4, 6, 99]
```

#### Push and pop method:

The **pop method** removes and returns the **last element** from the array object to which it is sent. The push method takes a scalar or an array literal as a parameter to insert an element at the end of an array.

##### Example:

```
>> list = [2, 4, 6, 99]
=> [2, 4, 6, 99]
>> list.pop
=> 99
```

```
>> list
=> [2, 4, 6]
>> list.push(3,10)
=> [2, 4, 6,10]
```

#### Concatenation method:

If an array is to be concatenated to the end of the array, we use **concat method**.

##### Example:

```
>> list1 =[1,2,3,4]
=> [1,2,3,4]
>> list2 =[6,7,9]
=> [6,7,9]
>> list1.concat(list2)
=> [1,2,3,4,6,7,9]
```

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

**Example:**

```
>> list1 =[1,2,3,4]
=> [1,2,3,4]
>> list2 =[6,7,9]
=> [6,7,9]
>> list1+list2
=> [1,2,3,4,6,7,9]
```

**Reverse and reverse! Method:**

The reverse method does what its name implies.

**Example:**

```
>> list1 =[1,2,3,4]
=> [1,2,3,4]
>> list1.reverse
=> [4,3,2,1]
>> list1
=> [1,2,3,4]
>> list2 =[6,7,9]
=> [6,7,9]
>> list2.reverse!
=> [9,7,6]
>> list2
=> [9,7,6]
```

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

**The Include? Method:**

The include? Predicate method searches an array for a specific object.



**Example:**

```
>> list1 =[1,2,3,4]
=> [1,2,3,4]
>> list1.include?(2)
=> true
>> list1.include?(12)
=> false
```

**Sort and sort! method.**

The sort method sorts the elements of an array, as long as Ruby knows how compare those elements. The most commonly sorted elements are either numbers or strings.

**Example:**

```
>> list1 =[1,6,21,4]
=> [1,6,21,4]
>> list1.sort
=> [1,4,6,21]
>> list1
=> [1,6,21,4]
>> list2 =[6,1,9]
=> [6,1,9]
>> list2.sort!
=> [1,6,9]
>> list2
=> [1,6,9]
```

In some situations, arrays represents sets. There are three methods that perform set operations on two arrays. All are used as binary infix operators. They are ,

- & for set intersection,
- for set difference
- | for set union

**Example:**

```
>> list1 =[2,4,6,8]
=> [2,4,6,8]
>> list2 =[4,6,8,10]
=> [4,6,8,10]
>> list1 & list2
=> [4,6,8]
>> list1 - list2
=> [2]
>> list1 | list2
=> [2,4,6,8,10]
```

## Hashes

- A Hash is a collection of key-value pairs like this: "employee" => "salary". It is similar to an Array, except that indexing is done via arbitrary keys of any object type, not an integer index.
- The order in which you traverse a hash by either key or value may seem arbitrary and will generally not be in the insertion order. If you attempt to access a hash with a key that does not exist, the method will return *nil*.

### Creating Hashes:

As with arrays, there is a variety of ways to create hashes. You can create an empty hash with the *new* class method –

```
months = Hash.new
```

we can also use *new* to create a hash with a default value, which is otherwise just *nil* –

```
months = Hash.new( "month" ) or
months = Hash.new "month"
```

The way to create a Hash by simply assigning the values to a variable:

### Example:

```
H = Hash["a" => 100, "b" => 200]
```

When you access any key in a hash that has a default value, if the key or value doesn't exist, accessing the hash will return the default value –

### Example:

```
H = Hash["a" => 100, "b" => 200]
puts "#{H['a']}"
puts "#{H['b']}"
```

This will produce the following result –

```
100
200
```

### Hash Built-in Methods:

- **Delete method:** an element removed from a hash with the **delete** method, which takes an element as a parameter.

### Example:

```
>> usn=[{"john"=>1, "abhi"=> 5, "janvi"=>"9"}]
=> [{"john"=>1, "abhi"=> 5, "janvi"=>"9"}]
```

```
>> usn.delete("janvi")
=> [{"john"=>1, "abhi"=> 5}]
```

**Clear method:** A hash can be set to empty in two ways: First, an empty hash literal can be assigned to the hash. Second, the *clear* method can be used on the hash.

**Example:**

```
>> usn={"john"=>1, "abhi"=> 5, "janvi"=>9}
=> {"john"=>1, "abhi"=> 5, "janvi"=>9}
>> usn={}
=> {}

>> usn={"john"=>1, "abhi"=> 5, "janvi"=>9}
=> {"john"=>1, "abhi"=> 5, "janvi"=>9}
>> usn.clear
=> {}
```

- **Has\_key?:** the has\_key? Predicate method is used to determine whether an element with a specific key is in a hash.

**Example:**

```
>> usn={"john"=>1, "abhi"=> 5, "janvi"=>9}
=> {"john"=>1, "abhi"=> 5, "janvi"=>9}
>> usn.has_key?("abhi")
=> true
>> usn.has_key?("abhishek")
=> false
```

- **Keys and values method:** The keys and values of a hash can be extracted into arrays with the methods keys and values.

**Example:**

```
>> usn={"john"=>1, "abhi"=> 5, "janvi"=>9}
=> {"john"=>1, "abhi"=> 5, "janvi"=>9}
>> usn.keys
=> ["john", "abhi", "janvi"]
>> usn.values
=> [1,5,9]
```

## Methods

- Ruby methods are very similar to functions in any other programming language. Ruby methods are used to bundle one or more repeatable statements into a single unit.
- Method names should begin with a **lowercase letter**. If you begin a method name with an uppercase letter, Ruby might think that it is a constant and hence can parse the call incorrectly.

- Methods should be defined before calling them; otherwise Ruby will raise an exception for

undefined method invoking.

- A method definition includes the method's header and a sequence of statements, ending with the end reserved word, which describes its actions. A method header is the reserved word **def**, the method's name, and optionally a parenthesized list of formal parameters.

**Syntax:**

```
def method_name
  expr..
end
```

We can represent a method that accepts parameters like this –

```
def method_name (var1, var2)
  expr..
end
```

We can set default values for the parameters, which will be used if method is called without passing the required parameters –

```
def method_name (var1 = value1, var2 = value2)
  expr..
end
```

Whenever you call the simple method, you write only the method name as follows –

```
method_name
```

However, when you call a method with parameters, you write the method name along with the parameters, such as –

```
method_name 25, 30
```

**Example:**

```
def test(a1 = "Ruby", a2 = "Perl")
  puts "The programming language is #{a1}" puts "The
  programming language is #{a2}"
end
test "C", "C++" // call a function with parameters test // simple
function call
```

### Output:

```
The programming language is C The  
programming language is C++ The  
programming language is Ruby The  
programming language is Perl
```

### Ruby return statement:

The *return* statement in ruby is used to return one or more values from a Ruby Method.

### Syntax:

```
return [expr[, 'expr...]]
```

If more than two expressions are given, the array containing these values will be the return value. If no expression given, nil will be the return value.

### Example:

```
def test  
  i = 100  
  j = 200  
  k = 300  
  return i,  
  j, k end  
var = test  
puts var
```

This will produce the following result –

```
100  
200  
300
```

### Parameters

- The parameters values that appear in a call to a method are called *actual parameters*.
- The parameter names in the method, which correspond to the actual parameters, are called as *formal parameters*.

**Example:**

```
>> def swap(x,y) // actual parameters
>> t=x
>> x=y
>> y=t
>> end
=> nil
>> a= 1
>> b=2
>> swap(a,b) // formal parameters
=> 1
>> a
=> 1
>> b
=> 2
```

Suppose you declare a method that takes two parameters, whenever you call this method, you need to pass two parameters along with it.

However, Ruby allows you to declare methods that work with a variable number of parameters. However a method can be defined to take a variable number of parameters by defining it with a parameter that is preceded by an asterisk (\*). Such a parameter is called an *asterisk parameter*. Let us examine a sample of this –

```
def sample (*test)
  puts "The number of parameters is #{test.length}"
  for i in 0...test.length
    puts "The parameters are #{test[i]}"
  end
end
sample "Zara", "6", "F" // with three parameters
sample "Mac", "36", "M", "MCA"v// with four parameters
```

In this code, you have declared a method sample that accepts one parameter test. However, this parameter is a variable parameter. This means that this parameter can take in any number of variables. So, the above code will produce the following result –

```
The number of parameters is 3 The
parameters are Zara
The parameters are 6 The
parameters are F
The number of parameters is 4 The
parameters are Mac
The parameters are 36 The
parameters are M The
parameters are MCA
```

## Class Methods

- When a method is defined outside of the class definition, the method is marked as *private* by default. On the other hand, the methods defined in the class definition are marked as public by default. The default visibility and the *private* mark of the methods can be changed by *public* or *private* of the Module.
- Whenever you want to access a method of a class, you first need to instantiate the class. Then, using the object, you can access any member of the class.
- Ruby gives you a way to access a method without instantiating a class. Let us see how a class method is declared and accessed –

```
lass Accounts
  def reading_charge
  end
  def Accounts.return_date
  end
end
```

See how the method `return_date` is declared. It is declared with the class name followed by a period, which is followed by the name of the method. You can access this class method directly as follows –

```
Accounts.return_date
```



## Code Blocks and Iterators

- We have seen how Ruby defines methods where you can put number of statements and then you call that method. Similarly, Ruby has a concept of Block.
  - A block consists of chunks of code.
  - We assign a name to a block.
  - The code in the block is always enclosed within braces ({}).
  - A block is always invoked from a function with the same name as that of the block. This means that if you have a block with the name *test*, then you use the function *test* to invoke this block.
  - You invoke a block by using the *yield* statement.

### Syntax:

```
block_name {  
  statement1  
  statement2  
  .....  
}
```

Here, we will learn to invoke a block by using a simple *yield* statement. We will also learn to use a *yield* statement with parameters for invoking a block. We will check the sample code with both types of *yield* statements.

### The yield Statement;

Let's look at an example of the yield statement –

```
def test  
  puts "You are in the method"  
  yield  
  puts "You are again back to the method"  
  yield  
end  
test {puts "You are in the block"}
```

This will produce the following result –

```
You are in the method You are  
in the block  
You are again back to the method You are  
in the block
```

### Types of Iterators:

- The *times* iterator: The time iterator method provides a way to build simple counting loops. Typical *times* number of times. The *times*

**Example:**

**iterator**

is sent to a number, which repeats the attached block, that method repeatedly executes the block.

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

Hey!

Block

- The ***each*** iterator: the most commonly used iterator is ***each***, which is often used to go through arrays and apply a block to each element.

**Example:**

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

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

**Example:**

```
>> 5.upto(9) { |value| puts value}
5
6
7
8
9
=> 5
```

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

**Example:**

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

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

**Example:**

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

## Pattern Matching

In Ruby, the pattern matching operation is specified with the matching operators, **=~**, which is for positive matches, and **!~**, which is for negative matches. Patterns are placed between slashes (/).

**For Example:**

```
>> street = "Hammel"
=> "Hammel"
>> street =~ /mm/
=> 2 // returns matched index value
```

The result of evaluating a pattern-matching expression is the position in the string where the pattern is matched.

## Remembering Matches

The part of the string that matched a part of the pattern can be saved in an implicit variable for later use. The part of the pattern whose match you want to save is placed in parenthesis. The substring matched the first parenthesized part of the pattern is saved in \$1, the second in \$2, and so on.

**For example:**

```
>> str = "4 july 1776"
=> "4 july 1776"
>> str =~ /(d+) (w+) (d+)/ //d- digits, w- letters
=> 0
>> puts "#{$2} #{ $1} #{ $3}"
=> july 4, 1776
```

## Substitutions

Some of the most important String methods that use regular expressions are **sub** and **gsub**, and their in-place variants **sub!** and **gsub!**.

Some of the most important String methods that use regular expressions are **sub** and **gsub**, and their in-place variants **sub!** and **gsub!**.

The **sub** and **gsub** returns a new string, leaving the original unmodified where as **sub!** and **gsub!** modify the string on which they are called.

Consider the following examples:

```
>> str = " the old car is great, but old"
=> " the old car is great, but old"
>> str.sub(/old/, "new")
=> " the new car is great, but old"
```

The **gsub** method is similar to **sub**, but finds all substring matches and replaces all of them with its second parameter.

Example:

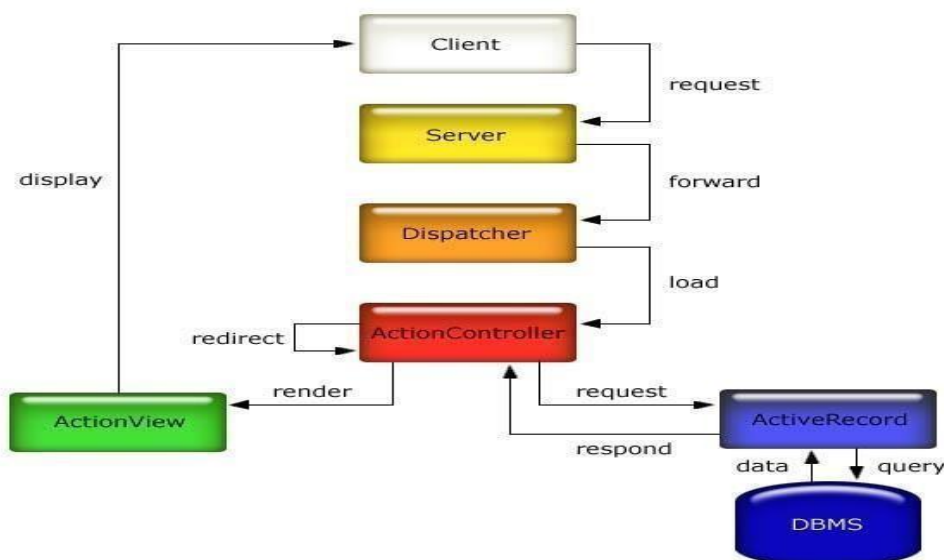
```
>> str = " the old car is great, but old"
=> " the old car is great, but old"
>> str.gsub(/old/, "new")
=> " the new car is great, but new"
>> str
=>> " the old car is great, but old"
>> str.gsub!(/old/, "new")
=> " the new car is great, but new"
>> str
=>" the new car is great, but new"
```

## Overview of Rails

- Rails is a web application development framework written in the Ruby language.
- A web application development framework is a software framework that is designed to support the development of websites, web applications, web services and web resources.
- Rails, because of its intimate connection with Ruby, is often called Ruby on Rails or simply RoR.
- Rails was developed by David Heinemeier Hansson in the early 2000s and was released to the public in July 2004

## MVC Architecture

- Rails is based on Model-View-Controller (MVC) architecture for applications.
- MVC was developed by Trygve Reenskaug, a Norwegian, in 1978-1979.
- The MVC architecture clearly separates applications into three parts.
- The **model** (ActiveRecord) belongs to the Data or **Database** and all its queries.
- The **view** (ActionView) is the part of an application that prepares and presents results to the user nothing but on the browser. The view documents are generated three categories of view documents, XHTML, XML, and JavaScript.
- The **controller** (ActionController), the controller part of MVC, controls the interactions between the data model, the user, and the view.



## Working:

- In an MVC web application, a browser submits request to the controller, which consults the model

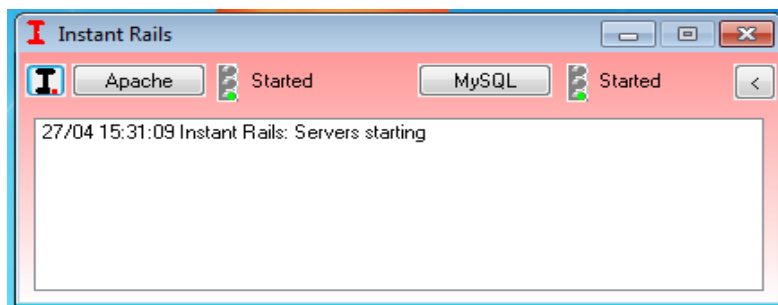
and reports results back to the controller and indirectly to the view.

- The controller then instructs the view to produce a result document that is then transmitted to the client for display.
- The intent of MVC is to reduce the coupling among the three parts of an application, making the application easier to write and maintain.
- The view and controller parts of MVC is supported with the ActionPack component of Rails. They are packed in the same component, the code support for the view and controller are cleanly separated.
- Ruby on Rails is completely Object-Oriented-Programming language, whereas the Database is completely Relational-Database.
- These two are different platforms, so, this is not a natural connection. For this Rails uses an Object- Relational-Mapping (ORM) approach.
- Each relational database tables is implicitly mapped to a class.
  - **Example:**
    - An ORM maps database Tables to Classes,
    - Rows to Objects, and
    - Table Columns to the fields of the Object methods.

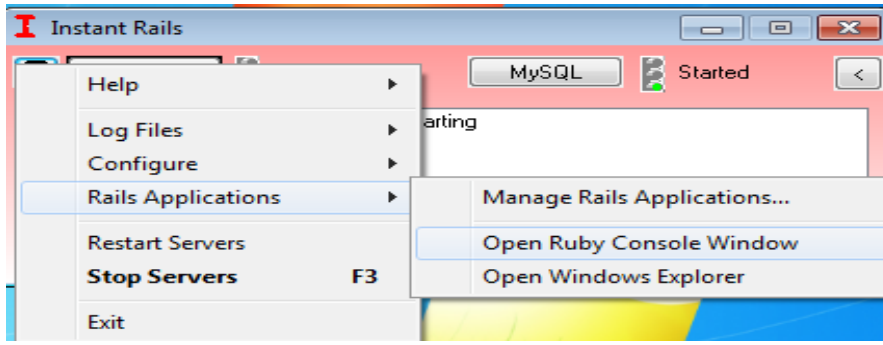
### Document Request

#### Static documents

- This topic describes how to demonstrate the structure of the simplest possible Rails application and showing what files must be created and where they must be reside in the directory structure.
- First, you just click on InstantRails icon. You will get the following small window



- Here, you click on black I on the left border of this window produces a small menu.



- Click the Rails Application entry in this menu, which opens another menu.
- Clicking on the Open Ruby Console Window entry in this menu opens a DOS command window in the following directory.

**C:\InstantRails-2.0\rails\_apps**

- To this base directory, users usually add a new subdirectory for all their Rails applications.
- We named ours **lab1**. In the new lab1 directory, the new application rails is created with the following command.

**rails lab1**

- Rails responds by creating more than 40 files in more than 15 directories. This is part of the framework to support a Rails application. In this case **lab1**, 12 subdirectories are created.
- Now change to new directory, just typing: **cd lab1**
- The most interesting of which at this point is **app**. The **app directory** has four subdirectories- **models**, **views**, **controllers** and **helpers**. The **helpers** subdirectory contains Rails-provided methods that aid in constructing applications.
- One of the directories created by the rails command is **script**, which has several important Ruby scripts that perform services. This script creates two Ruby controller methods and also a subdirectory of the **views directory** where views code will be stored.
- **generate** is used to create part of an application controller.
- For our application, we pass two parameters to **generate**, the **first** of which is **controller**, which indicates that we want the **controller class built**. **Second** parameter is the **name** we chose for the **controller**.
- The following command is given in the **lab1 directory to create the controller**.

**ruby script/generate controller say**

- Here **say** is the name of the **controller**. The response produced by the execution of this command as follows:

exists app/controllers/ exists



```
app/helpers/  
create app/views/say  
create app/controllers/say_controller.rb
```

In the above output **exists** lines indicate files and directories that are verified to already exist.

- The **create** lines show the newly created directories and files.
- There are now **two Ruby** classes in the **controllers directory**, **application.rb** and **say\_controller.rb**  
where the **say\_controller.rb class is a subclass of application.rb.**

**Example: Class SayController < ApplicationController end**

- **Saycontroller** is an empty class, other than what it inherits from **application.rb**. The **controller** produces, at least indirectly, the **response to requests**, so **a method must be added** to **SayController** subclasses.

```
> notepad app\Controllers\say_controller.rb
```

**say\_controller.rb**

```
Class SayController < ApplicationController def hello  
end end
```

- Next we need to build the view file which will be simple like XHTML file to produce the greeting. The view document is often called mplate. The following is the template for the lab1 applications

```
> notepad app\Views\say\hello.rhtml
```

**Hello.rhtml**

```
<DOCTYPE >  
<html>  
<head> <title> Simple Document </title> </head>  
<body>  
  <h1> Hello Welcome </h1>  
</body>  
</html>
```

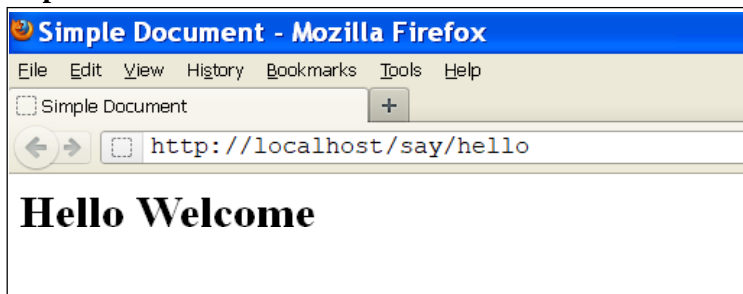
- The extension on this file is **.rhtml**, templates can include ruby code, which is interpreted by a ruby interpreter named ERb (Embedded Ruby), before the template is return to the requesting browser.
- Before the application can be tested, a rails web server must be started. A server is started with server script from the script directory

## **ruby script/server**

- Rails there are three different server available. The default server is Mongrel, but apache and WEBrick are also available in rails.
- The default port is 3000. If a different port must be used, because 3000 is already being used by some other program on the system, the port number is given as a parameter to the server script.
- The server name is always localhost because its running in the same machine on which applications resides.
- The complete URL of our application is:

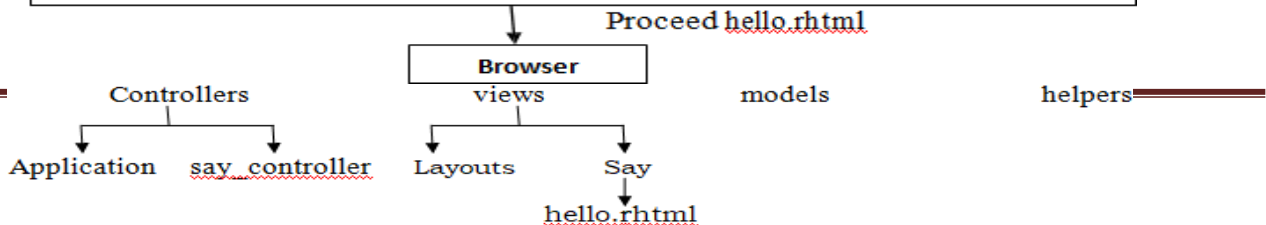
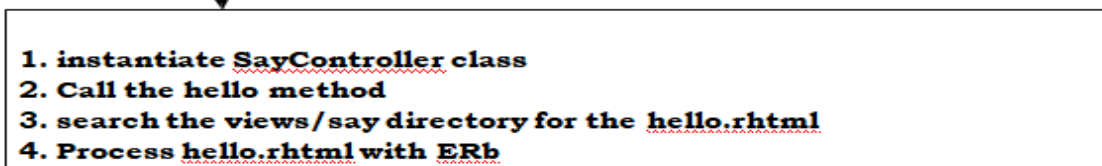
<http://localhost/say/hello>

**Output:**



## **The directory structure for the rails1 application**

The activities of Rails in response to a simple request are shown below  
<http://localhost/say/hello>



### Dynamic Documents

- Rails offers three different approaches to producing dynamic documents. Here we discussed only one, which is to embed Ruby code in a template file. This is similar to some other approaches we have discussed, in particular PHP, ASP.NET and JSP.
- 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, and equal sign (=) is attached to the opening marker.

**Example:**        The number of seconds in a day: <%=60 \* 60 \* 24%>

**Output:**         The number of seconds in a day: 86400

- The date can be obtained by calling Ruby's **Time.now** method, this method returns current day of the week, month, day of the month, time, time zone, and year, as a string.

**Example:**        <p>It is now<%= Time.now %></p>

**Output:**         It is now May 06 10:12:23 2015

[**Note:** The above (static document) procedure is same to develop a new rails applications]

- 1) **Create a new directory:**                                > rails lab2
- 2) **Now change to new directory:**                        > cd lab2
- 3) **Generate a new Controller :**                            > ruby script/generate controller main
- 4) **Open timer Controller:**                                > notepad app\Controllers\main\_controller.rb

#### **Maincontroller.rb**

```
Class MainController < ApplicationController def timer
  @t = Time.now end
end
```

**Create a timer.rhtml under views\Main:** > notepad app\Views\Main\timer.rhtml

#### **Timer.html**

```
<!DOCTYPE >
<html>
<head> <title> Simple Document </title> </head>
<body>
<h1> Hello Welcome </h1>
<p> It is now <%= @t %> </p>
</body>
</html>
```

**Start Ruby Server:** > ruby script/server

**URL link is:**      http://localhost:3000/Main/timer

## Processing forms

### 1) Setting Up the Application

- The new directory and generate a controller named home with the following command.  
**>ruby script/generate controller home**
- The contents of the **form.rhtml** is exactly the same as the HTML files, except that the opening form tag appears as follows.

```
<form action = "result" method = "post"> Phone:
  <input type="text" name="phone"/>
  <input type="submit" value="OK"/>
</form>
```

- This specifies that the name of the action method in the application's controller, as well as the template for the result of submitting the form, is **result**.
- This tag specifies the **POST HTTP** method. Rails requires that POST be used.
- Now, we point our browser to the following: **http://localhost/home/the\_form**

### 2) The controller and the View

- Next step of the construction of the application is to build the action method in `home_controller.rb` to process the form data when the form is submitted.
- In the initial template file, `the_form.rhtml`, this method is named `result` in the `action` attribute of the form tag.
- The `result` method has two tasks, the first of which is to fetch the form data. This data is used to display back to the customer and to compute the result.
- The form data is made available to the controller class through the Rails-defined object, `params`. `Params` is a hash-like object that contains all of the form data. It is a hash-like because it is a hash that can be indexed with either symbols or actual keys. The common Rails convention is to index `params` with symbols.

**Example:** `@phone_no = params[:phone]`

- In above example `:phone` is a key and `params` is hash-like object, and the value of the `phone` will be stored into instance variable `phone_no`.

**Homecontroller.rb**

```
Class HomeController<
  ApplicationController def
    form
    end
  def result
    @phone_no = params[:phone]
    end
end
```

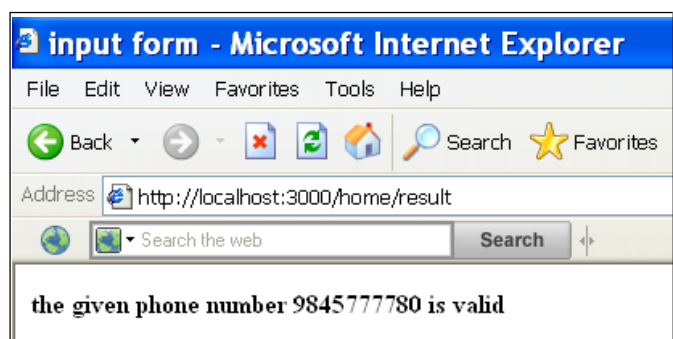
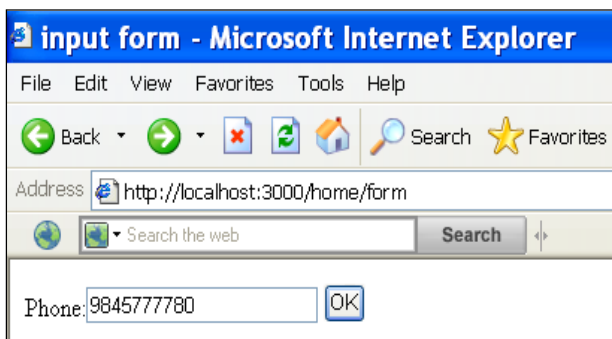
### form.rhtml

```
<html>
<head><title>input form</title></head>
<body>
<form action = "result" method = "post"> Phone:<input
  type="text" name="phone"/>
  <input type="submit" value="OK"/>
</form>
</body>
```

### result.rhtml

```
<html>
<head><title>input form</title></head>
<body>
  <h4> the given phone number<%= @phone_no%>is valid</h4>
</body>
</html>
```

### OUTPUT:



### Layouts.

- The views directory of each application has two subdirectories, one that has the name of the controller and another subdirectory named layouts.
- Rails creates the layouts subdirectory, but leaves it empty.
- The user can create a layout template and place it in the layouts directory.

- A layout template is a template for other templates.
- It provides a way to put some boilerplate markup into each template file in the application.

```
!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>CARS</title></head>
<body>
  <h1>aidan's used car Lot</h1>
  <%= @content_for_layout %>
</body>
</html>
```

- Notice that the value of the predefined instance variable **@content\_for\_layout** is inserted in the body of the layout document.

## Rails applications with Databases.

### 1. Building the Database

- MySQL must be started. This is done with the following command given at the command prompt in the: C:\InstantRails-2.0\rails\_apps>

**mysql [-h host] [-u username] [database\_name] [-p password]**

C:\InstantRails-2.0\rails\_apps> mysql -u root

- Now it will login to the mysql command prompt

mysql>

- Now we can create a database and its tables.
- To construct the database, the Rails applications to use three copies of the database, one for development, one for testing and one for production.

For this example, we are using a single database will be constructed, **database\_name\_development**.

**Example:**

```
mysql> create database
```

```
roopa_development; mysql> use
```

```
roopa_development;
```

```
create table Books (id int not null auto_increment,
                    name varchar(80),description text,
```

price decimal(8,2) primary key(id) );
--

## 2) Building the Application

- The inbuilt methods are count and find

### Count method:

- The number of rows in a table can be determined by calling the count method on the table's object.

**@num\_books = Books.count**

### Find method

- Which searches its table for rows that satisfy given criteria.
- The find method has two parameters. The first parameter to find is :all, find searches can be controlled by a second parameter, which is specified as the value of the :conditions symbol

**@book\_name = params[:sname]**

**@bookz = find(:all, conditions => "name = ' #{@book\_name} ' ")**

## MODULE-3

### Building Rich Internet Applications with Ajax-1

#### Introduction:

AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

#### AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

#### Building Rich Internet Applications with AJAX

- Ajax (shorthand for Asynchronous JavaScript and XML) is a group of interrelated web development methods used on the client-side to create interactive web applications.



- With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
- Data is usually retrieved using the XMLHttpRequest object. Despite the name, the use of XML is not needed, and the requests need not be asynchronous.
- Like DHTML and LAMP, Ajax is not one technology, but a group of technologies. Ajax uses a combination of HTML and CSS to mark up and style information.
- The DOM is accessed with JavaScript to dynamically display, and to allow the user to interact with, the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.
- The term Ajax has come to represent a broad group of web technologies that can be used to implement a web application that communicates with a server in the background, without interfering with the current state of the page.

In the article that coined the term Ajax, Jesse James Garrett explained that the following technologies are incorporated:

- HTML or XHTML and CSS for presentation
- the Document Object Model (DOM) for dynamic display of and interaction with data
- XML for the interchange of data, and XSLT for its manipulation
- the XMLHttpRequest object for asynchronous communication
- JavaScript to bring these technologies together
- Since then, however, there have been a number of developments in the technologies used in an Ajax application, and the definition of the term Ajax.
- In particular, it has been noted that JavaScript is not the only client-side scripting language that can be used for implementing an Ajax application; other languages such as VBScript are also capable of the required functionality. (However, JavaScript is the most popular language for Ajax programming due to its inclusion in and compatibility with the majority of modern web browsers.)
- Also, XML is not required for data interchange and therefore XSLT is not required for the manipulation of data.
- JavaScript Object Notation (JSON) is often used as an alternative format for data interchange, although other formats such as preformatted HTML or plain text can also be used.

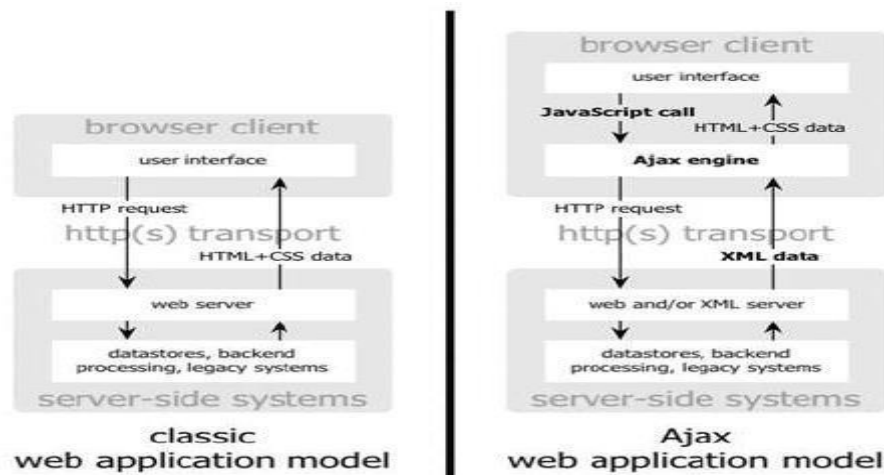
### **Limitations of Classic Web application model**

- The classic web application model works like this: Most user actions in the interface

trigger an HTTP request back to a web server.

- The server does some processing —retrieving data, crunching numbers, talking to various legacy systems —and then returns an HTML page to the client.
- It's a model adapted from the Web's original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications.

The following figure shows that the difference between classic web application model and Ajax web application model:



- This approach makes a lot of technical sense, but it doesn't make for a great user experience. While the server is doing its thing, what's the user doing? That's right, waiting.
- And at every step in a task, the user waits some more. Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait around.
- Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

### AJAX principles

- As a new web application model, Ajax is still in its infancy. However, several web developers have taken this new development as a challenge.
- The challenge is to define what makes a good Ajax web application versus what makes a bad or mediocre one. Michael Mahemoff (<http://mahemoff.com/>), a software developer and usability expert, identified several key principles of good.
- Ajax applications that are worth repeating:
  - **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.
- The common thread in all these principles is usability. Ajax is, primarily, about enhancing the web experience for your users; the technology behind it is merely a means to that end.
- By adhering to the preceding principles, you can be reasonably assured that your Ajax application will be useful and usable.

### Technologies behind AJAX

- JavaScript

- Loosely typed scripting language

- JavaScript function is called when an event in a page occurs
- Glue for the whole AJAX operation
- DOM
  - API for accessing and manipulating structured documents
  - Represents the structure of XML and HTML documents
- CSS
  - Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript
- XMLHttpRequest
  - JavaScript object that performs asynchronous interaction with the server.

### **Examples of usage of AJAX; Asynchronous communication and AJAX application model.**

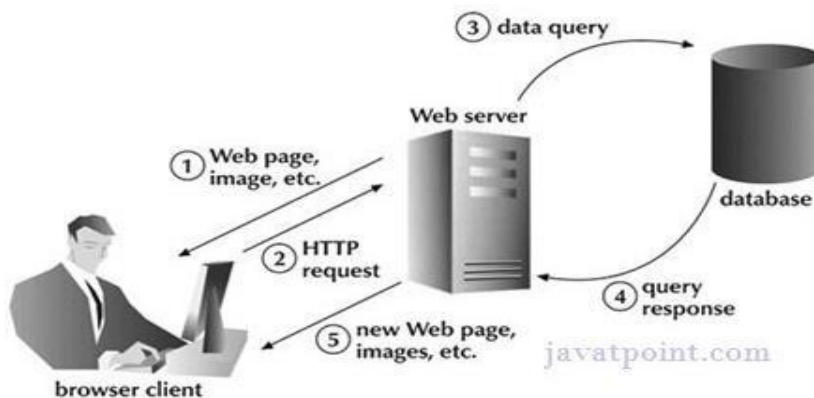
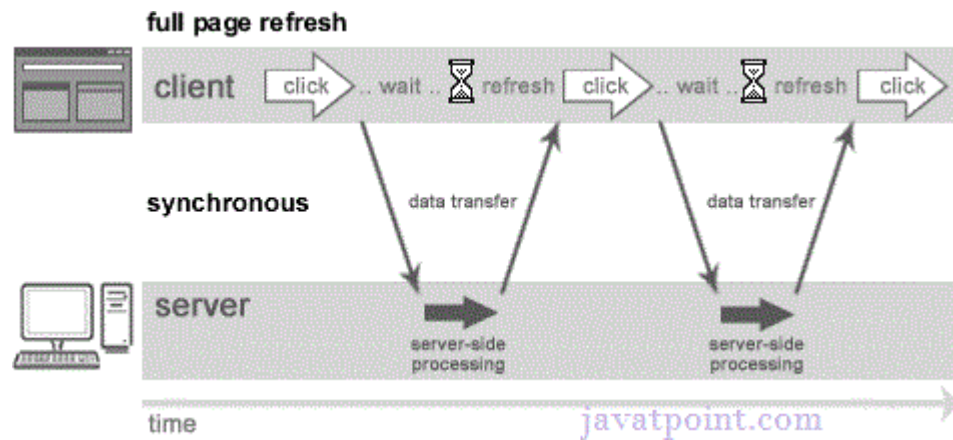
Here is a list of some famous web applications that make use of AJAX.

- **Google Maps**  
A user can drag an entire map by using the mouse, rather than clicking on a button.
- **Google Suggest**  
As you type, Google offers suggestions. Use the arrow keys to navigate the results.
- **Gmail**  
Gmail is a webmail built on the idea that emails can be more intuitive, efficient, and useful.
- **Yahoo Maps (new)**  
Now it's even easier and more fun to get where you're going!

### **Synchronous Vs. Asynchronous Application**

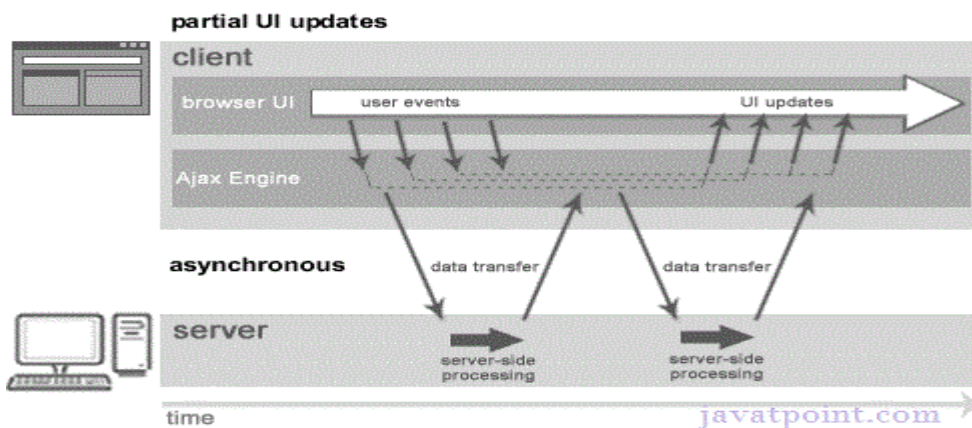
Before understanding AJAX, let's understand classic web application model and AJAX Web application model.

- **Synchronous (Classic Web-Application Model):** A synchronous request blocks the client until operation completes i.e. browser is not unresponsive. In such case, JavaScript Engine of the browser is blocked.



As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

- **Asynchronous (AJAX Web-Application Model):** An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform other operations also. In such case, JavaScript Engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user



gets response from the AJAX Engine. Let's try to understand asynchronous communication by the image given below.

## Ajax with XMLHttpRequest object

The keystone of AJAX is the XMLHttpRequest object.

An AJAX Example program:

### Htmlfile

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
  <h2>Let AJAX change this text</h2>
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>

</body>
</html>
```

- The HTML page contains a <div> section and a <button>.
- The <div> section is used to display information from a server.
- The <button> calls a function (if it is clicked).
- The function requests data from a web server and displays it:

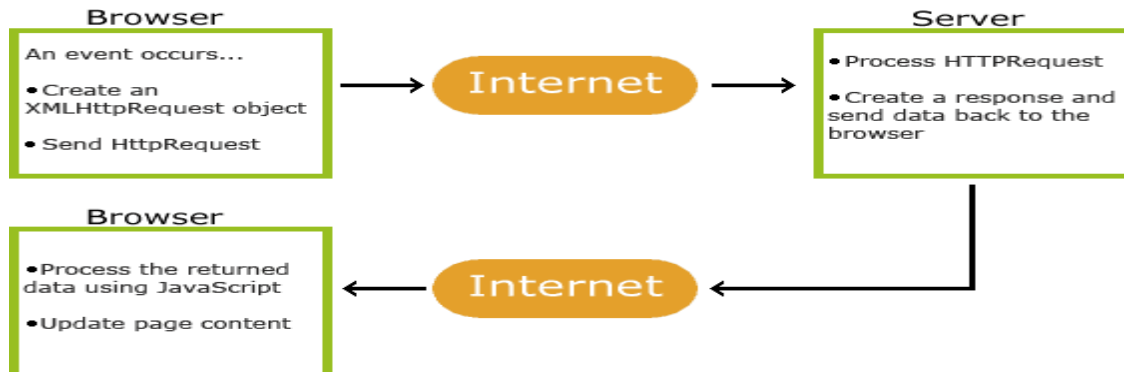
### loadDoc()

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

### "ajax\_info.txt" looks like this:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

### How Ajax Works:



An event occurs in a web page (the page is loaded, a button is clicked)

2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript.

## The XMLHttpRequest Object

All modern browsers support the XMLHttpRequest object.

The **XMLHttpRequest** object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

### Create an XMLHttpRequest Object

All modern browsers (Chrome, Firefox, IE7+, Edge, Safari Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

```
variable = new XMLHttpRequest();
```

Example:

```
<html>
<body>

<h1>The XMLHttpRequest Object</h1>

<p id="demo">Let AJAX change this text.</p>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
```

```
function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            document.getElementById("demo").innerHTML = this.responseText;  
        }  
    };  
    xhttp.open("GET", "ajax_info.txt", true);  
    xhttp.send();  
}  
</script>  
  
</body>  
</html>
```

### Old Versions of Internet Explorer (IE5 and IE6):

Old versions of Internet Explorer (IE5 and IE6) use an ActiveX object instead of the XMLHttpRequest object:

Syntax:

```
variable = new ActiveXObject("Microsoft.XMLHTTP");
```

To handle IE5 and IE6, check if the browser supports the XMLHttpRequest object, or else create an ActiveX object:

Example:

```
<html>  
<body>  
  
<h1>The XMLHttpRequest Object</h1>  
  
<p id="demo">Let AJAX change this text.</p>  
  
<button type="button" onclick="loadDoc()">Change Content</button>  
  
<script>  
function loadDoc() {  
    var xhttp;  
    if (window.XMLHttpRequest) {  
        // code for modern browsers  
        xhttp = new XMLHttpRequest();  
    } else {  
        // code for IE6, IE5  
        xhttp = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
}
```

```
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200) {  
        document.getElementById("demo").innerHTML = this.responseText;  
    }  
};  
xhttp.open("GET", "ajax_info.txt", true);  
xhttp.send();  
}  
</script>  
  
</body>  
</html>
```

## AJAX - Send a Request To a Server

The **XMLHttpRequest** object is used to exchange data with a server

### Send a Request To a Server

To send a request to a server, we use the `open()` and `send()` methods of the **XMLHttpRequest** object:

Syntax:

```
xhttp.open("GET", "ajax_info.txt", true);  
xhttp.send();
```

Method	Description
<code>open(method, url, async)</code>	Specifies the type of request  <i>method</i> : the type of request: GET or POST <i>url</i> : the server (file) location <i>async</i> : true (asynchronous) or false (synchronous)
<code>send()</code>	Sends the request to the server (used for GET)
Sends the request to the server (used for GET)	Sends the request to the server (used for POST)

### GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server).
- Sending a large amount of data to the server (POST has no size limitations).
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

### GET Requests

A simple GET request:

```
<html>
<body>

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "demo_get.asp", true);
  xhttp.send();
}
</script>

</body>
</html>
```

## POST Requests

A simple POST request:

Example:

```
<html>
<body>

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("POST", "demo_post.asp", true);
  xhttp.send();
}
</script>

</body>
</html>
```

```
    };  
    xmlhttp.open("POST", "demo_post.asp", true);  
    xmlhttp.send();  
  }  
</script>  
  
</body>  
</html>
```

### The onreadystatechange Property

- With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.
- The function is defined in the **onreadystatechange** property of the XMLHttpRequest object:

Example:

```
<html>  
<body>  
  
  <div id="demo">  
    <h1>The XMLHttpRequest Object</h1>  
    <button type="button" onclick="loadDoc()">Change Content</button>  
  </div>  
  
  <script>  
    function loadDoc() {  
      var xmlhttp = new XMLHttpRequest();  
      xmlhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
          document.getElementById("demo").innerHTML =  
            this.responseText;  
        }  
      };  
      xmlhttp.open("GET", "ajax_info.txt", true);  
      xmlhttp.send();  
    }  
  </script>  
  
</body>  
</html>
```



## Dynamic web applications through Hidden frames for both GET and POST methods.

### HiddenFrameGET

#### Example:

```
<html>
<head>
    <title>Get Customer Data</title>
<?php

    //customer ID
    $sID = $_GET["id"];

    //variable to hold customer info
    $sInfo = "";
    //database information
    $sDBServer = "localhost";
    $sDBName = "ajax";
    $sDBUsername = "root";
    $sDBPassword = "";

    //create the SQL query string
    $sQuery = "Select * from Customers where CustomerId=".$sID;

    //make the database connection
    $oLink =
    mysql_connect($sDBServer,$sDBUsername,$sDBPassword)
    ; @mysql_select_db($sDBName) or $sInfo = "Unable to
    open database";

    if($sInfo == "") {
        if($oResult = mysql_query($sQuery) and mysql_num_rows($oResult) > 0) {
            $aValues = mysql_fetch_array($oResult,MYSQL_ASSOC);
            $sInfo = $aValues['Name']."<br />".$aValues['Address']."<br />".
                $aValues['City']."<br />".$aValues['State']."<br />".
                $aValues['Zip']."<br /><br />Phone: ".$aValues['Phone']."<br />".
```

```
        "<a href=\"mailto:\".$aValues['E-mail'].\">\".$aValues['E-mail'].\"</a>";
    } else {
        $sInfo = "Customer with ID $sID doesn't exist.";
    }
}

mysql_close($oLink);

?>
<script
    type="text/javascript">
    window.onload =
    function () {
        var divInfoToReturn = document.getElementById("divInfoToReturn");
        top.frames["displayFrame"].displayCustomerInfo(divInfoToReturn.innerHTML);
    };

</script>
</head>
<body>
    <div id="divInfoToReturn"><?php echo $sInfo ?></div>
</body>
</html>
```

- **HiddenFramePOST**

**Example:**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
    <title>Create New Customer</title>
<?php
    //get information
    $sName = $_POST["txtName"];
    $sAddress = $_POST["txtAddress"];
    $sCity = $_POST["txtCity"];
```

```
$sState = $_POST["txtState"];
$sZipCode = $_POST["txtZipCode"];
$sPhone = $_POST["txtPhone"];
$sEmail = $_POST["txtEmail"];

//status message
$sStatus = "";

//database information
$sDBServer = "localhost";
$sDBName = "ajax";
$sDBUsername = "root";
$sDBPassword = "";

//create the SQL query string
$sSQL = "Insert into
        Customers(Name,Address,City,State,Zip,Phone,`E-mail`) ". "
        values ('$sName','$sAddress','$sCity','$sState', '$sZipCode'".
        ", '$sPhone', '$sEmail')";

$link = mysql_connect($sDBServer,$sDBUsername,$sDBPassword);
@mysql_select_db($sDBName) or $sStatus = "Unable to open
database";

if($sStatus == ""){
    if($oResult = mysql_query($sSQL)) {
        $sStatus = "Added customer; customer ID is ".mysql_insert_id();
    } else {
        $sStatus = "An error occurred while inserting; customer not saved.";
    }
}

mysql_close($link);
?>
```

## Department of MCA

---

<script type="text/javascript">

```
        window.onload = function () {  
            top.frames["displayFrame"].saveResult("<?php echo  
            $sStatus ?>");  
        }  
  
    </script>  
</head>  
<body>  
</body>  
</html>
```

## Ajax with XMLHttpRequest object

### Handling multiple XMLHttpRequest objects in the same page

One of the unsung heroes in the HTML5 universe is XMLHttpRequest. Strictly speaking XHR2 isn't HTML5. However, it's part of the incremental improvements browser vendors are making to the core platform.

**XMLHttpRequest (XHR)** is an API in the form of an object whose methods transfer data between a web browser and a web server. The object is provided by the browser's JavaScript environment. Particularly, retrieval of data from XHR for the purpose of continually modifying a loaded web page is the underlying concept of Ajax design. Despite the name, XHR can be used with protocols other than HTTP and data can be in the form of not only XML, but also JSON, HTML or plain text.

#### Using multiple XMLHttpRequest objects:

One solution to the problem of having multiple choices is to create one **XMLHttpRequest** object per request. For example, you might modify **lunch.html** into, for example, **double.html**, which creates two **XMLHttpRequest** objects: **XMLHttpRequestObject** and **XMLHttpRequestObject2**, like this:

```
var XMLHttpRequestObject = false;
if (window.XMLHttpRequest)
{
XMLHttpRequestObject=newXMLHttpRequest();
XMLHttpRequestObject.overrideMimeType("text/xml");
}
else if
(window.ActiveXObject)
{
XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");
}
var XMLHttpRequestObject2 = false;
if (window.XMLHttpRequest2)
{
XMLHttpRequestObject2=new XMLHttpRequest();
XMLHttpRequestObject2.overrideMimeType("text/xml");
}
else if (window.ActiveXObject)
```

```
{ XMLHttpRequestObject2 = new ActiveXObject("Microsoft.XMLHTTP");  
}
```

This way, when the user clicks one button, one **XMLHttpRequest** object responds. Because there are two buttons and two **XMLHttpRequest** objects.

### Using an array of XMLHttpRequest objects

```
var index = ["ESL_SC2", "OgamingSC2", "cretetion", "freecodecamp",  
"storbeck", "habathcx", "RobotCaleb", "noobs2ninja"];  
var request = new XMLHttpRequest();  
(function loop(i, length) {  
    if (i >= length) {  
        return;  
    }  
    var url = "https://wind-bow.glitch.me/twitch-api/channels/" + index[i];  
  
    request.open("GET", url);  
    request.onreadystatechange = function() {  
        if (request.readyState === XMLHttpRequest.DONE && request.status ===  
200) {  
            var data = JSON.parse(request.responseText);  
            console.log('--> ' + i + ' id: ' + data._id);  
            loop(i + 1, length);  
        }  
    }  
    request.send();  
})(0, index.length);
```

## Ajax Patterns

A design pattern describes programming techniques to solve common problems.

### Predictive Fetch Algorithm

- In case of traditional web applications, the application reacts only when there is an interaction.
- This is called “fetch on demand”. The user action tells the server what data should be retrieved.
- In the predictive fetch algorithm, the application guesses what the user is going to do next and retrieves the appropriate data.
- Determining the future action of the user is just or guess based on the users intentions.



- For eg: say a user is reading an online article of 3 pages. It can be assumed here that if the user is reading the 1<sup>st</sup> page for few seconds, the person will also be interested in reading the 2<sup>nd</sup> page.
- Hence the 2<sup>nd</sup> page can be downloaded at the background before the user explicitly clicks on the 'Next'.
- Therefore when the user clicks on next the 2<sup>nd</sup> page instantaneously appears reducing the response time.
- Similarly the 3<sup>rd</sup> page can be downloaded when the user reads 2<sup>nd</sup> page for a few seconds.
- This extra data being downloaded is cached on the client.
- Some approach can be applied in emails. If a person starts composing a mail, it is logical to anticipate that the mail would be sent to someone in the address book so this can be preloaded and kept.
- By using ajax to fetch information related to any possible next step, can overload the server.
- Therefore this algorithm has to be implemented only when it is logical to assume that information will be requisite to completing the user's next request

### **Submission Throttling**

- If retrieving data from the server is one part of the problem sending data to server is another.
- Since in AJAX page refreshes needs to be avoided, it is important to know when to send user data to the server.
- One approach that could be taken is to send data to server on every user interaction. But this results in a lot of requests submitted to the server in a short period.
- In case of submission throttling, the data to be sent to the server is buffered on the client.
- This data is then sent to the server at predefined times.
- The delay from typing to sending data is fine tuned such that it doesn't seem like a delay to the user.
- Then a client side function is invoked that begins buffering the data.
- Then a client side function is invoked that begins buffering the data.
- It can be sent at a predefined time interval.
- This determination depends on the usecase being used.
- After the data is sent, the application continues to gather data.

### **Periodic Refresh**

- This algorithm is basically used to notify users of updated information.
- It describes the process of checking for new server information in specific intervals.
- This approach is called 'polling' and it requires the browser to keep track of when another request to the server should take place.

### **Multi-Stage Download**

- In the modern web experience, a webpage is loaded with information, pictures, flash animations etc. Therefore to download a page all these elements need to be download leading to very slow speeds.
- Multistage download is an AJAX pattern wherein only the most basic functionality is loaded into a page initially.
- Upon completion of this, the page then begins to download other components that should appear on the page.
- If the user leaves the page before all components are downloaded, then it is not much of consequence as all useful info was already displayed/
- If however the user stays on the page for some extended period of time, the extra functionality is loaded at the background and available to the user.
- The developer decides what is downloaded at what point in time.
- This can be dealt with by providing a basic and simple interface for the browser that don't support AJAX and a richer interface for browsers that do.

### **Fall Back Patterns**

- All the above methods work fine when there is no problem at the server side.
- The following problems can occur:
  - The request might never make it to the server. An error might occur at the server.

### **Cancel Pending Requests**

- If error occurs on the server like a file not found or an internal server error, then it
  - doesn't make sense to try again after a few minutes.
- Such problems need an administrator to fix it.
- In such a situation all the pending requests are simply cancelled.
- This can be done by setting a flag somewhere in the code that says "don't send any more request".

- This solution has maximum impact on the periodic refresh pattern.

### **Try Again**

- Another option is to silently keep trying for either a specified amount of time or a particular number of files.
- This problem is handled behind the scenes without generally notifying the user.
- Unless the ajax functionality is key to the user's experience, it is not needed to notify about the failures.

### **XMLHTTP Object**

- The xmlhttp object was created to enable developers to initiate HTTP requests from anywhere in an application.
- These requests were intended to return an XML so the XMLHttpRequest object provided an easy way to access this info in the form of an XML document.
- This XMLHttpRequest Object was modified to suit various browsers and this version known as the "XMLHttpRequest Object" or the XHR.

### **Creating an XHR object**

- Since most of the Microsoft's implementation is on ActiveX control, the ActiveX
  - object class should be used in the javascript.
  - `var oxhr=new ActiveXObject("Microsoft.xmlHttp");`
- The signatures differ for various versions of the MSXML library.
- The various signatures are:
  - Microsoft.xmlHttp
  - MSXML2.xmlHttp
  - MSXML2.xmlHttp.3.0
  - MSXML2.xmlHttp.4.0
  - MSXML2.xmlHttp.5.0
  - MSXML2.xmlHttp .6.0
- The best way to find out which version is supported is to create all the probable ones. The ActiveX Control throws an error if it is not supported.

```
function createXHR() {  
    var aVersions = [ "MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0"];  
    for (var i = 0; i < aVersions.length; i++)  
    {  
        try  
        {  
            var oXHR = new ActiveXObject(aVersions[i]);
```

```
return oXHR;
} catch (oError)
{
//Do nothing
}
}
throw new Error("MSXML is not installed.");
}
```

- In case of browsers apart from IE like Mozilla, Safari etc and also IE7 uses a simple code

**var oXHR=new XMLHttpRequest();**

- In order to create a cross-browser way of creating XHR objects, both the methods are combined as shown:

```
function createXHR()
{
if (typeof XMLHttpRequest != "undefined")
{
return new XMLHttpRequest();
}
else if (window.ActiveXObject)
{
var aVersions = [ "MSXML2.XMLHttp.6.0", "MSXML2.XMLHttp.3.0"];
for (var i = 0; i < aVersions.length; i++)
{
Try
{
var oXHR = new ActiveXObject(aVersions[i]); return oXHR;
} catch (oError)
{
//Do nothing
}
}
}
throw new Error("XMLHttpRequest object could not be created.");
}
```

- Another way of creating cross-browser XHR Objects is to use the ZXML library which already has a cross-browser code written.
- Therefore a single function can be used to any browser.

**var oXHR = xmlhttp.createRequest();**

### Using XHR

- In order to send the http request from javascript, the first step is to call the open() method which initializes the object.
- The following are the 3 arguments for this method.
  - Request Type: A string indicating the request type to be made-typically,GET or POST
  - URL: A string indicating the URL to send the request to
  - Async:A Boolean value indicating whether the request should be made asynchronously
- The next step is to define an 'onreadystatechange' event handler.
- The XHR object has a property called 'readyState' that changes as the request goes through and the response is received.
- Every time the readyState property changes from one value to another, the readyStateChange event fires and the onreadystatechange event handler is called.

```
var oXHR = XMLHttpRequest.createRequest();
oXHR.open("get", "info.txt", true);
oXHR.onreadystatechange = function ()
{
if (oXHR.readyState == 4)
{
alert("Got response.");
}
};
oXHR.send(null);
```

- The send method is called at the end which actually sends the request.
- If the request doesn't have a body, then a null should be passed in.
- In the above code, after response is received, an alert is displayed.
- This returns the context of info.txt. if there are any errors like the file didn't exist and so on, it needs to be handled.

```
if (oXHR.status == 200)
{
alert("Data returned is: " + oXHR.responseText);
}
else
{
alert("An error occurred: " + oXHR.statusText);
}
```

- The response Header can be accessed using the 'getResponseHeader()' method and passing the name of header to be retrieved.

```
var sContentType = oXHR.getResponseHeader("Content-Type");
if (ContentType == "text/xml")
{
    alert("XML content received.");
}
else if (sContentType == "text/plain")
{
    alert("Plain text content received.");
}
else
{
    alert("Unexpected content received.");
}
```

- The getAllResponseHeader() method can be used to return a string containing all the headers. Each header in this string is separated by '\n' or '\r\n'.

```
var sHeaders = oXHR.getAllResponseHeaders();
var aHeaders = sHeaders.split(/\r?\n/);
for (var i=0; i < aHeaders.length; i++)
{
    alert(aHeaders[i]);
}
```

- The 'setRequestHeader()' method can be set headon on the request before it is sent.

```
var oXHR = XMLHttpRequest.createRequest();

oXHR.open("get", "info.txt", true);
oXHR.onreadystatechange = function ()
{
    if (oXHR.readyState == 4)
    {
        alert("Got response.");
    }
};
oXHR.setRequestHeader("myheader", "myvalue");
oXHR.send(null);
```

### **Synchronous Requests**

- In case of synchronous requests, it is not required to assign the onreadystatechange event handler.

- The response will have been received by the time the send() method returns.



```
var oXHR = XMLHttpRequest.createRequest();
oXHR.open("get", "info.txt", false);
oXHR.send(null);
if (oXHR.status == 200)
{
    alert("Data returned is: " + oXHR.responseText);
}
else
    alert("An error occurred: " + oXHR.statusText);
}
```

### Cache Control

- Whenever repeated calls to the same page all dealt with browser caching needs to be considered.
- Web browsers tend to cache certain resources to improve the speed with which sites are downloaded and displayed.
- This increases the speed on frequently visited web sites.
- Caching can be dealt with by including a header with caching information on any data being sent from the server to the browser.
  - Cache\_Control : no\_cache
  - Expires : Fri, 30 Oct 1998 14:19:41 GMT
- This tells the browser not to cache the data coming from the specific URL. Instead the browser always calls a new version from the server instead of a saved version from its own cache.

## MODULE-5

### Introduction to Bootstrap.

#### What is Bootstrap?

- Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
- Bootstrap is completely free to download and use!
- In other words , Bootstrap is a collection of CSS classes and JS Functions the way you get to ready to use.
- Bootstrap also gives you the ability to easily create responsive designs.

#### What is responsive design?

Responsive web design is about creating web sites which automatically adjust themselves to look good on all devices, from small phones to large desktops.



#### Bootstrap Environment Setup

It is very easy to setup and start using Bootstrap. This chapter will explain how to download and setup Bootstrap. It will also discuss the Bootstrap file structure, and demonstrate its usage with an example.

## Download Bootstrap

You can download the latest version of Bootstrap from <http://getbootstrap.com/>. When you get into this link, you will get to see a screen as below:



Here you can see two buttons:

- **Download Bootstrap:** Clicking this, you can download the precompiled and minified versions of Bootstrap CSS, JavaScript, and fonts. No documentation or original source code files are included.
- **Download Source:** Clicking this, you can get the latest Bootstrap LESS and JavaScript source code directly from GitHub.

### Installation:

#### Local Installation

You will need to include these files:

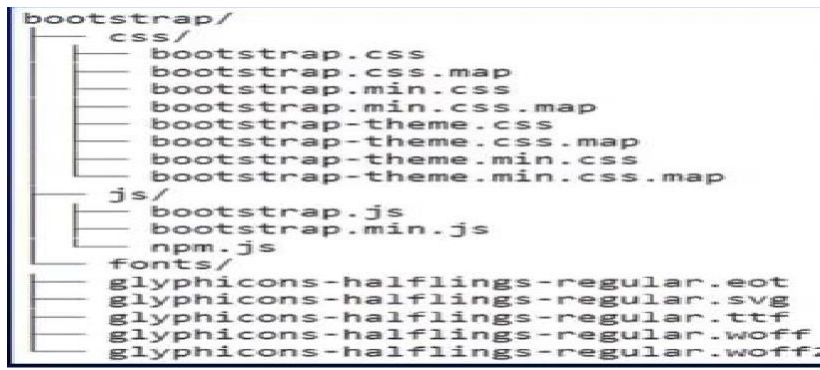
- Bootstrap.min.css
- Bootstrap.min.js
- JQuery.min.js

#### CDN:

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
```

## Bootstrap File structure

The Bootstrap download includes three folders: css, js, and img. For simplicity, add these to the root of your project. Included are also minified versions of the CSS and Javascript. Both the uncompressed and the minified versions do not need to be included. For the sake of brevity, I use the uncompressed during development, and then switch to the compressed version in production.



## Basic HTML Template

Normally, a web project looks something like this: Basic HTML Layout.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://code.jquery.com/jquery.js"></script>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Here you can see the jquery.js and bootstrap.min.js and bootstrap.min.css files are included to make a normal HTML file to Bootstrapped Template.

## Global Styles

With Bootstrap, a number of items come prebuilt. In particular, these default styles give special treatment to typography and links.

- margin has been removed from the body, and content will snug up to the edges of the browser window.
- background-color: white; is applied to the body

- Bootstrap is using the @baseFontFamily, @baseFontSize, and @baseLine Height attributes as our typographic base. This allows the height of headings, and other content around the site to maintain a similar line height.
- Bootstrap sets the global link color via @linkColor and applies link underlines only on :hover

## Grid System

- In graphic design, a grid is a structure (usually two-dimensional) made up of a series of intersecting straight (vertical, horizontal) lines used to structure the content.
- It is widely used to design layout and content structure in print design. In web design, it is a very effective method to create a consistent layout rapidly and effectively using HTML and CSS.

### Bootstrap Default Grid System:

- Bootstrap's grid system allows up to 12 columns across the page.
- If you do not want to use all 12 columns individually, you can group the columns together to create wider columns:

span 1	span 1	:	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span			span 4					span 4			
span			span 8								
span 6						span 6					
span 12											

## Grid Classes

The Bootstrap grid system has four classes:

- **xs** (for phones - screens less than 768px wide)
- **sm** (for tablets - screens equal to or greater than 768px wide)
- **md** (for small laptops - screens equal to or greater than 992px wide)
- **lg** (for laptops and desktops - screens equal to or greater than 1200px wide)

The classes above can be combined to create more dynamic and flexible layouts.

## Grid System Rules

Some Bootstrap grid system rules:

- Rows must be placed within a **.container** (fixed-width) or **.container-fluid** (full-width) for proper alignment and padding
- Use rows to create horizontal groups of columns
- Content should be placed within columns, and only columns may be immediate children

of rows

- Predefined classes like `.row` and `.col-sm-4` are available for quickly making grid layouts
- Columns create gutters (gaps between column content) via padding. That padding is offset in rows for the first and last column via negative margin on `.rows`
- Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three `.col-sm-4`
- Column widths are in percentage, so they are always fluid and sized relative to their parent element

## Basic grid HTML

```
<div class="container">
  <div class="row">
    <div class="col-*-*"></div>
    <div class="col-*-*"></div>
  </div>
  <div class="row">...</div>
</div>
<div class="container">....
```

## Offsetting Columns

Offsets are a useful feature for more specialized layouts. They can be used to push columns over for more spacing, for example. The `.col-xs=*` classes don't support offsets, but they are easily replicated by using an empty cell.

To use offsets on large displays, use the `.col-md-offset-*` classes. These classes increase the left margin of a column by \* columns where \* range from 1 to 11.

In the following example we have `<div class="col-md-6">..</div>`, we will center this using class `.col-md-offset-3`.

Example:

```
div class="container">

  <h1>Hello, world!</h1>
  <div class="row" >
    <div class="col-xs-6 col-md-offset-3"
      style="background-color: #dedef8;box-
      shadow:
      inset 1px -1px 1px #444, inset -1px 1px 1px #444;">
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit.
```

```

    </p>
  </div>

</div>
</div>

```

Output:

Hello, world!

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

### Nesting columns

To nest your content with the default grid, add a new `.row` and set of `.col-md-*` columns within an existing `.col-md-*` column. Nested rows should include a set of columns that add up to 12.

Example:

```

<div class="row">
  <div class="col-sm-8">
    .col-sm-8
    <div class="row">
      <div class="col-sm-6">.col-sm-6</div>
      <div class="col-sm-6">.col-sm-6</div>
    </div>
  </div>
  <div class="col-sm-4">.col-sm-4</div>
</div>

```

### Fluid Grid System

- The fluid grid system uses percents 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 the exact same, making it easy to flip between fixed and fluid grids.
- To offset, you operate in the same way as the fixed grid system works by adding `.offset*` to any column to shift by your desired number of columns.

```

<div class="row-fluid">
  <div class="col-md-4">...</div>
  <div class="col-md-4">...</div>
</div>

<div class="row-fluid">

```



```
<div class=" col-md-4">...</div>
<div class=" col-md-4 offset2">...</div>
</div>
```

## 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>`.

## Typography

- Starting with Typography, with the default font stack, Bootstrap uses Helvetica Neue, Helvetica, Arial, and sans-serif.
- These are all standard fonts, and included as defaults on all major computers, falling back to sans-serif, the catch all to tell the browser to use the default font that the user has decided.
- All body copy has the font-size set at 14 pixels, with the line-height set at 20 pixels. The `<p>` tag has a margin-bottom of 10 pixels, or half the line-height.

## Headings

All six standard heading levels have been styled in Bootstrap, with the `<h1>` at 36 pixels high, and the `<h6>` down to 12 pixels in height (for reference, body text is 14 pixels in height by default).

In addition, to add an inline sub-heading 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.

Example:

```
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 `<strong>` tag. This will add font-weight- bold to the selected text.

## Italics

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

## Emphasis classes

Along with `<strong>` and `<em>`, Bootstrap offers a few other classes that can be used to provide emphasis. These could be applied to paragraphs, or spans.

Emphasis Classes:

```
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>
```

## Bootstrap Emphasis Classes

This content is muted

This content carries a warning class

This content carries an error class

This content carries an info class

This content carries a success class

This content has *emphasis*, and can be **bold**

## Lists

Bootstrap supports ordered lists, unordered lists, and definition lists.

- **Ordered lists:** An ordered list is a list that falls in some sort of sequential order and is prefaced by numbers.
- **Unordered lists:** An unordered list is a list that doesn't have any particular order and is traditionally styled with bullets. If you do not want the bullets to appear then you can remove the styling by using the class `.list-unstyled`. You can also place all list items on a single line using the class `.list-inline`.
- **Definition lists:** In this type of list, each list item can consist of both the `<dt>` and the `<dd>` elements. `<dt>` stands for *definition term*, and like a dictionary, this is the term (or phrase) that is being defined. Subsequently, the `<dd>` is the definition of the `<dt>`.

You can make terms and descriptions in `<dl>` line up side-by-side using class `dl-horizontal`.

### Example:

```

<h4>Example of Ordered List</h4>
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ol>
<h4>Example of UnOrdered List</h4>
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>
<h4>Example of Unstyled List</h4>
<ul class="list-unstyled">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>
<h4>Example of Inline List</h4>
<ul class="list-inline">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
</ul>

```

```

<h4>Example of Definition List</h4>

```

```

<dl>
  <dt>Description 1</dt>
  <dd>Item 1</dd>
  <dt>Description 2</dt>
  <dd>Item 2</dd>
</dl>
<h4>Example of Horizontal Definition List</h4>
<dl class="dl-horizontal">
  <dt>Description 1</dt>
  <dd>Item 1</dd>
  <dt>Description 2</dt>
  <dd>Item 2</dd>
</dl>

```

## Output:

### Example of Ordered List

1. Item 1
2. Item 2
3. Item 3
4. Item 4

### Example of UnOrdered List

- Item 1
- Item 2
- Item 3
- Item 4

### Example of Unstyled List

Item 1  
Item 2  
Item 3  
Item 4

### Example of Inline List

Item 1 Item 2 Item 3 Item 4

### Example of Definition List

**Description 1**  
Item 1  
**Description 2**  
Item 2

### Example of Horizontal Definition List

<b>Description 1</b>	Item 1
<b>Description 2</b>	Item 2

## Codes

Bootstrap allows you to display code with two different key ways:

- The first is the `<code>` tag. If you are going to be displaying code inline, you should use the `<code>` tag.
- Second is the `<pre>` tag. If the code needs to be displayed as a standalone block element or if it

has multiple lines, then you should use the <pre> tag.

Make sure that when you use the `<pre>` and `<code>` tags, you use the unicode variants for the opening and closing tags: `&lt;` and `&gt;`.

### Example:

```
<p><code>&lt;header&gt;</code> is wrapped as an inline element.</p>
<p>To display code as a standalone block element use &lt;pre&gt; tag as:
<pre>
  &lt;article&gt;
    &lt;h1&gt;Article
    Heading&lt;/h1&gt;
  &lt;/article&gt;
</pre>
```

### Output:

`<header>` is wrapped as an inline element.

To display code as a standalone block element use `<pre>` tag as:

```
<article>
<h1>Article Heading</h1>
</article>
```

## Tables

Bootstrap provides a clean layout for building tables. Some of the table elements supported by Bootstrap are:

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

### Basic Table

If you want a nice, basic table style with just some light padding and horizontal dividers, add the base class of *.table* to any table as shown in the following example:

```
<table class="table">
  <caption>Basic Table Layout</caption>
  <thead>
    <tr>
      <th>Name</th>
      <th>City</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Tanmay</td>
      <td>Bangalore</td>
    </tr>
    <tr>
      <td>Sachin</td>
      <td>Mumbai</td>
    </tr>
  </tbody>
</table>
```

Name	City
Tanmay	Bangalore
Sachin	Mumbai

## Optional Table Classes

Along with the base table markup and the *.table* class, there are a few additional classes that you can use to style the markup. Following sections will give you a glimpse of all these classes.

### Striped table

By adding the *.table-striped* class, you will get stripes on rows within the `<tbody>` as seen in the

following example:

```
<table class="table table-striped">
  <caption>Striped Table Layout</caption>
  <thead>
    <tr>
      <th>Name</th>
      <th>City</th>
      <th>Pincode</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Tanmay</td>
      <td>Bangalore</td>
      <td>560001</td>
    </tr>
    <tr>
      <td>Sachin</td>
      <td>Mumbai</td>
      <td>400003</td>
    </tr>
    <tr>
      <td>Uma</td>
      <td>Pune</td>
      <td>411027</td>
    </tr>
  </tbody>
</table>
```

**Output:**

### Bordered table

Name	City	Pincode
Tanmay	Bangalore	560001
Sachin	Mumbai	400003
Uma	Pune	411027



By adding the *.table-bordered* class, you will get borders surrounding every element and rounded corners around the entire table as seen in the following example:

```
<table class="table table-bordered">
  <caption>Bordered Table Layout</caption>
  <thead>
    <tr>
      <th>Name</th>
      <th>City</th>
      <th>Pincode</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Tanmay</td>
      <td>Bangalore</td>
      <td>560001</td>
    </tr>
    <tr>
      <td>Sachin</td>
      <td>Mumbai</td>
      <td>400003</td>
    </tr>
    <tr>
      <td>Uma</td>
      <td>Pune</td>
      <td></td>
    </tr>
  </tbody>
</table>
```

Bordered Table Layout

Name	City	Pincode
Tanmay	Bangalore	560001
Sachin	Mumbai	400003
Uma	Pune	411027

### Hover table

By adding the *.table-hover* class, a light gray background will be added to rows while the cursor hovers over them, as seen in the following example:

```
<table class="table table-hover">
  <caption>Hover Table Layout</caption>
  <thead>
    <tr>
      <th>Name</th>
      <th>City</th>
      <th>Pincode</th>
    </tr>
```

```

</thead>
<tbody>
<tr>
    <td>Tanmay</td>
    <td>Bangalore</td>
    <td>560001</td>
</tr>
<tr>
    <td>Sachin</td>
    <td>Mumbai</td>
    <td>400003</td>
</tr>
<tr>
    <td>Uma</td>
    <td>Pune</td>
    <td>411027</td>
</tr>
</tbody>
</table>

```

Hover Table Layout

Name	City	Pincode
Tanmay	Bangalore	560001
Sachin	Mumbai	400003
Uma	Pune	411027

### Condensed table

By adding the *.table-condensed* class, row padding is cut in half to condense the table. as seen in the following example. This is useful if you want denser information.

```

<table class="table table-condensed">
  <caption>Condensed Table Layout</caption>
  <thead>
  <tr>
    <th>Name</th>
    <th>City</th>
    <th>Pincode</th>
  </tr>

```

```

</thead>
<tbody>
<tr>
    <td>Tanmay</td>
    <td>Bangalore</td>
    <td>560001</td>
</tr>
<tr>
    <td>Sachin</td>
    <td>Mumbai</td>
    <td>400003</td>
</tr>
<tr>
    <td>Uma</td>
    <td>Pune</td>
    <td>411027</td>
</tr>
</tbody>
</table>

```

Condensed Table Layout

Name	City	Pincode
Tanmay	Bangalore	560001
Sachin	Mumbai	400003
Uma	Pune	411027

## Forms

Another one of the highlights of using Bootstrap is the attention that is paid to forms. As a web developer, one of my least favorite things to do is style forms. Bootstrap makes it easy to do with the simple HTML markup and extended classes for different styles of forms.

The basic form structure comes styled in Bootstrap, without needing to add any extra helper classes. If you use the placeholder, it is only supported in newer browsers. In older ones, no text will be displayed.

### Basic Form Structure.

```

<form>
<fieldset>
<legend>Legend</legend>
<label for="name">Label name</label>
<input type="text" id="name" placeholder="Type something...">

```

```

<span class="help-block">Example block-level help text here.</span>
<label class="checkbox" for="checkbox">
<input type="checkbox" id="checkbox"> Check me out
</label>
<button type="submit" class="btn">Submit</button>
</fieldset>
</form>

```

**Output:**
**Optional Form Layouts**

With a few helper classes, you can dynamically update the layout of your form. Bootstrap comes with a few preset styles you can use.

**Search Form**

Add `.form-search` to the form tag, and then `.search-query` to the `<input>` for an input box with rounded corners, and an inline submit button.

**Form Structure:**

```

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

```

**Output:**

## Inline Form

To create a form where all of the elements are inline, and labels are along side, add the class .form-inline to the form tag. To have the label and the input on the same line, use the horizontal form below.

### Inline form structure

```
<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>
```

### Output:

## Horizontal Form

Bootstrap also comes with a pre-baked 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 this, but Bootstrap manages to do it without. 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

### Horizontal Form Code

```
<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>

```

### Supported Form Controls

Bootstrap natively supports the most common form controls. Chief among them, in put, textarea, checkbox and radio, and select.

#### Inputs

The most common form text field is the input—this is where users will enter most of the essential form data. Bootstrap offers support for all native HTML5 input types: text, password, datetime, datetime-local, date, month, time, week, number, email, url, search, tel, and color.

#### Input example:

```
<input type="text" placeholder="Text input">
```



#### Textarea

The textarea is used when you need multiple lines of input. You'll find you mainly **Example:** modify the rows attribute, changing it to the number of rows that you need to support (fewer rows = smaller box, more rows = bigger box).

#### Example:

```
<textarea rows="3"></textarea>
```

## Output

## Checkboxes and radios

Checkboxes and radio buttons are great for when you want users to be able to choose from a list of preset options. When building a form, use checkbox if you want the user to select any number of options from a list, and radio if you want to limit them to just one selection.

Example:

```
<label class="checkbox">
<input type="checkbox" value="">
Option one is this and that—be sure to include why it's great
</label>

<label class="radio">
<input type="radio" name="optionsRadios" id="optionsRadios1" value="option1" checked>
Option one is this and that—be sure to include why it's great
</label>
<label class="radio">
<input type="radio" name="optionsRadios" id="optionsRadios2" value="option2"> Option two
can be something else and selecting it will deselect option one
</label>
```

## Output:

## Selects

A select is used when you want to allow the user to pick from multiple options, but by default it only allows one. It's best to use `<select>` for list options of which the user is familiar such as states or numbers. Use `multiple="multiple"` to allow the user to select more than one option. If



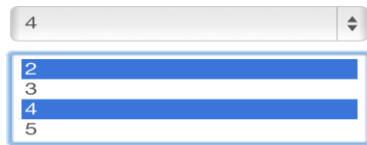
you only want the user to choose one option, use type="radio".

Example:

```
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
</select>

<select multiple="multiple">
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
</select>
```

**Output:**



### Extending Form Controls

In addition to the basic form controls listed in the previous section, Bootstrap offers a few other form components to complement the standard HTML form elements; for example, it lets you easily prepend and append content to inputs.

#### Prepended and Appended Inputs

By adding prepended and appended content to an input field, you can add common elements to the text users input, like the dollar symbol, the @ for a Twitter username or anything else that might be common for your application interface. To use, wrap the input in a div with class input-prepend (to add the extra content before the user input) or input-append (to add it after). Then, within that same <div>, place your extra content inside a <span> with an add-on class, and place the <span> either before or after the <input> element.

Example:

```
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">.00</span>
</div>
```

**Output:**
**Attach Multiple Buttons Code Example.**

If you are appending a button to a search form, you will get the same nice rounded corners that you would expect.

```
<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>
```

**Output:**
**Form Control Sizing**

With the default grid system that is inherent in Bootstrap, you can use the .span\* system for sizing form controls. In addition to the span column-sizing method, you can also use a handful of classes that take a relative approach to sizing. If you want the input to act as a block level element, you can add .input-block-level and it will be the full width of the container element.

**Example:**

```
<input class="input-block-level" type="text" placeholder=".input-block-level">
```

**Output:**

## Relative Input Controls

### Example

```
<input class="input-mini" type="text" placeholder=".input-mini">
<input class="input-small" type="text" placeholder=".input-small">
<input class="input-medium" type="text" placeholder=".input-medium">
<input class="input-large" type="text" placeholder=".input-large">
<input class="input-xlarge" type="text" placeholder=".input-xlarge">
<input class="input-xxlarge" type="text" placeholder=".input-xxlarge">
```

### output:

## Uneditable Text

If you want to present a form control, but not have it editable, simply add the `.uneditable-input` class.

### Example:

```
<span class="input-xlarge uneditable-input">Some value here</span>
```

### Example:

## Form Actions

At the bottom of a horizontal-form you can place the form actions. Then inputs will correctly line up with the floated form controls.

### Example:

```
<div class="form-actions">
<button type="submit" class="btn btn-primary">Save changes</button>
<button type="button" class="btn">Cancel</button>
</div>
```

**Output:**


 A screenshot of a Bootstrap form control. It features a light gray background with a white border. Inside, there are two buttons: a blue 'Save changes' button and a light gray 'Cancel' button.

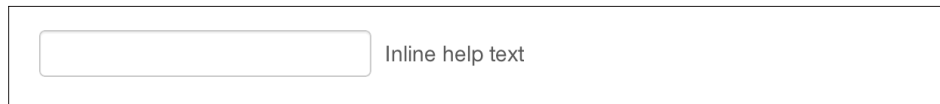
## Help Text

Bootstrap form controls can have either block or inline text that flows with the inputs.

### Example

```
<input type="text"><span class="help-inline">Inline help text</span>
```

### Output:


 A screenshot of the output of the example code. It shows a text input field followed by the text 'Inline help text' in a smaller font size, indicating it is help text.

## Validation States

Bootstrap includes validation styles for error, warning, info, and success messages. To use, simply add the appropriate class to the surrounding .control-group.

### Example:

```
<div class="control-group warning">
<label class="control-label" for="inputWarning">Input with warning</label>
<div class="controls">
<input type="text" id="inputWarning">
<span class="help-inline">Something may have gone wrong</span>
</div>
</div>
<div class="control-group error">
<label class="control-label" for="inputError">Input with error</label>
<div class="controls">
<input type="text" id="inputError">
<span class="help-inline">Please correct the error</span>
</div>
</div>
<div class="control-group success">
<label class="control-label" for="inputSuccess">Input with success</label>
<div class="controls">
<input type="text" id="inputSuccess">
<span class="help-inline">Woohoo!</span>
```

&lt;/div&gt;

&lt;/div&gt;

**Output:**









Input with warning	<input type="text"/>	Something may have gone wrong
Input with error	<input type="text"/>	Please correct the error
Input with info	<input type="text"/>	Username is taken
Input with success	<input type="text"/>	Woohoo!

**Buttons**

Bootstrap provides different styles of buttons:

Now, buttons, and links can all look alike with Bootstrap, anything that is given that class of btn will inherit the default look of a grey button with rounded corners. Adding extra classes will add colors to the buttons.

The following table shows different buttons:

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
	btn btn-info	Used as an alternative to the default styles
	btn btn-success	Indicates a successful or positive action
	btn btn-warning	Standard gray button with gradient
	btn btn-danger	Indicates a dangerous or potentially negative action
	btn btn-inverse	Alternate dark gray button, not tied to a semantic action or use
	btn btn-link	Deemphasize a button by making it look like a link while maintaining button behavior

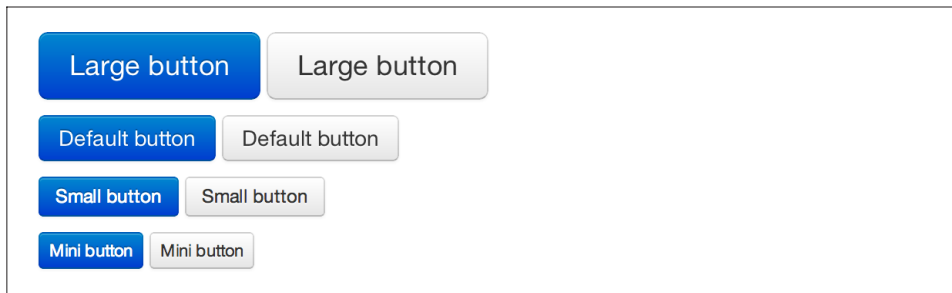
## Button Sizes

If you need larger or smaller buttons, simply add `.btn-large`, `.btn-small`, or `.btn-mini` to links or buttons.

### Example:

```
<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>
```

### Output:



If you want to create buttons that display like a block level element, simply add the `btn-block` class. These buttons will display at 100% width.

### Example:

```
<button class="btn btn-large btn-block btn-primary" type="button">Block level
button</button>
<button class="btn btn-large btn-block" type="button">Block level button</button>
```

### Output:





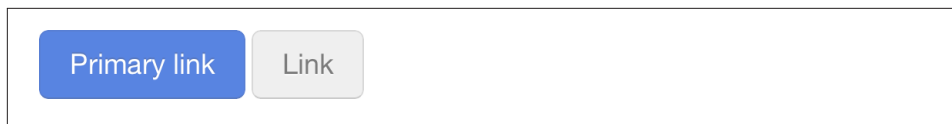
### Disabled Button Styling

For anchor elements, simply add the class of `.disabled` to the tag, and the link will drop back in color, and will lose the gradient.

#### Example:

```
<a href="#" class="btn btn-large btn-primary disabled">Primary link</a>
<a href="#" class="btn btn-large disabled">Link</a>
```

#### Output:



### Images

Images have three classes to apply some simple styles. They are `.img-rounded` that adds `border-radius:6px` to give the image rounded corners, `.img-circle` that adds makes the entire image a circle by adding `border-radius:500px` making the image round, and lastly, `img-polaroid`, that adds a bit of padding and a grey border.

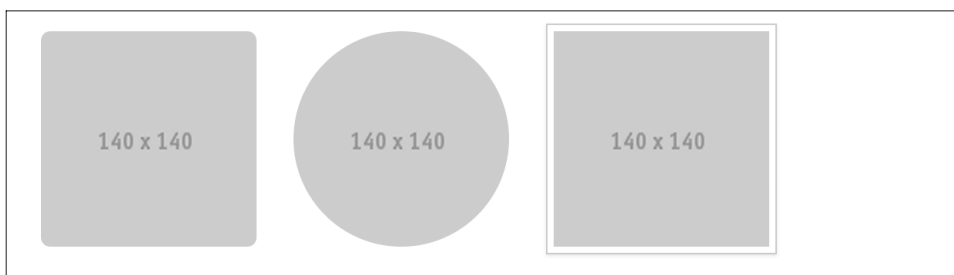
#### Example:

```



```

#### Output:



## Icons

Bootstrap bundles 140 icons into one sprite that can be used with buttons, links, navigation, and form fields. The icons are provided by Glyphicons.

 icon-glass	 icon-music	 icon-search	 icon-envelope
 icon-heart	 icon-star	 icon-star-empty	 icon-user
 icon-film	 icon-th-large	 icon-th	 icon-th-list
 icon-ok	 icon-remove	 icon-zoom-in	 icon-zoom-out
 icon-off	 icon-signal	 icon-cog	 icon-trash
 icon-home	 icon-file	 icon-time	 icon-road
 icon-download-alt	 icon-download	 icon-upload	 icon-inbox
 icon-play-circle	 icon-repeat	 icon-refresh	 icon-list-alt
 icon-lock	 icon-flag	 icon-headphones	 icon-volume-off
 icon-volume-down	 icon-volume-up	 icon-qrcode	 icon-barcode
 icon-tag	 icon-tags	 icon-book	 icon-bookmark
 icon-print	 icon-camera	 icon-font	 icon-bold
 icon-italic	 icon-text-height	 icon-text-width	 icon-align-left
 icon-align-center	 icon-align-right	 icon-align-justify	 icon-list
 icon-indent-left	 icon-indent-right	 icon-facetime-video	 icon-picture
 icon-pencil	 icon-map-marker	 icon-adjust	 icon-tint
 icon-edit	 icon-share	 icon-check	 icon-move
 icon-step-backward	 icon-fast-backward	 icon-backward	 icon-play
 icon-pause	 icon-stop	 icon-forward	 icon-fast-forward
 icon-step-forward	 icon-eject	 icon-chevron-left	 icon-chevron-right
 icon-plus-sign	 icon-minus-sign	 icon-remove-sign	 icon-ok-sign
 icon-question-sign	 icon-info-sign	 icon-screenshot	 icon-remove-circle
 icon-ok-circle	 icon-ban-circle	 icon-arrow-left	 icon-arrow-right
 icon-arrow-up	 icon-arrow-down	 icon-share-alt	 icon-resize-full
 icon-resize-small	 icon-plus	 icon-minus	 icon-asterisk
 icon-exclamation-sign	 icon-gift	 icon-leaf	 icon-fire
 icon-eye-open	 icon-eye-close	 icon-warning-sign	 icon-plane
 icon-calendar	 icon-random	 icon-comment	 icon-magnet
 icon-chevron-up	 icon-chevron-down	 icon-retweet	 icon-shopping-cart
 icon-folder-close	 icon-folder-open	 icon-resize-vertical	 icon-resize-horizontal
 icon-hdd	 icon-bullhorn	 icon-bell	 icon-certificate
 icon-thumbs-up	 icon-thumbs-down	 icon-hand-right	 icon-hand-left
 icon-hand-up	 icon-hand-down	 icon-circle-arrow-right	 icon-circle-arrow-left
 icon-circle-arrow-up	 icon-circle-arrow-down	 icon-globe	 icon-wrench
 icon-tasks	 icon-filter	 icon-briefcase	 icon-fullscreen

## Glyphicon Attribution

Users of Bootstrap are fortunate to use the Glyphicons free of use on Bootstrap projects. The developers have asked that you use a link back to Glyphicons when practical.

Glyphicons Halflings are normally not available for free, but an arrangement between Bootstrap and the Glyphicons creators have made this possible at no cost to you as de-

## Usage

To use the icons, simply use an `<i>` tag with the namespaced `.icon-` class. For example, if you wanted to use the edit icon, you would simply add the `.icon-edit` class to the

`<i>` tag.

```
<i class="icon-edit"></i>
```

If you want to use the white icon, simply add the `.icon-white` class to the tag.

```
<i class="icon-edit icon-white"></i>
```

## Button Groups

Using button groups, combined with icons, you can create nice interface elements with minimal markup.

### Example:

```
<div class="btn-toolbar">
<div class="btn-group">
<a class="btn" href="#"><i class="icon-align-left"></i></a>
<a class="btn" href="#"><i class="icon-align-center"></i></a>
<a class="btn" href="#"><i class="icon-align-right"></i></a>
<a class="btn" href="#"><i class="icon-align-justify"></i></a>
</div>
</div>
```

### Output:

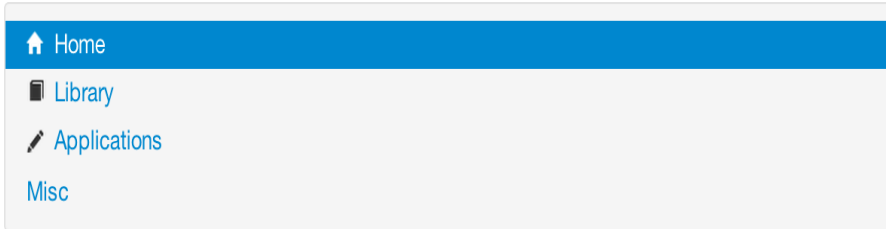


## Navigation

When you are using icons next to a string of text, make sure to add a space to provide the proper alignment of the image. More of navigation code will be covered in the next chapter.

### Example:

```
<ul class="nav nav-list">
<li class="active"><a href="#"><i class="icon-home icon-white"></i> Home</a></li>
<li><a href="#"><i class="icon-book"></i> Library</a></li>
<li><a href="#"><i class="icon-pencil"></i> Applications</a></li>
<li><a href="#"><i class="i"></i> Misc</a></li>
</ul>
```



**Output:**