1. **Presentation Logic:**
   html, js, angular js, jQuery, AJAX, etc
2. **Business Logic:**
   array functions, string functions, etc
3. **Database Logic:**
   sql queries etc.

M – Model = DB Logic

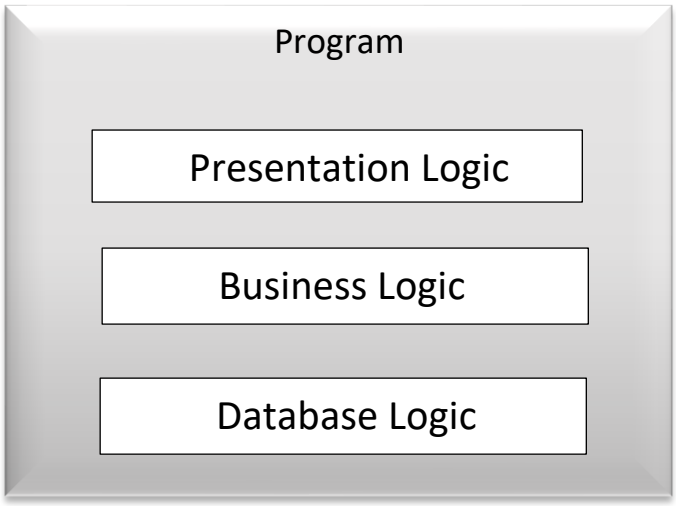
V – View = Presentation Logic.

C – Controller = Business Logic.

PROGRAM:

A program is collection of statements. In real time every program contains 3 types of logic.

1) Presentation Logic. 2) Business Logic 3) Database Logic



1. **Presentation Logic:**
   It is nothing but user interface (UI). This Layer contains pure Html, CSS, jQuery etc.
   In Real time web designers are responsible to develop presentation logic.
2. **Business Logic:**
   Business Logic is nothing but all mathematical operations, server-side validations etc, this logic will be taken care by server- side developers.

3. **Database Logic:**

   Database Logic is nothing but code belongs to purely to communicate database, like connecting to database, storing information in DB, reading etc.
   This Logic will be taken care by server-side developers.

MVC (MODEL VIEW CONTROLLER):

- MVC is a design pattern (designing the code) and it stands for model view controller.
- The objective of MVC is separating the above three logic into three different pages.
- Every framework by default follows MVC architecture.

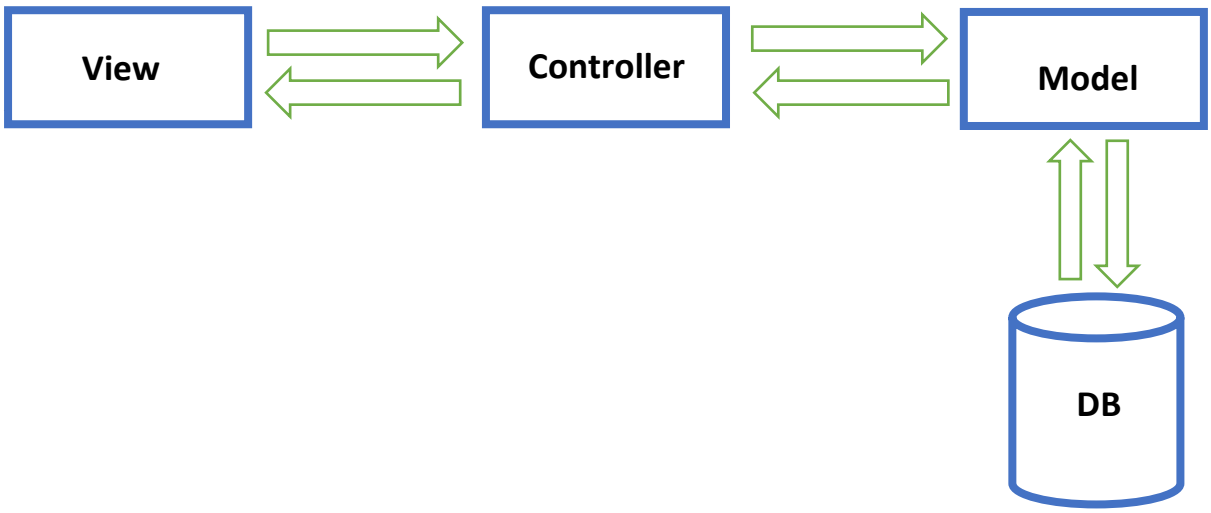
1. **MODEL:**

   Model is a server-side page which contains pure database logic. In case of php model file extension should be **.php**.
2. **VIEW:**

   View is a client-side page and it contains pure presentation logic. View file may be .html or server-side page.
3. **CONTROLLER:**

   Controller is a server-side page and contains pure business logic. Controller is a middle layer between view and model.

MVC Architecture:



Advantages:

1. We can avoid dependencies between designers and developers.
2. We can save time.
3. We can develop application rapidly.
4. We can reduce cost effective for company.
5. Better segregation / separation of three types of logic for better readability and accessibility.

What is a framework?

1. A framework is a semi-finished project and it is well organised (MVC).
2. A framework can be defined as a set of libraries that say, “Don’t Call Us, We’ll Call You”.
3. It will help us to accomplish job better and faster with industry standards.
4. Ex. CodeIgniter, Laravel, Zend, YII (Yes, it is),…etc.

Advantages of Framework:

1. The biggest advantage of the software framework is that it reduces the time and efforts in developing any software.
2. Framework separates business logic from user interface making the code cleaner and extensible.
3. Framework helps you to develop the project rapidly.

What framework provides?

1. URL Routing.
2. Database Manipulation.
3. Templating
4. Security
5. Rich set of libraries. (session, email, validations, ..etc)

Introduction to CodeIgniter (C.I)

CI is open source web application framework introduced by Mr. Ellis from Ellis Labs in 2006, but its not PHP substitute. It is an application development framework for people who build websites using PHP. Its goal is to enable you to develop project much faster than you could if you were codding from scratch, by providing a rich set of libraries for commonly needed tasks.

Features of CI

1. Model-View-Controller based system.
2. Extremely light weight
3. Fully featured database classes and support for several platforms.
4. Query builder database support.
5. Form and data validations.
6. Security and xss filtering.
7. Session management.
8. Image manipulation libraries.
9. Pagination.
10. Email sending features.

How to install CI.

The following steps we need to follow to install CI framework.

1. Download the CI from google.
2. Unzip the downloaded file and rename with your project name.
3. Upload this folder in server root directory. (htdocs)
4. localhost/project_name →to access

Important folders in CI

1. System – This folder contains readymade code (libraries) which are provided by CI framework. This folder is loaded inside root directory i.e. 'famouskart' or 'project_name'.
2. Application – As a developer we have to store all views, controllers, models etc inside this application folder. This folder is located inside root directory of our project.

Important folders in application folder

1. Controller – Inside this folder we have to store all controller files. This folder is located inside application folder.
   Ex – newfk/application/controller.
2. Models – All model files we have to store inside this folder. This folder is located inside application folder. Ex – newfk/applications/models
3. Views – All view files we have to store inside this folder. This folder is located inside application folder. Ex – newfk/applications/views
4. Config – This folder contains all configuration files related to CI framework. Like database configuration, constants configuration, url routings etc

Important files in config folder

1. **database.php** – Inside this file we have to configure all database related configurations like hostname, db-username etc. This file is located inside: newfk/application/config.
2. **constants.php** – All constants variables which we require inside the php project those we have to store inside this file.
3. **routes.php** – Using this file we can develop url routing i.e. assigning alias names for physical files.
4. **config.php** – using this file we can configure base url of project i.e. http://localhost/newfk
   Ex: $config[‘base_url’] = ‘http://localhost/newfk’

How to create view file

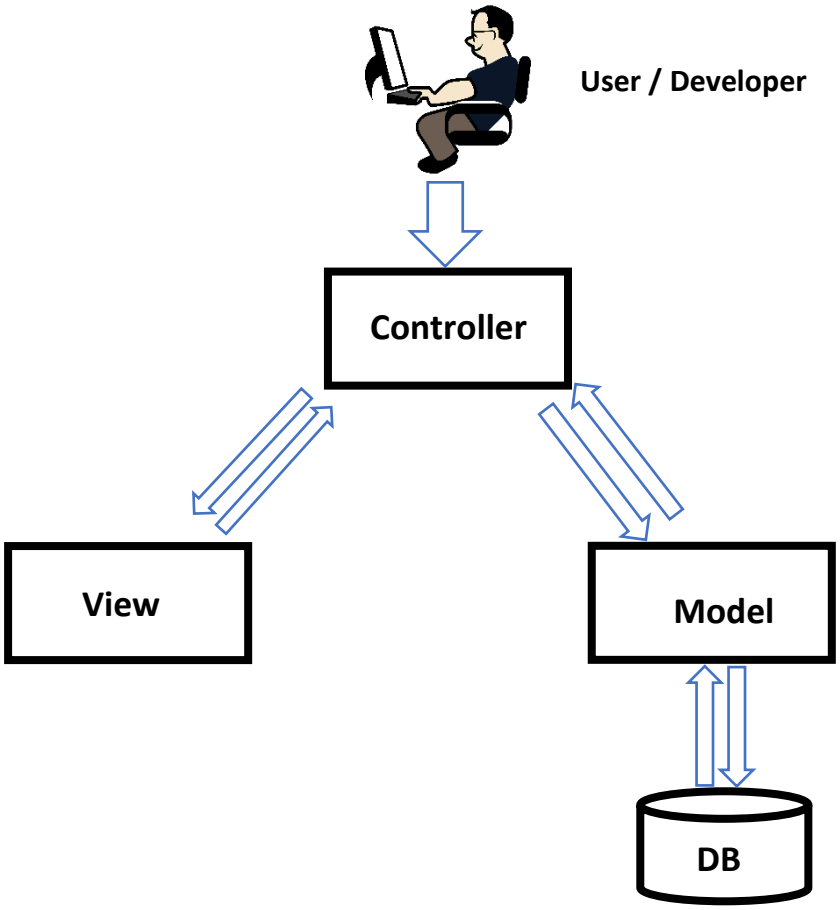
The following steps we need to follow to create view files.

1. Go to views folder in application folder.
2. Create a new file with either .html or .php extension.
3. Write your code inside this file related to presentation logic.

Execution flow of CI

In case of CI framework to run either view or a model for sure we need to depend on controller.

As a user/developer first we need to request controller then controller will execute respective views or models.



How to create Controller?

Following steps, we have to follow to create controller.

1. Go to controller folder.
2. Create .php file with starting character in uppercase.
3. Inside this file we have to write php code in object-oriented approach.
4. Inside a class by extending CI_controller.
5. Inside the class we have to write atleast function.

My_first_controller.php

```
<?php
    class My_first_controller extends CI_controller {
        public function f1( ) {
            $this →load→view('my_first_view');
        }
        public function f2( ) {
            $this→load→view('my_second_view');
        }
    }
?>
```

How to request controller method

To run controller method, we have to type below url.

Syntax: http://localhost/project_name/index.php/controller_name/method_name

Ex: http://localhost/newfk/index.php/my_first_controller/f1

Loader Class

1. Loader is a built-in class inside CI framework with a reference name called load.
2. Loader is used to load elements. These elements can be libraries (classes), view files, helpers, models or your own files.

Note: This class is instantiated automatically by the system there is no need to create object manually.

Ex: $this→load→view ( ); $this→load→model ( );

Functions of Loader Class

1) $this→load→view ( )

Using this function, we can load a view file into controller or a view file into another view. For this function we have to pass minimum one argument i.e. view file name.

Syntax: $this→load→view('view_file_name');

Example: $this→load→view('register_view');

2) $this→load→model ( )

Using this function, we can load a model into controller or a model into another model. For this function we have to pass minimum one argument i.e. model file name.

3) $this→load→helper ( )

Using this function, we can load a helper into required controller or a view. For this function we have to pass minimum one argument i.e. helper name.

4) $this→load→library ( )

Using this function, we can load libraries into required controller/model. For this function we have to pass minimum one argument i.e. library name.

What is index method?

If we create a method as index inside controller that method will be executed automatically when we will type url up to controller name.

If it is other than index those methods we have to type in browser url address after controller name.

```
<?php
    public function index ( ) {
        $this→load→view('index_view');
    }
?>
```

Default controller:

When we set up CI framework by default one controller will be provided as default controller i.e. Welcome.

If we make any controller as default controller it will execute automatically when we will type in browser url up to project name.

To change the default controller, we have to follow below steps:

1. Go to routes.php
2. For $route['default_controller'] we have to assign our controller name instead of Welcome.
3. Example: $route['default_controller'] = 'home_controller';

Note:

Default controller will execute automatically when we will type url up to project name.

Example: http://localhost/project_name

404_override:

In CI whenever will type invalid url, CI framework will display error code as 404.

If we want to override the design with other output we can.

Following steps, we have to use override 404 page.

1. Create a new file under 404_view.php and write code what we need to display.
2. Inside any controller create a method and load this view i.e. 404_view page

Example:

```
public function page_not_found ( ) {
        $this→load→view('404_view');
}
```

3. Go to routes.php
4. Inside routes.php we have to assign value for $route['404_override'] that value is controller name & method name

Example:

```
$route['404_override'] = 'home_controller/page_not_found';
```

base_url ( ):

1. Using this function, we can get base url name which we configured inside config.php
2. This function return type is string
3. To access base url we have to load url helper then only we can access the function otherwise it will give fatal error.

   Example:

```
public function test ( ) {
$this→load→helper('url')
echo base_url ( );
    }
```

Autoloading:

Whenever we are loading either helper or libraries into controllers or views and models there are two types of loading available in CI

1. Manual loading
2. Auto loading

1) Manual loading:

It is a process of loading libraries or helpers whenever we required. If we load with manual loading it will applicable to that area when we are doing manual loading.

2) Auto loading:

If we load any library or helper with auto loading we can access its features throughout the project and it will load one time.

Note:

When we have a requirement to use same code again and again better to go with autoloading.

How to auto load helpers or libraries:

In CI framework to develop auto loading mechanism CI provided a separate page with the name called autoload.php which is available inside config folders.

Steps to auto load for helpers:

1. Go to config folder

2. Open autoload.php
3. Look for the variable i.e. $autoload[‘helper’]
4. For this variable inside array we have to specify list of helpers with comma (,) separator

Example:

$autoload[‘helper’] = array(‘url’); //$autoload[‘helper’] is a configuration variable

How to auto load libraries:

Inside autoload.php there is a configuration with the name called $autoload[‘libraries’]. For this variable we have to specify list of library name with (,) separated.

Constants:

Constant is nothing but a fixed value and we cannot override constant variable values. Constants are similar to variable but variable allows overriding the value but variable allows overriding the value but constant doesn’t allow.

To declare constants variable, we have to use a function called define ( ).

In CI framework all constant variables we have to declare or define in constants.php

Example:

define(‘CSS_PATH’,’assets/css/’);

How to create a model:

Model file creation is similar to controller creation only. When extending we have to use CI_model

Steps to execute model method:

1. Load a model into required controller by using
   $this→load→model ( ) //assignment file name
2. Once model is loaded we can start model model methods by using following syntax i.e.
   $this→model_class_name→method_name ( );

Example:

$this→test_method→f1( );

Test_model.php

```
<?php
        Class Test_model extends CI_model {
        Public function f1( ) {
        echo  "Welcome to model";
                }
        }
?>
```

Test_controller.php

```
<?php
        Class Test_controller extends CI_controller {
        Public function test ( ) {
                $this->load->model('test_model');
                $this->test_model->f1( );
                }
        }
?>
```

Requesting url: http://localhost/newfk/index.php/test_controller/test

Note:

1.If we want to send data from controller to model we have to use arguments & parameters

2. if we want to send data from model data to controller we have to use return keyword.

Database Class

1. Automatic Connection

Automatic connections can be done by using the file application/config/autoload.php. Automatic connection will load the database for each and every page. We just need to add the database library as shown below.

```
$autoload['libraries'] = array('database');
```

2. **Manual Connection**

   If you want database connectivity for only some of the pages, then we can go for manual connecting. We can connect database manually by adding the following line in the class.

   $this→load→database ( );

Functions of database class

1. **$this→db→insert ( )**

   Using this function, we can store data into database table. For this function we have to pass two arguments,

   1) Table Name
   2) Associative Array (Keys as table columns)

   Return type of this function is Boolean.

2. **$this→db→get( )**

   Using this function, we can get data from database table. For this function we need to pass table name as a argument, it will generate the following query ie

   select * from table_name;

   This function return type is result_set.

   Ex: $res = $this→db→get(‘user_tbl’);

   Output Query : select * from user_tbl

3. **$this→db→select( )**

   Using this function we can customize select query columns section, i.e. instead of all columns if we want to fetch specific column we require this function.

   Ex: $this→db→select(‘user_id’, ‘email’);
   $this→db→get(‘user_tbl’);
   Output Query: select user_id,email from user_tbl

4. **$this→db→where( )**

This function will add where condition for the sql query. For this function we can pass two argument as column name and value or one argument as associative array.

Ex:

$this→db→where('email', 'seshu@gmail.com');

Or

$condition = array('email' => 'seshu@gmail.com');

$this→db→where($condition);

5. **$this→db→or_where ( )**

Using this function we can generate or conditions for sql queries. For this function we need to pass 1 argument as associative array or 2 argument as column name and value.

Ex.

```
$this→db→select('email');
$this→db→from('user_tbl');
$this→db→where('user_id', 1);
$this→db→or_where('user_id', 5);
$this→db→get( );
```

Output query:

Select email from user_tbl where user_id = 1 or user_id = 5;

6. **$this→db→from( )**

Using this function, we can add from section to the select query. When you don't want to pass table name for get we can use this function.

Ex:

```
$this→db→select('email', 'user_id');
$this→db→from('user_tbl');
$this→db→where('email', 'seshu@gmail.com');
$this→db→get( );
```

Output Query:

Select email, user_id from user_tbl where email = 'seshu@gmail.com'

7. **$this→db→update( )**

Using this function, we can records/record in the specific table. For this function we have to pass 2 arguments.

- Table Name
- Associative array which contains columns name as keys to update and its values.

Ex. For updating all records.

$update_data = array(‘password’, ‘123123’);

$this→db→update(‘users_tbl’, ‘$update_data’);

Ex. For updating specific record.

$update_data = array(‘password’, ‘123123’);

$this→db→where(‘user_id’, 2);

$this→db→update(‘users_tbl’, ‘$update_data’);

8. $this→db→delete( )

Using this function, we can delete specific records from database table. Using this function we cannot delete all records. For this function we have to pass one argument i.e. table name

Ex. $this→db→where(‘user_id’, 2);
$this→db→delete(‘users_tbl’);

9. $this→db→empty_table( )

Using this function, we can delete all records from given table name. For this function we have to pass one argument i.e. table name.

Ex. $this→db→empty_table (‘users_tbl’);

10. $this→db→order_by()

Using this function we can fetch db records either in ascending or descending order. For this function we have to pass 2 argument

- Column name
- Order type (asc or desc)

Example: $this→db→order_by(‘user_id’, ‘desc’);

11. $this→db→count_all( )

Using this function we can find number of record count inside db table. For this function we have to pass one argument i.e. table name.

Example: $count = $this→db→count_all(‘users_table’);

12.$this→db→like( )

Using this function we can search for db records based on string. For this function we have to pas 3 arguments

- Column name
- Value to be search
- Search type (after /before/ both)
  In realtime this function can be used for search functionality development.
  Example: $this→db→like('user_name', 'p', 'after');

Joins in Code Igniter

$this→db→join( )

Using this function we can implement join query for select queries. As we know there are 3 types of joins.

1. Inner join
2. Left join
3. Right join

For this function we need to pass 3 arguments

1. Table name to join
2. Condition
3. Join type (inner/left/right)

Ex. *public function get_users(){*

$this→db→select('u.name, u.email, u.user_id, c.country_name');

$this→db→from('users_tbl u');

$this→db→join('country_tbl c', 'u.country_id = c.country_id', 'inner');

$result = $this→db→get();

return $result;

}

Finding rows count in result set

result_set→num_rows( )

Using this function we can find number of records inside result set.

Ex: $res→num_rows( );

How to send data from controller to view

To send some information from controller to view we need to store the data in form of associative array and this associative array we need to pass as a second argument for $this→load→view ( ).

$data = array ( );

$data[‘msg’] = “Registration is successful”;

$this→load→view(‘register_view’,$data);

How to read data inside view which is coming from controller

Once we will send data to view from controller. Inside view we need to call associative array keys as variables.

Ex: <?php echo $msg; ?>

form_validation class

This class can be used to develop validation at controller level. To work with form validation, we have to load form_validation library with either autoloading or manual loading.

Suggested loading is manual.

Ex: $this→load→library(‘form_validation’);

Functions of form_validation class

1) $this→form_validation→set_rules ( )

Using this function, we can specify list of rules for specify form fields. For this function we have to pass three arguments.

1) Form fields name 2) Label 3) Rules

$this→form_validation→set_rules (‘name’, ‘name’, ‘required’);

Once we configured all rules using set_rules function finally we need to call this function. This function will return true if all validation rules are satisfied for all fields which we configured otherwise it will return false.

```
<?php
public function register_now ( ) {
    $this→load→library('form_validation');
    $this→form_validation→set_rules ('name', 'name', 'required');
    $this→ form_validation→set_rules ('email', 'email', 'required');
    if ($this→form_validation→run ( ))
        echo "Success";
    else
        echo "Failed";
}
?>
```

Form Helpers

Using this helper file, we can get or read error messages which are returned by form validation class. To work with this helper, we need to autoload or manual load. Suggested loading in autoload.

Ex: $autoload['helper'] = array ('url', 'form');

Functions of form helper

1. **validation_errors ( )** – Using this function we can get all errors at a time which are generated by form validation class. Return type is string.
   Ex: <?php echo validation_errors ( ); ?>

2. **form_error ( )** – Using this function we can get error message of specific form field. For this function we have to pass one argument i.e. form field name. Return type is string.
   Ex: <?php echo form_error ( ); ?>

3. **set_value ( )** – Using this function we can get user submitted values into form when validation becomes failed. For this function we have to pass one argument i.e. form field name. Return type is string.
   Ex: value = "<?php echo set_value('name'); ?>"

Can we customize error message of form validation?

Yes, we can customize error messages of form validation class for this we have to pass fourth argument as associative array for set_rules function which contains rules as keys and values as custom message.

Ex:

$this→form_validation→set_rules ('name', 'name', 'required', 'required'=>'name is required');

2) $this→form_validation→set_error_delimiters ( )

Using this function, we can override the error message tag from paragraph <p> to any other tag. For this function we have to pass two argument

1) Opening Tag
2) Closing tag

Ex: $this→form_validaion→set_error_delimiters('', '');

Rules of form validation

Sr. No	*Rules*	*Description*
1	required	To check empty
2	alpha	It allows only alphabets
3	alpha_numeric	It allows only alpha numeric
4	numeric	It allows only numbers or numeric values
5	alpha_numeric_spaces	It allows only alphabets, numbers and spaces
6	min_length	We can specify minimum length $this→form_validation→set_values ('name', ' ', 'min_length [2]');
7	max_length	We can specify maximum length
8	exact_length	To specify exact length ex. exact_length [6];
9	valid_email	Using this rule, we can check email is valid or not
10	valid_emails	We can check multiple emails separated with commas.
11	greater_than	greater_than [5];
12	less_than	less_than [5];
13	greater_than_equal_to	
14	less_than_equal_to	
15	trim	It will remove white spaces from both sides.
16	valid_ip	It will check given ip address is valid or not
17	valid_url	It will check valid url.

Difference between helpers and libraries:

#	*Helpers*	*Libraries*
1	Helpers especially developed for both views and controllers.	Libraries especially developed for both models and controllers.
2	Helpers are developed by using POP programming approach.	Libraries are developed by using OPPs Programming approach.
3	To load helper we have to use $this→load→helper( ) function.	To load library we have to ise $this→load→library( ) function.
4	All helpers provided by CI kept in helpers folder inside system folder. system/helpers/	All libraries provided by CI are kept in libraries folder inside system folder. system/libraries/
5	Ex: url, form	Ex: db, form_validation

Can we create own helpers and libraries?

Yes, we can create our own helpers and libraries. We have to keep these user-defined helpers or libraries inside application folder in separate folders helpers or libraries.

How to create user-defined helpers

1) Create .php file which contains file name as lower case letter and suffix should be _helper.
   Ex. test_helper.php
   Note: This file we have to store inside application folder of helper.
2) Start writing code POP approach.
   Ex. test_helper.php

```
<?php
        Function test_fun( ){
                echo “welcome to ”;
        }
?>
```

3) Now if we require above code we need to load helper.
   Ex. $this→load→helper(‘test’);

test_helper.php

```
<?php
function reverse_words($pstr){
        $arr = explode(" ",$pstr);
        $count = count($arr);
        $rstr = "";
        for ($i=$count-1;$i>=0; $i--){
                $rstr = $rstr.$arr[$i]." ";
        }
        return $rstr;
}?>
```

controller.php

```
<?php
        public function f1( ){
                $this->load->helper('test');
                $str = "Welcome to PHP";
                echo reverse_words($str);
        }
?>
```

How to create user-defined library

1. Create .php file which contains file whose name starts with upper case.
   Ex. Test_lib.php
   Note: This file we have to store inside application folder of library.
2. Start writing code in OPP's approach.
   Ex:

```
<?php
class Test_lib{
        public function reverse_words($pstr){
                $arr = explode(" ",$pstr);
                $count = count($arr);
                $rstr = "";
                for ($i = $count-1;$i>=0; $i--){
                        $rstr = $rstr.$arr[$i]." ";
                }
                return $rstr;
        }
}
?>
```

3. Now if we require above code we need to load library.
   Ex: $this→load→library('test_lib');

How to call library method:

To call library method we need to use below syntax.

Syntax: $this→library_class_name→method_name( );

Ex: $this→test_lib→reverse_words( );

What is CI_controller

CI_controller is a parent controller for controllers which we are creating. The necessary libraries are loaded by CI_controller. If you want CI_controller features in to child class then we should extend CI_controller.

Constructor in Code Igniter

Constructor is a special method inside the class and it can be used to reuse the code automatically throughout other methods of same class.

When you are loading libraries, helpers and models better to load inside constructor.

Ex:

```
<?php
class C1 extends CI_controller {
    public function __construct ( ) {
        parent :: __construct ( );
        $this→load→library('test_lib');
    }
    public function f1( ) {
    …….;
    }
    public function f2( ) {
    …….;
    }
}
```

?>

DB Queries type

In CI framework DB Queries we can write in two ways.

1) Standard Queries
2) Query Builder

Standard Queries

Writing queries manually by the developer and executing it is nothing but standard queries.

To execute standard queries in CI we have a function called $this→db→query( ), this function return type is boolean and result.

Note: This function is similar to mysql_query( ) in core php.

Ex:

```
public function register ( ) {
$sql_query = "insert into users_tbl (name,mobile) values ('xyz', '1231231234')";
$resp = $this→db→query($sql_query);
if($resp)
    echo "inserted";
else
    echo "Not inserted";
}
```

Query Builder

Query Builder is a db concept in CI which can generate and execute the SQL queries automatically based on developer inputs.

Database class contains query builder functions.

Ex: $this→db→insert( );

$this→db→select( ); ….etc

Query Helper Functions

1. **$this→db→insert_id ( )**
   Using this function, we can get last inserted auto incremented record id.
   Note: We have to write this function after insert query.
2. **$this→db→affected_rows( )**
   Using this function, we can get no. of records count how many modified inside db table in case of insert, update and delete queries.
3. **$this→db→last_query( )**
   Using this function, we can print last executed query. This function return type is string.

How to read records from result set in Code Igniter

In CI to read records from result set there are four functions available.

1) result_set→row( )
2) result_set→row_array( )
3) result_set→result( )
4) result_set→result_array( )

result_set→row( ) – Using this function we can read single record from result set in object format.

Ex: $row→name;

$row→email;

result_set→row_array( ) – Using this function we can read single record from result set in associative array format.

Ex: $row['name'];

$row['email'];

result_set→result( ) – Using this function we can read multiple records from result set in object format.

Ex: $row→name;

Note: To read multiple records here we will use foreach( ) iteration statement.

result_set→result_array( ) – Using this function we can read multiple records from result set in associative array format.

Ex: $row['name'];

user_model.php

```
<?php
    public function get_users( ){
    $this→db→select('name', 'email');
    $this→db→from('users_tbl');
    $this→db→where('user_id', 1);
    $result_set = $this→db→get( );
    return $result_set;
    }
?>
```

User_controller.php

```
<?php
    public function read_user( ){
    $data = array( );
    $res = $this→user_model→get_user( );
    $data['resultset'] = $res;
    $this→load→view('display_view', $data);
    }
?>
```

display_view.php

```
<?php
    $count = $resultset→num_rows( );
    if($count>0) {
        $row = $resultset→row( );
        echo $row→name; }
    else{
```

echo "Data not found"; } ?>

Segments in CodeIgniter

- Segments are similar to query string.
- Segment can be used to transfer some information from one page to another page via browser URL address.
- To pass some segment values after method name have to specify segment values with separator called (forward slash) ' / '.

How to read segment values

- To read segment values in CI there is a function with the name called – $this →uri→segment( )
- For this function we have to pass one argument.
- **Note:** Segment position starts from controller and controller position is 1 method position is 2 etc.

Ex: http://localhost/newfk/index.php/controller_name/method_name/20
$this→uri→segment(3);

redirect( )

It is a function using this function we can redirect from one location to another location which contain controller_name and also method_name.

Ex: redirect('user_controller/read_user');

Note: If we are not specifying method name it will redirect to index method. And it is a function of **url helper.**

Updating Records

For this we have to follow bellow stapes

1. Get existing information from the db table by passing user_id

   Note: Here we required one model method to get existing information we required controller method to call model method and are required one view file to display existing information inside the form.
2. Create a form and display existing information
3. When user modifies information click on update then we have to perform update query.

Pagination

$this→db→limit

This function can be used to get limited records from database table. For this function we can pass one argument as no of records count.

Example: $this→db→limit(2, i) //2 is no of record & (i) is starting index
We can also pass 2 arguments i.e. no of records
And starting index.
Example:

Pagination Class

Pagination is nothing but loading limited records per page when we have many records at a time is not correct that's why we need to implement pagination.

To work with pagination in CI we have a library with the name called pagination. To start working with this library we have to load library first.
Note: Suggested loading is manual loading
Example: $this→load→library('pagination');
Under pagination class we have few configurations and also functions available.

Configuration of Pagination Class

1. **$config['total_rows']**
   For this configuration we have to specify the list of numbers of records count.
   Example: $config['total_rows'] = 100;
2. **$config['base_url']**
   Ex.
   $config['base_url'] =
   base_url()."index.php/pagination_controller/pagging";
3. **$config['per_page']**
   For this configuration we have to specify per page how many we wanted to load.
   Example: $config['per_page'] = 10;
4. **$config['num_links']**
   Using configuration, we can increase or decrease number of pages links after current page.
   Note: By default, 2 is page limit.
   Example: $config['num_links'] = 10;

5. **$config['attribute']**

Using this configuration, we can apply attributes for the pagination links which are generated we have assign associative array which contains key as attribute names and values as attribute value.

Example: $config[‘attributes’] = array (‘class’ => ‘pdesign’);

6. **$config[‘use_page_numbers’]**

Using this configuration, we can pass numbers as segment instead of starting index. For this configuration just we have to assign Boolean value true.

$config[‘use_page_numbers’] = true;

Functions of Pagination Class

1. **$this→pagination→create_links( )**

This function will return no of pages links which we require for pagination.

This function we need to call after above configuration.

2. **$this→pagination→initialize( )**

Using this function, we can initialize required configuration for pagination class for this function we have to pass one argument i.e. associative array.

Email Class (Library)

To send emails to users using CI framework, we have one library class with the name called “email”. To work with this library, we have to load this library with either auto loading or manual loading.
Note: Suggested loading is - manual

Functions of Email Class

1. **$this→email→to( )**
   For this function we specify list of emails with comma separated to whom we need to send email.
   Example: $this→email→to (‘m1$gmail.com, m2@gmail.com’);
2. **$this→email→from(info@richlabz.in)**
   Using this function, we can specify from email address.
   Note: From address mandatory otherwise mail will go to spam folder
3. **$this→email→subject( )**
4. **$this→email→message( )**
5. **$this→email→attach( )**
6. **$this→email→cc( )**
7. **$this→email→bcc( )**
8. **$this→email→send( )**

Upload Class (Library)

To work with file uploading concept in CI framework we have a built-in library class with the name called “*upload*”, We have to load this class either with auto loading or manual loading.

Note: Suggested loading is manual
Example: $this→load→library(‘upload’);
To work with upload class, we have few configuration and functions are available.

Configurations of Upload Class

1. **$config[‘allowed_type’]**
   Using this config we can specify which we would like to use each extention seperated with pipe (|) symbol.
   Example: $config[‘allowed_type’] = “jpg|png|gif”;
2. **$config[‘upload_path’]**
   For this configuration we have to specify the path of where we need to upload files.
   Example: $config[‘upload_path’] = “uploads/”
3. **$config[‘max_size’]**

For this configuration we have to specify the maximum size of a file to upload by the user size should specify in MB.
Example: $config[‘max_size’] = 100;

4. **$config[‘max_width’]**
   For this configuration we have to specify maximum size of a file value should be in pixel. This configuration we can use for only image files
5. **$config[‘max_height’]**
   For this configuration we can specify max height of a file.
6. **$config[‘file_name’]**
   Using this configuration, we can define the file name with which name to upload into target folder.

Functions of Upload Class

1. **$this→upload→initialize( )**
2. **$this→upload→do_upload( )**
   Using this function, we can start uploading return type is Boolean. For this function we have to pass one argument i.e. form field name
   <input type = “file” name = “image”/>
   Example: $this→upload→do_upload(‘image’);
3. **$this→upload→data(‘file_name’)**
   Using this function, we can get file name which is uploaded into *uploads* folder
4. **$this→upload→display_error( )**
   Using this function, we can get errors in case file uploading. This function return type is string.

Session Class (Library)

Using this library class, we can work with session concept before working with this library we have to load this library with auto loading or manual loading.
Suggested loading: it depends
Example: Gmail – autoloading
Amazon – manual loading

1. **$this→session→set_userdata()**
   Using this function, we can store data into session. For this function we can pass two types of arguments
   - Two arguments which contain session name and value
     Example: $this→session→set_userdata('name', 'xyx');
   - One argument as associative array
   - $sess_data = array (
     'name' => 'xyz',
     'user_id' =>20
     );

   Example: $this→session→set_userdata($sess_data);

How to read Session Data

To read session data we have a function called
$this→session→userdata( )
For this we have to one argument as session name
Example: $this→session→userdata('name');
$this→session→userdata('user_id');

How to delete Session Data

$this→session→unset_userdata( )

Using this function, we delete session variable values by passing 1 argument as session variable name.

Example: $this→session→unset_userdata('name');

site_url ( )

Using this function, we can get url upto index.php. This function is available inside a helper called url. Return type is string.
Ex: echo site_url( )
Output: http://localhost/newfk/index.php

current_url ( )

Using this function, we can get the current URL what we have in browser url address. This function is also kept in URL helper and return type is string.
Ex: echo current_url ( )

uri_string ( )

Using this function we can get all the segment values which we have inside a url. This function return type is string.
Ex. Requested url:
http://localhost/newfk/index.php/user_controller/register/3
echo uri_string ( );
output: user_controller/register/3

String Helpers

Apart from core string functions code igniter provided few new functions under this helper. To access those functions first we have to load this helper with autoload or manual load. Suggested loading is manual.
Ex: $this→load→helper('string');

1. **random_string ( )**

   Using this function, we can generate random string which includes either alphabets or numbers etc.
   There are four types of random strings we can generate,
   a) alpha
   b) alnum
   c) numeric
   d) nozero
   For this function we have to pass 2 arguments.
   1. Random string type 2. Length
   Return Type is string.

Ex: *public function test( ){*
$this→load→helper('string');
echo random_string ('alpha', 10);
}

2. repeater ( )

Using this function, we can print a string repeatedly a number of times. For this function we have to pass 2 arguments
1) string to repeat 2) number of times to repeat

Ex. *public function test ( ){*
$this→load→helper('string');
$str = 'CI';
echo repeater($str,3);
}

3. reduce_double_slashes ( )

Using this function, we can remove double slaches which are invalid inside the url. For this function we have to pass one argument, i.e. URL. Return type is string.

Ex. *public function test(){*
$this→load→helper('string');
$url = http://localhost/newfk//index.php//c1/test ;
echo reduce_double_slashes($url);
}

4. strip_slashes( )

Using this function, we can remove slashes from a give string or array of string. For this function we can pass a string variable or array variable also.

```
$str = array(
        'question'=> 'Is your name O\' reilly?',
        'answer' => 'No, my name is O\' connor.'
        );
$str = strip_slashes($str);
```

Form helper othe functions

1) form_error ( )
2) validation_errors( )
3) set_value( )

4) form_open ( )

Using this function we can generate form open tag. For this function we can pass different types of arguments. If you are not passing any arguments by default it will take action as current url method type as post.

Return type is string.

Ex. <?php echo form_open ( ); ?>

How to customize action:

To customize action attribute we can pass first argument as controller name and method name.

Ex. echo form_open('user_controller/register');

How to customize additional attributes

Ex. *$attr = array(*
'method' => 'get',
'onsubmit' => 'return validate()
);
echo form_open('C1/register', $attr);

5) form_close( )

Using this function, we can close a form tag. Return type is sting.

6) form_open_multipart()

This function is similar to form_open only, but this function will add one additional attribute i.e. enctype. Return type is string.

Ex. echo form_open_multipart();

7) form_input( )

Using this function we can generate input tags like textbox, password field, checkbox etc. For this function we can pass different types of arguments.

Ex. echo form_input();

Output: <input type= "text" value= " "/>

Ex. echo form_input('name', 'xyz');

Output: <input type= "text" name = "name" value= "xyz"/>

Passing associative array:

```
<?php
        echo form_open();
        $input_attr = array(
                        'type' => 'password',
                        'name' => 'password',
                        );
        echo form_input($input_attr);
        echo form_close();
?>
```

What is index.php in CodeIgniter Framework?

index.php in CI framework we will call it as front controller. index.php responsibility are as follows.

- It will load necessary libraries required by ci framework.
- It will create required objects internally.
- It will handle error hiding and displaying.

How to hide error in CI framework?

Inside index.php change ENVIRONMENT constant variable value from development to production, so that error will be hidden throughout framework.
Note: index.php we can finf inside root directory of your CI project folder.

How to hide index.php from URL?

In real-time in CI better to hide the index.php to keep url user friendly and also to stop knowing other people in which technology we built this application. To hide index.php we have to follow the below steps.

1) Create .htaccess file inside the root directory of your project.(ex.newfk)
2) Copy the below code into .htaccess file and save.

```
DirectoryIndex index.php
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ ./index.php?$1 [L,QSA]
```

What is .htaccess file?

It is a configuration file for apache server, if you want to override few configuration of apache and also php.ini we can do with this file.
Note: This changes configurations for temporary.

URL Routing

- URL routing is a process of assigning reference / alias URL for physical URL.
- Using URL routing we can develop SEO friendly URLs.

Steps to develop URL Routing.

1) Goto routes.php from config folder.
2) Decide for which url we need to need to take alias name.
3) We have to use below syntax to configure url .
   $route['ref_name'] = 'physical_url';

Ex. $route['user/register'] = 'user_controller/register_view';

To change only method name:

Ex. $route['user_controller/register'] = 'user_controller/register_view';

$config['permitted_uri_chars']

In case of url CI framework doesn't allow all characters in the url, it allows only few characters, when you are passing chars which are not allowed CI will display error message as follows.

"The URI you submitted has disallowed characters"

To solve the above errors we have to specify other characters which are not configured or make the config value as empty.

Ex. $config['permitted_uri_chars'] = " " ; //now it will allow all characters

Note: This configuration available in config.php.

Calendar Class

To work with calendar CI has one library class with the name 'calendar'. To use this calendar library, we have to load calendar library with either autoload or manual load, suggested loading is manual.

Ex. $this→load→library('calendar');

Calendar class has few functions and configurations.

$this→calendar→generate()

Using this function, we can get calendar and this function return present month and year calendar. Return type is string.

Ex.
```
public function test( ){
        $this→load→library('calendar');
        echo $this→calendar→generate( );
}
```

By passing two argument (year, month) we get specific year and month calendar.

Ex. echo $this→calendar→generate(2016,6);

Configurations of calendar class

1) **$config[‘show_next_prev’]**
   Using this configuration we can enable next and previous links. For this configuration we have to assign boolean value true.
2) **$config[‘next_prev_url’]**
   Using this configuration, we can specify url name where it has to redirect when we click on next or prev link.
   Note: Better to request same method link
   Public function test(){
   $config[‘show_next_prev’] =true;
   $config[‘next_prev_url’] = base_url().“C1/test”;
   }
3) **$config[‘month_type’]**
   Using this configuration we can display month name with full name or we can display starting three charecters also.
   There are 2 values for this configuration.
   1. long 2. short
   Ex: $config[‘month_type’] = ‘long’;
   Ex: $config[‘month_type’] = ‘short’;
4) **$config[‘day_type’] = ‘long’/’short’;**
5) **$config[‘start_day’]**
   Using this configuration we start a calendar day with any other day instead of Sunday. We have to assign full day name for this configuration.
   Ex. $config[‘start_day’] = ‘saturday’;

Can we assign hyperlink to specific dates

Yes, we can assign some hyperlink for specific dates for this we have to prepare an array which contains day number as keys and values as its url.

This array we have to supply as a third argument for generate function.

Ex. *$data = array(5 => base_url().‘C1/get_users/2018-09-05’);*
echo $this→calendar→generate($year, $month, $data);

Hooks

Hooks are similar to events, which can be used when we require common piece of code in all controllers and in all methods the we will go for hooks concept.

Without disturbing code CI functionality we can apply hooks code to controllers. Hooks can be enabled and disabled whenever we require.

To work with hooks concept we need to focus in 3 areas.

1) Hooks folder (application/hooks)
2) hooks.php (config folder)
3) Enable hooks (from config.php file)

Hook events

There are 4 types of events available for hooks and these can be used when we have to run hook code.

- pre_controller
- post_controller
- pre_constructor
- post _constructor

Steps to develop hooks concept

1) Enable hooks from config.php file
   Ex. $config['enable_hooks'] = TRUE;
2) Create a hook file with .php extension and write code in opps or pop style.
   Ex: Testhook.php

```
<?php
    class Testhook{
        public function msg(){
            echo "Welcome to ";
        }
    }
?>
```

3) Once a hook is developed now we have to specify to CI framework when a hook code should execute. These configurations we have to specify in hooks.php file from config folder.

Ex. <?php

```
$hook['pre_controller'] = array(
                        'class' => 'Test_controller',
                        'function' => 'msg',
                        'filename' => 'Testhook.php',
                        'filepath' => 'hooks'
                        );
?>
```