



PHP- Handbook with CI

Version, Oct-2020

Contents

Module – 01[Fundamental].....	12
Object Oriented Programming.....	12
Software Engineering	31
Definitions	32
Software Evolution	32
Software Evolution Laws.....	33
Software Paradigms.....	34
Software Development Paradigm	34
Software Design Paradigm	34
Programming Paradigm.....	34
Need of Software Engineering	34
Characteristics of good software.....	35
Operational	35
Transitional	35
Maintenance	36
Various Syntax in SQL	36
SQL SELECT Statement	36
SQL DISTINCT Clause.....	36
SQL WHERE Clause	37
SQL AND/OR Clause	37
SQL IN Clause.....	37
SQL BETWEEN Clause.....	37
SQL LIKE Clause.....	37
SQL ORDER BY Clause.....	37
SQL GROUP BY Clause	37
SQL COUNT Clause.....	38
SQL HAVING Clause.....	38
SQL CREATE TABLE Statement	38
SQL DROP TABLE Statement.....	38
SQL CREATE INDEX Statement.....	39
SQL DROP INDEX Statement.....	39
SQL DESC Statement	39
SQL TRUNCATE TABLE Statement	39
SQL ALTER TABLE Statement	39
SQL ALTER TABLE Statement (Rename)	39
SQL INSERT INTO Statement.....	39
SQL UPDATE Statement.....	39
SQL DELETE Statement	40
SQL CREATE DATABASE Statement.....	40
SQL DROP DATABASE Statement.....	40
SQL USE Statement.....	40
SQL COMMIT Statement.....	40
SQL ROLLBACK Statement.....	40
Characteristics	41

TOPS Technologies

Users	42
Types of Normal Forms	43
First Normal Form (1NF)	44
Second Normal Form (2NF)	45
Third Normal Form (3NF)	46
Module - 2 [HTML & HTML5]	31
What You Can Do with HTML	50
Step 1: Creating the HTML file	51
Step 2: Type some HTML code	51
Step 3: Saving the file	51
HTML Tags and Elements	52
HTML Elements	52
HTML Element Syntax	53
HTML Tags Vs Elements	53
Case Insensitivity in HTML Tags and Attributes	53
Empty HTML Elements	53
Nesting HTML Elements	54
Writing Comments in HTML	55
HTML Elements Types	55
HTML Styles	56
Styling HTML Elements	56
Adding Styles to HTML Elements	56
Inline Styles	56
This is a heading	57
Embedded Style Sheets	57
External Style Sheets	58
Linking External Style Sheets	58
HTML Images	58
Inserting Images into Web Pages	59
Setting the Width and Height of an Image	59
HTML Tables	60
Creating Tables in HTML	60
Spanning Multiple Rows and Columns	62
HTML Lists	63
Working with HTML Lists	63
HTML Unordered Lists	63
HTML Ordered Lists	64
HTML Forms	65
What is HTML Form	65
Input Element	65
Text Fields	65
Password Field	66
Radio Buttons	66
Checkboxes	66
File Select box	67
Textarea	67
Select Boxes	67
Select Boxes	67
Submit and Reset Buttons	68

TOPS Technologies

Grouping Form Controls	68
Frequently Used Form Attributes	69
Module – 3 [CSS & CSS3]	69
CSS Border	70
CSS Border Properties	70
Understanding the Different Border Styles.....	70
Setting the Border Width.....	71
CSS Box Shadow	72
CSS box-shadow Property	72
CSS Multiple Backgrounds	73
CSS Text Shadow.....	75
Module – 4 [Learning The Language].....	76
PHP	76
What You Can Do with PHP	77
Advantages of PHP over Other Languages	77
Getting Started	78
Getting Started with PHP	78
Setting Up a Local Web Server.....	78
Creating Your First PHP Script	79
PHP Syntax.....	79
Standard PHP Syntax	80
Embedding PHP within HTML.....	80
PHP Comments	80
Case Sensitivity in PHP	81
PHP Variables	82
What is Variable in PHP.....	82
Naming Conventions for PHP Variables	83
PHP Constants.....	83
What is Constant in PHP	83
Naming Conventions for PHP Constants	84
PHP Echo and Print Statements	84
The PHP echo Statement	84
Display Strings of Text.....	84
Display HTML Code.....	84
Display Variables	85
The PHP print Statement	85
Display Strings of Text.....	86
Display HTML Code.....	86
Display Variables	86
PHP Data Types	87
Data Types in PHP	87
PHP Integers.....	87
PHP Strings	88
PHP Floating Point Numbers or Doubles.....	88
PHP Booleans.....	89
PHP Arrays	90
PHP Objects	90
PHP NULL	92

TOPS Technologies

PHP Resources	93
PHP Strings.....	93
What is String in PHP	93
Manipulating PHP Strings	95
Calculating the Length of a String	95
Counting Number of Words in a String	95
Replacing Text within Strings.....	96
Reversing a String	97
PHP Operators.....	97
What is Operators in PHP	97
PHP Arithmetic Operators.....	97
PHP Assignment Operators	98
PHP Comparison Operators	100
PHP Incrementing and Decrementing Operators.....	102
PHP Logical Operators	104
PHP String Operators	105
PHP Array Operators	105
PHP Spaceship Operator PHP 7	107
PHP If...Else Statements	109
PHP Conditional Statements.....	109
The if Statement.....	110
The if...else Statement	110
The if...elseif...else Statement.....	111
The Ternary Operator	112
The Null Coalescing Operator PHP 7	113
PHP Switch...Case Statements	113
PHP If...Else Vs Switch...Case	113
PHP Arrays	115
What is PHP Arrays	115
Types of Arrays in PHP.....	116
Indexed Arrays.....	116
Associative Arrays	116
Multidimensional Arrays	117
PHP Sorting Arrays	118
PHP Functions For Sorting Arrays	118
Sorting Indexed Arrays in Ascending Order.....	118
Sorting Indexed Arrays in Descending Order.....	119
Sorting Associative Arrays in Ascending Order By Value	119
Sorting Associative Arrays in Descending Order By Value	120
Sorting Associative Arrays in Ascending Order By Key	120
Sorting Associative Arrays in Descending Order By Key	120
PHP Loops	121
Different Types of Loops in PHP	121
PHP while Loop	121
PHP do...while Loop	122
Difference Between while and do...while Loop	124
PHP for Loop	124
PHP foreach Loop.....	124
PHP Functions	126

TOPS Technologies

PHP Built-in Functions	126
PHP User-Defined Functions	126
Creating and Invoking Functions.....	126
Functions with Parameters.....	127
Functions with Optional Parameters and Default Values	128
Returning Values from a Function.....	128
Passing Arguments to a Function by Reference	130
Understanding the Variable Scope	130
The global Keyword	132
Creating Recursive Functions	132
PHP Superglobals	135
PHP Date and Time	140
The PHP Date() Function.....	140
Formatting the Dates and Times with PHP	140
The PHP time() Function.....	141
PHP Include and Require Files	143
Including a PHP File into Another PHP File	143
Difference Between include and require Statements	143
The include_once and require_once Statements.....	144
Module – 5 [Database Connectivity].....	146
PHP - MySQL.....	146
What is MySQL	146
Talking to MySQL Databases with SQL.....	146
PHP Connect to MySQL Server	147
Ways of Connecting to MySQL through PHP.....	147
Connecting to MySQL Database Server	147
Closing the MySQL Database Server Connection	149
Creating MySQL Database Using PHP.....	149
PHP MySQL Create Tables	150
PHP MySQL INSERT Query	151
Inserting Data into a MySQL Database Table.....	151
Inserting Multiple Rows into a Table	152
Insert Data into a Database from an HTML Form	153
Step 1: Creating the HTML Form.....	153
Step 2: Retrieving and Inserting the Form Data.....	154
PHP MySQL Last Inserted ID.....	155
How to Get the ID of Last Inserted Row.....	155
PHP MySQL SELECT Query	156
Selecting Data From Database Tables	156
Explanation of Code (Procedural style).....	158
PHP MySQL WHERE Clause	158
Filtering the Records	158
PHP MySQL LIMIT Clause.....	161
Limiting Result Sets	161
PHP MySQL ORDER BY Clause	164
Ordering the Result Set	164
PHP MySQL UPDATE Query.....	166
Updating Database Table Data.....	166
PHP MySQL DELETE Query.....	168

TOPS Technologies

Deleting Database Table Data	168
SQL Join	169
SQL Injection.....	174
SQL Injection	174
SQL in Web Pages	174
SQL Injection Based on Batched SQL Statements	176
The Anatomy of a Cookie	177
Setting Cookies with PHP	178
Accessing Cookies with PHP	179
Deleting Cookie with PHP	180
PHP Sessions	180
Starting a PHP Session	181
Storing and Accessing Session Data	181
Destroying a Session	182
PHP File Upload.....	182
Uploading Files with PHP.....	182
Step 1: Creating an HTML form to upload the file.....	183
Step 2: Processing the uploaded file	183
Explanation of Code.....	185
Classes, objects, methods and properties.....	186
How to create classes?.....	186
How to add properties to a class?	187
How to create objects from a class?	187
Objects, what are they good for?	188
How to get an object's properties?	189
How to set the object's properties?	189
How to add methods to a class?	190
The \$this keyword	192
The \$this keyword	192
Chaining methods and properties	196
Access modifiers: public vs. private	198
The public access modifier.....	198
The private access modifier	199
How to access a private property?	200
Why do we need access modifiers?	202
Overloading in PHP.....	204
PHP's Overloading.....	204
Dynamic Entities on PHP Overloading	205
PHP's Overloading Types	205
Property Overloading	205
Example: PHP Property Overloading.....	206
Method Overloading.....	207
PHP example for Method Overloading	207
Final keyword in PHP	208
Scope Resolution operator in PHP.....	210
Abstraction Interface	213
PHP - What are Abstract Classes and Methods?	213
Interface	215
Traits	217

TOPS Technologies

PHP - What are Traits?	217
Type Hinting	218
Inheritance	219
Constructor.....	220
PHP - The __construct Function	220
Destructor.....	221
PHP - The __destruct Function	221
Object Iteration.....	222
Introduction	222
Using foreach loop	222
Output	223
PHP Magic Methods.....	224
PHP Magic Methods and Purposes	224
PHP Magic Methods Invoked on Creating Class Instance.....	225
Magic Methods with PHP Overloading	225
Magic Methods with PHP Object Serialization	225
Other Magic Methods in PHP.....	225
PHP File Handling.....	226
PHP Manipulating Files.....	226
PHP readfile() Function.....	226
PHP MVC.....	227
Advantages:.....	227
Visual Representation for Data flow of MVC Architecture represented below:	227
Model:.....	228
View:.....	228
Controller:.....	228
MVC Structure.....	229
Hello World Example using PHP MVC	229
How to insert data in database using PHP MVC with example	230
How to Update data in database using PHP MVC with example	234
How to View data in database using PHP MVC with example	237
How to Delete data in database using PHP MVC with example.....	239
Hello World using Javascript.....	242
Applications of Javascript Programming	242
What is JavaScript ?	243
Client-Side JavaScript.....	243
Advantages of JavaScript	243
Limitations of JavaScript	244
JavaScript Events.....	244
Mouse events:.....	244
Keyboard events:	245
Form events:.....	245
Window/Document events	245
Click Event.....	246
MouseOver Event	246
Load event	247
AJAX	247
What Is AJAX?	248

TOPS Technologies

How AJAX Works Using Vanilla JavaScript	249
How to Use JavaScript Promises for AJAX.....	251
How AJAX Works Using the jQuery Library	252
A Real-World AJAX Example With PHP	253
Using Promises for AJAX With jQuery.....	255
Module – 7[Application for Industrial Projects]	266
Web Services in PHP	266
PHP Shopping Cart	270
PHP Shopping Cart Software Development Overview.....	270
Create a Product Gallery for Shopping Cart.....	271
Adding Products to Shopping Cart.....	272
List Cart Items from the PHP Session.....	274
Removing or Clearing Cart Item.....	276
Database Product Table for Shopping Cart	277
Database Product Table for Shopping Cart	277
Simple PHP Shopping Cart Output	278
What is inside?.....	280
About Stripe Checkout	280
Steps to setup Stripe one-time payment flow.....	281
About Stripe payment integration example	281
Get and configure Stripe API keys	282
Create webhook and map events	283
Required libraries to access Stripe API.....	284
Create client-server code to process checkout	284
HTML page with Stripe checkout button	285
JavaScript event handler initiates checkout session.....	286
PHP endpoint processing create-checkout-session request.....	286
Capture and process response	290
Showing success page after payment	294
Database script	294
Evaluate integration with test data	297
Stripe one-time payment example output	297
PHP Send Emails	299
The PHP mail() Function.....	299
Sending attachments with email	299
Sending Plain Text Emails	300
Software Engineering	31
Definitions	32
Software Evolution	32
Software Evolution Laws.....	33
Software Paradigms.....	34
Software Development Paradigm	34
Software Design Paradigm	34
Programming Paradigm.....	34
Need of Software Engineering	34
Characteristics of good software.....	35
Operational	35
Transitional	35

TOPS Technologies

Maintenance	36
Various Syntax in SQL	36
SQL SELECT Statement	36
SQL DISTINCT Clause	36
SQL WHERE Clause	37
SQL AND/OR Clause	37
SQL IN Clause	37
SQL BETWEEN Clause	37
SQL LIKE Clause	37
SQL ORDER BY Clause	37
SQL GROUP BY Clause	37
SQL COUNT Clause	38
SQL HAVING Clause	38
SQL CREATE TABLE Statement	38
SQL DROP TABLE Statement	38
SQL CREATE INDEX Statement	39
SQL DROP INDEX Statement	39
SQL DESC Statement	39
SQL TRUNCATE TABLE Statement	39
SQL ALTER TABLE Statement	39
SQL ALTER TABLE Statement (Rename)	39
SQL INSERT INTO Statement	39
SQL UPDATE Statement	39
SQL DELETE Statement	40
SQL CREATE DATABASE Statement	40
SQL DROP DATABASE Statement	40
SQL USE Statement	40
SQL COMMIT Statement	40
SQL ROLLBACK Statement	40
Characteristics	41
Users	42
Types of Normal Forms	43
First Normal Form (1NF)	44
Second Normal Form (2NF)	45
Third Normal Form (3NF)	46
Module - 2 [HTML & HTML5]	31
What You Can Do with HTML	50
Step 1: Creating the HTML file	51
Step 2: Type some HTML code	51
Step 3: Saving the file	51
HTML Tags and Elements	52
HTML Elements	52
HTML Element Syntax	53
HTML Tags Vs Elements	53
Case Insensitivity in HTML Tags and Attributes	53
Empty HTML Elements	53
Nesting HTML Elements	54
Writing Comments in HTML	55

TOPS Technologies

HTML Elements Types	55
HTML Styles.....	56
Styling HTML Elements	56
Adding Styles to HTML Elements	56
Inline Styles.....	56
This is a heading	57
Embedded Style Sheets	57
External Style Sheets.....	58
Linking External Style Sheets	58
HTML Images.....	58
Inserting Images into Web Pages	59
Setting the Width and Height of an Image	59
HTML Tables.....	60
Creating Tables in HTML	60
Spanning Multiple Rows and Columns	62
HTML Lists	63
Working with HTML Lists	63
HTML Unordered Lists	63
HTML Ordered Lists.....	64
HTML Forms	65
What is HTML Form.....	65
Input Element.....	65
Text Fields	65
Password Field	66
Radio Buttons	66
Checkboxes	66
File Select box	67
Textarea.....	67
Select Boxes	67
Select Boxes	67
Submit and Reset Buttons	68
Grouping Form Controls	68
Frequently Used Form Attributes	69
Module – 3 [CSS & CSS3]	69
CSS Border	70
CSS Border Properties	70
Understanding the Different Border Styles.....	70
Setting the Border Width	71
CSS Box Shadow	72
CSS box-shadow Property	72
CSS Multiple Backgrounds	73
CSS Text Shadow.....	75
Module – 4 [Learning The Language].....	76
PHP.....	76
What You Can Do with PHP	77
Advantages of PHP over Other Languages	77
Getting Started	78
Getting Started with PHP	78

TOPS Technologies

Setting Up a Local Web Server.....	78
Creating Your First PHP Script	79
PHP Syntax.....	79
Standard PHP Syntax.....	80
Embedding PHP within HTML.....	80
PHP Comments	80
Case Sensitivity in PHP	81
PHP Variables	82
What is Variable in PHP.....	82
Naming Conventions for PHP Variables	83
PHP Constants.....	83
What is Constant in PHP	83
Naming Conventions for PHP Constants	84
PHP Echo and Print Statements	84
The PHP echo Statement	84
Display Strings of Text.....	84
Display HTML Code.....	84
Display Variables	85
The PHP print Statement	85
Display Strings of Text.....	86
Display HTML Code.....	86
Display Variables	86
PHP Data Types	87
Data Types in PHP	87
PHP Integers.....	87
PHP Strings	88
PHP Floating Point Numbers or Doubles	88
PHP Booleans.....	89
PHP Arrays	90
PHP Objects	90
PHP NULL	92
PHP Resources	93
PHP Strings.....	93
What is String in PHP	93
Manipulating PHP Strings	95
Calculating the Length of a String	95
Counting Number of Words in a String	95
Replacing Text within Strings.....	96
Reversing a String	97
PHP Operators.....	97
What is Operators in PHP	97
PHP Arithmetic Operators.....	97
PHP Assignment Operators	98
PHP Comparison Operators	100
PHP Incrementing and Decrementing Operators.....	102
PHP Logical Operators	104
PHP String Operators	105
PHP Array Operators	105

TOPS Technologies

PHP Spaceship Operator PHP 7	107
PHP If...Else Statements	109
PHP Conditional Statements	109
The if Statement	110
The if...else Statement	110
The if...elseif...else Statement	111
The Ternary Operator	112
The Null Coalescing Operator PHP 7	113
PHP Switch...Case Statements	113
PHP If...Else Vs Switch...Case	113
PHP Arrays	115
What is PHP Arrays	115
Types of Arrays in PHP	116
Indexed Arrays	116
Associative Arrays	116
Multidimensional Arrays	117
PHP Sorting Arrays	118
PHP Functions For Sorting Arrays	118
Sorting Indexed Arrays in Ascending Order	118
Sorting Indexed Arrays in Descending Order	119
Sorting Associative Arrays in Ascending Order By Value	119
Sorting Associative Arrays in Descending Order By Value	120
Sorting Associative Arrays in Ascending Order By Key	120
Sorting Associative Arrays in Descending Order By Key	120
PHP Loops	121
Different Types of Loops in PHP	121
PHP while Loop	121
PHP do...while Loop	122
Difference Between while and do...while Loop	124
PHP for Loop	124
PHP foreach Loop	124
PHP Functions	126
PHP Built-in Functions	126
PHP User-Defined Functions	126
Creating and Invoking Functions	126
Functions with Parameters	127
Functions with Optional Parameters and Default Values	128
Returning Values from a Function	128
Passing Arguments to a Function by Reference	130
Understanding the Variable Scope	130
The global Keyword	132
Creating Recursive Functions	132
PHP Superglobals	135
PHP Date and Time	140
The PHP Date() Function	140
Formatting the Dates and Times with PHP	140
The PHP time() Function	141
PHP Include and Require Files	143
Including a PHP File into Another PHP File	143

TOPS Technologies

Difference Between include and require Statements	143
The include_once and require_once Statements.....	144
Module – 5 [Database Connectivity].....	146
PHP - MySQL.....	146
What is MySQL	146
Talking to MySQL Databases with SQL.....	146
PHP Connect to MySQL Server	147
Ways of Connecting to MySQL through PHP.....	147
Connecting to MySQL Database Server	147
Closing the MySQL Database Server Connection	149
Creating MySQL Database Using PHP.....	149
PHP MySQL Create Tables	150
PHP MySQL INSERT Query	151
Inserting Data into a MySQL Database Table.....	151
Inserting Multiple Rows into a Table	152
Insert Data into a Database from an HTML Form	153
Step 1: Creating the HTML Form.....	153
Step 2: Retrieving and Inserting the Form Data.....	154
PHP MySQL Last Inserted ID.....	155
How to Get the ID of Last Inserted Row.....	155
PHP MySQL SELECT Query	156
Selecting Data From Database Tables	156
Explanation of Code (Procedural style).....	158
PHP MySQL WHERE Clause	158
Filtering the Records	158
PHP MySQL LIMIT Clause.....	161
Limiting Result Sets	161
PHP MySQL ORDER BY Clause	164
Ordering the Result Set	164
PHP MySQL UPDATE Query.....	166
Updating Database Table Data.....	166
PHP MySQL DELETE Query	168
Deleting Database Table Data.....	168
SQL Join	169
SQL Injection.....	174
SQL Injection	174
SQL in Web Pages	174
SQL Injection Based on Batched SQL Statements	176
The Anatomy of a Cookie	177
Setting Cookies with PHP	178
Accessing Cookies with PHP	179
Deleting Cookie with PHP	180
PHP Sessions	180
Starting a PHP Session	181
Storing and Accessing Session Data	181
Destroying a Session	182
PHP File Upload.....	182
Uploading Files with PHP.....	182
Step 1: Creating an HTML form to upload the file.....	183

TOPS Technologies

Step 2: Processing the uploaded file	183
Explanation of Code.....	185
Software Engineering	31
Definitions	32
Software Evolution	32
Software Evolution Laws.....	33
Software Paradigms.....	34
Software Development Paradigm	34
Software Design Paradigm	34
Programming Paradigm.....	34
Need of Software Engineering	34
Characteristics of good software.....	35
Operational	35
Transitional	35
Maintenance	36
Various Syntax in SQL	36
SQL SELECT Statement	36
SQL DISTINCT Clause	36
SQL WHERE Clause	37
SQL AND/OR Clause	37
SQL IN Clause.....	37
SQL BETWEEN Clause.....	37
SQL LIKE Clause.....	37
SQL ORDER BY Clause.....	37
SQL GROUP BY Clause	37
SQL COUNT Clause.....	38
SQL HAVING Clause.....	38
SQL CREATE TABLE Statement	38
SQL DROP TABLE Statement.....	38
SQL CREATE INDEX Statement.....	39
SQL DROP INDEX Statement.....	39
SQL DESC Statement	39
SQL TRUNCATE TABLE Statement	39
SQL ALTER TABLE Statement	39
SQL ALTER TABLE Statement (Rename)	39
SQL INSERT INTO Statement.....	39
SQL UPDATE Statement.....	39
SQL DELETE Statement	40
SQL CREATE DATABASE Statement	40
SQL DROP DATABASE Statement.....	40
SQL USE Statement.....	40
SQL COMMIT Statement.....	40
SQL ROLLBACK Statement.....	40
Characteristics	41
Users	42
Types of Normal Forms	43
First Normal Form (1NF)	44
Second Normal Form (2NF)	45

TOPS Technologies

Third Normal Form (3NF)	46
Module - 2 [HTML & HTML5]	31
What You Can Do with HTML	50
Step 1: Creating the HTML file	51
Step 2: Type some HTML code	51
Step 3: Saving the file	51
HTML Tags and Elements	52
HTML Elements	52
HTML Element Syntax	53
HTML Tags Vs Elements	53
Case Insensitivity in HTML Tags and Attributes	53
Empty HTML Elements	53
Nesting HTML Elements	54
Writing Comments in HTML	55
HTML Elements Types	55
HTML Styles	56
Styling HTML Elements	56
Adding Styles to HTML Elements	56
Inline Styles	56
This is a heading	57
Embedded Style Sheets	57
External Style Sheets	58
Linking External Style Sheets	58
HTML Images	58
Inserting Images into Web Pages	59
Setting the Width and Height of an Image	59
HTML Tables	60
Creating Tables in HTML	60
Spanning Multiple Rows and Columns	62
HTML Lists	63
Working with HTML Lists	63
HTML Unordered Lists	63
HTML Ordered Lists	64
HTML Forms	65
What is HTML Form	65
Input Element	65
Text Fields	65
Password Field	66
Radio Buttons	66
Checkboxes	66
File Select box	67
Textarea	67
Select Boxes	67
Select Boxes	67
Submit and Reset Buttons	68
Grouping Form Controls	68
Frequently Used Form Attributes	69
Module – 3 [CSS & CSS3]	69
CSS Border	70

TOPS Technologies

CSS Border Properties	70
Understanding the Different Border Styles.....	70
Setting the Border Width	71
CSS Box Shadow	72
CSS box-shadow Property	72
CSS Multiple Backgrounds	73
CSS Text Shadow.....	75
Module – 4 [Learning The Language].....	76
PHP	76
What You Can Do with PHP	77
Advantages of PHP over Other Languages	77
Getting Started	78
Getting Started with PHP	78
Setting Up a Local Web Server.....	78
Creating Your First PHP Script	79
PHP Syntax	79
Standard PHP Syntax	80
Embedding PHP within HTML.....	80
PHP Comments	80
Case Sensitivity in PHP	81
PHP Variables	82
What is Variable in PHP.....	82
Naming Conventions for PHP Variables	83
PHP Constants.....	83
What is Constant in PHP	83
Naming Conventions for PHP Constants	84
PHP Echo and Print Statements	84
The PHP echo Statement	84
Display Strings of Text.....	84
Display HTML Code.....	84
Display Variables	85
The PHP print Statement.....	85
Display Strings of Text.....	86
Display HTML Code.....	86
Display Variables	86
PHP Data Types	87
Data Types in PHP	87
PHP Integers.....	87
PHP Strings	88
PHP Floating Point Numbers or Doubles.....	88
PHP Booleans.....	89
PHP Arrays	90
PHP Objects	90
PHP NULL	92
PHP Resources	93
PHP Strings.....	93
What is String in PHP	93
Manipulating PHP Strings	95

TOPS Technologies

Calculating the Length of a String	95
Counting Number of Words in a String	95
Replacing Text within Strings.....	96
Reversing a String	97
PHP Operators	97
What is Operators in PHP	97
PHP Arithmetic Operators.....	97
PHP Assignment Operators	98
PHP Comparison Operators	100
PHP Incrementing and Decrementing Operators.....	102
PHP Logical Operators	104
PHP String Operators	105
PHP Array Operators	105
PHP Spaceship Operator PHP 7	107
PHP If...Else Statements	109
PHP Conditional Statements.....	109
The if Statement.....	110
The if...else Statement	110
The if...elseif...else Statement.....	111
The Ternary Operator	112
The Null Coalescing Operator PHP 7	113
PHP Switch...Case Statements	113
PHP If...Else Vs Switch...Case	113
PHP Arrays	115
What is PHP Arrays	115
Types of Arrays in PHP.....	116
Indexed Arrays.....	116
Associative Arrays	116
Multidimensional Arrays	117
PHP Sorting Arrays	118
PHP Functions For Sorting Arrays	118
Sorting Indexed Arrays in Ascending Order.....	118
Sorting Indexed Arrays in Descending Order.....	119
Sorting Associative Arrays in Ascending Order By Value	119
Sorting Associative Arrays in Descending Order By Value	120
Sorting Associative Arrays in Ascending Order By Key	120
Sorting Associative Arrays in Descending Order By Key	120
PHP Loops	121
Different Types of Loops in PHP	121
PHP while Loop	121
PHP do...while Loop	122
Difference Between while and do...while Loop	124
PHP for Loop	124
PHP foreach Loop.....	124
PHP Functions	126
PHP Built-in Functions	126
PHP User-Defined Functions	126
Creating and Invoking Functions.....	126
Functions with Parameters	127

TOPS Technologies

Functions with Optional Parameters and Default Values	128
Returning Values from a Function.....	128
Passing Arguments to a Function by Reference	130
Understanding the Variable Scope	130
The global Keyword	132
Creating Recursive Functions	132
PHP Superglobals	135
PHP Date and Time	140
The PHP Date() Function.....	140
Formatting the Dates and Times with PHP	140
The PHP time() Function.....	141
PHP Include and Require Files	143
Including a PHP File into Another PHP File	143
Difference Between include and require Statements	143
The include_once and require_once Statements.....	144
Module – 5 [Database Connectivity].....	146
PHP - MySQL.....	146
What is MySQL	146
Talking to MySQL Databases with SQL.....	146
PHP Connect to MySQL Server	147
Ways of Connecting to MySQL through PHP.....	147
Connecting to MySQL Database Server	147
Closing the MySQL Database Server Connection	149
Creating MySQL Database Using PHP.....	149
PHP MySQL Create Tables	150
PHP MySQL INSERT Query	151
Inserting Data into a MySQL Database Table	151
Inserting Multiple Rows into a Table	152
Insert Data into a Database from an HTML Form	153
Step 1: Creating the HTML Form.....	153
Step 2: Retrieving and Inserting the Form Data.....	154
PHP MySQL Last Inserted ID.....	155
How to Get the ID of Last Inserted Row.....	155
PHP MySQL SELECT Query	156
Selecting Data From Database Tables	156
Explanation of Code (Procedural style).....	158
PHP MySQL WHERE Clause	158
Filtering the Records	158
PHP MySQL LIMIT Clause.....	161
Limiting Result Sets	161
PHP MySQL ORDER BY Clause	164
Ordering the Result Set	164
PHP MySQL UPDATE Query.....	166
Updating Database Table Data.....	166
PHP MySQL DELETE Query	168
Deleting Database Table Data.....	168
SQL Join	169
SQL Injection.....	174
SQL Injection	174

TOPS Technologies

SQL in Web Pages	174
SQL Injection Based on Batched SQL Statements	176
The Anatomy of a Cookie	177
Setting Cookies with PHP	178
Accessing Cookies with PHP	179
Deleting Cookie with PHP	180
PHP Sessions	180
Starting a PHP Session	181
Storing and Accessing Session Data	181
Destroying a Session	182
PHP File Upload.....	182
Uploading Files with PHP.....	182
Step 1: Creating an HTML form to upload the file.....	183
Step 2: Processing the uploaded file	183
Explanation of Code.....	185
Classes, objects, methods and properties.....	186
How to create classes?.....	186
How to add properties to a class?	187
How to create objects from a class?	187
Objects, what are they good for?	188
How to get an object's properties?	189
How to set the object's properties?	189
How to add methods to a class?	190
The \$this keyword	192
The \$this keyword	192
Chaining methods and properties	196
Access modifiers: public vs. private	198
The public access modifier.....	198
The private access modifier	199
How to access a private property?	200
Why do we need access modifiers?.....	202
Overloading in PHP	204
PHP's Overloading.....	204
Dynamic Entities on PHP Overloading	205
PHP's Overloading Types	205
Property Overloading	205
Example: PHP Property Overloading.....	206
Method Overloading.....	207
PHP example for Method Overloading	207
Final keyword in PHP	208
Scope Resolution operator in PHP.....	210
Abstraction Interface	213
PHP - What are Abstract Classes and Methods?	213
Interface	215
Traits	217
PHP - What are Traits?.....	217
Type Hinting	218
Inheritance	219
Constructor	220

TOPS Technologies

PHP - The __construct Function	220
Destructor.....	221
PHP - The __destruct Function.....	221
Object Iteration.....	222
Introduction	222
Using foreach loop	222
Output	223
PHP Magic Methods.....	224
PHP Magic Methods and Purposes	224
PHP Magic Methods Invoked on Creating Class Instance.....	225
Magic Methods with PHP Overloading	225
Magic Methods with PHP Object Serialization	225
Other Magic Methods in PHP.....	225
PHP File Handling	226
PHP Manipulating Files.....	226
PHP readfile() Function.....	226
PHP MVC	227
Advantages:.....	227
Visual Representation for Data flow of MVC Architecture represented below:	227
Model:.....	228
View:.....	228
Controller:.....	228
MVC Structure.....	229
Hello World Example using PHP MVC	229
How to insert data in database using PHP MVC with example	230
How to Update data in database using PHP MVC with example	234
How to View data in database using PHP MVC with example	237
How to Delete data in database using PHP MVC with example.....	239
Hello World using Javascript.....	242
Applications of Javascript Programming	242
What is JavaScript ?	243
Client-Side JavaScript.....	243
Advantages of JavaScript	243
Limitations of JavaScript	244
JavaScript Events.....	244
Mouse events:.....	244
Keyboard events:	245
Form events:	245
Window/Document events	245
Click Event.....	246
MouseOver Event	246
Load event	247
AJAX	247
What Is AJAX?	248
How AJAX Works Using Vanilla JavaScript	249
How to Use JavaScript Promises for AJAX.....	251
How AJAX Works Using the jQuery Library	252
A Real-World AJAX Example With PHP	253

TOPS Technologies

Using Promises for AJAX With jQuery.....	255
Module – 7[Application for Industrial Projects]	266
Web Services in PHP	266
PHP Shopping Cart.....	270
PHP Shopping Cart Software Development Overview.....	270
Create a Product Gallery for Shopping Cart.....	271
Adding Products to Shopping Cart.....	272
List Cart Items from the PHP Session.....	274
Removing or Clearing Cart Item.....	276
Database Product Table for Shopping Cart	277
Database Product Table for Shopping Cart	277
Simple PHP Shopping Cart Output	278
What is inside?.....	280
About Stripe Checkout	280
Steps to setup Stripe one-time payment flow.....	281
About Stripe payment integration example	281
Get and configure Stripe API keys	282
Create webhook and map events	283
Required libraries to access Stripe API.....	284
Create client-server code to process checkout	284
HTML page with Stripe checkout button	285
JavaScript event handler initiates checkout session.....	286
PHP endpoint processing create-checkout-session request.....	286
Capture and process response	290
Showing success page after payment	294
Database script	294
Evaluate integration with test data	297
Stripe one-time payment example output	297
PHP Send Emails.....	299
The PHP mail() Function.....	299
Sending attachments with email	299
Sending Plain Text Emails	300
Module – 6 [Advance PHP]	170
Classes, objects, methods and properties.....	186
How to create classes?	186
How to add properties to a class?	187
How to create objects from a class?	187
Objects, what are they good for?	188
How to get an object's properties?	189
How to set the object's properties?	189
How to add methods to a class?	190
The \$this keyword	192
The \$this keyword	192
Chaining methods and properties	196
Access modifiers: public vs. private	198
The public access modifier.....	198
The private access modifier	199
How to access a private property?	200
Why do we need access modifiers?	202

TOPS Technologies

Overloading in PHP	204
PHP's Overloading.....	204
Dynamic Entities on PHP Overloading	205
PHP's Overloading Types	205
Property Overloading	205
Example: PHP Property Overloading.....	206
Method Overloading.....	207
PHP example for Method Overloading	207
Final keyword in PHP	208
Scope Resolution operator in PHP.....	210
Abstraction Interface	213
PHP - What are Abstract Classes and Methods?	213
Interface	215
Traits	217
PHP - What are Traits?.....	217
Type Hinting	218
Inheritance	219
Constructor.....	220
PHP - The __construct Function.....	220
Destructor.....	221
PHP - The __destruct Function.....	221
Object Iteration.....	222
Introduction	222
Using foreach loop	222
Output	223
PHP Magic Methods.....	224
PHP Magic Methods and Purposes	224
PHP Magic Methods Invoked on Creating Class Instance.....	225
Magic Methods with PHP Overloading	225
Magic Methods with PHP Object Serialization.....	225
Other Magic Methods in PHP.....	225
PHP File Handling	226
PHP Manipulating Files.....	226
PHP readfile() Function.....	226
PHP MVC	227
Advantages:.....	227
Visual Representation for Data flow of MVC Architecture represented below:	227
Model:.....	228
View:.....	228
Controller:.....	228
MVC Structure.....	229
Hello World Example using PHP MVC	229
How to insert data in database using PHP MVC with example	230
How to Update data in database using PHP MVC with example	234
How to View data in database using PHP MVC with example	237
How to Delete data in database using PHP MVC with example.....	239
Hello World using Javascript.....	242
Applications of Javascript Programming	242

TOPS Technologies

What is JavaScript ?	243
Client-Side JavaScript.....	243
Advantages of JavaScript	243
Limitations of JavaScript	244
JavaScript Events.....	244
Mouse events:.....	244
Keyboard events:	245
Form events:	245
Window/Document events	245
Click Event.....	246
MouseOver Event	246
Load event	247
AJAX	247
What Is AJAX?.....	248
How AJAX Works Using Vanilla JavaScript	249
How to Use JavaScript Promises for AJAX.....	251
How AJAX Works Using the jQuery Library	252
A Real-World AJAX Example With PHP	253
Using Promises for AJAX With jQuery.....	255
Module – 7[Application for Industrial Projects]	266
Web Services in PHP	266
PHP Shopping Cart.....	270
PHP Shopping Cart Software Development Overview.....	270
Create a Product Gallery for Shopping Cart.....	271
Adding Products to Shopping Cart.....	272
List Cart Items from the PHP Session.....	274
Removing or Clearing Cart Item.....	276
Database Product Table for Shopping Cart	277
Database Product Table for Shopping Cart	277
Simple PHP Shopping Cart Output	278
What is inside?.....	280
About Stripe Checkout	280
Steps to setup Stripe one-time payment flow.....	281
About Stripe payment integration example	281
Get and configure Stripe API keys	282
Create webhook and map events	283
Required libraries to access Stripe API.....	284
Create client-server code to process checkout	284
HTML page with Stripe checkout button	285
JavaScript event handler initiates checkout session.....	286
PHP endpoint processing create-checkout-session request.....	286
Capture and process response	290
Showing success page after payment	294
Database script	294
Evaluate integration with test data	297
Stripe one-time payment example output	297
PHP Send Emails	299
The PHP mail() Function.....	299
Sending attachments with email	299

TOPS Technologies

Sending Plain Text Emails	300
Module – 8 [CodeIgniter]	305
What is CodeIgniter.....	305
CodeIgniter License	305
License Agreement.....	306
CodeIgniter Versions.....	306
GitHub.....	306
Features of CodeIgniter.....	306
Important Features.....	306
Some other Features	307
CodeIgniter Installation.....	307
File structure in CodeIgniter	309
Application	310
System.....	312
User_guide.....	313
CodeIgniter Architecture.....	313
Data flow in CodeIgniter.....	313
Model-View-Controller (MVC)	314
Model	314
View	314
Controller	314
Models.....	315
Loading a Model.....	315
Connecting Models to Database	315
Views.....	316
What is view	316
View syntax.....	316
Loading View	316
Loading Multiple Views	316
Controller.....	316
What is Controller	316
Controller Syntax	316
What is Default Controller	316
CodeIgniter First Example.....	317
An Example to print Hello World	317
CodeIgniter URL	318
Basic URL structure	318
What is site_url();	318
What is base_url();.....	319
What is uri_string();.....	319
What is current_url();	319
What is index_page();	319
anchor().....	319
anchor_popup().....	319
mailto().....	320
url_title().....	320
Basic Site creation in CodeIgniter	320
CodeIgniter Methods	323
Method other than index().....	323

TOPS Technologies

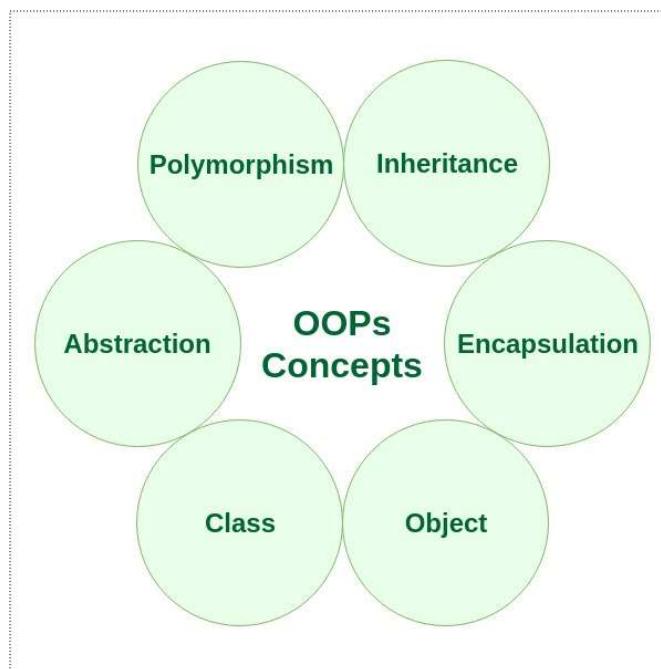
Remapping Method Calls.....	325
CodeIgniter Helper	325
What is Helper	325
Loading a Helper.....	326
Loading Multiple Helpers.....	326
html Helper Example.....	326
CodeIgniter Library.....	331
What is a Library	331
 Loading Library	331
Creating Libraries.....	331
Creating an entire new library	331
Extending Native Libraries	332
Replacing Native Libraries	332
URL Routing.....	332
Setting your own Routing Rules.....	332
Wildcards	333
Regular Expression.....	333
URL Suffix.....	333
CodeIgniter Hooks.....	334
Enabling Hooks.....	334
Defining a Hook	334
Multiple calls to the same Hook	334
Hook Points.....	335
Hook Example.....	336
Passing Parameters in CodeIgniter.....	337
CodeIgniter Driver	339
What are Drivers	339
Initializing Driver.....	339
Creating Own Drivers.....	339
Making file structure	339
Making Driver List	340
Database Configuration.....	342
Automatically connecting Database	344
Manually connecting Database	344
Connecting multiple Database	345
CodeIgniter SELECT Database record	348
Login Form in CodeIgniter (without MySQL)	354
Login page (with database).....	359
XSS Filtering	378
CSRF (Cross-site Request Forgery)	379
Class Reference	379
Preventing, Enabling from CSRF	380
What is CSRF attack.....	380
Token Method	380
Enabling CSRF Protection	380
Token Generation	380

Module-1 [Fundamentals]

Object Oriented Programming

Object-oriented programming – As the name suggests uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Characteristics of an Object Oriented Programming language



Class: The building block of C++ that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object

TOPS Technologies

For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user-defined data-type which has data members and member functions.
 - Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions define the properties and behaviour of the objects in a Class.
 - In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.
-
- We can say that a **Class in C++** is a blue-print representing a group of objects which shares some common properties and behaviours.

Object: An Object is an identifiable entity with some characteristics and behaviour. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

```
class person
{
    char name[20];
    int id;
public:
    void getdetails(){}
};

int main()
{
person p1; // p1 is a object
}
```

Object take up space in memory and have an associated address like a record in pascal or structure or union in C.

When a program is executed the objects interact by sending messages to one another.

Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

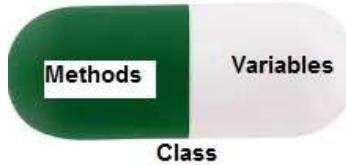
Encapsulation: In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.

Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and

TOPS Technologies

keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

Encapsulation in C++



Encapsulation also leads to *data abstraction or hiding*. As using encapsulation also hides the data. In the above example, the data of any of the section like sales, finance or accounts are hidden from any other section.

Abstraction: Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

- *Abstraction using Classes:* We can implement Abstraction in C++ using classes. The class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to the outside world and which is not.
- *Abstraction in Header files:* One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate the power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating the power of numbers.

Polymorphism: The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

TOPS Technologies

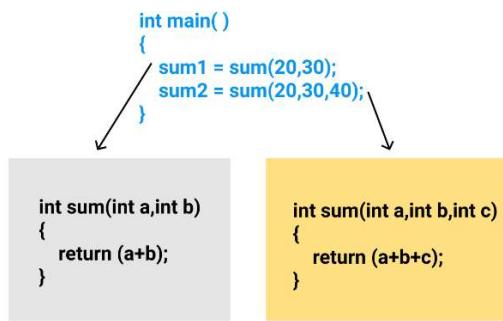
An operation may exhibit different behaviours in different instances. The behaviour depends upon the types of data used in the operation.

C++ supports operator overloading and function overloading.

- **Operator Overloading:** The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
- **Function Overloading:** Function overloading is using a single function name to perform different types of tasks.

Polymorphism is extensively used in implementing inheritance.

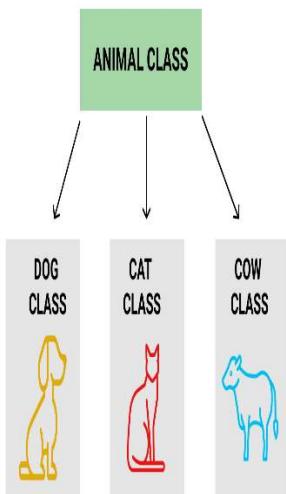
Example: Suppose we have to write a function to add some integers, sometimes there are 2 integers, sometimes there are 3 integers. We can write the Addition Method with the same name having different parameters, the concerned method will be called according to parameters.



Inheritance: The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by sub class is called Base Class or Super class.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Example: Dog, Cat, Cow can be Derived Class of Animal Base Class.



Dynamic Binding: In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has [virtual functions](#) to support this.

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Software Engineering

Let us first understand what software engineering stands for. The term is made of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods



Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

IEEE defines software engineering as:

- (1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
- (2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Software Evolution

TOPS Technologies

The process of developing a software product using software engineering principles and methods is referred to as software evolution. This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software Evolution Laws

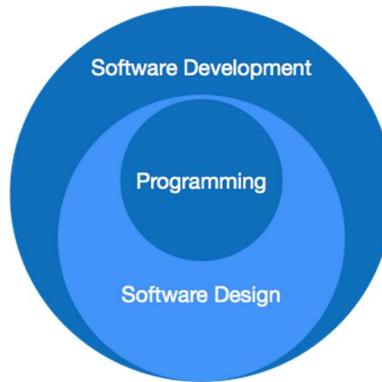
Lehman has given laws for software evolution. He divided the software into three different categories:

- S-type (static-type) - This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.
- P-type (practical-type) - This is a software with a collection of procedures. This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.
- E-type (embedded-type) - This software works closely as the requirement of real-world environment. This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

TOPS Technologies

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand. These can be combined into various categories, though each of them is contained in one another:



Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This Paradigm is known as software engineering paradigms where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirement gathering
- Software design
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

Need of Software Engineering

TOPS Technologies

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software** - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability**- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost**- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature**- The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management**- Better process of software development provides better and quality software product.

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability

TOPS Technologies

- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

SQL Queries

SQL is followed by a unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is case insensitive, which means SELECT and select have same meaning in SQL statements. Whereas, MySQL makes difference in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

Various Syntax in SQL

All the examples given in this tutorial have been tested with a MySQL server.

SQL SELECT Statement

```
SELECT column1, column2....columnN  
FROM    table_name;
```

SQL DISTINCT Clause

```
SELECT DISTINCT column1, column2....columnN  
FROM    table_name;
```

TOPS Technologies

SQL WHERE Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   CONDITION;
```

SQL AND/OR Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name IN (val-1, val-2,...val-N);
```

SQL BETWEEN Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   column_name LIKE { PATTERN };
```

SQL ORDER BY Clause

```
SELECT column1, column2....columnN  
FROM    table_name  
WHERE   CONDITION  
ORDER  BY column_name {ASC|DESC};
```

SQL GROUP BY Clause

TOPS Technologies

```
SELECT SUM(column_name)  
FROM table_name  
WHERE CONDITION  
GROUP BY column_name;
```

SQL COUNT Clause

```
SELECT COUNT(column_name)  
FROM table_name  
WHERE CONDITION;
```

SQL HAVING Clause

```
SELECT SUM(column_name)  
FROM table_name  
WHERE CONDITION  
GROUP BY column_name  
HAVING (arithmetic function condition);
```

SQL CREATE TABLE Statement

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
    columnN datatype,  
    PRIMARY KEY( one or more columns )  
);
```

SQL DROP TABLE Statement

```
DROP TABLE table_name;
```

SQL CREATE INDEX Statement

```
CREATE UNIQUE INDEX index_name  
ON table_name ( column1, column2,...columnN);
```

SQL DROP INDEX Statement

```
ALTER TABLE table_name  
DROP INDEX index_name;
```

SQL DESC Statement

```
DESC table_name;
```

SQL TRUNCATE TABLE Statement

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

```
ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_type};
```

SQL ALTER TABLE Statement (Rename)

```
ALTER TABLE table_name RENAME TO new_table_name;
```

SQL INSERT INTO Statement

```
INSERT INTO table_name( column1, column2....columnN)  
VALUES ( value1, value2....valueN);
```

SQL UPDATE Statement

```
UPDATE table_name
```

TOPS Technologies

```
SET column1 = value1, column2 = value2....columnN=valueN  
[ WHERE CONDITION ];
```

SQL DELETE Statement

```
DELETE FROM table_name  
WHERE {CONDITION};
```

SQL CREATE DATABASE Statement

```
CREATE DATABASE database_name;
```

SQL DROP DATABASE Statement

```
DROP DATABASE database_name;
```

SQL USE Statement

```
USE database_name;
```

SQL COMMIT Statement

```
COMMIT;
```

SQL ROLLBACK Statement

```
ROLLBACK;
```

Database

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

TOPS Technologies

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

Characteristics

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics –

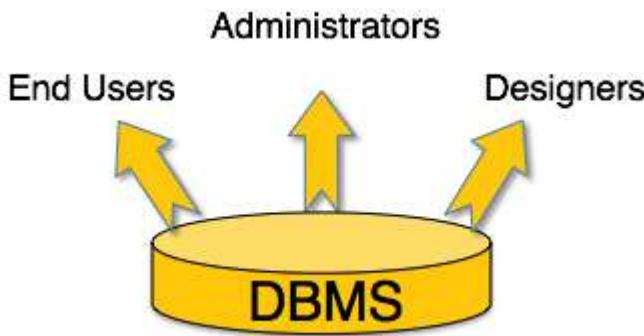
- **Real-world entity** – A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** – DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** – A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- **Less redundancy** – DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- **Consistency** – Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.
- **Query Language** – DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- **ACID Properties** – DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multiuser and Concurrent Access** – DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** – DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** – Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user

TOPS Technologies

in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

Users

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows –



- Administrators – Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.
- Designers – Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- End Users – End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

Constraints –

terms that needs to be satisfied on data

For ex all students must have unique roll number

Can be defined as primary key, foreign key, unique key etc

Primary key – column of table whose value can be used to uniquely identify records **Foreign key** – column inside table that is primary key of another table

Unique key – like primary key can be used to uniquely identify a record

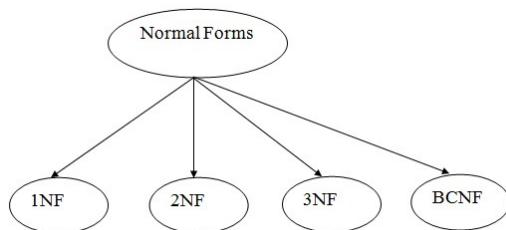
Difference between primary key and unique key is primary key will never allow null where as unique key will allow it for once

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

There are the four types of normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multivalued dependency.
5NF	A relation is in 5NF if it is in 4NF and does not contain any join dependency and joining should be lossless.

TOPS Technologies

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar

TOPS Technologies

12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TOPS Technologies

TEACHER_ID	TEACHER AGE
25	30
47	35
83	38

TEACHER SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

1. X is a super key.

TOPS Technologies

2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010

TOPS Technologies

333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Web Programming

Scripting language, script language or extension language is a programming language that allows control of one or more software applications. "Scripts" are distinct from the core code of the application, which is usually written in a different language, and are often created or at least modified by the end-user. Scripts are often interpreted from source code or byte code, whereas the applications they control are traditionally compiled to native machine code. Scripting languages are nearly always embedded in the applications they control.

What is Website?

A website is a collection of web pages and related content that is identified by a common domain name and published on at least one web server. Notable examples are wikipedia.org, google.com, and amazon.com. All publicly accessible websites collectively constitute the World Wide Web.

Website – basic parts

TOPS Technologies

Websites are consist of three parts

- GUI – web pages that you visit
- Coding – logic that provides functionality or makes website dynamic
- Database – manages data provided by end user
- For GUI building HTML is used from long time

Web browsers

Main article: Client-side scripting

Web browsers are applications for displaying web pages. A host of special-purpose languages has developed to control their operation. These include JavaScript, a scripting language superficially resembling Java; VBScript by Microsoft, which only works in Internet Explorer; XUL by the Mozilla project, which only works in Firefox; and XSLT, a presentation language that transforms XML content into a new form.

Web servers

Main article: Server-side scripting

On the server side of the HTTP link, application servers and other dynamic content servers such as Web content management systems provide content through a large variety of techniques and technologies typified by the scripting approach.

CLIENTS AND SERVERS

- Web programming languages are usually classified as server-side or client-side. Some languages, such as JavaScript, can be used as both client-side and server-side languages, but most Web programming languages are server-side languages
- The client is the Web browser, so client-side scripting uses a Web browser to run scripts. The same script may produce different effects when different browsers run it.
- A Web server is a combination of software and hardware that outputs Web pages after receiving a request from a client. Server-side scripting takes place on the server. If you look at the source of a page created from a server-side script, you'll see only the HTML code the script has generated. The source code for the script is on the server and doesn't need to be downloaded with the page that's sent back to the client

Example of Web Programming Language

- o PHP
- o ASP
- o Perl
- o JAVA

Module – 2 [HTML & HTML5]

HTML

HTML is the main markup language for describing the structure of web pages.



HTML stands for HyperText Markup Language. HTML is the basic building block of World Wide Web.

Hypertext is text displayed on a computer or other electronic device with references to other text that the user can immediately access, usually by a mouse click or key press.

Apart from text, hypertext may contain tables, lists, forms, images, and other presentational elements. It is an easy-to-use and flexible format to share information over the Internet.

Markup languages use sets of markup tags to characterize text elements within a document, which gives instructions to the web browsers on how the document should appear.

HTML was originally developed by Tim Berners-Lee in 1990. He is also known as the father of the web. In 1996, the World Wide Web Consortium (W3C) became the authority to maintain the HTML specifications. HTML also became an international standard (ISO) in 2000. HTML5 is the latest version of HTML. HTML5 provides a faster and more robust approach to web development.

What You Can Do with HTML

There are lot more things you can do with HTML.

- You can publish documents online with text, images, lists, tables, etc.
- You can access web resources such as images, videos or other HTML document via hyperlinks.

TOPS Technologies

- You can create forms to collect user inputs like name, e-mail address, comments, etc.
- You can include images, videos, sound clips, flash movies, applications and other HTML documents directly inside an HTML document.
- You can create offline version of your website that work without internet.
- You can store data in the user's web browser and access later on.
- You can find the current location of your website's visitor.

Step 1: Creating the HTML file

Open up your computer's plain text editor and create a new file.

Step 2: Type some HTML code

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>A simple HTML document</title>

</head>

<body>

<p>Hello World!<p>

</body>

</html>
```

Step 3: Saving the file

Now save the file on your desktop as "myfirstpage.html".

To open the file in a browser. Navigate to your file then double click on it. It will open in your default Web browser. If it does not, open your browser and drag the file to it.

Explanation of code

TOPS Technologies

You might think what that code was all about. Well, let's find out.

The first line <!DOCTYPE html> is the document type declaration. It instructs the web browser that this document is an HTML5 document. It is case-insensitive.

The <head> element is a container for the tags that provides information about the document, for example, <title> tag defines the title of the document.

The <body> element contains the document's actual content (paragraphs, links, images, tables, and so on) that is rendered in the web browser and displayed to the user.

You will learn about the different HTML elements in detail in the upcoming chapters. For now, just focus on the basic structure of the HTML document.

HTML Tags and Elements

HTML is written in the form of HTML elements consisting of markup tags. These markup tags are the fundamental characteristic of HTML. Every markup tag is composed of a keyword, surrounded by angle brackets, such as <html>, <head>, <body>, <title>, <p>, and so on.

HTML tags normally come in pairs like <html> and </html>. The first tag in a pair is often called the opening tag (or start tag), and the second tag is called the closing tag (or end tag).

An opening tag and a closing tag are identical, except a slash (/) after the opening angle bracket of the closing tag, to tell the browser that the command has been completed.

In between the start and end tags you can place appropriate contents. For example, a paragraph, which is represented by the p element, would be written as:

```
<p>This is a paragraph.</p>
```

```
<!-- Paragraph with nested element -->
```

```
<p>
```

```
    This is <b>another</b> paragraph.
```

```
</p>
```

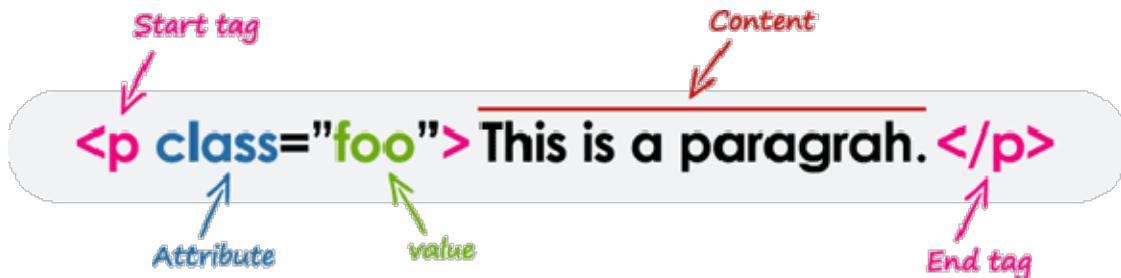
HTML Elements

TOPS Technologies

HTML Element Syntax

An HTML element is an individual component of an HTML document. It represents semantics, or meaning. For example, the `title` element represents the title of the document.

Most HTML elements are written with a *start tag* (or opening tag) and an *end tag* (or closing tag), with content in between. Elements can also contain attributes that defines its additional properties. For example, a paragraph, which is represented by the `p` element, would be written as:



HTML Tags Vs Elements

Technically, an HTML element is the collection of start tag, its attributes, an end tag and everything in between. On the other hand an HTML tag (either opening or closing) is used to mark the start or end of an element, as you can see in the above illustration.

However, in common usage the terms HTML element and HTML tag are interchangeable i.e. a tag is an element is a tag. For simplicity's sake of this website, the terms "tag" and "element" are used to mean the same thing — as it will define something on your web page.

Case Insensitivity in HTML Tags and Attributes

In HTML, tag and attribute names are not case-sensitive (but most attribute values are case-sensitive). It means the tag `<P>`, and the tag `<p>` defines the same thing in HTML which is a paragraph.

But in XHTML they are case-sensitive and the tag `<P>` is different from the tag `<p>`.

`<p>This is a paragraph.</p>`

`<P>This is also a valid paragraph.</P>`

Empty HTML Elements

Empty elements (also called self-closing or void elements) are not container tags — that means, you can not write `<hr>some content</hr>` or `
some content</br>`.

TOPS Technologies

A typical example of an empty element, is the `
` element, which represents a line break. Some other common empty elements are ``, `<input>`, `<link>`, `<meta>`, `<hr>`, etc.

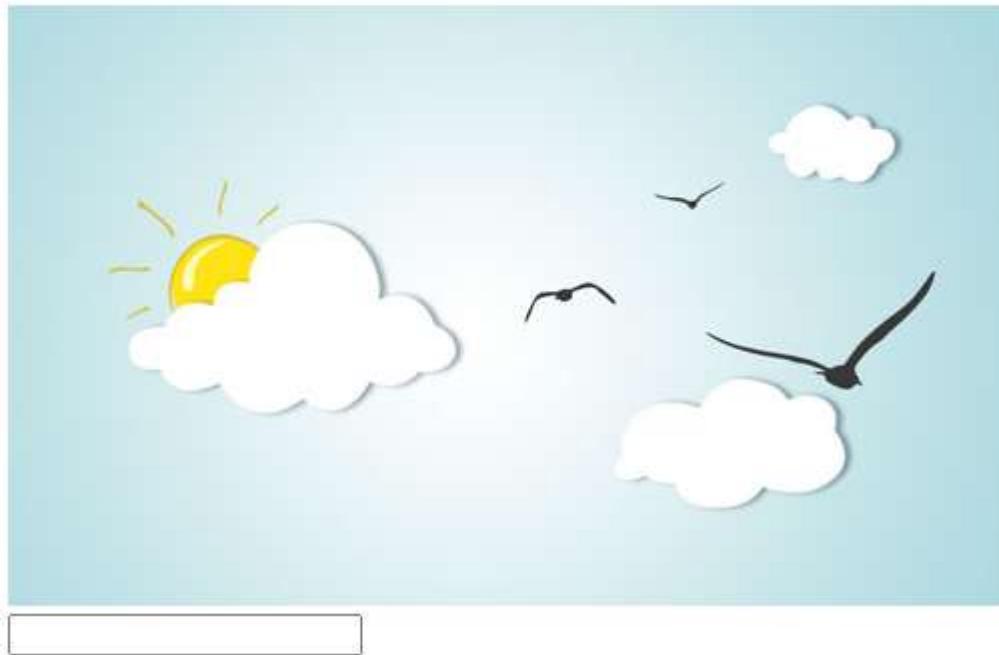
`<p>This paragraph contains
 a line break.</p>`

``

`<input type="text" name="username">`



This paragraph contains
a line break.



Nesting HTML Elements

Most HTML elements can contain any number of further elements (except [empty elements](#)), which are, in turn, made up of tags, attributes, and content or other elements.

The following example shows some elements nested inside the `<p>` element.

`<p>Here is some bold text.</p>`

`<p>Here is some emphasized text.</p>`

TOPS Technologies

<p>Here is some <mark>highlighted</mark> text.</p>

Output

Here is some **bold** text.

Here is some *emphasized* text.

Here is some highlighted text.

HTML tags should be nested in correct order. They must be closed in the inverse order of how they are defined, that means the last tag opened must be closed first

<p>These tags are nested properly.</p>

<p>These tags are not nested properly.</p>

Output

These tags are nested properly.

These tags are not nested properly.

Writing Comments in HTML

Comments are usually added with the purpose of making the source code easier to understand. It may help other developer (or you in the future when you edit the source code) to understand what you were trying to do with the HTML. Comments are not displayed in the browser.

An HTML comment begins with <!--, and ends with -->, as shown in the example below:

<!-- This is an HTML comment -->

<!-- This is a multi-line HTML comment

that spans across more than one line -->

<p>This is a normal piece of text.</p>

You can also comment out part of your HTML code for debugging purpose, as shown here:

<!-- Hiding this image for testing -->

HTML Elements Types

TOPS Technologies

Elements can be placed in two distinct groups: [block level](#) and [inline level](#) elements. The former make up the document's structure, while the latter dress up the contents of a block.

Also, a block element occupies 100% of the available width and it is rendered with a line break before and after. Whereas, an inline element will take up only as much space as it needs.

The most commonly used block-level elements are `<div>`, `<p>`, `<h1>` through `<h6>`, `<form>`, ``, ``, ``, and so on. Whereas, the commonly used inline-level elements are ``, `<a>`, ``, ``, ``, ``, `<i>`, `<code>`, `<input>`, `<button>`, etc.

HTML Styles

Styling HTML Elements

HTML is quite limited when it comes to the presentation of a web page. It was originally designed as a simple way of presenting information. [CSS \(Cascading Style Sheets\)](#) was introduced in December 1996 by the [World Wide Web Consortium \(W3C\)](#) to provide a better way to style HTML elements.

With CSS, it becomes very easy to specify the things like, size and typeface for the fonts, colors for the text and backgrounds, alignment of the text and images, amount of space between the elements, border and outlines for the elements, and lots of other styling properties.

Adding Styles to HTML Elements

Style information can either be attached as a separate document or embedded in the HTML document itself. These are the three methods of implementing styling information to an HTML document.

- **Inline styles** — Using the `style` attribute in the HTML start tag.
- **Embedded style** — Using the `<style>` element in the head section of the document.
- **External style sheet** — Using the `<link>` element, pointing to an external CSS files.

Inline Styles

Inline styles are used to apply the unique style rules to an element, by putting the CSS rules directly into the start tag. It can be attached to an element using the `style` attribute.

The `style` attribute includes a series of CSS property and value pairs. Each property: value pair is separated by a semicolon (;), just as you would write into an embedded or external style sheet. But it needs to be all in one line i.e. no line break after the semicolon.

The following example demonstrates how to set the [color](#) and [font-size](#) of the text:

```
<h1 style="color:red; font-size:30px;">This is a heading</h1>
```

```
<p style="color:green; font-size:18px;">This is a paragraph.</p>
```

TOPS Technologies

```
<div style="color:green; font-size:18px;">This is some text.</div>
```

Output

This is a heading

This is a paragraph.

This is some text.

Using the inline styles are generally considered as a bad practice. Because style rules are embedded directly inside the html tag, it causes the presentation to become mixed with the content of the document, which makes updating or maintaining a website very difficult.

Embedded Style Sheets

Embedded or internal style sheets only affect the document they are embedded in.

Embedded style sheets are defined in the `<head>` section of an HTML document using the `<style>` tag. You can define any number of `<style>` elements inside the `<head>` section.

The following example demonstrates how style rules are embedded inside a web page.

```
<head>
  <style>
    body { background-color: YellowGreen; }

    h1 { color: blue; }

    p { color: red; }

  </style>
</head>
```

Output

This is a heading

This is a paragraph.

External Style Sheets

An external style sheet is ideal when the style is applied to many pages.

An external style sheet holds all the style rules in a separate document that you can link from any HTML document on your site. External style sheets are the most flexible because with an external style sheet, you can change the look of an entire website by updating just one file.

You can attach external style sheets in two ways — *linking* and *importing*:

Linking External Style Sheets

An external style sheet can be linked to an HTML document using the `<link>` tag.

The `<link>` tag goes inside the `<head>` section, as shown here:

```
<head>
```

```
  <link rel="stylesheet" href="css/style.css">
```

```
</head>
```

Linking External Style Sheet

The styles of this HTML document are defined in linked style sheet.

HTML Images

TOPS Technologies

Inserting Images into Web Pages

Images enhance visual appearance of the web pages by making them more interesting and colorful.

The `` tag is used to insert images in the HTML documents. It is an empty element and contains attributes only.

The syntax of the `` tag can be given with:

```

```

The following example inserts three images on the web page:

```



```

Output



Setting the Width and Height of an Image

The `width` and `height` attributes are used to specify the width and height of an image.

The values of these attributes are interpreted in pixels by default.

TOPS Technologies

```



```



You can also use the `style` attribute to specify width and height for the images. It prevents style sheets from changing the image size accidentally, since inline style has the highest priority.

```



```



HTML Tables

Creating Tables in HTML

HTML table allows you to arrange data into rows and columns. They are commonly used to display tabular data like product listings, customer's details, financial reports, and so on.

You can create a table using the `<table>` element. Inside the `<table>` element, you can use the `<tr>` elements to create rows, and to create columns inside a row you can use the `<td>` elements. You can also define a cell as a header for a group of table cells using the `<th>` element.

The following example demonstrates the most basic structure of a table.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Creating Tables in HTML</title>
</head>
<body>
  <h2>HTML Table (Default Style)</h2>
  <table>
    <tr>
      <th>No.</th>
```

TOPS Technologies

```
<th>Name</th>
<th>Age</th>
</tr>
<tr>
    <td>1</td>
    <td>Peter Parker</td>
    <td>16</td>
</tr>
<tr>
    <td>2</td>
    <td>Clark Kent</td>
    <td>34</td>
</tr>
<tr>
    <td>3</td>
    <td>Harry Potter</td>
    <td>11</td>
</tr>
</table>
</body>
</html>
```

Output

HTML Table (Default Style)

No.	Name	Age
1	Peter Parker	16
2	Clark Kent	34
3	Harry Potter	11

Tables do not have any borders by default. You can use the CSS [border](#) property to add borders to the tables. Also, table cells are sized just large enough to fit the contents by default. To add more space around the content in the table cells you can use the CSS [padding](#) property.

The following style rules add a 1-pixel border to the table and 10-pixels of padding to its cells.

```
table, th, td {
    border: 1px solid black;
}
th, td {
    padding: 10px;
}
```

TOPS Technologies

Table with Separated Borders

No.	Name	Age
1	Peter Parker	16
2	Clark Kent	34
3	Harry Potter	11

By default, borders around the table and their cells are separated from each other. But you can collapse them into one by using the [border-collapse](#) property on the `<table>` element.

Also, text inside the `<th>` elements are displayed in bold font, aligned horizontally center in the cell by default. To change the default alignment you can use the CSS [text-align](#) property.

The following style rules collapse the table borders and align the table header text to left.

```
table {  
    border-collapse: collapse;  
}  
  
th {  
    text-align: left;  
}
```

Table with Collapsed Borders

No.	Name	Age
1	Peter Parker	16
2	Clark Kent	34
3	Harry Potter	11

Spanning Multiple Rows and Columns

Spanning allow you to extend table rows and columns across multiple other rows and columns.

Normally, a table cell cannot pass over into the space below or above another table cell. But, you can use the `rowspan` or `colspan` attributes to span multiple rows or columns in a table.

Let's try out the following example to understand how `colspan` basically works:

```
<table>  
    <tr>  
        <th>Name</th>  
        <th colspan="2">Phone</th>  
    </tr>  
    <tr>  
        <td>John Carter</td>  
        <td>5550192</td>
```

TOPS Technologies

```
<td>5550152</td>
</tr>
</table>
```

Spanning Columns

Name	Phone	
John Carter	5550192	5550152

Similarly, you can use the `rowspan` attribute to create a cell that spans more than one row. Let's try out an example to understand how row spanning basically works:

```
<table>
<tr>
<th>Name:</th>
<td>John Carter</td>
</tr>
<tr>
<th rowspan="2">Phone:</th>
<td>55577854</td>
</tr>
<tr>
<td>55577855</td>
</tr>
</table>
```

Spanning Rows

Name:	John Carter
Phone:	55577854
	55577855

HTML Lists

Working with HTML Lists

HTML lists are used to present list of information in well formed and semantic way. There are three different types of list in HTML and each one has a specific purpose and meaning.

- **Unordered list** — Used to create a list of related items, in no particular order.
- **Ordered list** — Used to create a list of related items, in a specific order.
- **Description list** — Used to create a list of terms and their descriptions.

In the following sections we will cover all the three types of list one by one:

HTML Unordered Lists

TOPS Technologies

An unordered list created using the `` element, and each list item starts with the `` element.

The list items in unordered lists are marked with bullets. Here's an example:

```
<ul>
  <li>Chocolate Cake</li>
  <li>Black Forest Cake</li>
  <li>Pineapple Cake</li>
</ul>
```

— The output of the above example will look something like this:

- Chocolate Cake
- Black Forest Cake
- Pineapple Cake

You can also change the bullet type in your unordered list using the CSS `list-style-type` property. The following style rule changes the type of bullet from the default `disc` to `square`

```
ul {
  list-style-type: square;
}
```

HTML Ordered Lists

An ordered list created using the `` element, and each list item starts with the `` element. Ordered lists are used when the order of the list's items is important.

The list items in an ordered list are marked with numbers. Here's an example:

```
<ol>
  <li>Fasten your seatbelt</li>
  <li>Starts the car's engine</li>
  <li>Look around and go</li>
</ol>
```

— The output of the above example will look something like this:

1. Fasten your seatbelt
2. Starts the car's engine
3. Look around and go

The numbering of items in an ordered list typically starts with 1. However, if you want to change that you can use the `start` attribute, as shown in the following example:

```
<ol start="10">
  <li>Mix ingredients</li>
  <li>Bake in oven for an hour</li>
  <li>Allow to stand for ten minutes</li>
</ol>
```

TOPS Technologies

The output of the above example will look something like this:

10. Mix ingredients
11. Bake in oven for an hour
12. Allow to stand for ten minutes

HTML Forms

What is HTML Form

HTML Forms are required to collect different kinds of user inputs, such as contact details like name, email address, phone numbers, or details like credit card information, etc.

Forms contain special elements called controls like inputbox, checkboxes, radio-buttons, submit buttons, etc. Users generally complete a form by modifying its controls e.g. entering text, selecting items, etc. and submitting this form to a web server for further processing.

The `<form>` tag is used to create an HTML form. Here's a simple example of a login form:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Simple HTML Form</title>
</head>
<body>
    <form action="/examples/actions/confirmation.php" method="post">
        <label>Username: <input type="text" name="username"></label>
        <label>Password: <input type="password" name="userpass"></label>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Output

Username: Password:

Input Element

This is the most commonly used element within HTML forms.

It allows you to specify various types of user input fields, depending on the type attribute. An input element can be of type *text field*, *password field*, *checkbox*, *radio button*, *submit button*, *reset button*, *file select box*, as well as several [new input types](#) introduced in HTML5.

The most frequently used input types are described below.

Text Fields

Text fields are one line areas that allow the user to input text.

Single-line text input controls are created using an `<input>` element, whose type attribute has a value of `text`. Here's an example of a single-line text input used to take username:

TOPS Technologies

```
<form>
  <label for="username">Username:</label>
  <input type="text" name="username" id="username">
</form>
```

— The output of the above example will look something like this:

Username:

The `<label>` tag is used to define the labels for `<input>` elements. If you want your user to enter several lines you should use a `<textarea>` instead.

Password Field

Password fields are similar to text fields. The only difference is; characters in a password field are masked, i.e. they are shown as asterisks or dots. This is to prevent someone else from reading the password on the screen. This is also a single-line text input controls created using an `<input>` element whose type attribute has a value of `password`. Here's an example of a single-line password input used to take user password:

```
<form>
  <label for="user-pwd">Password:</label>
  <input type="password" name="user-password" id="user-pwd">
</form>
```

— The output of the above example will look something like this:

Password:

Radio Buttons

Radio buttons are used to let the user select exactly one option from a pre-defined set of options. It is created using an `<input>` element whose type attribute has a value of `radio`.

Here's an example of radio buttons that can be used to collect user's gender information:

```
<form>
  <input type="radio" name="gender" id="male">
  <label for="male">Male</label>
  <input type="radio" name="gender" id="female">
  <label for="female">Female</label>
</form>
```

— The output of the above example will look something like this:

Male Female

Checkboxes

Checkboxes allows the user to select one or more option from a pre-defined set of options. It is created using an `<input>` element whose type attribute has a value of `checkbox`.

Here's an example of checkboxes that can be used to collect information about user's hobbies:

```
<form>
  <input type="checkbox" name="sports" id="soccer">
  <label for="soccer">Soccer</label>
  <input type="checkbox" name="sports" id="cricket">
```

TOPS Technologies

```
<label for="cricket">Cricket</label>
<input type="checkbox" name="sports" id="baseball">
<label for="baseball">Baseball</label>
</form>
```

— The output of the above example will look something like this:

Soccer Cricket Baseball

File Select box

The file fields allow a user to browse for a local file and send it as an attachment with the form data. Web browsers such as Google Chrome and Firefox render a file select input field with a Browse button that enables the user to navigate the local hard drive and select a file.

This is also created using an `<input>` element, whose `type` attribute value is set to `file`.

```
<form>
  <label for="file-select">Upload:</label>
  <input type="file" name="upload" id="file-select">
</form>
```

— The output of the above example will look something like this:

Upload: No file chosen

Textarea

Textarea is a multiple-line text input control that allows a user to enter more than one line of text. Multi-line text input controls are created using an `<textarea>` element.

```
<form>
  <label for="address">Address:</label>
  <textarea rows="3" cols="30" name="address" id="address"></textarea>
</form>
```

— The output of the above example will look something like this:

Address:

Select Boxes

A select box is a dropdown list of options that allows user to select one or more option from a pull-down list of options. Select box is created using the `<select>` element and `<option>` element.

The `<option>` elements within the `<select>` element define each list item.

Select Boxes

A select box is a dropdown list of options that allows user to select one or more option from a pull-down list of options. Select box is created using the `<select>` element and `<option>` element.

The `<option>` elements within the `<select>` element define each list item.

```
<form>
  <label for="city">City:</label>
  <select name="city" id="city">
    <option value="sydney">Sydney</option>
```

TOPS Technologies

```
<option value="melbourne">Melbourne</option>
<option value="cromwell">Cromwell</option>
</select>
</form>
```

— The output of the above example will look something like this:

City: ▾

Submit and Reset Buttons

A submit button is used to send the form data to a web server. When submit button is clicked the form data is sent to the file specified in the form's `action` attribute to process the submitted data.

A reset button resets all the forms control to default values. Try out the following example by typing your name in the text field, and click on submit button to see it in action.

```
<form action="action.php" method="post">
  <label for="first-name">First Name:</label>
  <input type="text" name="first-name" id="first-name">
  <input type="submit" value="Submit">
  <input type="reset" value="Reset">
</form>
```

First Name:

Type your name in the text field above, and click on submit button to see it in action.

Grouping Form Controls

You also group logically related controls and labels within a web form using the `<legend>` element. Grouping form controls into categories makes it easier for users to locate a control which makes the form more user-friendly. Let's try out the following example to see how it works:

```
<form>
  <fieldset>
    <legend>Contact Details</legend>
    <label>Email Address: <input type="email" name="email"></label>
    <label>Phone Number: <input type="text" name="phone"></label>
  </fieldset>
</form>
```

TOPS Technologies

Name			
Firstname:	<input type="text"/>	Lastname:	<input type="text"/>
Gender			
<input type="radio"/> Male <input type="radio"/> Female			
Hobbies			
<input type="checkbox"/> Soccer <input type="checkbox"/> Cricket <input type="checkbox"/> Baseball			
Contact Details			
Email Address:	<input type="text"/>	Phone Number:	<input type="text"/>

Frequently Used Form Attributes

The following table lists the most frequently used form element's attributes:

Attribute	Description
name	Specifies the name of the form.
action	Specifies the URL of the program or script on the web server that will be used for processing the information submitted via form.
method	Specifies the HTTP method used for sending the data to the web server by the browser. The value can be either <code>get</code> (the default) and <code>post</code> .
target	Specifies where to display the response that is received after submitting the form. Possible values are <code>_blank</code> , <code>_self</code> , <code>_parent</code> and <code>_top</code> .
enctype	Specifies how the form data should be encoded when submitting the form to the server. Applicable only when the value of the <code>method</code> attribute is <code>post</code> .

Module – 3 [CSS & CSS3]

TOPS Technologies

CSS Border

CSS Border Properties

The CSS border properties allow you to define the border area of an element's box.

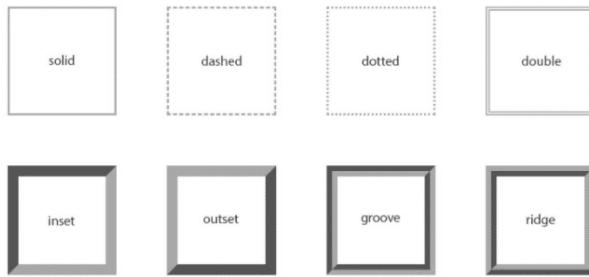
Borders appear directly between the margin and padding of an element. The border can either be a predefined style like, solid line, dotted line, double line, etc. or [an image](#).

The following section describes how to set the style, color, and width of the border.

Understanding the Different Border Styles

The [border-style](#) property sets the style of a box's border such as: solid, dotted, etc. It is a shorthand property for setting the line style for all four sides of the elements border.

The [border-style](#) property can have the following values: none, hidden, solid, dashed, dotted, double, inset, outset, groove, and ridge. Now, let's take a look at the following illustration, it gives you a sense of the differences between the border style types.



The values `none` and `hidden` displays no border, however, there is a slight difference between these two values. In the case of table cell and border collapsing, the `none` value has the *lowest* priority, whereas the `hidden` value has the *highest* priority, if any other conflicting border is set.

The values `inset`, `outset`, `groove`, and `ridge` creates a 3D like effect which essentially depends on the `border-color` value. This is typically achieved by creating a "shadow" from two colors that are slightly lighter and darker than the border color. Let's check out an example:

```
h1 {  
    border-style: dotted;  
}  
  
p {  
    border-style: ridge;  
}
```

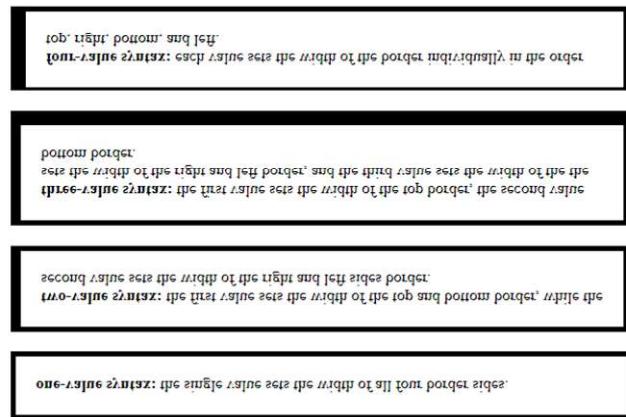
TOPS Technologies

Setting the Border Width

The [border-width](#) property specifies the width of the border area. It is a shorthand property for setting the thickness of all the four sides of an element's border at the same time.

Let's try out the following example to understand how it works:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>CSS border-width Property</title>
<style>
p {
    border-style: solid;
    padding: 20px;
    margin: 20px;
}
p.one {
    border-width: 5px;
}
p.two {
    border-width: 5px 10px;
}
p.three {
    border-width: 5px 10px 15px;
}
p.four {
    border-width: medium 10px thick 15px;
}
</style>
</head>
<body>
    <p class="one"><strong>one-value syntax:</strong> the single value sets the width of all four border sides.</p>
    <p class="two"><strong>two-value syntax:</strong> the first value sets the width of the top and bottom border, while the second value sets the width of the right and left sides border.</p>
    <p class="three"><strong>three-value syntax:</strong> the first value sets the width of the top border, the second value sets the width of the right and left border, and the third value sets the width of the the bottom border.</p>
    <p class="four"><strong>four-value syntax:</strong> each value sets the width of the border individually in the order top, right, bottom, and left.</p>
</body>
</html>
```



CSS Box Shadow

CSS box-shadow Property

The CSS **box-shadow** property applies shadow to elements.

In its simplest use, you only specify the horizontal shadow and the vertical shadow

This is a yellow <div> element
with a black box-shadow

Code

```
<!DOCTYPE html>

<html>
<head>
<style>
div {
```

width: 300px;

height: 100px;

padding: 15px;

```
background-color: yellow;  
box-shadow: 10px 10px;  
}  
</style>  
</head>  
<body>
```

```
<h1>The box-shadow Property</h1>
```

```
<div>This is a div element with a box-shadow</div>
```

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

CSS Multiple Backgrounds

CSS allows you to add multiple background images for an element, through the **background-image** property.

The different background images are separated by commas, and the images are stacked on top of each other, where the first image is closest to the viewer.

The following example has two background images, the first image is a flower (aligned to the bottom and right) and the second image is a paper background (aligned to the top-left corner):

```
<!DOCTYPE html>  
<html>  
<head>  
<style>  
#example1 {  
background-image: url(img_flwr.gif), url(paper.gif);
```

TOPS Technologies

```
background-position: right bottom, left top;  
background-repeat: no-repeat, repeat;  
padding: 15px;  
}  
</style>  
</head>  
<body>
```

<h1>Multiple Backgrounds</h1>

<p>The following div element has two background images:</p>

```
<div id="example1">  
  <h1>Lorem Ipsum Dolor</h1>  
  <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut  
  laoreet dolore magna aliquam erat volutpat.</p>  
  <p>Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea  
  commodo consequat.</p>  
</div>  
  
</body>  
</html>
```

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.



CSS Text Shadow

The CSS `text-shadow` property applies shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

Text shadow effect!

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<style>
```

```
h1 {
```

```
    text-shadow: 2px 2px;
```

```
}
```

```
</style>
```

```
</head>
```

TOPS Technologies

```
<body>

<h1>Text-shadow effect!</h1></body>

</html>
add a color to the shadow:
h1 {
    text-shadow: 2px 2px red;
}

add a blur effect to the shadow:
text-shadow: 2px 2px 5px red;
white text with black shadow:
h1 {
    color: white;
    text-shadow: 2px 2px 4px #000000;
}

red neon glow shadow:
h1 {
    text-shadow: 0 0 3px #FF0000;
}
```

Module – 3 [Learning The Language]

PHP

PHP is the most popular server-side scripting language for creating dynamic web pages.



PHP stands for Hypertext Preprocessor. PHP is a very popular and widely-used open source server-side scripting language to write dynamically generated web pages. PHP was originally created by Rasmus Lerdorf in 1994. It was initially known as Personal Home Page.

PHP scripts are executed on the server and the result is sent to the web browser as plain HTML. PHP can be integrated with the number of popular databases, including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, Sybase, and so on. The current major version of PHP is 7. All of the code in this tutorial has been tested and validated against the most recent release of PHP 7.

PHP is very powerful language yet easy to learn and use. So bookmark this website and continued on.

What You Can Do with PHP

There are lot more things you can do with PHP.

- You can generate pages and files dynamically.
- You can create, open, read, write and close files on the server.
- You can collect data from a web form such as user information, email, phone no, etc.
- You can send emails to the users of your website.
- You can send and receive cookies to track the visitor of your website.
- You can store, delete, and modify information in your database.
- You can restrict unauthorized access to your website.
- You can encrypt data for safe transmission over internet.

The list does not end here, there are many other interesting things that you can do with PHP. You will learn about all of them in detail in upcoming chapters.

Advantages of PHP over Other Languages

If you're familiar with other server-side languages like ASP.NET or Java, you might be wondering what makes PHP so special. There are several advantages why one should choose PHP.

TOPS Technologies

- **Easy to learn:** PHP is easy to learn and use. For beginner programmers who just started out in web development, PHP is often considered as the preferable choice of language to learn.
- **Open source:** PHP is an open-source project. It is developed and maintained by a worldwide community of developers who make its source code freely available to download and use.
- **Portability:** PHP runs on various platforms such as Microsoft Windows, Linux, Mac OS, etc. and it is compatible with almost all servers used today such Apache, IIS, etc.
- **Fast Performance:** Scripts written in PHP usually execute or runs faster than those written in other scripting languages like ASP, Ruby, Python, Java, etc.
- **Vast Community:** Since PHP is supported by the worldwide community, finding help or documentation related to PHP online is extremely easy.

Getting Started

Install Wampserver or XAMPP on your PC to quickly create web applications with Apache, PHP and a MySQL database.

Getting Started with PHP

Here, you will learn how easy it is to create dynamic web pages using PHP. Before begin, be sure to have a code editor and some working knowledge of HTML and CSS.

If you're just starting out in web development

Setting Up a Local Web Server

PHP script execute on a web server running PHP. So before you start writing any PHP program you need the following program installed on your computer.

- The Apache Web server
- The PHP engine
- The MySQL database server

You can either install them individually or choose a pre-configured package for your operating system like Linux and Windows. Popular pre-configured package are XAMPP and WampServer.

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. It will also provide the MySQL administrative tool PhpMyAdmin to easily manage your databases using a web browser.

The official website for downloading and installation instructions for the WampServer:
<http://www.wampserver.com/en/>

Creating Your First PHP Script

Now that you have successfully installed WampServer on your computer. In this section we will create a very simple PHP script that displays the text "Hello, world!" in the browser window.

Ok, click on the WampServer icon somewhere on your Windows task bar and select the "www directory". Alternatively, you can access the "www" directory through navigating the C:\wamp\www. Create a subdirectory in "www" directory let's say "project".

Now open up your favorite code editor and create a new PHP file then type the following code:

```
<?php  
// Display greeting message  
echo "Hello, world!";  
?>
```

Now save this file as "hello.php" in your project folder (located at C:\wamp\www\project), and view the result in your browser through visiting this URL: <http://localhost/project/hello.php>.

Alternatively, you can access the "hello.php" file through selecting the localhost option and then select the project folder from the WampSever menu on the taskbar.

PHP can be embedded within a normal HTML web page. That means inside your HTML document you can write the PHP statements, as demonstrated in the follwoing example:

```
<!DOCTYPE HTML>  
<html>  
<head>  
    <title>PHP Application</title>  
</head>  
<body>  
<?php  
// Display greeting message  
echo 'Hello World!';  
?>  
</body>  
</html>
```

PHP Syntax

The PHP script can be embedded within HTML web pages.

TOPS Technologies

Standard PHP Syntax

A PHP script starts with the `<?php` and ends with the `?>` tag.

The PHP delimiter `<?php` and `?>` in the following example simply tells the PHP engine to treat the enclosed code block as PHP code, rather than simple HTML.

```
<?php  
// Some code to be executed  
echo "Hello, world!";  
?>
```

Every PHP statement end with a semicolon (`;`) — this tells the PHP engine that the end of the current statement has been reached.

Embedding PHP within HTML

PHP files are plain text files with `.php` extension. Inside a PHP file you can write HTML like you do in regular HTML pages as well as embed PHP codes for server side execution.

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>A Simple PHP File</title>  
</head>  
<body>  
    <h1><?php echo "Hello, world!"; ?></h1>  
</body>  
</html>
```

The above example shows how you can embed PHP codes within HTML to create well-formed dynamic web pages. If you view the source code of the resulting web page in your browser, the only difference you will see is the PHP code `<?php echo "Hello, world!"; ?>` has been replaced with the output "Hello, world!".

What happen here is? when you run this code the PHP engine exacuted the instructions between the `<?php ... ?>` tags and leave rest of the thing as it is. At the end the web server send the final output back to your browser which is completely in HTML.

PHP Comments

A comment is simply text that is ignored by the PHP engine. The purpose of comments is to make the code more readable. It may help other developer (or you in the future when you edit the source code) to understand what you were trying to do with the PHP.

TOPS Technologies

PHP support single-line as well as multi-line comments. To write a single-line comment either start the line with either two slashes (//) or a hash symbol (#). For example:

```
<?php  
// This is a single line comment  
# This is also a single line comment  
echo "Hello, world!";  
?>
```

However to write multi-line comments, start the comment with a slash followed by an asterisk /*) and end the comment with an asterisk followed by a slash (*/), like this:

```
<?php  
/*  
This is a multiple line comment block  
that spans across more than  
one line  
*/  
echo "Hello, world!";  
?>
```

Case Sensitivity in PHP

Variable names in PHP are case-sensitive. As a result the variables \$color, \$Color and \$COLOR are treated as three different variables.

```
<?php  
// Assign value to variable  
$color = "blue";  
  
// Try to print variable value  
echo "The color of the sky is " . $color . "<br>";  
echo "The color of the sky is " . $Color . "<br>";  
echo "The color of the sky is " . $COLOR . "<br>";  
?>
```

If you try to run the above example code it will only display the value of the variable \$color and produce the "Undefined variable" warning for the variable \$Color and \$COLOR.

However the keywords, function and classes names are case-insensitive. As a result calling the `gettype()` or `GETTYPE()` produce the same result.

```
<?php  
// Assign value to variable
```

```
$color = "blue";  
  
// Get the type of a variable  
echo gettype($color) . "<br>";  
echo GETTYPE($color) . "<br>";  
?>
```

Output:

```
string  
String
```

If you try to run the above example code both the functions `gettype()` and `GETTYPE()` gives the same output, which is: string.

PHP Variables

What is Variable in PHP

Variables are used to store data, like string of text, numbers, etc. Variable values can change over the course of a script. Here're some important things to know about variables:

- In PHP, a variable does not need to be declared before adding a value to it. PHP automatically converts the variable to the correct data type, depending on its value.
- After declaring a variable it can be reused throughout the code.
- The assignment operator (=) used to assign value to a variable.

In PHP variable can be declared as: `$var_name = value;`

```
<?php  
  
// Declaring variables  
$txt = "Hello World!";  
$number = 10;  
  
// Displaying variables value  
echo $txt; // Output: Hello World!  
echo $number; // Output: 10  
?>
```

Output

```
← → ⌂ ⓘ localhost/php_handbook/  
Hello World!10
```

In the above example we have created two variables where first one has assigned with a string value and the second has assigned with a number. Later we've displayed the variable values in the browser using the `echo` statement. The PHP [echo statement](#) is often used to output data to the browser.

Naming Conventions for PHP Variables

These are the following rules for naming a PHP variable:

- All variables in PHP start with a \$ 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 in PHP can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- A variable name cannot contain spaces.

PHP Constants

What is Constant in PHP

A constant is a name or an identifier for a fixed value. Constant are like variables, except that once they are defined, they cannot be undefined or changed (except [magic constants](#)).

Constants are very useful for storing data that doesn't change while the script is running. Common examples of such data include configuration settings such as database username and password, website's base URL, company name, etc.

Constants are defined using PHP's `define()` function, which accepts two arguments: the name of the constant, and its value. Once defined the constant value can be accessed at any time just by referring to its name. Here is a simple example:

Output

TOPS Technologies

← → ⌂ ⓘ localhost/php_handbook/

Thank you for visiting - <https://www.tutorialrepublic.com/>

The PHP echo statement is often used to display or output data to the web browser. We will learn more about this statement in the next chapter.

Naming Conventions for PHP Constants

Name of constants must follow the same rules as [variable names](#), which means a valid constant name must starts with a letter or underscore, followed by any number of letters, numbers or underscores with one exception: the \$ prefix is not required for constant names.

PHP Echo and Print Statements

The PHP echo Statement

The echo statement can output one or more strings. In general terms, the echo statement can display anything that can be displayed to the browser, such as string, numbers, variables values, the results of expressions etc.

Since echo is a language construct not actually a function (like [if](#) statement), you can use it without parentheses e.g. echo or echo(). However, if you want to pass more than one parameter to echo, the parameters must not be enclosed within parentheses.

Display Strings of Text

The following example will show you how to display a string of text with the echo statement:

```
<?php  
// Displaying string of text  
echo "Hello World!";  
?>
```

The output of the above PHP code will look something like this:

Hello World!

Display HTML Code

The following example will show you how to display HTML code using the echo statement:

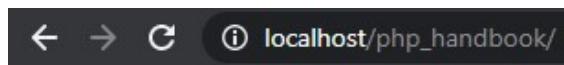
```
<?php  
// Displaying HTML code  
echo "<h4>This is a simple heading.</h4>";
```

TOPS Technologies

```
echo "<h4 style='color: red;'>This is heading with style.</h4>";
```

```
?>
```

```
Output
```



This is a simple heading.

This is heading with style.

Display Variables

The following example will show you how to display variable using the echo statement:

```
<?php  
// Defining variables  
$txt = "Hello World!";  
$num = 123456789;  
$colors = array("Red", "Green", "Blue");  
  
// Displaying variables  
echo $txt;  
echo "<br>";  
echo $num;  
echo "<br>";  
echo $colors[0];  
?>
```

The output of the above PHP code will look something like this:

Hello World!

123456789

Red

The PHP print Statement

You can also use the print statement (an alternative to echo) to display output to the browser. Like echo the print is also a language construct not a real function. So you can also use it without parentheses like: print or print().

Both echo and print statement works exactly the same way except that the print statement can only output one string, and always returns 1. That's why the echo statement considered marginally faster than the print statement since it doesn't return any value.

TOPS Technologies

Display Strings of Text

The following example will show you how to display a string of text with the print statement:

```
<?php  
// Displaying string of text  
print "Hello World!";  
?>
```

The output of the above PHP code will look something like this:

Hello World!

Display HTML Code

The following example will show you how to display HTML code using the print statement:

```
<?php  
// Displaying HTML code  
print "<h4>This is a simple heading.</h4>";  
print "<h4 style='color: red;'>This is heading with style.</h4>";  
?>
```

The output of the above PHP code will look something like this:

This is a simple heading.

This is heading with style.

Display Variables

The following example will show you how to display variable using the print statement:

```
<?php  
// Defining variables  
$txt = "Hello World!";  
$num = 123456789;  
$colors = array("Red", "Green", "Blue");  
// Displaying variables  
print $txt;  
print "<br>";  
print $num;  
print "<br>";  
print $colors[0];  
?>
```

The output of the above PHP code will look something like this:

Hello World!

PHP Data Types

Data Types in PHP

The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects.

PHP supports total eight primitive data types: Integer, Floating point number or Float, String, Booleans, Array, Object, resource and NULL. These data types are used to construct variables. Now let's discuss each one of them in detail.

PHP Integers

Integers are whole numbers, without a decimal point (...,-2,-1,0,1,2,...). Integers can be specified in decimal (base 10), hexadecimal (base 16 - prefixed with 0x) or octal (base 8 - prefixed with 0) notation, optionally preceded by a sign (- or +).

```
<?php  
  
$a = 123; // decimal number  
  
var_dump($a);  
  
echo "<br>";  
  
$b = -123; // a negative number  
  
var_dump($b);  
  
echo "<br>";  
  
$c = 0x1A; // hexadecimal number  
  
var_dump($c);  
  
echo "<br>";  
  
$d = 0123; // octal number  
  
var_dump($d);  
  
?>
```

TOPS Technologies

Output
int(123)
int(-123)
int(26)
int(83)

Note: Since PHP 5.4+ you can also specify integers in binary (base 2) notation. To use binary notation precede the number with 0b (e.g. \$var = 0b11111111;).

PHP Strings

Strings are sequences of characters, where every character is the same as a byte.

A string can hold letters, numbers, and special characters and it can be as large as up to 2GB (2147483647 bytes maximum). The simplest way to specify a string is to enclose it in single quotes (e.g. 'Hello world!'), however you can also use double quotes ("Hello world!").

```
<?php  
  
$a = 'Hello world!';  
  
echo $a;  
  
echo "<br>";  
  
$b = "Hello world!";  
  
echo $b;  
  
echo "<br>";  
  
$c = 'Stay here, I\'ll be back.';  
  
echo $c;  
  
?>  
Output  
Helloworld!  
Hello world!  
Stay here, I'll be back.
```

PHP Floating Point Numbers or Doubles

TOPS Technologies

Floating point numbers (also known as "floats", "doubles", or "real numbers") are decimal or fractional numbers, like demonstrated in the example below.

```
<?php  
  
$a = 1.234;  
  
var_dump($a);  
  
echo "<br>";  
  
  
  
$b = 10.2e3;  
  
var_dump($b);  
  
echo "<br>";  
  
$c = 4E-10;  
  
var_dump($c);  
  
?>  
Output  
float(1.234)  
float(10200)  
float(4.0E-10)
```

PHP Booleans

Booleans are like a switch it has only two possible values either 1 (true) or 0 (false).

```
<?php  
  
// Assign the value TRUE to a variable  
  
$show_error = true;  
  
var_dump($show_error);  
  
?>  
Output
```

bool(true)

PHP Arrays

An array is a variable that can hold more than one value at a time. It is useful to aggregate a series of related items together, for example a set of country or city names.

An array is formally defined as an indexed collection of data values. Each index (also known as the key) of an array is unique and references a corresponding value.

```
<?php

$colors = array("Red", "Green", "Blue");

var_dump($colors);

echo "<br>";

$color_codes = array( "Red" => "#ff0000",

"Green" => "#00ff00",

"Blue" => "#0000ff"

);

var_dump($color_codes);

?>

Output

array(3) { [0]=> string(3) "Red" [1]=> string(5) "Green" [2]=> string(4) "Blue" }
array(3) { ["Red"]=> string(7) "#ff0000" ["Green"]=> string(7) "#00ff00" ["Blue"]=> string(7) "#0000ff" }
```

PHP Objects

An object is a data type that not only allows storing data but also information on, how to process that data. An object is a specific instance of a class which serve as templates for objects. Objects are created based on this template via the new keyword.

Every object has properties and methods corresponding to those of its parent class. Every object instance is completely independent, with its own properties and methods, and can thus be manipulated independently of other objects of the same class.

Here's a simple example of a class definition followed by the object creation.

TOPS Technologies

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>PHP Objects</title>

</head>

<body>

<?php

// Class definition

class greeting{

    // properties

    public $str = "Hello World!";


    // methods

    function show_greeting(){

        return $this->str;

    }

}

// Create object from class

$message = new greeting;

var_dump($message);

?>
```

```
</body>

</html>
Output
object(greeting)#1 (1) { ["str"]=> string(12) "Hello World!" }
```

PHP NULL

The special NULL value is used to represent empty variables in PHP. A variable of type NULL is a variable without any data. NULL is the only possible value of type null.

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>PHP NULL Value</title>

</head>

<body>

<?php

$a = NULL;

var_dump($a);

echo "<br>";

$b = "Hello World!";

$b = NULL;

var_dump($b);

?>

</body>

</html>
Output
```

TOPS Technologies

NULL

NULL

When a variable is created without a value in PHP like \$var; it is automatically assigned a value of null. Many novice PHP developers mistakenly considered both \$var1 = NULL; and \$var2 = ""; are same, but this is not true. Both variables are different — the \$var1 has null value while \$var2 indicates no value assigned to it.

PHP Resources

A resource is a special variable, holding a reference to an external resource.

Resource variables typically hold special handlers to opened files and database connections.

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>PHP Resources</title>

</head>

<body>

<?php

// Open a file for reading

$handle = fopen("note.txt", "r");

var_dump($handle);

?>

</body>

</html>
```

Output
resource(2) of type (stream)

PHP Strings

What is String in PHP

TOPS Technologies

A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all. The simplest way to create a string is to enclose the string literal (i.e. string characters) in single quotation marks ('), like this:

```
$my_string = 'Hello World';
```

You can also use double quotation marks ("). However, single and double quotation marks work in different ways.

Strings enclosed in single-quotes are treated almost literally, whereas the strings delimited by the double quotes replaces variables with the string representations of their values as well as specially interpreting certain escape sequences.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

Here's an example to clarify the differences between single and double quoted strings:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <title>PHP Single and Double Quoted Strings</title>

</head>

<body>

    <?php

        $my_str = 'World';

        echo "Hello, $my_str!<br>";

        echo 'Hello, $my_str!<br>';

        echo '<pre>Hello\tWorld!</pre>';

    </?php>
```

TOPS Technologies

```
echo "<pre>Hello\\tWorld!</pre>";
```

```
echo 'I\\'ll be back';
```

```
?>
```

```
</body>
```

```
</html>
```

```
Output
```

```
Hello, World!
```

```
Hello, $my_str!
```

```
Hello\\tWorld!
```

```
Hello \\t World!
```

```
I\\'ll be back
```

Manipulating PHP Strings

PHP provides many built-in functions for manipulating strings like calculating the length of a string, find substrings or characters, replacing part of a string with different characters, take a string apart, and many others. Here are the examples of some of these functions.

Calculating the Length of a String

The `strlen()` function is used to calculate the number of characters inside a string. It also includes the blank spaces inside the string.

```
<?php
```

```
$my_str = 'Welcome to Tutorial Republic';
```

```
// Calculating and displaying string length
```

```
echo strlen($my_str);
```

```
?>
```

```
Output
```

```
28
```

Counting Number of Words in a String

The `str_word_count()` function counts the number of words in a string.

```
<?php

$my_str = 'The quick brown fox jumps over the lazy dog.';

// Calculating and displaying number of words

echo str_word_count($my_str);

?>

Output

9
```

Replacing Text within Strings

The `str_replace()` replaces all occurrences of the search text within the target string.

```
<?php

$my_str = 'If the facts do not fit the theory, change the facts.';

// Display replaced string

echo str_replace("facts", "truth", $my_str);

?>

Output

If the truth do not fit the theory, change the truth.
```

You can optionally pass the fourth argument to the `str_replace()` function to know how many times the string replacements was performed, like this.

```
<?php

$my_str = 'If the facts do not fit the theory, change the facts.';

// Perform string replacement

str_replace("facts", "truth", $my_str, $count);

// Display number of replacements performed

echo "The text was replaced $count times.;"
```

TOPS Technologies

?>

The output of the above code will be:

The text was replaced 2 times.

Reversing a String

The `strrev()` function reverses a string.

```
<?php  
  
$my_str = 'You can do anything, but not everything.';  
  
// Display reversed string  
  
echo strrev($my_str);  
  
?>
```

The output of the above code will be:

.gnihtyreve ton tub ,gnihtyna od nac uoY

PHP Operators

What is Operators in PHP

Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than (>) symbol is an operator that tells PHP to compare two values.

The following lists describe the different operators used in PHP.

PHP Arithmetic Operators

The arithmetic operators are used to perform common arithmetical operations, such as addition, subtraction, multiplication etc. Here's a complete list of PHP's arithmetic operators:

Operator	Description	Example	Result
+	Addition	<code>\$x + \$y</code>	Sum of \$x and \$y
-	Subtraction	<code>\$x - \$y</code>	Difference of \$x and \$y.
*	Multiplication	<code>\$x * \$y</code>	Product of \$x and \$y.

TOPS Technologies

/	Division	\$x / \$y	Quotient of \$x and \$y
%	Modulus	\$x % \$y	Remainder of \$x divided by \$y

The following example will show you these arithmetic operators in action:

```
<?php  
  
$x = 10;  
  
$y = 4;  
  
echo($x + $y);  
  
echo "<br>";  
  
echo($x - $y);  
  
echo "<br>";  
  
echo($x * $y);  
  
echo "<br>";  
  
echo($x / $y);  
  
echo "<br>";  
  
echo($x % $y);  
  
?>
```

Output

```
14  
6  
40  
2.5  
2
```

PHP Assignment Operators

The assignment operators are used to assign values to variables.

Operator	Description	Example	Is The Same As
----------	-------------	---------	----------------

TOPS Technologies

=	Assign	\$x = \$y	\$x = \$y
+=	Add and assign	\$x += \$y	\$x = \$x + \$y
-=	Subtract and assign	\$x -= \$y	\$x = \$x - \$y
*=	Multiply and assign	\$x *= \$y	\$x = \$x * \$y
/=	Divide and assign quotient	\$x /= \$y	\$x = \$x / \$y
%=	Divide and assign modulus	\$x %= \$y	\$x = \$x % \$y

The following example will show you these assignment operators in action:

```
<?php
```

```
$x = 10;
```

```
echo $x;
```

```
echo "<br>";
```

```
$x = 20;
```

```
$x += 30;
```

```
echo $x;
```

```
echo "<br>";
```

```
$x = 50;
```

```
$x -= 20;
```

```
echo $x;
```

```
echo "<br>";
```

```
$x = 5;
```

```
$x *= 25;
```

```
echo $x;
```

```
echo "<br>";
```

```
$x = 50;
```

TOPS Technologies

```
$x /= 10;
```

```
echo $x;
```

```
echo "<br>";
```

```
$x = 100;
```

```
$x %= 15;
```

```
echo $x;
```

```
?>
```

Output

```
10
```

```
50
```

```
30
```

```
125
```

```
5
```

```
10
```

PHP Comparison Operators

The comparison operators are used to compare two values in a Boolean fashion.

Operator	Name	Example	Result
==	Equal	\$x == \$y	True if \$x is equal to \$y
===	Identical	\$x === \$y	True if \$x is equal to \$y, and they are of the same type
!=	Not equal	\$x != \$y	True if \$x is not equal to \$y
<>	Not equal	\$x <> \$y	True if \$x is not equal to \$y
!==	Not identical	\$x !== \$y	True if \$x is not equal to \$y, or they are

TOPS Technologies

			not of the same type
<	Less than	\$x < \$y	True if \$x is less than \$y
>	Greater than	\$x > \$y	True if \$x is greater than \$y
>=	Greater than or equal to	\$x >= \$y	True if \$x is greater than or equal to \$y
<=	Less than or equal to	\$x <= \$y	True if \$x is less than or equal to \$y

The following example will show you these comparison operators in action:

```
<?php  
  
$x = 25;  
  
$y = 35;  
  
$z = "25";  
  
var_dump($x == $z);  
  
echo "<br>";  
  
var_dump($x === $z);  
  
echo "<br>";  
  
var_dump($x != $y);  
  
echo "<br>";  
  
var_dump($x !== $z);  
  
echo "<br>";  
  
var_dump($x < $y);  
  
echo "<br>";  
  
var_dump($x > $y);  
  
echo "<br>";
```

```
var_dump($x <= $y);
```

```
echo "<br>";
```

```
var_dump($x >= $y);
```

```
?>
```

```
OutPut
```

```
bool(true)
```

```
bool(false)
```

```
bool(true)
```

```
bool(true)
```

```
bool(true)
```

```
bool(false)
```

```
echo $x;
```

TOPS Technologies

```
echo "<hr>";
```

```
$x = 10;
```

```
echo $x++;
```

```
echo "<br>";
```

```
echo $x;
```

```
echo "<hr>";
```

```
$x = 10;
```

```
echo --$x;
```

```
echo "<br>";
```

```
echo $x;
```

```
echo "<hr>";
```

```
$x = 10;
```

```
echo $x--;
```

```
echo "<br>";
```

```
echo $x;
```

```
?>
```

Output

```
11
```

```
11
```

```
10
```

```
11
```

9

9

10

9

PHP Logical Operators

The logical operators are typically used to combine conditional statements.

Operator	Name	Example	Result
and	And	\$x and \$y	True if both \$x and \$y are true
or	Or	\$x or \$y	True if either \$x or \$y is true
xor	Xor	\$x xor \$y	True if either \$x or \$y is true, but not both
&&	And	\$x && \$y	True if both \$x and \$y are true
	Or	\$x \$y	True if either \$x or \$y is true
!	Not	!\$x	True if \$x is not true

The following example will show you these logical operators in action:

```
<?php  
  
$year = 2014;  
  
// Leap years are divisible by 400 or by 4 but not 100  
  
if(($year % 400 == 0) || (($year % 100 != 0) && ($year % 4 == 0))){  
  
    echo "$year is a leap year.";  
  
}  
  
} else{  
  
    echo "$year is not a leap year.";  
  
}  
  
?>
```

TOPS Technologies

Output

2014 is not a leap year.

PHP String Operators

There are two operators which are specifically designed for [strings](#).

Operator	Description	Example	Result
.	Concatenation	\$str1 . \$str2	Concatenation of \$str1 and \$str2
.=	Concatenation assignment	\$str1 .= \$str2	Appends the \$str2 to the \$str1

The following example will show you these string operators in action:

```
<?php  
  
$x = "Hello";  
  
$y = " World!";  
  
echo $x . $y; // Outputs: Hello World!  
  
echo "<br>";  
  
$x .= $y;  
  
echo $x; // Outputs: Hello World!  
  
?>
```

PHP Array Operators

The array operators are used to compare arrays:

Operator	Name	Example	Result
+	Union	\$x + \$y	Union of \$x and \$y
==	Equality	\$x == \$y	True if \$x and \$y have the same key/value pairs
==	Identity	\$x === \$y	True if \$x and \$y have the same key/value pairs in the same

TOPS Technologies

			order and of the same types
!=	Inequality	<code>\$x != \$y</code>	True if \$x is not equal to \$y
<>	Inequality	<code>\$x <> \$y</code>	True if \$x is not equal to \$y
!==	Non-identity	<code>\$x !== \$y</code>	True if \$x is not identical to \$y

The following example will show you these array operators in action:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <title>PHP Array Operators</title>

</head>

<body>

<?php

$x = array("a" => "Red", "b" => "Green", "c" => "Blue");

$y = array("u" => "Yellow", "v" => "Orange", "w" => "Pink");

$z = $x + $y; // Union of $x and $y

var_dump($z);

echo "<hr>";

var_dump($x == $y);

echo "<br>";

var_dump($x === $y);

echo "<br>";
```

TOPS Technologies

```
var_dump($x != $y);
```

```
echo "<br>";
```

```
var_dump($x <> $y);
```

```
echo "<br>";
```

```
var_dump($x !== $y);
```

```
?>
```

```
</body>
```

```
</html>
```

Output

```
array(6) { ["a"]=> string(3) "Red" ["b"]=> string(5) "Green" ["c"]=> string(4) "Blue" ["u"]=> string(6) "Yellow" ["v"]=> string(6) "Orange" ["w"]=> string(4) "Pink" }
```

```
bool(false)
```

```
bool(false)
```

```
bool(true)
```

```
bool(true)
```

```
bool(true)
```

PHP Spaceship Operator [PHP 7](#)

PHP 7 introduces a new spaceship operator (`<=>`) which can be used for comparing two expressions. It is also known as combined comparison operator.

The spaceship operator returns 0 if both operands are equal, 1 if the left is greater, and -1 if the right is greater. It basically provides three-way comparison as shown in the following table:

Operator	<=> Equivalent
<code>\$x < \$y</code>	<code>(\$x <=> \$y) === -1</code>
<code>\$x <= \$y</code>	<code>(\$x <=> \$y) === -1 (\$x <=> \$y) === 0</code>

TOPS Technologies

Operator	<=> Equivalent
$\$x == \y	$(\$x <=> \$y) === 0$
$\$x != \y	$(\$x <=> \$y) !== 0$
$\$x >= \y	$(\$x <=> \$y) === 1 \text{ } (\$x <=> \$y) === 0$
$\$x > \y	$(\$x <=> \$y) === 1$

The following example will show you how spaceship operator actually works:

```
<?php

// Comparing Integers

echo 1 <=> 1; // Outputs: 0

echo "<br>";

echo 1 <=> 2; // Outputs: -1

echo "<br>";

echo 2 <=> 1; // Outputs: 1

echo "<hr>";

// Comparing Floats

echo 1.5 <=> 1.5; // Outputs: 0

echo "<br>";

echo 1.5 <=> 2.5; // Outputs: -1

echo "<br>";

echo 2.5 <=> 1.5; // Outputs: 1

echo "<hr>";

// Comparing Strings
```

TOPS Technologies

```
echo "x" <=> "x"; // Outputs: 0
```

```
echo "<br>";
```

```
echo "x" <=> "y"; // Outputs: -1
```

```
echo "<br>";
```

```
echo "y" <=> "x"; // Outputs: 1
```

```
?>
```

Output

0

-1

1

0

-1

1

0

-1

1

PHP If...Else Statements

PHP Conditional Statements

Like most programming languages, PHP also allows you to write code that perform different actions based on the results of a logical or comparative test conditions at run time. This means, you can create test conditions in the form of expressions that evaluates to either true or false and based on these results you can perform certain actions.

There are several statements in PHP that you can use to make decisions:

- The **if** statement
- The **if...else** statement
- The **if...elseif....else** statement
- The **switch...case** statement

We will explore each of these statements in the coming sections.

TOPS Technologies

The if Statement

The *if* statement is used to execute a block of code only if the specified condition evaluates to true. This is the simplest PHP's conditional statements and can be written like:

```
if(condition){  
    // Code to be executed  
}
```

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<?php  
  
$d = date("D");  
  
if($d == "Fri"){  
  
    echo "Have a nice weekend!";  
  
}  
  
?>
```

The if...else Statement

You can enhance the decision making process by providing an alternative choice through adding an *else* statement to the *if* statement. The *if...else* statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false. It can be written, like this:

```
if(condition){  
    // Code to be executed if condition is true  
} else{  
    // Code to be executed if condition is false  
}
```

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

```
<?php
```

```
$d = date("D");

if($d == "Fri"){

    echo "Have a nice weekend!";

} else{

    echo "Have a nice day!";

}

?>
```

Output

Have a nice day!

The if...elseif...else Statement

The *if...elseif...else* a special statement that is used to combine multiple *if...else* statements.

```
if(condition1){

    // Code to be executed if condition1 is true

} elseif(condition2){

    // Code to be executed if the condition1 is false and condition2 is true

} else{

    // Code to be executed if both condition1 and condition2 are false

}
```

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday, otherwise it will output "Have a nice day!"

```
<?php

$d = date("D");

if($d == "Fri"){

    echo "Have a nice weekend!";

} elseif($d == "Sun"){

    echo "Have a nice Sunday!";

} else{
```

```
echo "Have a nice day!";  
}  
?  
Have a nice Sunday!
```

The Ternary Operator

The ternary operator provides a shorthand way of writing the *if...else* statements. The ternary operator is represented by the question mark (?) symbol and it takes three operands: a condition to check, a result for true, and a result for false.

To understand how this operator works, consider the following examples:

```
<?php  
  
$age = 15;  
  
if($age < 18){  
  
    echo 'Child'; // Display Child if age is less than 18  
  
} else{  
  
    echo 'Adult'; // Display Adult if age is greater than or equal to 18  
  
}  
?  
Output
```

Using the ternary operator the same code could be written in a more compact way:

```
<?php echo ($age < 18) ? 'Child' : 'Adult'; ?>
```

Output

Child

TOPS Technologies

The ternary operator in the example above selects the value on the left of the colon (i.e. 'Child') if the condition evaluates to true (i.e. if \$age is less than 18), and selects the value on the right of the colon (i.e. 'Adult') if the condition evaluates to false.

The Null Coalescing Operator PHP 7

PHP 7 introduces a new null coalescing operator (??) which you can use as a shorthand where you need to use a ternary operator in conjunction with `isset()` function.

To understand this in a better way consider the following line of code. It fetches the value of `$_GET['name']`, if it does not exist or `NULL`, it returns 'anonymous'.

```
<?php
```

```
$name = isset($_GET['name']) ? $_GET['name'] : 'anonymous';
```

```
echo $name;
```

```
?>
```

Output

Anonymous

```
<?php
```

```
$name = $_GET['name'] ?? 'anonymous';
```

```
echo $name;
```

```
?>
```

Output

Anonymous

PHP Switch...Case Statements

PHP If...Else Vs Switch...Case

The switch-case statement is an alternative to the if-elseif-else statement, which does almost the same thing. The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

```
switch(n){  
    case label1:
```

TOPS Technologies

```
// Code to be executed if n=label1  
break;  
case label2:  
// Code to be executed if n=label2  
break;  
...  
default:  
// Code to be executed if n is different from all labels  
}
```

Consider the following example, which display a different message for each day.

```
<?php  
  
$today = date("D");  
  
switch($today){  
  
    case "Mon":  
  
        echo "Today is Monday. Clean your house.";  
  
        break;  
  
    case "Tue":  
  
        echo "Today is Tuesday. Buy some food.";  
  
        break;  
  
    case "Wed":  
  
        echo "Today is Wednesday. Visit a doctor.";  
  
        break;  
  
    case "Thu":  
  
        echo "Today is Thursday. Repair your car.";  
  
        break;  
  
    case "Fri":  
  
        echo "Today is Friday. Party tonight.";  
  
        break;
```

TOPS Technologies

```
case "Sat":
```

```
    echo "Today is Saturday. Its movie time.;"
```

```
    break;
```

```
case "Sun":
```

```
    echo "Today is Sunday. Do some rest.;"
```

```
    break;
```

```
default:
```

```
    echo "No information available for that day.;"
```

```
    break;
```

```
}
```

```
?>
```

Output

Today is Sunday. Do some rest.

The switch-case statement differs from the if-elseif-else statement in one important way. The switch statement executes line by line (i.e. statement by statement) and once PHP finds a case statement that evaluates to true, it's not only executes the code corresponding to that case statement, but also executes all the subsequent case statements till the end of the switch block automatically.

To prevent this add a break statement to the end of each case block. The break statement tells PHP to break out of the switch-case statement block once it executes the code associated with the first true case.

PHP Arrays

What is PHP Arrays

Arrays are complex variables that allow us to store more than one value or a group of values under a single variable name. Let's suppose you want to store colors in your PHP script. Storing the colors one by one in a variable could look something like this:

```
<?php  
$color1 = "Red";  
$color2 = "Green";  
$color3 = "Blue";  
?>
```

TOPS Technologies

But what, if you want to store the states or city names of a country in variables and this time this not just three may be hundred. It is quite hard, boring, and bad idea to store each city name in a separate variable. And here array comes into play.

Types of Arrays in PHP

There are three types of arrays that you can create. These are:

- **Indexed array** — An array with a numeric key.
- **Associative array** — An array where each key has its own specific value.
- **Multidimensional array** — An array containing one or more arrays within itself.

Indexed Arrays

An indexed or numeric array stores each array element with a numeric index. The following examples shows two ways of creating an indexed array, the easiest way is:

```
<?php  
$colors = array("Red", "Green", "Blue");  
  
// Printing array structure  
print_r($colors);  
?  
Output  
Array ( [0] => Red [1] => Green [2] => Blue )
```

This is equivalent to the following example, in which indexes are assigned manually:

```
<?php  
$colors[0] = "Red";  
$colors[1] = "Green";  
$colors[2] = "Blue";  
?>
```

Associative Arrays

In an associative array, the keys assigned to values can be arbitrary and user defined strings. In the following example the array uses keys instead of index numbers:

```
<?php
```

```
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
```

```
// Printing array structure  
print_r($ages);  
?>
```

TOPS Technologies

Array ([Peter] => 22 [Clark] => 32 [John] => 28)

The following example is equivalent to the previous example, but shows a different way of creating associative arrays:

```
<?php  
$ages["Peter"] = "22";  
$ages["Clark"] = "32";  
$ages["John"] = "28";  
?>
```

Multidimensional Arrays

The multidimensional array is an array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on. An example of a multidimensional array will look something like this:

```
<?php  
// Define nested array  
$contacts = array(  
    array(  
        "name" => "Peter Parker",  
        "email" => "peterparker@mail.com",  
    ),  
    array(  
        "name" => "Clark Kent",  
        "email" => "clarkkent@mail.com",  
    ),  
    array(  
        "name" => "Harry Potter",  
        "email" => "harrypotter@mail.com",  
    )  
);  
// Access nested value  
echo "Peter Parker's Email-id is: " . $contacts[0]["email"];  
?>  
Output  
Peter Parker's Email-id is: peterparker@mail.com
```

PHP Sorting Arrays

PHP Functions For Sorting Arrays

In the previous chapter you've learnt the essentials of PHP arrays i.e. what arrays are, how to create them, how to view their structure, how to access their elements etc. You can do even more things with arrays like sorting the elements in any order you like.

PHP comes with a number of built-in functions designed specifically for sorting array elements in different ways like alphabetically or numerically in ascending or descending order. Here we'll explore some of these functions most commonly used for sorting arrays.

- `sort()` and `rsort()` — For sorting indexed arrays
- `asort()` and `arsort()` — For sorting associative arrays by value
- `ksort()` and `krsort()` — For sorting associative arrays by key

Sorting Indexed Arrays in Ascending Order

The `sort()` function is used for sorting the elements of the indexed array in ascending order (alphabetically for letters and numerically for numbers).

```
<?php  
// Define array  
  
$colors = array("Red", "Green", "Blue", "Yellow");  
  
// Sorting and printing array  
  
sort($colors);  
print_r($colors);  
?>
```

This `print_r()` statement gives the following output:

Array ([0] => Blue [1] => Green [2] => Red [3] => Yellow)
Similarly you can sort the numeric elements of the array in ascending order.

```
<?php  
// Define array  
  
$numbers = array(1, 2, 2.5, 4, 7, 10);  
  
// Sorting and printing array  
  
sort($numbers);  
print_r($numbers);  
?>
```

This `print_r()` statement gives the following output:

Array ([0] => 1 [1] => 2 [2] => 2.5 [3] => 4 [4] => 7 [5] => 10)

TOPS Technologies

Sorting Indexed Arrays in Descending Order

The `rsort()` function is used for sorting the elements of the indexed array in descending order (alphabetically for letters and numerically for numbers).

```
<?php  
// Define array  
  
$colors = array("Red", "Green", "Blue", "Yellow");  
  
// Sorting and printing array  
  
rsort($colors);  
  
print_r($colors);  
?>
```

This `print_r()` statement gives the following output:

Array ([0] => Yellow [1] => Red [2] => Green [3] => Blue)
Similarly you can sort the numeric elements of the array in descending order.

```
<?php  
// Define array  
  
$numbers = array(1, 2, 2.5, 4, 7, 10);  
  
// Sorting and printing array  
  
rsort($numbers);  
  
print_r($numbers);  
?>
```

This `print_r()` statement gives the following output:

Array ([0] => 10 [1] => 7 [2] => 4 [3] => 2.5 [4] => 2 [5] => 1)

Sorting Associative Arrays in Ascending Order By Value

The `asort()` function sorts the elements of an associative array in ascending order according to the value. It works just like `sort()`, but it preserves the association between keys and its values while sorting.

```
<?php  
// Define array  
  
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);  
  
// Sorting array by value and print  
  
asort($age);  
  
print_r($age);
```

TOPS Technologies

?>

This print_r() statement gives the following output:

```
Array ( [Harry] => 14 [Peter] => 20 [Clark] => 35 [John] => 45 )
```

Sorting Associative Arrays in Descending Order By Value

The arsort() function sorts the elements of an associative array in descending order according to the value. It works just like rsort(), but it preserves the association between keys and its values while sorting.

```
<?php  
// Define array  
  
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);  
  
// Sorting array by value and print  
  
arsort($age);  
  
print_r($age);  
?>
```

This print_r() statement gives the following output:

```
Array ( [John] => 45 [Clark] => 35 [Peter] => 20 [Harry] => 14 )
```

Sorting Associative Arrays in Ascending Order By Key

The ksort() function sorts the elements of an associative array in ascending order by their keys. It preserves the association between keys and its values while sorting, same as asort() function.

```
<?php  
// Define array  
  
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);  
  
// Sorting array by key and print  
  
ksort($age);  
  
print_r($age);  
?>
```

This print_r() statement gives the following output:

```
Array ( [Clark] => 35 [Harry] => 14 [John] => 45 [Peter] => 20 )
```

Sorting Associative Arrays in Descending Order By Key

The krsort() function sorts the elements of an associative array in descending order by their keys. It preserves the association between keys and its values while sorting, same as arsort() function.

```
<?php  
// Define array
```

TOPS Technologies

```
$age = array("Peter"=>20, "Harry"=>14, "John"=>45, "Clark"=>35);
```

```
// Sorting array by key and print  
krsort($age);  
print_r($age);  
?>
```

This `print_r()` statement gives the following output:

```
Array ( [Peter] => 20 [John] => 45 [Harry] => 14 [Clark] => 35 )
```

PHP Loops

how to repeat a series of actions using loops in PHP.

Different Types of Loops in PHP

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. PHP supports four different types of loops.

- **while** — loops through a block of code as long as the condition specified evaluates to true.
- **do...while** — the block of code executed once and then condition is evaluated. If the condition is true the statement is repeated as long as the specified condition is true.
- **for** — loops through a block of code until the counter reaches a specified number.
- **foreach** — loops through a block of code for each element in an array.

You will also learn how to loop through the values of array using `foreach()` loop at the end of this chapter. The `foreach()` loop work specifically with arrays.

PHP while Loop

The while statement will loops through a block of code as long as the condition specified in the `while` statement evaluate to true.

```
while(condition){  
    // Code to be executed  
}
```

The example below define a loop that starts with `$i=1`. The loop will continue to run as long as `$i` is less than or equal to 3. The `$i` will increase by 1 each time the loop runs:

```
<?php
```

```
$i = 1;  
  
while($i <= 3){  
  
    $i++;  
  
    echo "The number is " . $i . "<br>";  
  
}  
  
?>
```

Output

```
The number is 2  
The number is 3  
The number is 4
```

PHP do...while Loop

The do-while loop is a variant of while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true.

```
do{  
    // Code to be executed  
}  
while(condition);
```

The following example define a loop that starts with \$i=1. It will then increase \$i with 1, and print the output. Then the condition is evaluated, and the loop will continue to run as long as \$i is less than, or equal to 3.

```
<?php  
  
$i = 1;  
  
do{  
  
    $i++;  
  
    echo "The number is " . $i . "<br>";  
  
}  
  
while($i <= 3);  
  
?>
```

Output

TOPS Technologies

The number is 2

The number is 3

The number is 4

TOPS Technologies

Difference Between while and do...while Loop

The while loop differs from the do-while loop in one important way — with a while loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a do-while loop, on the other hand, the loop will always be executed once, even if the conditional expression is false, because the condition is evaluated at the end of the loop iteration rather than the beginning.

PHP for Loop

The for loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times.

```
for(initialization; condition; increment){  
    // Code to be executed  
}
```

The parameters of for loop have following meanings:

- initialization — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
- condition — in the beginning of each iteration, condition is evaluated. If it evaluates to true, the loop continues and the nested statements are executed. If it evaluates to false, the execution of the loop ends.
- increment — it updates the loop counter with a new value. It is evaluate at the end of each iteration.

The example below defines a loop that starts with \$i=1. The loop will continued until \$i is less than, or equal to 3. The variable \$i will increase by 1 each time the loop runs:

```
<?php  
  
for($i=1; $i<=3; $i++){  
  
    echo "The number is " . $i . "<br>";  
  
}  
  
?>
```

Output

```
The number is 2  
The number is 3  
The number is 4
```

PHP foreach Loop

The foreach loop is used to iterate over arrays.

```
foreach($array as $value){  
    // Code to be executed  
}
```

The following example demonstrates a loop that will print the values of the given array:

TOPS Technologies

```
<?php
```

```
$colors = array("Red", "Green", "Blue");
```

```
// Loop through colors array
```

```
foreach($colors as $value){
```

```
    echo $value . "<br>";
```

```
}
```

```
?>
```

Output

Red

Green

Blue

There is one more syntax of `foreach` loop, which is extension of the first.

```
foreach($array as $key => $value){  
    // Code to be executed  
}
```

```
<?php
```

```
$superhero = array(
```

```
    "name" => "Peter Parker",
```

```
    "email" => "peterparker@mail.com",
```

```
    "age" => 18
```

```
);
```

```
// Loop through superhero array
```

```
foreach($superhero as $key => $value){
```

```
    echo $key . ":" . $value . "<br>";
```

```
}
```

```
?>
```

Output

```
name : Peter Parker  
email : peterparker@mail.com  
age : 18
```

PHP Functions

how to create your own custom functions in PHP.

PHP Built-in Functions

A function is a self-contained block of code that performs a specific task.

PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like `gettype()`, `print_r()`, `var_dump`, etc.

Please check out PHP reference section for a complete list of useful PHP built-in functions.

PHP User-Defined Functions

In addition to the built-in functions, PHP also allows you to define your own functions. It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program. Here are some advantages of using functions:

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.
- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occurs you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.
- **Functions can be reused in other application** — Because a function is separated from the rest of the script, it's easy to reuse the same function in other applications just by including the php file containing those functions.

The following section will show you how easily you can define your own function in PHP.

Creating and Invoking Functions

The basic syntax of creating a custom function can be given with:

```
function functionName(){  
    // Code to be executed  
}
```

The declaration of a user-defined function starts with the word `function`, followed by the name of the function you want to create followed by parentheses i.e. `()` and finally place your function's code between curly brackets `{}`.

This is a simple example of an user-defined function, that displays today's date:

```
<?php
```

// Defining function

```
function whatIsToday(){  
    echo "Today is " . date("l", mktime());  
}
```

// Calling function

```
whatIsToday();
```

```
?>
```

Output

Today is Sunday

Functions with Parameters

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

```
function myFunc($oneParameter, $anotherParameter){  
    // Code to be executed  
}
```

You can define as many parameters as you like. However for each parameter you specify, a corresponding argument needs to be passed to the function when it is called.

The getSum() function in following example takes two integer values as arguments, simply add them together and then display the result in the browser.

```
<?php
```

// Defining function

```
function getSum($num1, $num2){  
  
    $sum = $num1 + $num2;  
  
    echo "Sum of the two numbers $num1 and $num2 is : $sum";  
  
}
```

// Calling function

TOPS Technologies

```
getSum(10, 20);
```

```
?>
```

The output of the above code will be:

```
Sum of the two numbers 10 and 20 is : 30
```

Functions with Optional Parameters and Default Values

You can also create functions with optional parameters — just insert the parameter name, followed by an equals (=) sign, followed by a default value, like this.

```
<?php
```

```
// Defining function
```

```
function customFont($font, $size=1.5){
```

```
echo "<p style=\"font-family: $font; font-size: {$size}em;\">Hello, world!</p>";
```

```
}
```

```
// Calling function
```

```
customFont("Arial", 2);
```

```
customFont("Times", 3);
```

```
customFont("Courier");
```

```
?>
```

Returning Values from a Function

A function can return a value back to the script that called the function using the return statement. The value may be of any type, including arrays and objects.

```
<?php
```

```
// Defining function
```

```
function getSum($num1, $num2){
```

TOPS Technologies

```
$total = $num1 + $num2;  
  
return $total;  
  
}
```

```
// Printing returned value  
  
echo getSum(5, 10); // Outputs: 15  
  
?>
```

A function can not return multiple values. However, you can obtain similar results by returning an array, as demonstrated in the following example.

```
<?php  
  
// Defining function  
  
function divideNumbers($dividend, $divisor){  
  
    $quotient = $dividend / $divisor;  
  
    $array = array($dividend, $divisor, $quotient);  
  
    return $array;  
  
}
```

```
// Assign variables as if they were an array  
  
list($dividend, $divisor, $quotient) = divideNumbers(10, 2);  
  
echo $dividend . "<br>"; // Outputs: 10  
  
echo $divisor . "<br>"; // Outputs: 2  
  
echo $quotient . "<br>"; // Outputs: 5  
  
?>
```

TOPS Technologies

Passing Arguments to a Function by Reference

In PHP there are two ways you can pass arguments to a function: *by value* and *by reference*. By default, function arguments are passed by value so that if the value of the argument within the function is changed, it does not get affected outside of the function. However, to allow a function to modify its arguments, they must be passed by reference.

Passing an argument by reference is done by prepending an ampersand (&) to the argument name in the function definition, as shown in the example below:

```
<?php

/* Defining a function that multiply a number
   by itself and return the new value */

function selfMultiply(&$number){

    $number *= $number;

    return $number;

}

$mynum = 5;

echo $mynum . "<br>"; // Outputs: 5

selfMultiply($mynum);

echo $mynum . "<br>"; // Outputs: 25

?>
```

Understanding the Variable Scope

However, you can declare the variables anywhere in a PHP script. But, the location of the declaration determines the extent of a variable's visibility within the PHP program i.e. where the variable can be used or accessed. This accessibility is known as *variable scope*.

By default, variables declared within a function are local and they cannot be viewed or manipulated from outside of that function, as demonstrated in the example below:

```
<?php
```

TOPS Technologies

// Defining function

```
function test(){  
  
    $greet = "Hello World!";  
  
    echo $greet;  
  
}
```

```
test(); // Outputs: Hello World!
```

```
echo $greet; // Generate undefined variable error
```

```
?>
```

Similarly, if you try to access or import an outside variable inside the function, you'll get an undefined variable error, as shown in the following example:

```
<?php  
  
$greet = "Hello World!";
```

// Defining function

```
function test(){
```

```
    echo $greet;
```

```
}
```

```
test(); // Generate undefined variable error
```

```
echo $greet; // Outputs: Hello World!
```

```
?>
```

The global Keyword

There may be a situation when you need to import a variable from the main program into a function, or vice versa. In such cases, you can use the `global` keyword before the variables inside a function. This keyword turns the variable into a global variable, making it visible or accessible both inside and outside the function, as shown in the example below:

```
<?php

$greet = "Hello World!";

// Defining function

function test(){

    global $greet;

    echo $greet;

}

test(); // Outputs: Hello World!

echo $greet; // Outputs: Hello World!

// Assign a new value to variable

$greet = "Goodbye";

test(); // Outputs: Goodbye

echo $greet; // Outputs: Goodbye

?>
```

Creating Recursive Functions

A recursive function is a function that calls itself again and again until a condition is satisfied. Recursive functions are often used to solve complex mathematical calculations, or to process deeply nested structures e.g., printing all the elements of a deeply nested array.

The following example demonstrates how a recursive function works.

```
<?php

// Defining recursive function
```

TOPS Technologies

```
function printValues($arr) {  
  
    global $count;  
  
    global $items;  
  
    // Check input is an array  
  
    if(!is_array($arr)){  
  
        die("ERROR: Input is not an array");  
  
    }  
  
    /*  
  
    Loop through array, if value is itself an array recursively call the function,  
  
    else add the value found to the output items array,  
  
    and increment counter by 1 for each value found  
  
    */  
  
    foreach($arr as $a){  
  
        if(is_array($a)){  
  
            printValues($a);  
  
        } else{  
  
            $items[] = $a;  
  
            $count++;  
  
        }  
  
    }  
  
    // Return total count and values found in array  
  
    return array('total' => $count, 'values' => $items); }  
  

```

TOPS Technologies

// Define nested array

```
$species = array(  
    "birds" => array(  
        "Eagle",  
        "Parrot",  
        "Swan"  
    ),  
  
    "mammals" => array(  
        "Human",  
        "cat" => array(  
            "Lion",  
            "Tiger",  
            "Jaguar"  
        ),  
  
        "Elephant",  
        "Monkey"  
    ),  
  
    "reptiles" => array(  
        "snake" => array(  
            "Cobra" => array(  
                "King Cobra",  
                "Egyptian cobra"  
            )  
        )  
    )  
)
```

```
    ),  
  
    "Viper",  
  
    "Anaconda"  
  
,  
  
    "Crocodile",  
  
    "Dinosaur" => array(  
  
        "T-rex",  
  
        "Alamosaurus"  
  
)  
  
)  
  
);  
  
// Count and print values in nested array  
  
$result = printValues($species);  
  
echo $result['total'] . ' value(s) found: ';  
  
echo implode(', ', $result['values']);  
  
?>  
  
Output  
  
16 value(s) found: Eagle, Parrot, Swan, Human, Lion, Tiger, Jaguar, Elephant, Monkey, King Cobra, Egyptian  
cobra, Viper, Anaconda, Crocodile, T-rex, Alamosaurus
```

PHP Superglobals

These are specially-defined array variables in PHP that make it easy for you to get information about a request or its context. The superglobals are available throughout your script. These variables can be accessed from any function, class or any file without doing any special task such as declaring any global variable etc. They are mainly used to store and get information from one page to another etc in an application.

Below is the list of superglobal variables available in PHP:

1. `$GLOBALS`
2. `$_SERVER`
3. `$_REQUEST`
4. `$_GET`
5. `$_POST`
6. `$_SESSION`
7. `$_COOKIE`
8. `$_FILES`
9. `$_ENV`

Let us now learn about some of these superglobals in details:

- **`$GLOBALS`** : It is a superglobal variable which is used to access global variables from anywhere in the PHP script. PHP stores all the global variables in array `$GLOBALS[]` where index holds the global variable name, which can be accessed.

Below program illustrates the use of `$GLOBALS` in PHP:

```
<?php  
$x = 300;  
$y = 200;  
  
function multiplication(){  
    $GLOBALS['z'] = $GLOBALS['x'] * $GLOBALS['y'];  
}  
multiplication();  
echo $z;  
?>  
Output :
```

60000

In the above code two global variables are declared `$x` and `$y` which are assigned some value to them. Then a function **multiplication()** is defined to multiply the values of `$x` and `$y` and store in another variable `$z` defined in the **GLOBAL array**.

TOPS Technologies

- **\$_SERVER** : It is a PHP super global variable that stores the information about headers, paths and script locations. Some of these elements are used to get the information from the superglobal variable **\$_SERVER**.

Below program illustrates the use of **\$_SERVER** in PHP:

```
filter_none
brightness_4
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
echo "<br>"
?>
```

Output :

```
/geek1/index.php
localhost
localhost:81
Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/
```

In the above code we used the **\$_SERVER** elements to get some information. We get the current file name which is worked on using '**PHP_SELF**' element. Then we get server name used currently using '**SERVER_NAME**' element. And then we get the host name through '**HTTP_HOST**'.

- **\$_REQUEST** : It is a superglobal variable which is used to collect the data after submitting a HTML form. **\$_REQUEST** is not used mostly, because **\$_POST** and **\$_GET** perform the same task and are widely used.

Below is the HTML and PHP code to explain how **\$_REQUEST** works:

```
<!DOCTYPE html>
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
NAME: <input type="text" name="fname">
<button type="submit">SUBMIT</button>
</form>
```

```
<?php  
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    $name = htmlspecialchars($_REQUEST['fname']);  
    if(empty($name)){  
        echo "Name is empty";  
    } else {  
        echo $name;  
    }  
}  
?  
</body>  
</html>  
Output :
```

NAME:
Sharvan

In the above code we have created a form that takes the name as input from the user and prints it's name on clicking of submit button. We transport the data accepted in the form to the same page using **\$_SERVER['PHP_SELF']** element as specified in the action attribute, because we manipulate the data in the same page using the PHP code. The data is retrieved using the **\$_REQUEST** superglobal array variable

- **\$_POST** : It is a super global variable used to collect data from the HTML form after submitting it. When form uses method post to transfer data, the data is not visible in the query string, because of which security levels are maintained in this method.

Below is the HTML and PHP code to explain how **\$_POST** works:

```
filter_none  
brightness_4  
<!DOCTYPE html>  
<html>  
<body>  
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">  
    <label for="name">Please enter your name: </label>  
    <input name="name" type="text"><br>  
    <label for="age">Please enter your age: </label>  
    <input name="age" type="text"><br>  
    <input type="submit" value="Submit">  
    <button type="submit">SUBMIT</button>
```

```
</form>
<?php
$nm=$_POST['name'];
$age=$_POST['age'];
echo "<strong>".$nm." is $age years old.</strong>";
?>
</body>
</html>
Output :
```

Please enter your name:

Please enter your age:

sharvan is 22 years old.

In the above code we have created a form that takes name and age of the user and accesses the data using `$_POST` super global variable when they submit the data. Since each superglobal variable is an array it can store more than one values. Hence we retrieved name and age from the `$_POST` variable and stored them in `$nm` and `$age` variables.

- **`$_GET`** : `$_GET` is a super global variable used to collect data from the HTML form after submitting it. When form uses method get to transfer data, the data is visible in the query string, therefore the values are not hidden. `$_GET` super global array variable stores the values that come in the URL.

Below is the HTML and PHP code to explain how `$_GET` works:

```
filter_none
brightness_4
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body bgcolor="cyan">
<?php
$name = $_GET['name'];
$city = $_GET['city'];
echo "<h1>This is ".$name." of ".$city."</h1><br>";
?>
<img src = "2.jpg" alt = "nanilake" height = "400" width="500" />
</body>
</html>
```

TOPS Technologies

We are actually seeing half of the logic just now. In the above code we have created a hyperlink image of Nainital Lake which will take us to picture.php page and with it will also take the parameters **name="Nainilake"** and **city="Nainital"**.

That is when we click on the small image of Nainital Lake we will be taken to the next page picture.php along with the parameters. As the default method is get, these parameters will be passed to the next page using get method and they will be visible in the address bar. When we want to pass values to an address they are attached to the address using a question mark (?).

Here the parameter **name=Nainilake** is attached to the address. If we want to add more values, we can add them using ampersand (&) after every key-value pair similarly as **city=Nainital** is added using ampersand after the name parameter. Now after clicking on the image of Nainital Lake we want the picture.php page to be displayed with the value of parameter displayed along with it.

PHP Date and Time

how to extract or format the date and time in PHP.

The PHP date() Function

The PHP date() function convert a timestamp to a more readable date and time.

The computer stores dates and times in a format called UNIX Timestamp, which measures time as a number of seconds since the beginning of the Unix epoch (midnight Greenwich Mean Time on January 1, 1970 i.e. January 1, 1970 00:00:00 GMT).

Since this is an impractical format for humans to read, PHP converts a timestamp to a format that is readable to humans and dates from your notation into a timestamp the computer understands. The syntax of the PHP date() function can be given with.

```
date(format, timestamp)
```

The *format* parameter in the date() function is required which specifies the format of returned date and time. However the *timestamp* is an optional parameter, if not included then current date and time will be used. The following statement displays today's date:

```
<?php  
$today = date("d/m/Y");  
echo $today;  
?  
18/10/2020
```

Formatting the Dates and Times with PHP

The format parameter of the date() function is in fact a string that can contain multiple characters allowing you to generate a date string containing various components of the date and time, like day of the week, AM or PM, etc. Here are some the date-related formatting characters that are commonly used in format string:

- d - Represent day of the month; two digits with leading zeros (01 or 31)
- D - Represent day of the week in text as an abbreviation (Mon to Sun)
- m - Represent month in numbers with leading zeros (01 or 12)
- M - Represent month in text, abbreviated (Jan to Dec)

TOPS Technologies

- y - Represent year in two digits (08 or 14)
- Y - Represent year in four digits (2008 or 2014)

The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

```
<?php  
echo date("d/m/Y") . "<br>";  
echo date("d-m-Y") . "<br>";  
echo date("d.m.Y");  
?>
```

Output

18/10/2020
18-10-2020
18.10.2020

Similarly you can use the following characters to format the time string:

- h - Represent hour in 12-hour format with leading zeros (01 to 12)
- H - Represent hour in 24-hour format with leading zeros (00 to 23)
- i - Represent minutes with leading zeros (00 to 59)
- s - Represent seconds with leading zeros (00 to 59)
- a - Represent lowercase ante meridiem and post meridiem (am or pm)
- A - Represent uppercase Ante meridiem and Post meridiem (AM or PM)

The PHP code in the following example displays the date in different formats:

```
<?php  
// Return current date and time from the remote server  
echo date("h:i:s") . "<br>";  
echo date("F d, Y h:i:s A") . "<br>";  
echo date("h:i a");  
?>  
Output  
12:03:49  
October 18, 2020 12:03:49 PM  
12:03 pm
```

The PHP time() Function

TOPS Technologies

The `time()` function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1 1970 00:00:00 GMT).

```
<?php  
// Executed at March 05, 2014 07:19:18  
  
$timestamp = time();  
  
echo($timestamp);  
  
?>
```

The above example produce the following output.

1394003958

We can convert this timestamp to a human readable date through passing it to the previously introduce `date()` function.

```
<?php  
  
$timestamp = 1394003958;  
  
echo(date("F d, Y h:i:s", $timestamp));  
  
?>
```

The above example produce the following output.

March 05, 2014 07:19:18

The above example produce the following output.

March 05, 2014 07:19:18

```
<?php  
// Create the timestamp for a particular date  
echo mktime(15, 20, 12, 5, 10, 2014);  
?>
```

The above example produce the following output.

1399735212

The `mktime()` function can be used to find the weekday name corresponding to a particular date. To do this, simply use the '`I`' (lowercase '`L`') character with your timestamp, as in the following example, which displays the day that falls on April 1, 2014:

```
<?php  
// Get the weekday name of a particular date  
echo date('l', mktime(0, 0, 0, 4, 1, 2014));  
?>
```

The above example produce the following output.

Tuesday

The `mktime()` function can also be used to find a particular date in future after a specific time period. As in the following example, which displays the date which falls on after 30 month from the current date?

```
<?php  
// Executed at March 05, 2014  
  
$futureDate = mktime(0, 0, 0, date("m") + 30, date("d"), date("Y"));  
  
echo date("d/m/Y", $futureDate);  
  
?>
```

The above example produce the following output.

05/09/2016

PHP Include and Require Files

how to include and evaluate the files in PHP.

Including a PHP File into Another PHP File

The include() and require() statement allow you to include the code contained in a PHP file within another PHP file. Including a file produces the same result as copying the script from the file specified and pasted into the location where it is called.

You can save a lot of time and work through including files — Just store a block of code in a separate file and include it wherever you want using the include() and require() statements instead of typing the entire block of code multiple times. A typical example is including the header, footer and menu file within all the pages of a website.

The basic syntax of the include() and require() statement can be given with:

```
include("path/to/filename"); -Or- include "path/to/filename";  
require("path/to/filename"); -Or- require "path/to/filename";
```

The following example will show you how to include the common header, footer and menu codes which are stored in separate 'header.php', 'footer.php' and 'menu.php' files respectively, within all the pages of your website. Using this technique you can update all pages of the website at once by making the changes to just one file, this saves a lot of repetitive work.

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <title>Tutorial Republic</title>  
</head>  
<body>  
    <?php include "header.php"; ?>  
    <?php include "menu.php"; ?>  
    <h1>Welcome to Our Website!</h1>  
    <p>Here you will find lots of useful information.</p>  
    <?php include "footer.php"; ?>  
</body>  
</html>
```

Difference Between include and require Statements

You might be thinking if we can include files using the include() statement then why we need require(). Typically the require() statement operates like include().

The only difference is — the include() statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the require() statement will generate a fatal error and stops the script execution.

```
<?php require "my_variables.php"; ?>  
<?php require "my_functions.php"; ?>  
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
    <title><?php displayTitle($home_page); ?></title>  
</head>  
  
<body>  
    <?php include "header.php"; ?>  
    <?php include "menu.php"; ?>  
    <h1>Welcome to Our Website!</h1>  
    <p>Here you will find lots of useful information.</p>  
    <?php include "footer.php"; ?>  
  
</body>  
  
</html>
```

The include_once and require_once Statements

If you accidentally include the same file (typically [functions](#) or [classes](#) files) more than one time within your code using the include or require statements, it may cause conflicts. To prevent this situation, PHP provides `include_once` and `require_once` statements. These statements behave in the same way as include and require statements with one exception.

The `include_once` and `require_once` statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again. To better understand how it works, let's check out an example. Suppose we've a 'my_functions.php' file with the following code:

```
<?php  
  
function multiplySelf($var){  
  
    $var *= $var; // multiply variable by itself  
  
    echo $var;  
  
}
```

TOPS Technologies

?>

Here's is the PHP script within which we've included the 'my_functions.php' file.

```
<?php  
  
// Including file  
  
require "my_functions.php";  
  
// Calling the function  
  
multiplySelf(2); // Output: 4  
  
echo "<br>";  
  
  
// Including file once again  
  
require "my_functions.php";  
  
// Calling the function  
  
multiplySelf(5); // Doesn't execute  
  
?>
```

When you run the above script, you will see the error message something like this: "**Fatal error: Cannot redeclare multiplySelf()**". This occurs because the 'my_functions.php' included twice, this means the function multiplySelf() is defined twice, which caused PHP to stop script execution and generate fatal error. Now rewrite the above example with require_once.

```
<?php  
  
// Including file  
  
require_once "my_functions.php";  
  
// Calling the function  
  
multiplySelf(2); // Output: 4  
  
echo "<br>";  
  
  
// Including file once again  
  
require_once "my_functions.php";  
  
// Calling the function  
  
multiplySelf(5); // Output: 25
```

?>

As you can see, by using `require_once` instead of `require`, the script works as we expected.

Module – 5 [Database Connectivity]

PHP - MySQL

MySQL is the most popular database system used with the PHP language.

What is MySQL

MySQL is one of the most popular relational database system being used on the Web today. It is freely available and easy to install, however if you have installed Wampserver it already there on your machine. MySQL database server offers several advantages:

- MySQL is easy to use, yet extremely powerful, fast, secure, and scalable.
- MySQL runs on a wide range of operating systems, including UNIX or Linux, Microsoft Windows, Apple Mac OS X, and others.
- MySQL supports standard SQL (Structured Query Language).
- MySQL is ideal database solution for both small and large applications.
- MySQL is developed, and distributed by Oracle Corporation.
- MySQL includes data security layers that protect sensitive data from intruders.

MySQL database stores data into tables like other relational database. A table is a collection of related data, and it is divided into rows and columns.

Each row in a table represents a data record that are inherently connected to each other such as information related to a particular person, whereas each column represents a specific field such as `id`, `first_name`, `last_name`, `email`, etc. The structure of a simple MySQL table that contains person's general information may look something like this:

+-----+	+-----+	+-----+	+-----+
id first_name last_name email			
+-----+	+-----+	+-----+	+-----+
1 Peter Parker peterparker@mail.com			
2 John Rambo johnrambo@mail.com			
3 Clark Kent clarkkent@mail.com			
4 John Carter johncarter@mail.com			
5 Harry Potter harrypotter@mail.com			
+-----+	+-----+	+-----+	+-----+

Talking to MySQL Databases with SQL

TOPS Technologies

SQL, the Structured Query Language, is a simple, standardized language for communicating with relational databases like MySQL. With SQL you can perform any database-related task, such as creating databases and tables, saving data in database tables, query a database for specific records, deleting and updating data in databases.

Look at the following standard SQL query that returns the email address of a person whose first name is equal to 'Peter' in the *persons* table:

```
SELECT email FROM persons WHERE first_name="Peter"
```

If you execute the SQL query above it will return the following record:

peterparker@mail.com

PHP Connect to MySQL Server

Ways of Connecting to MySQL through PHP

In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server. PHP offers two different ways to connect to MySQL server: **MySQLi** (Improved MySQL) and **PDO** (PHP Data Objects) extensions.

While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only. MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server. Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API which is relatively easy for beginners to understand.

Connecting to MySQL Database Server

In PHP you can easily do this using the `mysqli_connect()` function. All communication between PHP and the MySQL database server takes place through this connection. Here're the basic syntaxes for connecting to MySQL using MySQLi and PDO extensions:

Syntax: MySQLi, Procedural way

```
$link = mysqli_connect("hostname", "username", "password", "database");
```

Syntax: MySQLi, Object Oriented way

```
$mysqli = new mysqli("hostname", "username", "password", "database");
```

Syntax: PHP Data Objects (PDO) way

```
$pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");
```

TOPS Technologies

The *hostname* parameter in the above syntax specify the host name (e.g. `localhost`), or IP address of the MySQL server, whereas the *username* and *password* parameters specifies the credentials to access MySQL server, and the *database* parameter, if provided will specify the default MySQL database to be used when performing queries.

The following example shows how to connect to MySQL database server using MySQLi (both *procedural* and *object oriented* way) and PDO extension.

Procedural

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Print host information
echo "Connect Successfully. Host info: " . mysqli_get_host_info($link);
?>
```

Object Oriented

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$mysqli = new mysqli("localhost", "root", "", "demo");

// Check connection
if($mysqli === false){
    die("ERROR: Could not connect. " . $mysqli->connect_error());
}

// Print host information
echo "Connect Successfully. Host info: " . $mysqli->host_info;
?>
```

PDO

```
<?php
```

TOPS Technologies

```
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

try{
    $pdo = new PDO("mysql:host=localhost", "root", "");

    // Set the PDO error mode to exception
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Print host information
    echo "Connect Successfully. Host info: " .
    $pdo->getAttribute(constant("PDO::ATTR_CONNECTION_STATUS"));

} catch(PDOException $e){
    die("ERROR: Could not connect. " . $e->getMessage());
}

?>
```

Closing the MySQL Database Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, if you want to close it earlier you can do this by simply calling the PHP `mysqli_close()` function.

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Print host information
echo "Connect Successfully. Host info: " . mysqli_get_host_info($link);

// Close connection
mysqli_close($link);

?>
```

Creating MySQL Database Using PHP

TOPS Technologies

Now that you've understood how to open a connection to the MySQL database server. In this tutorial you will learn how to execute SQL query to create a database.

Before saving or accessing the data, we need to create a database first. The [CREATE DATABASE](#) statement is used to create a new database in MySQL.

Let's make a SQL query using the CREATE DATABASE statement, after that we will execute this SQL query through passing it to the PHP `mysqli_query()` function to finally create our database. The following example creates a database named `demo`.

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt create database query execution
$sql = "CREATE DATABASE demo";
if(mysqli_query($link, $sql)){
    echo "Database created successfully";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

PHP MySQL Create Tables

The SQL [CREATE TABLE](#) statement is used to create a table in database.

Let's make a SQL query using the CREATE TABLE statement, after that we will execute this SQL query through passing it to the PHP `mysqli_query()` function to finally create our table.

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
```

TOPS Technologies

```
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt create table query execution
$sql = "CREATE TABLE persons(
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    email VARCHAR(70) NOT NULL UNIQUE
)";

if(mysqli_query($link, $sql)){
    echo "Table created successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

The PHP code in the above example creates a table named *persons* with four columns *id*, *first_name*, *last_name* and *email* inside the *demo* database.

Notice that each field name is followed by a data type declaration; this declaration specifies what type of data the column can hold, whether integer, string, date, etc.

There are a few additional constraints (also called *modifiers*) that are specified after the column name in the preceding SQL statement, like NOT NULL, PRIMARY KEY, AUTO_INCREMENT, etc. Constraints define rules regarding the values allowed in columns.

Please check out the tutorial on [SQL CREATE TABLE statement](#) for the detailed information about syntax, as well as the data types and constraints available in MySQL database system

PHP MySQL INSERT Query

Inserting Data into a MySQL Database Table

The [INSERT INTO](#) statement is used to insert new rows in a database table.

TOPS Technologies

Let's make a SQL query using the `INSERT INTO` statement with appropriate values, after that we will execute this insert query through passing it to the PHP `mysqli_query()` function to insert data in table. Here's an example, which insert a new row to the `persons` table by specifying values for the `first_name`, `last_name` and `email` fields.

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('Peter', 'Parker', 'peterparker@mail.com')";
if(mysqli_query($link, $sql)){
    echo "Records inserted successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

Inserting Multiple Rows into a Table

You can also insert multiple rows into a table with a single insert query at once. To do this, include multiple lists of column values within the `INSERT INTO` statement, where column values for each row must be enclosed within parentheses and separated by a comma.

Let's insert few more rows into the `persons` table, like this:

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
```

TOPS Technologies

```
die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES
        ('John', 'Rambo', 'johnrambo@mail.com'),
        ('Clark', 'Kent', 'clarkkent@mail.com'),
        ('John', 'Carter', 'johncarter@mail.com'),
        ('Harry', 'Potter', 'harrypotter@mail.com');

if(mysqli_query($link, $sql)){
    echo "Records added successfully.";
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

Now, go to phpMyAdmin (<http://localhost/phpmyadmin/>) and check out the *persons* table data inside *demo* database. You will find the value for the *id* column is assigned automatically by incrementing the value of previous *id* by 1.

Insert Data into a Database from an HTML Form

In the previous section, we have learned how to insert data into database from a PHP script. Now, we'll see how we can insert data into database obtained from an HTML form. Let's create an HTML form that can be used to insert new records to *persons* table.

Step 1: Creating the HTML Form

Here's a simple HTML form that has three text `<input>` fields and a submit button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Add Record Form</title>
</head>
<body>
<form action="insert.php" method="post">
    <p>
```

TOPS Technologies

```
<label for="firstName">First Name:</label>
<input type="text" name="first_name" id="firstName">
</p>
<p>
    <label for="lastName">Last Name:</label>
    <input type="text" name="last_name" id="lastName">
</p>
<p>
    <label for="emailAddress">Email Address:</label>
    <input type="text" name="email" id="emailAddress">
</p>
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Step 2: Retrieving and Inserting the Form Data

When a user clicks the submit button of the add record HTML form, in the example above, the form data is sent to 'insert.php' file. The 'insert.php' file connects to the MySQL database server, retrieves forms fields using the PHP `$_REQUEST` variables and finally execute the insert query to add the records. Here is the complete code of our 'insert.php' file:

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Escape user inputs for security
$first_name = mysqli_real_escape_string($link, $_REQUEST['first_name']);
$last_name = mysqli_real_escape_string($link, $_REQUEST['last_name']);
$email = mysqli_real_escape_string($link, $_REQUEST['email']);

// Attempt insert query execution
```

TOPS Technologies

```
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('$first_name', '$last_name', '$email')";  
if(mysqli_query($link, $sql)){  
    echo "Records added successfully.>";  
} else{  
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);  
}  
  
// Close connection  
mysqli_close($link);  
?>
```

PHP MySQL Last Inserted ID

How to Get the ID of Last Inserted Row

MySQL automatically generate an unique ID for the `AUTO_INCREMENT` column each time you insert a new record or row into the table. However, there are certain situations when you need that automatically generated ID to insert it into a second table. In these situations you can use the PHP `mysqli_insert_id()` function to retrieve the most recently generated ID, as shown in the upcoming example.

For this example we'll use the same `persons` table that we've created in the [PHP MySQL create tables](#) chapter, which has four columns `id`, `first_name`, `last_name` and `email`, where `id` is the primary key column and marked with `AUTO_INCREMENT` flag

```
<?php  
/* Attempt MySQL server connection. Assuming you are running MySQL  
server with default setting (user 'root' with no password) */  
$link = mysqli_connect("localhost", "root", "", "demo");  
  
// Check connection  
if($link === false){  
    die("ERROR: Could not connect. " . mysqli_connect_error());  
}  
  
// Attempt insert query execution  
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('Ron', 'Weasley', 'ronweasley@mail.com')";  
if(mysqli_query($link, $sql)){  
    // Obtain last inserted id  
    $last_id = mysqli_insert_id($link);  
    echo "Records inserted successfully. Last inserted ID is: " . $last_id;  
} else{
```

TOPS Technologies

```
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

PHP MySQL SELECT Query

Selecting Data From Database Tables

So far you have learnt how to create database and table as well as inserting data. Now it's time to retrieve data what have inserted in the preceding tutorial. The SQL [SELECT](#) statement is used to select the records from database tables. Its basic syntax is as follows:

```
SELECT column1_name, column2_name, columnN_name FROM table_name;
```

Let's make a SQL query using the [SELECT](#) statement, after that we will execute this SQL query through passing it to the PHP [mysqli_query\(\)](#) function to retrieve the table data.

Consider our *persons* database table has the following records:

	id	first_name	last_name	email
1	1	Peter	Parker	peterparker@mail.com
2	2	John	Rambo	johnrambo@mail.com
3	3	Clark	Kent	clarkkent@mail.com
4	4	John	Carter	johncarter@mail.com
5	5	Harry	Potter	harrypotter@mail.com

The PHP code in the following example selects all the data stored in the *persons* table (using the asterisk character (*) in place of column name selects all the data in the table).

```
<?php
```

TOPS Technologies

```
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection

if($link === false){

    die("ERROR: Could not connect. " . mysqli_connect_error());

}

// Attempt select query execution

$sql = "SELECT * FROM persons";

if($result = mysqli_query($link, $sql)){

    if(mysqli_num_rows($result) > 0){

        echo "<table>";

        echo "<tr>";

        echo "<th>id</th>";

        echo "<th>first_name</th>";

        echo "<th>last_name</th>";

        echo "<th>email</th>";

        echo "</tr>";

        while($row = mysqli_fetch_array($result)){

            echo "<tr>";

            echo "<td>" . $row['id'] . "</td>";

            echo "<td>" . $row['first_name'] . "</td>";

            echo "<td>" . $row['last_name'] . "</td>";

            echo "<td>" . $row['email'] . "</td>";

        }

    }

}
```

```
echo "</tr>";  
}  
  
echo "</table>";  
  
// Free result set  
  
mysqli_free_result($result);  
  
} else{  
  
echo "No records matching your query were found.";  
}  
  
} else{  
  
echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);  
}  
  
// Close connection  
  
mysqli_close($link);  
  
?>
```

Explanation of Code (Procedural style)

In the example above, the data returned by the `mysqli_query()` function is stored in the `$result` variable. Each time `mysqli_fetch_array()` is invoked, it returns the next row from the result set as an array. The [while loop](#) is used to loops through all the rows in the result set. Finally the value of individual field can be accessed from the row either by passing the field index or field name to the `$row` variable like `$row['id']` or `$row[0]`, `$row['first_name']` or `$row[1]`, `$row['last_name']` or `$row[2]`, and `$row['email']` or `$row[3]`.

If you want to use the [for loop](#) you can obtain the loop counter value or the number of rows returned by the query by passing the `$result` variable to the `mysqli_num_rows()` function. This loop counter value determines how many times the loop should run.

PHP MySQL WHERE Clause

Filtering the Records

The [WHERE](#) clause is used to extract only those records that fulfill a specified condition.

The basic syntax of the WHERE clause can be given with:

```
SELECT column_name(s) FROM table_name WHERE column_name operator value
```

TOPS Technologies

Let's make a SQL query using the WHERE clause in SELECT statement, after that we'll execute this query through passing it to the PHP `mysqli_query()` function to get the filtered data.

Consider we've a *persons* table inside the *demo* database that has following records:

id first_name last_name email			
+-----+-----+-----+			
1 Peter Parker peterparker@mail.com			
2 John Rambo johnrambo@mail.com			
3 Clark Kent clarkkent@mail.com			
4 John Carter johncarter@mail.com			
5 Harry Potter harrypotter@mail.com			
+-----+-----+-----+			

The following PHP code selects all the rows from the *persons* table where `first_name='john'`:

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection

if($link === false){

    die("ERROR: Could not connect. " . mysqli_connect_error());

}

// Attempt select query execution

$sql = "SELECT * FROM persons WHERE first_name='john'";

if($result = mysqli_query($link, $sql)){

    if(mysqli_num_rows($result) > 0){
```

TOPS Technologies

```
echo "<table>";

    echo "<tr>";

        echo "<th>id</th>";

        echo "<th>first_name</th>";

        echo "<th>last_name</th>";

        echo "<th>email</th>";

    echo "</tr>";

while($row = mysqli_fetch_array($result)){

    echo "<tr>";

        echo "<td>" . $row['id'] . "</td>";

        echo "<td>" . $row['first_name'] . "</td>";

        echo "<td>" . $row['last_name'] . "</td>";

        echo "<td>" . $row['email'] . "</td>";

    echo "</tr>";

}

echo "</table>";

// Close result set

mysqli_free_result($result);

} else{

    echo "No records matching your query were found. ";

}

} else{

    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);

}

// Close connection

mysqli_close($link);
```

?>

After filtration the result set will look something like this:

id	first_name	last_name	email
2	John	Rambo	johnrambo@mail.com
4	John	Carter	johncarter@mail.com

PHP MySQL LIMIT Clause

Limiting Result Sets

The **LIMIT** clause is used to constrain the number of rows returned by the **SELECT** statement. This feature is very helpful for optimizing the page loading time as well as to enhance the readability of a website. For example you can divide the large number of records in multiple pages using pagination, where limited number of records will be loaded on every page from the database when a user request for that page by clicking on pagination link.

The basic syntax of the **LIMIT** clause can be given with:

```
SELECT column_name(s) FROM table_name LIMIT row_offset, row_count;
```

The **LIMIT** clause accepts one or two parameters which must be a nonnegative integer:

- When two parameters are specified, the first parameter specifies the offset of the first row to return i.e. the starting point, whereas the second parameter specifies the number of rows to return. The offset of the first row is 0 (not 1).
- Whereas, when only one parameter is given, it specifies the maximum number of rows to return from the beginning of the result set.

For example, to retrieve the first three rows, you can use the following query:

```
SELECT * FROM persons LIMIT 3;
```

To retrieve the rows 2-4 (inclusive) of a result set, you can use the following query:

```
SELECT * FROM persons LIMIT 1, 3;
```

Let's make a SQL query using the **LIMIT** clause in **SELECT** statement, after that we will execute this query through passing it to the PHP **mysqli_query()** function to get the limited number of records. Consider the following **persons** table inside the **demo** database:

TOPS Technologies

id first_name last_name email
1 Peter Parker peterparker@mail.com
2 John Rambo johnrambo@mail.com
3 Clark Kent clarkkent@mail.com
4 John Carter johnncarter@mail.com
5 Harry Potter harrypotter@mail.com

The PHP code in the following example will display just three rows from the *persons* table.

```
<?php

/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection

if($link === false){

    die("ERROR: Could not connect. " . mysqli_connect_error());

}

// Attempt select query execution

$sql = "SELECT * FROM persons LIMIT 3";

if($result = mysqli_query($link, $sql)) {

    if(mysqli_num_rows($result) > 0){

        echo "<table>";


```

TOPS Technologies

```
echo "<tr>";

    echo "<th>id</th>";

    echo "<th>first_name</th>";

    echo "<th>last_name</th>";

    echo "<th>email</th>";

    echo "</tr>",

while($row = mysqli_fetch_array($result)){

    echo "<tr>",

        echo "<td>" . $row['id'] . "</td>",

        echo "<td>" . $row['first_name'] . "</td>",

        echo "<td>" . $row['last_name'] . "</td>",

        echo "<td>" . $row['email'] . "</td>",

    echo "</tr>",

}

echo "</table>",

// Close result set

mysqli_free_result($result);

} else{

    echo "No records matching your query were found./";

}

echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
```

```
}
```

```
// Close connection
```

```
mysqli_close($link);
```

```
?>
```

After limiting the result set the output will look something like this:

id	first_name	last_name	email
1	Peter	Parker	peterparker@mail.com
2	John	Rambo	johnrambo@mail.com
3	Clark	Kent	clarkkent@mail.com

PHP MySQL ORDER BY Clause

Ordering the Result Set

The [ORDER BY](#) clause can be used in conjunction with the [SELECT](#) statement to see the data from a table ordered by a specific field. The ORDER BY clause lets you define the field name to sort against and the sort direction either ascending or descending.

The basic syntax of this clause can be given with:

```
SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC
```

Let's make a SQL query using the ORDER BY clause in SELECT statement, after that we will execute this query through passing it to the PHP `mysqli_query()` function to get the ordered data. Consider the following *persons* table inside the *demo* database:

TOPS Technologies

id first_name last_name email
1 Peter Parker peterparker@mail.com
2 John Rambo johnrambo@mail.com
3 Clark Kent clarkkent@mail.com
4 John Carter johnmorgan@mail.com
5 Harry Potter harrypotter@mail.com

The PHP code in the following example selects all rows from the *persons* table and sorts the result by the *first_name* column in the alphabetically ascending order.

```
<?php  
/* Attempt MySQL server connection. Assuming you are running MySQL  
server with default setting (user 'root' with no password) */  
$link = mysqli_connect("localhost", "root", "", "demo");  
  
// Check connection  
if($link === false){  
    die("ERROR: Could not connect. " . mysqli_connect_error());  
}  
  
// Attempt select query execution with order by clause  
$sql = "SELECT * FROM persons ORDER BY first_name";  
if($result = mysqli_query($link, $sql)){  
    if(mysqli_num_rows($result) > 0){  
        echo "<table>";  
        echo "<tr>";  
        echo "<th>id</th>";  
        echo "<th>first_name</th>";  
        echo "<th>last_name</th>";  
        echo "<th>email</th>";  
        echo "</tr>";  
        while($row = mysqli_fetch_array($result)){  
            echo "<tr>";  
            echo "<td>" . $row['id'] . "</td>";  
            echo "<td>" . $row['first_name'] . "</td>";  
            echo "<td>" . $row['last_name'] . "</td>";  
        }  
    }  
}
```

TOPS Technologies

```
echo "<td>" . $row['email'] . "</td>";
echo "</tr>";
}
echo "</table>";
// Close result set
mysqli_free_result($result);
} else{
    echo "No records matching your query were found.";
}
} else{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
}

// Close connection
mysqli_close($link);
?>
```

After ordering the result, the result set will look something like this:

id first_name last_name email
3 Clark Kent clarkkent@mail.com
5 Harry Potter harrypotter@mail.com
2 John Rambo johnrambo@mail.com
4 John Carter johnncarter@mail.com
1 Peter Parker peterparker@mail.com

PHP MySQL UPDATE Query

Updating Database Table Data

The [UPDATE](#) statement is used to change or modify the existing records in a database table. This statement is typically used in conjugation with the [WHERE](#) clause to apply the changes to only those records that matches specific criteria.

The basic syntax of the [UPDATE](#) statement can be given with:

UPDATE table_name **SET** column1=value, column2=value2,... **WHERE** column_name=some_value

TOPS Technologies

Let's make a SQL query using the `UPDATE` statement and `WHERE` clause, after that we will execute this query through passing it to the PHP `mysqli_query()` function to update the tables records. Consider the following `persons` table inside the `demo` database:

id first_name last_name email
1 Peter Parker peterparker@mail.com
2 John Rambo johnrambo@mail.com
3 Clark Kent clarkkent@mail.com
4 John Carter johncarter@mail.com
5 Harry Potter harrypotter@mail.com

The PHP code in the following example will update the email address of a person in the `persons` table whose `id` is equal to 1.

```
<?php  
/* Attempt MySQL server connection. Assuming you are running MySQL  
server with default setting (user 'root' with no password) */  
$link = mysqli_connect("localhost", "root", "", "demo");  
  
// Check connection  
if($link === false){  
    die("ERROR: Could not connect. " . mysqli_connect_error());  
}  
  
// Attempt update query execution  
$sql = "UPDATE persons SET email='peterparker_new@mail.com' WHERE id=1";  
if(mysqli_query($link, $sql)){  
    echo "Records were updated successfully.";  
} else {  
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);  
}  
  
// Close connection  
mysqli_close($link);  
?>
```

After update the `persons` table will look something like this:

+-----+-----+-----+

id	first_name	last_name	email
1	Peter	Parker	peterparker_new@mail.com
2	John	Rambo	johnrambo@mail.com
3	Clark	Kent	clarkkent@mail.com
4	John	Carter	johncarter@mail.com
5	Harry	Potter	harrypotter@mail.com

PHP MySQL DELETE Query

Deleting Database Table Data

Just as you insert records into tables, you can delete records from a table using the SQL [DELETE](#) statement. It is typically used in conjunction with the `WHERE` clause to delete only those records that matches specific criteria or condition.

The basic syntax of the `DELETE` statement can be given with:

```
DELETE FROM table_name WHERE column_name=some_value
```

Let's make a SQL query using the `DELETE` statement and `WHERE` clause, after that we will execute this query through passing it to the PHP `mysqli_query()` function to delete the tables records. Consider the following `persons` table inside the `demo` database:

id	first_name	last_name	email
1	Peter	Parker	peterparker@mail.com
2	John	Rambo	johnrambo@mail.com
3	Clark	Kent	clarkkent@mail.com
4	John	Carter	johncarter@mail.com
5	Harry	Potter	harrypotter@mail.com

The PHP code in the following example will delete the records of those persons from the `persons` table whose `first_name` is equal to John.

TOPS Technologies

```
<?php  
/* Attempt MySQL server connection. Assuming you are running MySQL  
server with default setting (user 'root' with no password) */  
$link = mysqli_connect("localhost", "root", "", "demo");  
  
// Check connection  
if($link === false){  
    die("ERROR: Could not connect. " . mysqli_connect_error());  
}  
  
// Attempt delete query execution  
$sql = "DELETE FROM persons WHERE first_name='John"';  
if(mysqli_query($link, $sql)){  
    echo "Records were deleted successfully.";  
} else{  
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);  
}  
  
// Close connection  
mysqli_close($link);  
?>
```

After the deletion the *persons* table will look something like this:

id first_name last_name email
1 Peter Parker peterparker@mail.com
3 Clark Kent clarkkent@mail.com
5 Harry Potter harrypotter@mail.com

As you can see the records has been deleted successfully from the persons table.

SQL Join

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

The simplest Join is INNER JOIN.

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

```
INNER JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

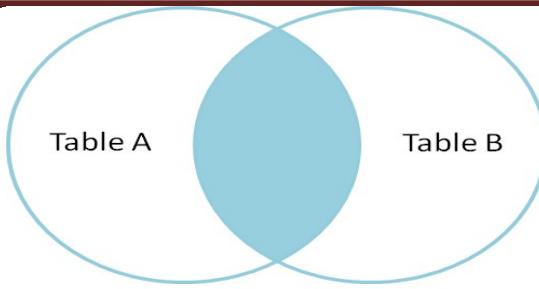
table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

TOPS Technologies



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student INNER JOIN StudentCourse  
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

LEFT JOIN: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN. Syntax:

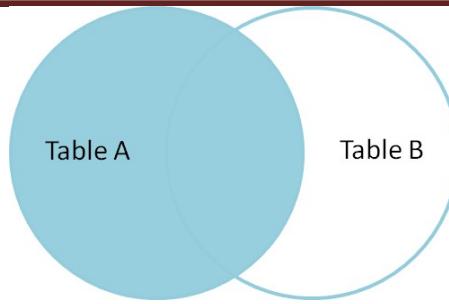
```
SELECT table1.column1,table1.column2,table2.column1,...  
FROM table1  
LEFT JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
LEFT JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL

RIGHT JOIN: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN. Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,...
```

```
FROM table1
```

```
RIGHT JOIN table2
```

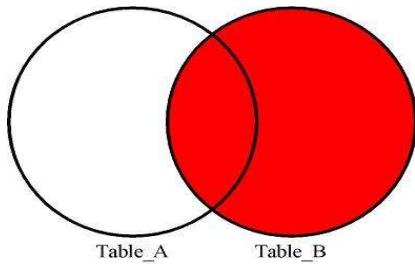
```
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example

Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
RIGHT JOIN StudentCourse
```

```
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

FULL JOIN: FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain **NULL** values. Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
```

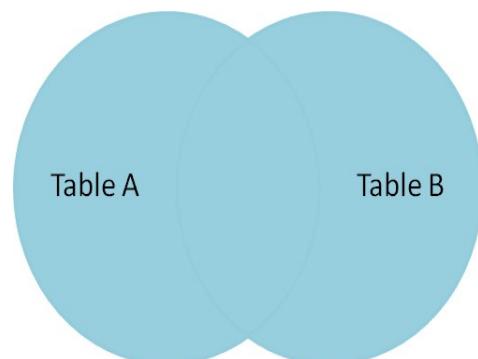
```
FULL JOIN table2
```

```
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.



Example Queries (FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
FULL JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	9
NULL	10
NULL	11

SQL Injection

SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

TOPS Technologies

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string.

The variable is fetched from user input (getRequestString):

Example:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

The rest of this chapter describes the potential dangers of using user input in SQL statements.

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId:

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

SQL Injection Based on ""="" is Always True

Here is an example of a user login on a web site:

Username:

Password:

Example

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
```

```
sql = 'SELECT * FROM Users WHERE Name =' + uName + ' AND Pass =' + uPass + '''
```

TOPS Technologies

Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""="" into the user name or password text box:

User Name:

" or ""=""

Password:

" or ""=""

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE.

SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

```
SELECT * FROM Users; DROP TABLE Suppliers
```

Look at the following example:

```
txtUserId = getRequestString("UserId");
```

```
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

And the following input:

User id: **105; DROP TABLE Suppliers**

The valid SQL statement would look like this:

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

TOPS Technologies

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this –

HTTP/1.1 200 OK

Date: Fri, 04 Feb 2000 21:03:38 GMT

Server: Apache/1.3.9 (UNIX) PHP/4.0b3

Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;

path=/; domain=tutorialspoint.com

Connection: close

Content-Type: text/html

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this –

GET / HTTP/1.0

Connection: Keep-Alive

User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)

Host: zink.demon.co.uk:1126

Accept: image/gif, */*

Accept-Encoding: gzip

Accept-Language: en

Accept-Charset: iso-8859-1,*;utf-8

Cookie: name=xyz

A PHP script will then have access to the cookie in the environmental variables `$_COOKIE` or `$HTTP_COOKIE_VARS[]` which holds all cookie names and values. Above cookie can be accessed using `$HTTP_COOKIE_VARS["name"]`.

TOPS Technologies

Setting Cookies with PHP

PHP provided setcookie() function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments –

- Name – This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- Value – This sets the value of the named variable and is the content that you actually want to store.
- Expiry – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- Path – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- Domain – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- Security – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies name and age these cookies will be expired after one hour.

```
<?php  
    setcookie("name", "John Watkin", time()+3600, "/", "", 0);  
    setcookie("age", "36", time()+3600, "/", "", 0);  
?  
<html>  
  
<head>  
    <title>Setting Cookies with PHP</title>  
</head>  
  
<body>  
    <?php echo "Set Cookies"?>  
</body>  
  
</html>
```

TOPS Technologies

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables. Following example will access all the cookies set in above example.

```
<html>

<head>
    <title>Accessing Cookies with PHP</title>
</head>

<body>

<?php
    echo $_COOKIE["name"]. "<br />";

    /* is equivalent to */
    echo $HTTP_COOKIE_VARS["name"]. "<br />";

    echo $_COOKIE["age"] . "<br />";

    /* is equivalent to */
    echo $HTTP_COOKIE_VARS["age"] . "<br />";
?

</body>
</html>
```

You can use `isset()` function to check if a cookie is set or not.

```
<html>

<head>
    <title>Accessing Cookies with PHP</title>
</head>

<body>

<?php
    if( isset($_COOKIE["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
```

```
else
    echo "Sorry... Not recognized" . "<br />";
?>

</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```
<?php
    setcookie( "name", "", time()- 60, "/", "", 0);
    setcookie( "age", "", time()- 60, "/", "", 0);
?>
<html>

<head>
    <title>Deleting Cookies with PHP</title>
</head>

<body>
    <?php echo "Deleted Cookies" ?>
</body>

</html>
```

PHP Sessions

Although you can store data using cookies but it has some security issues. Since cookies are stored on user's computer it is possible for an attacker to easily modify a cookie content to insert potentially harmful data in your application that might break your application.

Also every time the browser requests a URL to the server, all the cookie data for a website is automatically sent to the server within the request. It means if you have stored 5 cookies on user's system, each having 4KB in size, the browser needs to upload 20KB of data each time the user views a page, which can affect your site's performance.

TOPS Technologies

You can solve both of these issues by using the PHP session. A PHP session stores data on the server rather than user's computer. In a session based environment, every user is identified through a unique number called session identifier or SID. This unique session ID is used to link each user with their own information on the server like emails, posts, etc.

Starting a PHP Session

Before you can store any information in session variables, you must first start up the session. To begin a new session, simply call the PHP `session_start()` function. It will create a new session and generate a unique session ID for the user.

```
<?php  
// Starting session  
session_start();  
?>
```

The `session_start()` function first checks to see if a session already exists by looking for the presence of a session ID. If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID.

Storing and Accessing Session Data

You can store all your session data as key-value pairs in the `$_SESSION[]` superglobal array. The stored data can be accessed during lifetime of a session. Consider the following script, which creates a new session and registers two session variables.

```
<?php  
// Starting session  
session_start();  
  
// Storing session data  
$_SESSION["firstname"] = "Peter";  
$_SESSION["lastname"] = "Parker";  
?>
```

To access the session data we set on our previous example from any other page on the same web domain — simply recreate the session by calling `session_start()` and then pass the corresponding key to the `$_SESSION` associative array.

```
<?php  
// Starting session  
session_start();
```

TOPS Technologies

```
// Accessing session data
echo 'Hi, ' . $_SESSION["firstname"] . ' ' . $_SESSION["lastname"];
?>
```

The PHP code in the example above produce the following output.

Hi, Peter Parker

Destroying a Session

If you want to remove certain session data, simply unset the corresponding key of the `$_SESSION` associative array, as shown in the following example:

```
<?php
// Starting session
session_start();

// Removing session data
if(isset($_SESSION["lastname"])){
    unset($_SESSION["lastname"]);
}
?>
```

However, to destroy a session completely, simply call the `session_destroy()` function. This function does not need any argument and a single call destroys all the session data.

```
<?php
// Starting session
session_start();

// Destroying session
session_destroy();
?>
```

Every PHP session has a timeout value — a duration, measured in seconds — which determines how long a session should remain alive in the absence of any user activity. You can adjust this timeout duration by changing the value of `session.gc_maxlifetime` variable in the PHP configuration file (`php.ini`).

PHP File Upload

Uploading Files with PHP

TOPS Technologies

In this tutorial we will learn how to upload files on remote server using a Simple HTML form and PHP. You can upload any kind of file like images, videos, ZIP files, Microsoft Office documents, PDFs, as well as executables files and a wide range of other file types.

Step 1: Creating an HTML form to upload the file

The following example will create a simple HTML form that can be used to upload files.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>File Upload Form</title>
</head>
<body>
    <form action="upload-manager.php" method="post" enctype="multipart/form-data">
        <h2>Upload File</h2>
        <label for="fileSelect">Filename:</label>
        <input type="file" name="photo" id="fileSelect">
        <input type="submit" name="submit" value="Upload">
        <p><strong>Note:</strong> Only .jpg, .jpeg, .gif, .png formats allowed to a max size of 5 MB.</p>
    </form>
</body>
</html>
```

Note: In addition to a [file-select](#) field the upload form must use the [HTTP post](#) method and must contain an `enctype="multipart/form-data"` attribute. This attribute ensures that the form data is encoded as multipart MIME data — which is required for uploading the large quantities of binary data such as image, audio, video, etc

Step 2: Processing the uploaded file

Here's the complete code of our "upload-manager.php" file. It will store the uploaded file in a "upload" folder on permanent basis as well as implement some basic security check like file type and file size to ensure that users upload the correct file type and within the allowed limit.

```
<?php
// Check if the form was submitted
if($_SERVER["REQUEST_METHOD"] == "POST"){
    // Check if file was uploaded without errors
```

TOPS Technologies

```
if(isset($_FILES["photo"]) && $_FILES["photo"]["error"] == 0){

    $allowed = array("jpg" => "image/jpg", "jpeg" => "image/jpeg", "gif" => "image/gif", "png" => "image/png");

    $filename = $_FILES["photo"]["name"];

    $filetype = $_FILES["photo"]["type"];

    $filesize = $_FILES["photo"]["size"];



    // Verify file extension

    $ext = pathinfo($filename, PATHINFO_EXTENSION);

    if(!array_key_exists($ext, $allowed)) die("Error: Please select a valid file format.");



    // Verify file size - 5MB maximum

    $maxsize = 5 * 1024 * 1024;

    if($filesize > $maxsize) die("Error: File size is larger than the allowed limit.");



    // Verify MYME type of the file

    if(in_array($filetype, $allowed)){

        // Check whether file exists before uploading it

        if(file_exists("upload/" . $filename)){

            echo $filename . " is already exists.";

        } else{

            move_uploaded_file($_FILES["photo"]["tmp_name"], "upload/" . $filename);

            echo "Your file was uploaded successfully.";

        }

    } else{

        echo "Error: There was a problem uploading your file. Please try again.";

    }

} else{
```

TOPS Technologies

```
echo "Error: " . $_FILES["photo"]["error"];  
}  
}  
?>
```

Note: The above script prevents uploading a file with the same name as an existing file in the same folder. However, if you want to allow this just prepend the file name with a random string or timestamp, like \$filename = time() . '_' . \$_FILES["photo"]["name"];

You might be wondering what this code was all about. Well, let's go through each part of this example code one by one for a better understanding of this process.

Explanation of Code

Once the form is submitted information about the uploaded file can be accessed via PHP superglobal array called `$_FILES`. For example, our upload form contains a file select field called `photo` (i.e. `name="photo"`), if any user uploaded a file using this field, we can obtain its details like the name, type, size, temporary name or any error occurred while attempting the upload via the `$_FILES["photo"]` associative array, like this:

- `$_FILES["photo"]["name"]` — This array value specifies the original name of the file, including the file extension. It doesn't include the file path.
- `$_FILES["photo"]["type"]` — This array value specifies the MIME type of the file.
- `$_FILES["photo"]["size"]` — This array value specifies the file size, in bytes.
- `$_FILES["photo"]["tmp_name"]` — This array value specifies the temporary name including full path that is assigned to the file once it has been uploaded to the server.
- `$_FILES["photo"]["error"]` — This array value specifies error or status code associated with the file upload, e.g. it will be 0, if there is no error.

The PHP code in the following example will simply display the details of the uploaded file and stores it in a temporary directory on the web server.

```
<?php  
  
if($_FILES["photo"]["error"] > 0){  
  
    echo "Error: " . $_FILES["photo"]["error"] . "<br>";  
  
} else{  
  
    echo "File Name: " . $_FILES["photo"]["name"] . "<br>";  
  
    echo "File Type: " . $_FILES["photo"]["type"] . "<br>";  
  
    echo "File Size: " . ($_FILES["photo"]["size"] / 1024) . " KB<br>";
```

```
echo "Stored in: " . $_FILES["photo"]["tmp_name"];  
}  
  
?>
```

Tip: Once a file has been successfully uploaded, it is automatically stored in a temporary directory on the server.

To store this file on a permanent basis, you need to move it from the temporary directory to a permanent location using the PHP's `move_uploaded_file()` function.

Module – 6[Advance PHP]

Classes, objects, methods and properties

Object-oriented programming is a programming style in which it is customary to group all of the variables and functions of a particular topic into a single class. Object-oriented programming is considered to be more advanced and efficient than the procedural style of programming. This efficiency stems from the fact that it supports better code organization, provides modularity, and reduces the need to repeat ourselves. That being said, we may still prefer the procedural style in small and simple projects. However, as our projects grow in complexity, we are better off using the object oriented style.

- classes
- objects
- methods
- properties

How to create classes?

In order to create a class, we group the code that handles a certain topic into one place. For example, we can group all of the code that handles the users of a blog into one class, all of the code that is involved with the publication of the posts in the blog into a second class, and all the code that is devoted to comments into a third class.

To name the class, it is customary to use a singular noun that starts with a capital letter. For example, we can group a code that handles users into a `User` class, the code that handles posts into a `Post` class, and the code that is devoted to comments into a `Comment` class.

For the example given below, we are going to create a `Car` class into which we will group all of the code which

TOPS Technologies

has something to do with cars.

```
class Car {  
    // The code  
}
```

- We declare the class with the **class** keyword.
- We write the name of the class and capitalize the first letter.
- If the class name contains more than one word, we capitalize each word. This is known as upper camel case. For example, **JapaneseCars**, **AmericanIdol**, **EuropeTour**, etc.
- We circle the class body within curly braces. Inside the curly braces, we put our code.

How to add properties to a class?

We call properties to the variables inside a class. Properties can accept values like strings, integers, and booleans (true/false values), like any other variable. Let's add some properties to the **Car** class.

```
class Car {  
    public $comp;  
    public $color = 'beige';  
    public $hasSunRoof = true;  
}
```

- We put the **public** keyword in front of a class property.
- The naming convention is to start the property name with a lower case letter.
- If the name contains more than one word, all of the words, except for the first word, start with an upper case letter. For example, **\$color** or **\$hasSunRoof**.
- A property can have a default value. For example, **\$color = 'beige'**.
- We can also create a property without a default value. See the property **\$comp** in the above example.

How to create objects from a class?

We can create several objects from the same class, with each object having its own set of properties.

In order to work with a class, we need to create an object from it. In order to create an object, we use the **new** keyword. For example:

TOPS Technologies

```
$bmw = new Car();
```

- We created the object **\$bmw** from the class **Car** with the **new** keyword.
- The process of creating an object is also known as instantiation.

We can create more than one object from the same class.

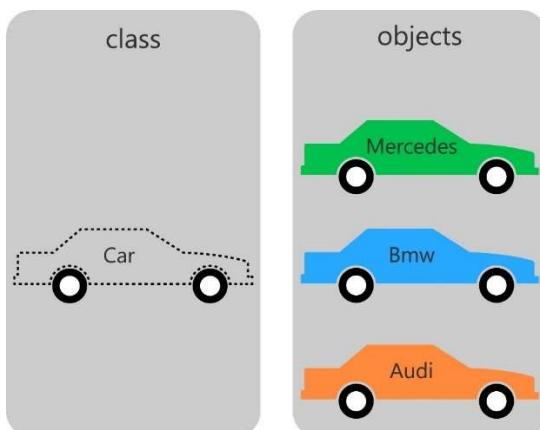
```
$bmw = new Car();  
$mercedes = new Car();
```

In fact, we can create as many objects as we like from the same class, and then give each object its own set of properties.

Objects, what are they good for?

While in the procedural style of programming, all of the functions and variables sit together in the global scope in a way that allows their use just by calling their name, the use of classes makes anything inside the classes hidden from the global scope. That's because the code inside the classes is encapsulated within the class scope, outside the reach of the global scope. So, we need a way to allow the code from the global scope to use the code within the class, and we do this by creating objects from a class.

I say objects, and not object, because we can create as many objects as we would like from the same class and they all will share the class's methods and properties. See the image below:



From the same [Car](#) class, we created three individual objects with the name of: [Mercedes](#), [Bmw](#), and [Audi](#).

Although all of the objects were created from the same class and thus have the class's methods and properties, they are still different. This is not only, because they have different names, but also because they may have different values assigned to their properties. For example, in the image above, they differ by the color property - the Mercedes is green while the Bmw is blue and the Audi is orange.

THE MESSAGE TO TAKE HOME IS:

A class holds the methods and properties that are shared by all of the objects that are created from it.

Although the objects share the same code, they can behave differently because they can have different values assigned to them.

[How to get an object's properties?](#)

Once we create an object, we can get its properties. For example:

```
echo $bmw -> color;
```

```
echo $mercedes -> color;
```

- In order to get a property, we write the object name, and then dash greater than (->), and then the property name.
- Note that the property name does not start with the \$ sign; only the object name starts with a \$

Result:

beige

Beige

[How to set the object's properties?](#)

In order to set an object property, we use a similar approach.

For example, in order to set the color to 'blue' in the bmw object:

TOPS Technologies

```
$bmw -> color = 'blue';
```

and in order to set the value of the `$comp` property for both objects:

```
$bmw -> comp = "BMW";  
$mercedes -> comp = "Mercedes Benz";
```

Once we set the value of a property, we can get its value.

In order to get the color of the `$bmw` object, we use the following line of code:

```
echo $bmw -> color;
```

Result:

```
blue
```

We can also get the company name and the color of the second car object.

```
echo $mercedes -> color;  
echo $mercedes -> comp;
```

Result:

```
beige
```

```
Mercedes Benz
```

How to add methods to a class?

The classes most often contain functions. A function inside a class is called a method. Here we add the method `hello()` to the class with the prefix `public`.

```
class Car {  
  
    public $comp;  
    public $color = 'beige';  
    public $hasSunRoof = true;  
  
    public function hello()  
    {  
        return "beep";
```

```
}
```

```
}
```

We put the **public** keyword in front of a method.

- The naming convention is to start the function name with a lower case letter.
- If the name contains more than one word, all of the words, except for the first word, start with an upper case letter. For example, **helloUser()** or **flyPanAm()**.

We can approach the methods similar to the way that we approach the properties, but we first need to create at least one object from the class.

```
$bmw = new Car();
```

```
$mercedes = new Car();
```

```
echo $bmw -> hello();
```

```
echo $mercedes -> hello();
```

Result:

```
beep
```

```
Beep
```

Here is the full code

```
<?php
// Declare the class
class Car {

    // properties
    public $comp;
    public $color = 'beige';
    public $hasSunRoof = true;

    // method that says hello
    public function hello()
    {
        return "beep";
    }
}

// Create an instance
```

TOPS Technologies

```
$bmw = new Car();  
$mercedes = new Car();  
  
// Get the values  
echo $bmw -> color; // beige  
echo "<br />";  
echo $mercedes -> color; // beige  
echo "<hr />";  
  
// Set the values  
$bmw -> color = 'blue';  
$bmw -> comp = "BMW";  
$mercedes -> comp = "Mercedes Benz";  
  
// Get the values again  
echo $bmw -> color; // blue  
echo "<br />";  
echo $mercedes -> color; // beige  
echo "<br />";  
echo $bmw -> comp; // BMW  
echo "<br />";  
echo $mercedes -> comp; // Mercedes Benz  
echo "<hr />";  
  
// Use the methods to get a beep  
echo $bmw -> hello(); // beep  
echo "<br />";  
echo $mercedes -> hello(); // beep
```

The \$this keyword

The \$this keyword

The \$this keyword indicates that we use the class's own methods and properties, and allows us to have access to them within the class's scope.

TOPS Technologies

The **\$this** keyword allows us to approach the class properties and methods from within the class using the following syntax:

- Only the **\$this** keyword starts with the **\$** sign, while the names of the properties and methods do not start with it.

\$this -> propertyName;

\$this -> methodName();

The **\$this** keyword indicates that we use the class's own methods and properties, and allows us to have access to them within the class's scope.

Let's illustrate what we have just said on the **Car** class. We will enable the **hello()** method to approach the class's own properties by using the **\$this** keyword.

In order to approach the class **\$comp** property. We use:

\$this -> comp

In order to approach the class **\$color** property. We use:

\$this -> color

That's what the code looks like:

```
class Car {
```

```
// The properties
```

```
public $comp;
```

```
public $color = 'beige';
```

```
public $hasSunRoof = true;
```

```
// The method can now approach the class properties

// with the $this keyword

public function hello()

{

    return "Beep I am a <i>" . $this -> comp . "</i>, and I am <i>" .

    $this -> color;

}

}
```

Let us create two objects from the class:

```
$bmw = new Car();
```

```
$mercedes = new Car();
```

and set the values for the class properties:

```
$bmw -> comp = "BMW";
```

```
$bmw -> color = "blue";
```

```
$mercedes -> comp = "Mercedes Benz";
```

```
$mercedes -> color = "green";
```

We can now call the [hello\(\)](#) method for the first car object:

```
echo $bmw -> hello();
```

TOPS Technologies

Result:

Beep I am a *BMW*, and I am *blue*.

And for the second car object.

```
echo $mercedes -> hello();
```

Result:

Beep I am a *Mercedes Benz*, and I am *green*.

Code

```
class Car {  
  
    // The properties  
    public $comp;  
    public $color = 'beige';  
    public $hasSunRoof = true;  
  
    // The method that says hello  
    public function hello()  
    {  
        return "Beep I am a <i>" . $this -> comp .  
            "</i>, and I am <i>" . $this -> color;  
    }  
}  
  
// We can now create an object from the class.  
$bmw = new Car();  
$mercedes = new Car();  
  
// Set the values of the class properties.  
$bmw -> color = 'blue';  
$bmw -> comp = "BMW";  
$mercedes -> comp = "Mercedes Benz";  
  
// Call the hello method for the $bmw object.  
echo $bmw -> hello();
```

Chaining methods and properties

This **method** is called **Method Chaining** in PHP's terminology. Each **method** in class in **Method Chaining**, that is, the **method** of the class returns the object of that class. For **Method Chaining**, instead of writing value return in class, we have to write return \$this;

we learned to use the \$this keyword to approach properties and methods within the scope of the class. In this chapter we will learn that, when a class's methods return the \$this keyword, they can be chained together to create much more streaming code.

For instance, in our Car class, let's say that we want to measure how much fuel we have in our car's tank. The amount of fuel in the tank is dependent upon the number of miles we have driven in our car, as well as the amount of fuel that we put in the tank.

In order to achieve our goal, we are going to put a public property \$tank to our class that represents the number of gallons of fuel that we have in the car's tank.

```
class Car {  
    public $tank;  
}
```

We must also add two methods to our Car class:

1. The `fill()` method adds gallons of fuel to our car's tank.
2. The `ride()` method calculates how much fuel we consume when we ride a certain distance, and then subtracts it from the tank. In our example, we assume that the car consumes 1 gallon of fuel every 50 miles.

```
class Car {  
  
    public $tank;  
  
    // Add gallons of fuel to the tank when we fill it.  
    public function fill($float)  
    {  
        $this->tank += $float;  
    }  
  
    // Subtract gallons of fuel from the tank as we ride the car.  
    public function ride($float)
```

TOPS Technologies

```
{  
    $miles = $float;  
    $gallons = $miles/50;  
    $this-> tank -= $gallons;  
}  
}
```

As we would like our code to look elegant, we will chain the methods and properties. Note the arrows in the code.

```
$tank = $car -> fill(10) -> ride(40) -> tank;
```

In words: How much fuel do we have left in our tank after putting in 10 gallons, and driving 40 miles?

In order for us to be able to perform the chaining, the methods should return the object and, since we are inside the class, the methods should return the `$this` keyword.

In the code below, we can see how each method returns the `$this` keyword in order to allow the chaining.

```
class Car {  
  
    public $tank;  
  
    // Add gallons of fuel to the tank when we fill it.  
    public function fill($float)  
    {  
        $this-> tank += $float;  
  
        return $this;  
    }  
  
    // Subtract gallons of fuel from the tank as we ride the car.  
    public function ride($float)  
    {  
        $miles = $float;  
        $gallons = $miles/50;  
        $this-> tank -= ($gallons);  
    }  
}
```

```
    return $this;  
}  
}
```

Now, we can create an object from the [Car](#) class with the name of `$bmw` and find out the number of gallons of fuel left in our car's tank after we have filled the tank with 10 gallons of fuel and driven 40 miles.

```
// Create a new object from the Car class.  
$bmw = new Car();  
  
// Add 10 gallons of fuel, then ride 40 miles,  
// and get the number of gallons in the tank.  
$tank = $bmw -> fill(10) -> ride(40) -> tank;  
  
// Print the results to the screen.  
echo "The number of gallons left in the tank: " . $tank . " gal.;"
```

Result:

The number of gallons left in the tank: 9.2 gal.

[Access modifiers: public vs. private](#)

we used the public access modifier in front of the methods and properties in our classes without any explanation. The public access modifier is only one of several modifiers that we use. In this tutorial, we will learn about another modifier called the private access modifier.

While the public access modifier allows a code from outside or inside the class to access the class's methods and properties, the private modifier prevents access to a class's methods or properties from any code that is outside the class.

[The public access modifier](#)

The following example should already be familiar to you. In the example, the class's property and method are defined as [public](#), so the code outside the class can directly interact with them.

```
<?php
```

TOPS Technologies

```
class Car {  
  
    // public methods and properties.  
  
    public $model;  
  
    public function getModel()  
  
        return "The car model is " . $this -> model;  
  
    }  
  
}  
  
$mercedes = new Car();  
  
//Here we access a property from outside the class  
  
$mercedes -> model = "Mercedes";  
  
//Here we access a method from outside the class  
  
echo $mercedes -> getModel();
```

Result:

The car model is Mercedes

The private access modifier

We can prevent access to the properties and methods inside our classes if we define them with the **private** access modifier instead of the **public** access modifier.

In the following example, we define the property **\$model** as private, and when we try to set its value from outside the class, we encounter a fatal error.

```
<?php
```

TOPS Technologies

```
class Car {  
  
    //private  
  
    private $model;  
  
    public function getModel()  
  
    {  
  
        return "The car model is " . $this -> model;  
  
    }  
  
}  
  
$mercedes = new Car();  
  
// We try to access a private property from outside the class.  
  
$mercedes -> model = "Mercedes benz";  
  
echo $mercedes -> getModel();  
  
?>
```

Result:

```
Fatal error: Cannot access private property Car::$model
```

How to access a private property?

We saw that we have no access to private properties from outside the class, but we still have to somehow set and

TOPS Technologies

get the properties' values. In order to interact with private properties, we use public methods because they can interact with both the code outside of the class's scope as well as the code inside the class. The public methods that can interact in this manner are commonly divided into two kinds of methods:

Setters that set the values of the private properties.

Getters that get the values of the private properties.

In the following example, we will see that we can get and set the value of a private property, `$carModel`, through the use of setter and getter methods. We will use the `setModel()` method in order to set the value of the car model, and the `getModel()` method to get the value of the property.

```
<?php

class Car {

    //the private access modifier denies access to the method from outside the class's scope

    private $model;

    //the public access modifier allows the access to the method from outside the class

    public function setModel($model)

    {

        $this -> model = $model;

    }

    public function getModel()

    {

        return "The car model is " . $this -> model;

    }

}
```

```
}

}

$mercedes = new Car();

//Sets the car's model

$mercedes -> setModel("Mercedes benz");

//Gets the car's model

echo $mercedes -> getModel();

?>
```

Result:

```
The car model is Mercedes benz
```

Why do we need access modifiers?

We need access modifiers in order to limit the modifications that code from outside the classes can do to the classes' methods and properties. Once we define a property or method as **private**, only methods that are within the class are allowed to approach it. So, in order to interact with private methods and properties, we need to provide public methods. Inside these methods, we can put logic that can validate and restrict data that comes from outside the class.

In our example, we can validate that only certain car models can make their way, and be assigned to the private **\$model** property, by defining the allowed alternatives for models in the public **setModel()** method. For this purpose, we define inside the **setModel()** method an array of allowed car models, and check that only these models are assigned to the **\$model** property.

```
<?php
```

```
class Car {
```

TOPS Technologies

//the private access modifier denies access to the property from outside the class's scope

```
private $model;
```

//the public access modifier allows the access to the method from outside the class

```
public function setModel($model)
```

```
{
```

//validate that only certain car models are assigned to the \$carModel property

```
$allowedModels = array("Mercedes benz","BMW");
```

```
if(in_array($model,$allowedModels))
```

```
{
```

```
    $this -> model = $model;
```

```
}
```

```
else
```

```
{
```

```
    $this -> model = "not in our list of models.;"
```

```
}
```

```
}
```

```
public function getModel()
```

```
{
```

```
return "The car model is " . $this -> model;

}

}

$mercedes = new Car();

//Sets the car's model

$mercedes -> setModel("Mercedes benz");

//Gets the car's model

echo $mercedes -> getModel();

?>
```

Overloading in PHP

We are going to see a surprise in this PHP tutorial. We all know about overloading and it is a basic concept in OOPS and we have beaten it enough in our colleges. We have operator overloading, function overloading etc. When different operations are performed using the same operator, then it is called operator overloading. Similarly, if we define functions having an identical name but different set of arguments, then it is called function overloading.

In PHP we have to overload. A lot of people say different things like we do not have to overload in PHP or there is only partial support for overloading in PHP and what PHP does is not overloading. So what is my take? I am going to go by PHP bible, the php.net manual. Following is what the PHP manual says about overloading,

PHP's interpretation of "overloading" is different than most object oriented languages. Overloading traditionally provides the ability to have multiple methods with the same name but different quantities and types of arguments.

This is kind of tricky, it says PHP's interpretation. There is for sure something muddy here. I would say, let us not worry about whether it is the traditional OOPS overloading or not and just skip to PHP's overloading ;-)

PHP's Overloading

TOPS Technologies

PHP's overloading is to create dynamic entities. In this tutorial, we will understand about those dynamic entities, how they are created and how to access them.

Dynamic Entities on PHP Overloading

Properties and methods are those entities created dynamically by using PHP overloading. After creating an object for a class, we can access set of entities, that are, properties or methods not defined within the scope of the class. Such entities are said to be overloaded properties or methods, and the process is called as overloading.

For working with these overloaded properties or functions, [PHP magic methods](#) are used. Most of the magic methods will be triggered in object context except `__callStatic()` method which is used in static context.



PHP's Overloading Types

Overloading in PHP can be classified as,

1. Property Overloading
2. Method Overloading

Property Overloading

PHP property overloading allows us to create dynamic properties in the object context. For creating those properties no separate line of code is needed. A property associated with the class instance, and it is not declared within the scope of the class, is considered as overloaded property.

We can perform the following operations with overloaded properties in PHP.

- Setting and getting overloaded properties.
- Evaluating overloaded properties setting.
- Undo such properties setting.

Before performing the above three operations, we should define appropriate magic methods. Those methods are,

- `__set()` – triggered while initializing overloaded properties.
- `get()` – triggered while using overloaded properties with [PHP print statements](#)

TOPS Technologies

- __isSet() – This magic method is invoked when we check overloaded properties with isSet() function
- __unset() – Similarly, this function will be invoked on using [PHP unset\(\)](#) for overloaded properties.

Example: PHP Property Overloading

This PHP example is to create property array elements dynamically.

```
<?php

class Toys{

private $str;

public function __set($name,$value){

$this->str[$name] = $value;

}

public function __get($name){

echo "Overloaded Property name = " . $this->str[$name] . "<br/>";

}

public function __isSet($name){

if(isset($this->str[$name])){

echo "Property \$name is set.<br/>";

} else {

echo "Property \$name is not set.<br/>";

}

}

public function __unset($name){

unset($this->str[$name]);

echo "\$name is unset <br/>";

}
```

```
}

}

$objToys = new Toys;

/* setters and getters on dynamic properties */

$objToys->overloaded_property = "new";

echo $objToys->overloaded_property . "\n\n";

/*Operations with dynamic properties values*/

isset($objToys->overloaded_property);

unset($objToys->overloaded_property);

isset($objToys->overloaded_property);

?>
```

In the above example PHP program, the dynamic property is created. While initializing this dynamic property, `__set()` is invoked with name and value pair to be initialized as the name and value of class property array element `$str`.

Added to that, `isset()` and `unset()` with overloaded property will trigger `__isset()` and `__unset()` magic methods. After `unset()`, if we call `isset()`, then it will print “property not set” message to the browser.

Method Overloading

This type of overloading is for creating dynamic methods that are not declared within the class scope. PHP method overloading also triggers magic methods dedicated for the appropriate purpose.

Unlike property overloading, PHP method overloading allows function call in both object and static context. The related magic functions are,

- `__call()` – triggered while invoking overloaded methods in the object context.
- `__callStatic()` – triggered while invoking overloaded methods in static context.

PHP example for Method Overloading

TOPS Technologies

Let us call the undefined class function with both object reference and with the class name itself. For accessing function from outside class with the name of the class itself, it should be a static member of that class. So accessing some overloaded method with the name of the class will trigger static magic member defined within the class. For example,

```
<?php

class Toys

{

public function __call($name,$param){

echo "Magic method invoked while method overloading with object reference<br/>";

}

public static function __callStatic($name,$param){

echo "Magic method invoked while method overloading with static access<br/>";

}

$objToys = new Toys;

$objToys->overloaded_method();

Toys::overloaded_property();

?>
```

Final keyword in PHP

Final keyword in PHP is used in different context. The final keyword is used only for methods and classes.

Final Methods → **Prevent Method Overriding**

Final Classes → **Prevent Inheritance**

Final methods: When a method is declared as final then overriding on that method can not be performed. Methods are declared as final due to some design reasons. Method should not be overridden due to security or any other reasons.

```
<?php
```

```
// Program to understand use of
// final keyword for methods

class Base {

    // Final method
    final function printdata() {
        echo " Base class final printdata function";
    }

    // Non final method
    function nonfinal() {
        echo "\n This is nonfinal function of base class";
    }
}

// Class that extend base class
class Derived extends Base {

    // Inheriting method nonfinal
    function nonfinal() {
        echo "\n Derived class non final function";
    }

    // Here printdata function can
    // not be overridden
}

$obj = new Derived;
$obj->printdata();
$obj->nonfinal();
?>
```

Output:

```
Base class final printdata function
```

```
Derived class non final function
```

Final Classes: A class declared as final can not be extended in future. Classes are declared as final due to some design level issue. Creator of class declare that class as final if he want that class should not be inherited due to some security or other reasons. A final class can contain final as well as non final methods.

TOPS Technologies

But there is no use of final methods in class when class is itself declared as final because inheritance is not possible.

Example:

```
<?php

// Program to understand final classes
// in php
final class Base {

    // Final method
    final function printdata() {
        echo "final base class final method";
    }

    // Non final method
    function nonfinal() {
        echo "\nnon final method of final base class";
    }
}

$obj = new Base;
$obj->printdata();
$obj->nonfinal();

/* If we uncomment these lines then it will
show Class Derived may not inherit from final
class (Base)
class Derived extends Base {

} */
?>
```

Output:

```
final base class final method
non final method of final base class
```

Scope Resolution operator in PHP

The scope resolution operator also known as *Paamayim Nekudotayim* or more commonly known as the double colon is a token that allows access to static, constant, and overridden properties or methods of a class.

TOPS Technologies

It is used to refer to blocks or codes in context to classes, objects, etc. An identifier is used with the scope resolution operator. The most common example of the application of the scope resolution operator in PHP is to access the properties and methods of the class.

The following examples show the usage of the scope resolution operator in various scenarios.

Example 1: This type of definition is used while defining constants within a class.

```
<?php
```

```
class democlass {  
    const PI = 3.14;  
}
```

```
echo democlass::PI;
```

```
?>
```

Output:

```
3.14
```

Example 2: Three special keywords self, parent, and static are used to access properties or methods from inside the class definition.

```
<?php
```

```
// Declaring parent class
```

```
class demo{
```

```
    public static $bar=10;
```

```
    public static function func(){
```

```
        echo static::$bar."\n";
```

```
    }
```

```
}
```

```
// Declaring child class
```

```
class Child extends demo{
```

```
    public static $bar=20;
```

```
}
```

```
// Calling for demo's version of func()  
demo::func();
```

```
// Calling for child's version of func()  
Child::func();
```

```
?>
```

Output:

```
10
```

```
20
```

Example 3: When an extending class overrides its parent's function, the compiler calls the child class's version of the method but it is up to the child class to call its parent's version of the method.

```
<?php
```

```
class demo{
```

```
    public function myfunc() {  
        echo "myfunc() of parent class\n";  
    }  
}
```

```
class child extends demo {
```

```
    public function myfunc(){  
  
        // Calling parent's version  
        // of myfunc() method  
        parent::myfunc();  
  
        echo "myfunc() of child class";  
    }  
}
```

```
$class = new child;  
$class -> myfunc()
```

```
?>
```

TOPS Technologies

Output:

```
myfunc() of parent class  
myfunc() of child class
```

Abstraction Interface

PHP - What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the **abstract** keyword:

Syntax

```
<?php  
abstract class ParentClass {  
    abstract public function someMethod1();  
    abstract public function someMethod2($name, $color);  
    abstract public function someMethod3() : string;  
}  
?>
```

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Let's look at an example:

```
<?php
```

TOPS Technologies

```
// Parent class
abstract class Car {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
    abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
    public function intro() : string {
        return "Choose German quality! I'm an $this->name!";
    }
}

class Volvo extends Car {
    public function intro() : string {
        return "Proud to be Swedish! I'm a $this->name!";
    }
}

class Citroen extends Car {
    public function intro() : string {
        return "French extravagance! I'm a $this->name!";
    }
}

// Create objects from the child classes
$audi = new Audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new Volvo("Volvo");
```

```
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

Output

Choose German quality! I'm an Audi!
Proud to be Swedish! I'm a Volvo!
French extravagance! I'm a Citroen!

Interface

An Interface allows the users to create programs, specifying the public methods that a class must implement, without involving the complexities and details of how the particular methods are implemented. It is generally referred to as the next level of abstraction. It resembles the abstract methods, resembling the abstract classes. An Interface is defined just like a class is defined but with the class keyword replaced by the interface keyword and just the function prototypes. The interface contains no data variables. The interface is helpful in a way that it ensures to maintain a sort of metadata for all the methods a programmer wishes to work on.

Creating an Interface

Following is an example of how to define an interface using the *interface* keyword.

```
<?php

interface MyInterfaceName {
    public function methodA();
    public function methodB();
}

?>
```

Few characteristics of an Interface are:

- An interface consists of methods that have no implementations, which means the interface methods are abstract methods.
- All the methods in interfaces must have public visibility scope.
- Interfaces are different from classes as the class can inherit from one class only whereas the class can implement one or more interfaces.

To implement an interface, use the **implements** operator as follows:

```
<?php
```

TOPS Technologies

```
class MyClassName implements MyInterfaceName{  
    public function methodA() {  
  
        // method A implementation  
    }  
    public function methodB(){  
  
        // method B implementation  
    }  
}
```

```
?>
```

Concrete Class: The class which implements an interface is called the Concrete Class. It must implement all the methods defined in an interface. Interfaces of the same name can't be implemented because of ambiguity error. Just like any class, an interface can be extended using the **extends** operator as follows:

```
<?php
```

```
interface MyInterfaceName1{
```

```
    public function methodA();  
  
}
```

```
interface MyInterfaceName2 extends MyInterfaceName1{
```

```
    public function methodB();  
}
```

```
?>
```

Example: -

```
<?php
```

```
interface MyInterfaceName{
```

```
    public function method1();  
    public function method2();
```

```
}
```

```
class MyClassName implements MyInterfaceName{
```

```
public function method1(){
    echo "Method1 Called" . "\n";
}
```

```
public function method2(){
    echo "Method2 Called". "\n";
}
}
```

```
$obj = new MyClassName;
$obj->method1();
$obj->method2();
```

```
?>
```

Output:

```
Method1 Called
```

```
Method2 Called
```

Advantages of PHP Interface

- An interface allows unrelated classes to implement the same set of methods, regardless of their positions in the class inheritance hierarchy.
- An interface can model multiple inheritances because a class can implement more than one interface whereas it can extend only one class.
- The implementation of an inheritance will save the caller from full implementation of object methods and focus on just the objects interface, therefore, the caller interface remains unaffected.

Traits

PHP - What are Traits?

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the **trait** keyword:

Syntax

```
<?php
```

TOPS Technologies

```
trait TraitName {  
    // some code...  
}  
?>
```

To use a trait in a class, use the `use` keyword:

Syntax

```
<?php  
class MyClass {  
    use TraitName;  
}  
?>
```

Let's look at an example:

```
<?php  
trait message1 {  
    public function msg1() {  
        echo "OOP is fun! ";  
    }  
}  
  
class Welcome {  
    use message1;  
}  
  
$obj = new Welcome();  
$obj->msg1();  
?>
```

Output

OOP is fun!

Type Hinting

- In simple word, type hinting means providing hints to function to only accept the given data type.
- In technical word we can say that Type Hinting is method by which we can force function to accept the desired data type.

In PHP, we can use type hinting for Object, Array and callable data type.

```
<?php
```

```
class a {  
    public $c= "hello world";  
}  
//create function with class name argument  
function display(a $a1)  
{
```

```
//call variable  
echo $a1->c;  
}  
display(new a());  
?>
```

localhost/example/oop.php

hello Tops

Inheritance

It is a concept of accessing the features of one class from another class. If we inherit the class features into another class, we can access both class properties. We can extends the features of a class by using 'extends' keyword.

- It supports the concept of **hierarchical classification**.
- Inheritance has three types, **single, multiple and multilevel Inheritance**.
- **PHP** supports only **single inheritance**, where only one class can be **derived from single parent class**.
- We can simulate multiple inheritance by using **interfaces**.

```
<?php
```

```
class a
```

```
{
```

```
    function fun1()
```

```
{
```

```
    echo "Tops";
```

```
}
```

```
}
```

```
class b extends a
```

```
{
```

```
    function fun2()
```

```
{  
    echo "SSSIT";  
}  
}  
  
$obj= new b();  
  
$obj->fun1();  
  
?>
```

Output

Tops

Constructor

PHP - The __construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (_)!

We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:

```
<?php  
class Fruit {  
    public $name;  
    public $color;  
  
    function __construct($name) {  
        $this->name = $name;  
    }  
    function get_name() {  
        return $this->name;  
    }  
}
```

TOPS Technologies

```
$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Output

Apple

```
<?php
class Fruit {
    public $name;
    public $color;

    function __construct($name, $color) {
        $this->name = $name;
        $this->color = $color;
    }
    function get_name() {
        return $this->name;
    }
    function get_color() {
        return $this->color;
    }
}
```

```
$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

Output

Apple
red

Destructor

PHP - The __destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

TOPS Technologies

If you create a `__destruct()` function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (`__`)!

The example below has a `__construct()` function that is automatically called when you create an object from a class, and a `__destruct()` function that is automatically called at the end of the script:

```
<?php  
class Fruit {  
    public $name;  
    public $color;  
  
    function __construct($name) {  
        $this->name = $name;  
    }  
    function __destruct() {  
        echo "The fruit is {$this->name}. "  
    }  
}  
  
$apple = new Fruit("Apple");  
?>
```

Output

The fruit is Apple.

Object Iteration Introduction

From PHP 5 onwards, it is possible to iterate through list of all visible items of an object. Iteration can be performed using `foreach` loop as well as `Iterator` interface. There is also `IteratorAggregate` interface in PHP, that can be used for this purpose

Using `foreach` loop

```
<?php  
class myclass{  
    private $var;  
    protected $var1;  
    public $x, $y, $z;  
    public function __construct(){
```

```
$this->var="private variable";  
  
$this->var1=TRUE;  
  
$this->x=100;  
  
$this->y=200;  
  
$this->z=300;  
  
}  
  
public function iterate(){  
  
foreach ($this as $key => $value) {  
  
print "$key => $value\n";  
  
}  
  
}  
  
}  
  
$obj = new myclass();  
  
foreach($obj as $key => $value) {  
  
print "$key => $value\n";  
  
}  
  
echo "\n";  
  
$obj->iterate();  
  
?>
```

Output

The output is as below –

```
x => 100  
y => 200  
z => 300  
  
var => private variable  
var1 => 1  
x => 100  
y => 200  
z => 300
```

To write to a log file and make a new one each day, you could use `date("j.n.Y")` as part of the filename.

```
//Something to write to txt log
$log = "User: ".$_SERVER['REMOTE_ADDR']. '-' .date("F j, Y, g:i a").PHP_EOL.
"Attempt: ".($result[0]['success']=='1'?'Success':'Failed').PHP_EOL.
"User: ".$username.PHP_EOL.
"-----".PHP_EOL;
//Save string to log, use FILE_APPEND to append.
file_put_contents('./log_'.date("j.n.Y").'.log', $log, FILE_APPEND);
```

PHP Magic Methods

Generally, for each PHP user defined function, it contains two portions, such as function definition and function call. In some special cases, [PHP functions](#) will have function declaration. For example, [PHP interfaces](#) functions or PHP abstract functions, that we have seen with [abstract access modifiers](#).

In PHP, we can define some special functions that will be called automatically. Such functions require no function call to execute the code inside these functions. With this special feature, they can be referred as magic functions or magic methods.

For using these magic methods in our PHP project, we need to be known about the following list of points.

- PHP magic methods names are limited to some list of PHP supported keywords, like construct, destruct and etc.
- And, these names are reserved. So, we should not define any function with the name of PHP magic methods.
- PHP magic methods should be started with `(__)` symbol.
- For defining the magic method with `(__)` followed by a different name, apart from the list of PHP supported naming keyword, we need to simulate PHP magic functionality.
- These special functions should be defined by the user. But no need to call them explicitly. Rather, it will be called on appropriate event occurrence. For example, class `__construct()` will be called while instantiating the class.
- PHP magic methods must be defined inside the class.

PHP Magic Methods and Purposes

TOPS Technologies

Now, we are going to list all available magic methods in PHP with their purposes. And, let us recollect some of PHP magic methods we have seen already in this blog.

PHP Magic Methods Invoked on Creating Class Instance

- `__construct()`/`__destruct()` – We have seen enough about these two magic methods while discussing [constructors and destructors](#) which is one of the object [oriented feature supported in PHP](#).

Magic Methods with PHP Overloading

The following list of PHP magic methods is used to deal with inaccessible class members created at run time by using PHP overloading concept.

- `__get()`/`__set()` – These are magic getters and setters for getting and putting values for class properties created dynamically by PHP property overloading.
- `__isSet()` – This magic method will be invoked automatically while checking whether a required overloaded property is set or not, by using the PHP `isSet()` function.
- `__unset()` – Similarly, when we call [PHP unset\(\) function](#) on such dynamically created properties, this magic method will automatically be invoked.
- `__call()`/`__callStatic()` – These two magic methods are dedicated for accessing dynamically created but invisible methods on PHP method overloading. These differ, where `__call()` will invoke normal PHP overloaded methods, and `__callStatic()` will invoke static methods

Magic Methods with PHP Object Serialization

The following magic methods will be invoked automatically while performing serialization with PHP object context. This automatic invocation will be triggered by using a pair of PHP function `serialize()`/`unserialize()` used in PHP Object Serialization.

- `sleep()` – This will be invoked on calling PHP `serialize()` function. It will return object's property array on cleaning PHP class objects before serialization.
- `wakeup()` – It will do the reverse work to restore objects properties and resources on invoking `unserialize()`.

Other Magic Methods in PHP

- `__toString()` – This method is expected return a string value while using class instances with [PHP printing statements](#). If we specify return values with any other data type, then the following error will occur.

Catchable fatal error: Method TestClass::__toString() must return a string value in...

TOPS Technologies

Without this function, we print class instance with PHP print and echo construct, then the following error will be displayed.

Catchable fatal error: Object of class TestClass could not be converted to string in...

- __invoke()-This method defined in a class will be called when we make a function call with the name of that class instance.
- __set_static()-This is a static method invoked while exporting objects property array and expects such array variable as its argument.
- __clone()-This function is used while [PHP Object Cloning](#).

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The `readfile()` function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML

CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language

PHP = PHP Hypertext Preprocessor

TOPS Technologies

SQL = Structured Query Language

SVG = Scalable Vector Graphics

XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the `readfile()` function returns the number of bytes read on success):

```
<?php  
echo readfile("webdictionary.txt");  
?>
```

The `readfile()` function is useful if all you want to do is open up a file and read its contents.

PHP MVC

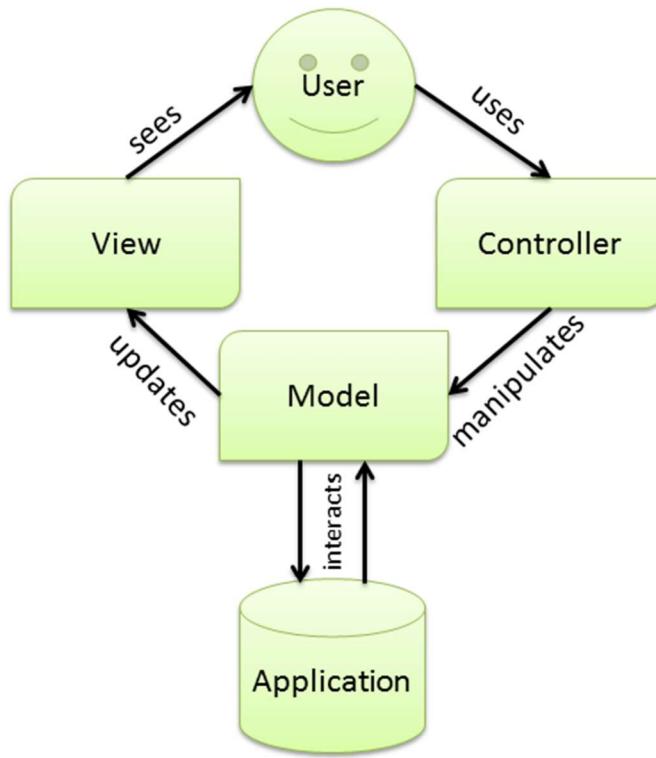
The Model-View-Controller (MVC) pattern, originally formulated in the late 1970s. It is an application design pattern that separates the application data and business logic (model) from the presentation (view). The controller mediates between the models and views.

Advantages:

Advantages of this architecture is reusability of code, data security and better application performance. In theory, a well-developed MVC system should allow a front-end developer and a back-end developer to work on the same system without interfering, sharing, or editing files.

Visual Representation for Data flow of MVC Architecture represented below:

MVC – Data flow



Model:

Model is the name given to the component that will communicate with the database to manipulate the data. It acts as a bridge between the View component and the Controller component in the overall architecture. This is responsible to manage the data. It stores and retrieves entities used by an application, usually from a database, and contains the logic implemented by the application.

View:

The view is responsible to display the data provided by the model in a specific format. This is responsible for mapping graphics onto a device. A view attaches to a model and renders its contents to the display surface and formats the data to be presented to the user, in a web application as an html output.

Controller:

A controller is the means by which the user interacts with the application. A controller accepts input from the user and instructs the model and view to perform actions based on that input.

It handles the model and view layers to work together. The controller receives a request from the client.

TOPS Technologies

invokes the model to perform the requested operations and sends the data to the View.

controller is responsible for mapping end-user action to application response

MVC Structure

- controllers
 - index.php
 - hello.php
- models
 - index_model.php
- views
 - hello.php
 - index.php
 - insert_index.php
 - update_index.php
 - view_index.php
- libs
 - Bootstrap.php
 - Controller.php
 - Database.php
 - Model.php
 - Session.php
 - View.php

Hello World Example using PHP MVC

MODEL:It is a class file which contains the Database connectivity queries inside it.

CONTROLLER:This file access the methods and variables declared.

VIEW:It contains all the html and view page.

In Controller create a file named "hello.php". This file will be saved in the Controller folder.

TOPS Technologies

controller/hello.php

```
<?php  
class Hello extends Controller {  
  
    function __construct() {  
        parent::__construct();  
    }  
  
    function index() {  
  
        $this->view->render('hello/index');  
  
    }  
}
```

Create a file index.php and place inside views folder.

views/hello/index.php

```
<!DOCTYPE html>  
<html>  
<head>  
    <title>Hello World</title>  
</head>  
<body>  
  
    <p>Hello World!!</p>  
</body>  
</html>
```

Now run the file in any browser: <http://localhost/mvc/hello/index>

How to insert data in database using PHP MVC with example

In this example we using Models, Views, Controller Structure for insert data into the database.

To insert data in the database first we have to create a Controller file.

controller/index.php

```
<?php
```

TOPS Technologies

```
class Index extends Controller {
    function __construct() {
        parent::__construct();
    }

    function index() {

        $this->view->allrecords = $this->model->getAllrecords();
        $this->view->render('index/index');

    }
    function edit_submit_index(){

        $action=$_POST['submit'];
        if ($action=='submit')
        {
            echo'$action';
            $arg=$_POST['id'];
            $data = array(
                'id' =>null,
                'name' =>$_POST['name'],
                'email' =>$_POST['email'],
                'contact' => $_POST['contact']
            );
            $this->model->submit_index($data);

        }
        header('location: index');
    }
}
```

Here is the model file which we are using for inserting data to database.

The file index_model.php is created under Models folder

TOPS Technologies

models/index_model.php

index_model.php

```
<?php

class Index_Model extends Model
{
    public function __construct()
    {
        parent::__construct();
    }
    public function getAllrecords()
    {
        return $this->db->select("SELECT * FROM `mvc` ORDER BY id DESC");
    }
    public function submit_index($data)
    {
        $this->db->insert('mvc', $data);
    }
}
```

Here is the view file index.php which inside views folder contains the form

views/index/index.php

TOPS Technologies

```
<?php
    $name=$this->content[0]['name'];
    $email=$this->content[0]['email'];
    $contact=$this->content[0]['contact'];
?
<div class="container">
    <div class="row clearfix">
        <div class="col-md-12 column"
            <form action=<?php echo URL; ?>index/edit_submit_index" method="post" enctype="multipart/form-data" onsubmit="return confirm('Do you really want to submit the form?');">
                <div class="col-xs-6 form-group">
                    <label class="control-label col-xs-6" for="name">name:</label>
                    <div class="col-xs-6">
                        <input class="form-control" name="name" id="name" placeholder="Enter name">
                    </div>
                </div>
                <div class="col-xs-6 form-group">
                    <label class="control-label col-xs-6" for="email">email:</label>
                    <div class="col-xs-6">
                        <input type="text" class="form-control" name="email" id="email" placeholder="Enter email">
                    </div>
                </div>
                <div class="col-xs-6 form-group">
                    <label class="control-label col-xs-6" for="contact">contact:</label>
                    <div class="col-xs-6">
                        <input type="text" class="form-control" name="contact" id="contact" placeholder="Enter contact">
                    </div>
                </div>
                <div class="col-sm-offset-2 col-xs-6">
                    <button type="submit" class=".btn-info form-control" value="submit" name="submit">Submit</button>
                </div>
            </div>
        </form>
    </div>
</div>
```

Path: localhost/project_folder_name/view_folder_name/view_filename

Example: localhost/mvc/index/index

TOPS Technologies

How to Update data in database using PHP MVC with example

In this example we using Models, Views, Controller Structure for Update the inserted data.

To update data in the database first we have to create a Controller file.

controller/index.php

```
<?php
class Index extends Controller {
    function __construct() {
        parent::__construct();
    }
    function index() {

        $this->view->allrecords = $this->model->getAllrecords();
        $this->view->render('index/index');

    }
    function edit_index() {
        $data = $_GET;
        if($_GET['id']=="") {
            $this->view->pick="";
            $this->view->data=$data;
        } else {
            $this->view->pick=$_GET['id'];
            $this->view->content= $this->model->getOnerecord($_GET['id']);
        }

        $this->view->render('index/edit_index');
    }
    function edit_submit_index(){
        $arg=$_POST['id'];
        $data = array(
            'name' =>$_POST['name'],
            'email' =>$_POST['email'],
            'contact' => $_POST['contact']
        );
        $this->model->edit_submit_index($data,$arg);
    }
    header('location: index');
}
```

TOPS Technologies

Here is the model file which we are using for Updating the inserted data to database.

The file index_model.php is created under Models folder

models/index_model.php

```
index_model.php

<?php
class Index_Model extends Model
{
    public function __construct()
    {
        parent::__construct();
    }
    public function getAllrecords()
    {
        return $this->db->select("SELECT * FROM `mvc` ORDER BY id DESC");
    }
    public function getOnerecord($id)
    {
        return $this->db->select("SELECT * FROM `mvc` WHERE id='".$id."' LIMIT 1");
    }
    public function edit_submit_index($data,$arg)
    {

        $this->db->update('mvc', $data,
                           "`id` = $arg");
    }
}
```

Here is the view file index.php which inside views folder contains the update form

views/index/index.php

```
<?php
$name=$this->content[0]['name'];
$email=$this->content[0]['email'];
$contact=$this->content[0]['contact'];
```

TOPS Technologies

```
?>

<div class="container">
  <div class="row clearfix">
    <div class="col-md-12 column">

      <form action="php echo URL; ?&gt;index/edit_submit_index" method="post" enctype="multipart/form-data"
onsubmit="return confirm('Do you really want to submit the form?');"&gt;

        &lt;div class="col-xs-6 form-group"&gt;
          &lt;label class="control-label col-xs-6" for="name"&gt;name:&lt;/label&gt;
          &lt;div class="col-xs-6"&gt;
            &lt;input class="form-control" name="name" id="name" placeholder="Enter name" value="<?php echo
$name; ?&gt;"&gt;
          &lt;/div&gt;
        &lt;/div&gt;

        &lt;div class="col-xs-6 form-group"&gt;
          &lt;label class="control-label col-xs-6" for="email"&gt;email:&lt;/label&gt;
          &lt;div class="col-xs-6"&gt;
            &lt;input type="text" class="form-control" name="email" id="email" placeholder="Enter email" value="<?php
echo $email; ?&gt;"&gt;
          &lt;/div&gt;
        &lt;/div&gt;

        &lt;div class="col-xs-6 form-group"&gt;
          &lt;label class="control-label col-xs-6" for="contact"&gt;contact:&lt;/label&gt;
          &lt;div class="col-xs-6"&gt;
            &lt;input type="text" class="form-control" name="contact" id="contact" placeholder="Enter contact"
value="<?php echo $contact; ?&gt;"&gt;
          &lt;/div&gt;
        &lt;/div&gt;

        &lt;div class="col-sm-offset-2 col-xs-6"&gt;
          &lt;button type="submit" class=".btn-info form-control" value="update" name="submit"&gt;Update&lt;/button&gt;
        &lt;/div&gt;
      &lt;/div&gt;
    &lt;/form&gt;
  &lt;/div&gt;
&lt;/div&gt;</pre
```

Path: localhost/project_folder_name/view_folder_name/view_filename

TOPS Technologies

Example: localhost/mvc/index/index

How to View data in database using PHP MVC with example

In this example we using Models, Views, Controller Structure for View the inserted data.

To View data in the database first we have to create a Controller file.

controller/index.php

```
<?php

class Index extends Controller {

    function __construct() {
        parent::__construct();
    }

    function index() {

        $this->view->allrecords = $this->model->getAllrecords();
        $this->view->render('index/index');

    }
    function view_index()
    {
        /* Auth::handleLogin(); */
        $data = $_GET;
        if($data['id']=="")
        {
            $this->view->pick="";
            $this->view->data=$data;
        }
        else
        {
            $this->view->pick=$data['id'];
            $this->view->content= $this->model->viewOnerecord($data['id']);
        }
    }

    $this->view->render('index/view_index');
```

TOPS Technologies

```
}
```

```
}
```

Here is the model file which we are using for View data from database.

The file index_model.php is created under Models folder

models/index_model.php

index_model.php

```
<?php

class Index_Model extends Model
{
    public function __construct()
    {
        parent::__construct();
    }

    public function getAllrecords()
    {
        return $this->db->select("SELECT * FROM `mvc` ORDER BY id DESC");
    }
    public function viewOnerecord($id)
    {
        return $this->db->select("SELECT * FROM `mvc` WHERE id='".$id."' LIMIT 1");
    }
}
```

Here is the view file index.php which inside views folder contains the table

views/index/index.php

```
<table border="2" id="internalActivities" style="width:100%" class="table table-bordered">
<tbody>
<tr>
    <th>id</th>
    <td><?php echo $this->content[0]['id']; ?></td>
    <th>name</th>
    <td><?php echo $this->content[0]['name']; ?></td>
```

TOPS Technologies

```
</tr>
<tr>
    <th>email</th>
    <td><?php echo $this->content[0]['email']; ?></td>
    <th>contact</th>
    <td><?php echo $this->content[0]['contact']; ?></td>
</tr>
</tbody>
</table>
```

Path: localhost/project_folder_name/view_folder_name/view_filename

Example: localhost/mvc/index/index

How to Delete data in database using PHP MVC with example

In this example we using Models, Views, Controller Structure for Delete the inserted data.

To Delete data in the database first we have to create a Controller file.

controller/index.php

```
<?php

class Index extends Controller {

    function __construct() {
        parent::__construct();
    }

    function index() {

        $this->view->allrecords = $this->model->getAllrecords();
        $this->view->render('index/index');

    }

    function delete_index($id)
    {
```

TOPS Technologies

```
$this->model->delete_index($id);
header('location: ../index/index');

}
```

Here is the model file which we are using for Delete data from database.

The file index_model.php is created under Models folder

models/index_model.php

```
index_model.php

<?php

class Index_Model extends Model
{
    public function __construct()
    {
        parent::__construct();
    }

    public function getAllrecords()
    {
        return $this->db->select("SELECT * FROM `mvc` ORDER BY id DESC");
    }
    public function delete_index($id)
    {
        $this->db->delete('mvc', "`id` = {$id}");
    }
}
```

Here is the view file index.php which inside views folder contains the table

views/index/index.php

```
<table border="2" id="internalActivities" style="width:100%" class="table table-bordered">
<tr>
```

TOPS Technologies

```
<th>Name</th>
<th>Email</th>
<th>contact</th>

<th></th>

</tr>
<?php foreach($this->allrecords AS $key=>$value){>
    ?>
    <tr>

        <td><?php echo $value['name']. "<br>"?></td>
        <td><?php echo $value['email']. "<br>"?></td>
        <td><?php echo $value['contact'];?></a>
        </td>

<td><a href="javascript:confirmDelete('delete_index/<?php echo $value['id'];?>')">Delete</a></td>

        <?php } ?>
    </tr>
}>
</table>

<script>
function confirmDelete(delUrl) {
    if (confirm("Are you sure you want to delete")) {
        document.location = delUrl;
    }
}
</script>
```

Path: localhost/project_folder_name/view_folder_name/view_filename Example: localhost/mvc/index/index Path:
localhost/project_folder_name/view_folder_name/view_filename Example:
localhost/mvc/changepassword/changepassword

JAVASCRIPT

JavaScript is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Hello World using Javascript

Just to give you a little excitement about **Javascript programming**, I'm going to give you a small conventional Javascript Hello World program, You can try it using Demo link

```
<html>
  <body>
    <script language = "javascript" type = "text/javascript">
      <!--
        document.write("Hello World!")
      //-->
    </script>
  </body>
</html>
```

Applications of Javascript Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has its presence in almost every area of software development. I'm going to list few of them here:

- **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validating those inputs at front-end itself.
- **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
- **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
- **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
- **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
- **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

What is JavaScript ?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript**, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The [ECMA-262 Specification](#) defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are –

- **Less server interaction** – You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** – They don't have to wait for a page reload to see if they have forgotten to enter something.

TOPS Technologies

- **Increased interactivity** – You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** – You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features –

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multi-threading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When `javascript` code is included in `HTML`, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the

TOPS Technologies

		element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
        function clickevent()
    {
        document.write("This is TopsTech ");
    }
    //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
<html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
    <!--
        function mouseoverevent()
    {
        alert("This is TopsTech");
    }
</script>
```

TOPS Technologies

//-->

```
</script>

<p onmouseover="mouseoverevent()"> Keep cursor over me</p>

</body>

</html>
```

Load event

```
<html>

<head>Javascript Events</head>

<br>

<body onload="window.alert('Page successfully loaded');">

<script>

<!--

document.write("The page is loaded successfully");

//-->

</script>

</body>

</html>
```

AJAX

The AJAX technique helps you to improve your application's user interface and enhance the overall end user experience.

TOPS Technologies

What Is AJAX?

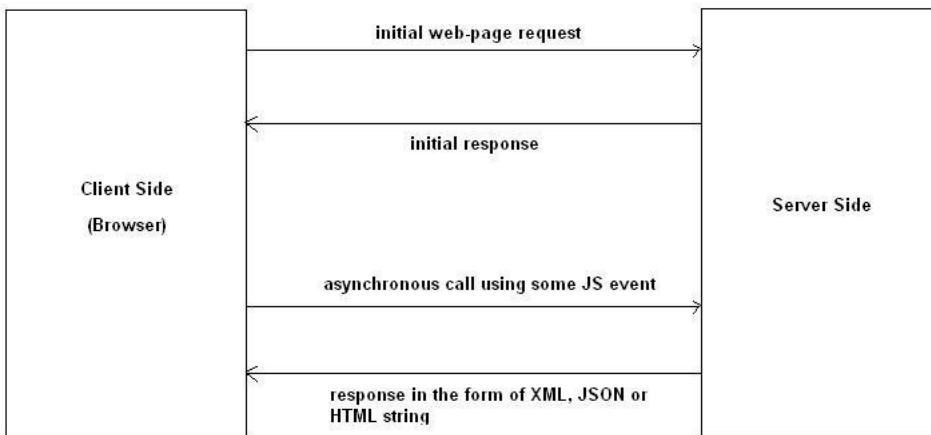
AJAX stands for Asynchronous JavaScript and XML, and it allows you to fetch content from the back-end server asynchronously, without a page refresh. Thus, it lets you update the content of a web page without reloading it.

Let's look at an example to understand how you could use AJAX in your day-to-day application development. Say you want to build a page that displays a user's profile information, with different sections like personal information, social information, notifications, messages, and so on.

The usual approach would be to build different web pages for each section. So for example, users would click the social information link to reload the browser and display a page with the social information. This makes it slower to navigate between sections, though, since the user has to wait for the browser to reload and the page to render again each time.

On the other hand, you could also use AJAX to build an interface that loads all the information without refreshing the page. In this case, you can display different tabs for all sections, and by clicking on the tab it fetches the corresponding content from the back-end server and updates the page without refreshing the browser. This helps you to improve the overall end-user experience.

The overall AJAX call works something like this:



Let's quickly go through the usual AJAX flow:

1. First, the user opens a web page as usual with a synchronous request.
2. Next, the user clicks on a DOM element—usually a button or link—that initiates an asynchronous request to the back-end server. The end user won't notice this since the call is made asynchronously and doesn't refresh the browser. However, you can spot these AJAX calls using a tool like Firebug.
3. In response to the AJAX request, the server may return XML, JSON, or HTML string data.
4. The response data is parsed using JavaScript.

TOPS Technologies

5. Finally, the parsed data is updated in the web page's DOM.

So as you can see, the web page is updated with real-time data from the server without the browser reloading.

In the next section, we'll see how to implement AJAX using vanilla JavaScript.

How AJAX Works Using Vanilla JavaScript

In this section, we'll see how AJAX works in vanilla JavaScript. Of course, there are JavaScript libraries available that make it easier to do AJAX calls, but it's always interesting to know what's happening under the hood.

Let's have a look at the following vanilla JavaScript code, which performs the AJAX call and fetches a response from the server asynchronously.

```
<script>

var objXMLHttpRequest = new XMLHttpRequest();

objXMLHttpRequest.onreadystatechange = function()

if(objXMLHttpRequest.readyState === 4) {

    if(objXMLHttpRequest.status === 200) {

        alert(objXMLHttpRequest.responseText);

    } else {

        alert('Error Code: ' + objXMLHttpRequest.status);

        alert('Error Message: ' + objXMLHttpRequest.statusText);

    }

}

}


```

TOPS Technologies

```
objXMLHttpRequest.open('GET', 'request_ajax_data.php');

objXMLHttpRequest.send();

</script>
```

Let's go through the above code to understand what's happening behind the scenes.

1. First, we initialize the `XMLHttpRequest` object, which is responsible for making AJAX calls.
2. The `XMLHttpRequest` object has a `readyState` property, and the value of that property changes during the request lifecycle. It can hold one of four values: `OPENED`, `HEADERS_RECEIVED`, `LOADING`, and `DONE`.
3. We can set up a listener function for state changes using the `onreadystatechange` property.
And that's what we've done in the above example: we've used a function which will be called every time the state property is changed.
4. In that function, we've checked if the `readyState` value equals `4`, which means the request is completed and we've got a response from the server. Next, we've checked if the status code equals `200`, which means the request was successful. Finally, we fetch the response which is stored in the `responseText` property of the `XMLHttpRequest` object.
5. After setting up the listener, we initiate the request by calling the `open` method of the `XMLHttpRequest` object. The `readyState` property value will be set to 1 after this call.
6. Finally, we've called the `send` method of the `XMLHttpRequest` object, which actually sends the request to the server. The `readyState` property value will be set to 2 after this call.
7. When the server responds, it will eventually set the `readyState` value to 4, and you should see an alert box displaying the response from the server.

So that's how AJAX works with vanilla JavaScript. The method here, using "callback functions" is the traditional way to code AJAX, but a cleaner and more modern way is with Promises.

In the next section, we'll see how to use the `Promise` object for AJAX.

TOPS Technologies

How to Use JavaScript Promises for AJAX

Promises in JavaScript provide a better way to manage asynchronous operations and callbacks that are dependent on other callbacks. In JavaScript, `Promise` is an object which may have one of the three states: pending, resolved, or rejected. Initially, the `Promise` object is in the pending state, but as the asynchronous operation is completed, it may evaluate to the resolved or rejected state.

Let's quickly revise the previous example with the `Promise` object.

```
function AjaxCallWithPromise() {  
    return new Promise(function (resolve, reject) {  
        const objXMLHttpRequest = new XMLHttpRequest();  
  
        objXMLHttpRequest.onreadystatechange = function () {  
            if (objXMLHttpRequest.readyState === 4) {  
                if (objXMLHttpRequest.status == 200) {  
                    resolve(objXMLHttpRequest.responseText);  
                } else {  
                    reject('Error Code: ' + objXMLHttpRequest.status + ' Error Message: ' +  
objXMLHttpRequest.statusText);  
                }  
            }  
        }  
    }  
  
    objXMLHttpRequest.open('GET', 'request_ajax_data.php');  
    objXMLHttpRequest.send();  
});  
  
AjaxCallWithPromise().then(  
    data => { console.log('Success Response: ' + data) },  
    error => { console.log(error) }  
);
```

TOPS Technologies

When the `AjaxCallWithPromise` function is called, it returns the promise object, and it's in the pending state initially. Based on the response, it'll call either the `resolve` or `reject` function.

Next, we use the `then` method, which is used to schedule callbacks when the promise object is successfully resolved. The `then` method takes two arguments. The first argument is a callback which will be executed when the promise is resolved, and the second argument is a callback for the rejected state.

So that's how you can use JavaScript Promises for AJAX. In the next section, we'll see how to use the jQuery library to perform AJAX calls.

How AJAX Works Using the jQuery Library

In the earlier section, we discussed how you could perform AJAX calls using vanilla JavaScript. In this section, we'll use the jQuery library to demonstrate this. I'll assume that you're aware of the basics of the jQuery library.

The jQuery library provides a few different methods to perform AJAX calls, although here we'll look at the standard `ajax` method, which is the most often used.

Take a look at the following example.

```
<script>
$.ajax(
  'request_ajax_data.php',
  {
    success: function(data) {
      alert('AJAX call was successful!');
      alert('Data from the server' + data);
    },
    error: function() {
      alert('There was some error performing the AJAX call!');
```

```
    }
}
);
</script>
```

As you already know, the `$` sign is used to refer to a jQuery object.

The first parameter of the `ajax` method is the URL that will be called in the background to fetch content from the server side. The second parameter is in JSON format and lets you specify values for some different options supported by the `ajax` method.

In most cases, you will need to specify the success and error callbacks. The success callback will be called after the successful completion of the AJAX call. The response returned by the server will be passed along to the success callback. On the other hand, the failure callback will be called if something goes wrong and there was an issue performing the AJAX call.

So as you can see, it's easy to perform AJAX operations using the jQuery library. In fact, the process is more or less the same, irrespective of the JavaScript library with which you choose to perform AJAX calls.

In the next section, we'll see a real-world example to understand how this all works with PHP.

A Real-World AJAX Example With PHP

In this section, we'll build an example that fetches JSON content from a PHP file on the server side using AJAX.

For demonstration purposes, we'll build an example which performs user login using AJAX and jQuery. To start with, let's make the `index.php` file, as shown in the following snippet, which renders a basic login form.

```
<!doctype html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.3.1.js" integrity="sha256-
```

TOPS Technologies

```
2Kok7MbOyxpathUVvAk/HJ2jigOSYS2auK4Pfzbn7uH60=" crossorigin="anonymous"></script>
</head>
<body>
<form id="loginform" method="post">
<div>
    Username:
    <input type="text" name="username" id="username" />
    Password:
    <input type="password" name="password" id="password" />
    <input type="submit" name="loginBtn" id="loginBtn" value="Login" />
</div>
</form>
<script type="text/javascript">
$(document).ready(function() {
    $('#loginform').submit(function(e) {
        e.preventDefault();
        $.ajax({
            type: "POST",
            url: 'login.php',
            data: $(this).serialize(),
            success: function(response)
            {
                var jsonData = JSON.parse(response);

                // user is logged in successfully in the back-end
                // let's redirect
                if (jsonData.success == "1")
                {
                    location.href = 'my_profile.php';
                }
                else
                {
                    alert('Invalid Credentials!');
                }
            }
        });
    });
});
</script>
</body>
```

TOPS Technologies

</html>

The **index.php** file is a pretty standard HTML form which contains username and password fields. It also contains a jQuery JavaScript snippet, which follows the outline we saw above.

We've used the `submit` event of the form element, which will be triggered when a user clicks on the submit button. In that event handler, we've initiated the AJAX call, which submits the form data to the **login.php** file using the POST method asynchronously. Once we receive a response from the server, we parse it using the `parse` method of the `JSON` object. And finally, based on the success or failure, we take the appropriate action.

Let's also see what **login.php** looks like.

```
<?php
if (isset($_POST['username']) && $_POST['username'] && isset($_POST['password']) &&
$_POST['password']) {
    // do user authentication as per your requirements
    // ...
    // ...
    // based on successful authentication
    echo json_encode(array('success' => 1));
} else {
    echo json_encode(array('success' => 0));
}
```

The **login.php** file contains the logic of authenticating users and returns a JSON response based on the success or failure of login.

Using Promises for AJAX With jQuery

Apart from this, the `$.ajax` method supports JavaScript Promises as well. It provides different methods like `then`, `done`, `fail` and `always` that you could use in the context of Promises.

Let's quickly revise the jQuery snippet which we've used in our example to show how to use it with the `then` method.

```
$ajax({  
    type: "POST",  
    url: 'login.php',  
    data: $(this).serialize()  
}).then(  
    // resolve/success callback  
    function(response)  
    {  
        var jsonData = JSON.parse(response);  
  
        // user is logged in successfully in the back-end  
        // let's redirect  
        if (jsonData.success == "1")  
        {  
            location.href = 'my_profile.php';  
        }  
        else  
        {  
            alert('Invalid Credentials!');  
        }  
    },  
    // reject/failure callback  
    function()  
    {  
        alert('There was some error!');  
    }  
);
```

jQuery Validation and Ajax

The Basic Components

So, in order to create our process we are going to use the following components:

A general HTML form format

Our standard CSS rules for styling the form

TOPS Technologies

Add default text for our input fields, including a function to remove the default text when the form is submitted – see earlier blog post – Add Default Text To Form Fields Using jQuery

jQuery validation rules, which are applied when a form is submitted

jQuery code for processing the form and submitting the contents via AJAX

1. HTML Form Format

Obviously everyone has their own preferences when it comes to how you code your form. After experimenting with several different formats I found that using an ordered list to handle the layout, with each form input field and associated label being one list item, has been easiest to use and so far has been robust enough to take most standard form requirements.

For the tutorial we will create a standard contact form with a name field, email, telephone and text area for comments:

```
<form id="form-contact" class="styled" action="/user_functions.php" method="post">
    <fieldset>
        <legend>Contact Form</legend>
        <ol>
            <li class="form-row">
                <label>Email:</label>
                <input id="input-email" type="text" class="text-input required email default" name="email" value="" title="Enter Your Email Address" />
            </li>
            <li class="form-row">
                <label>Name:</label>
                <input id="input-name" type="text" class="text-input required default" name="name" value="" title="Enter Your Full Name" />
            </li>
            <li class="form-row">
                <label>Phone:</label>
                <input id="input-phone" type="text" class="text-input" name="phone" value="" />
            </li>
            <li class="form-row">
                <label>Comments:</label>
                <textarea id="input-message" class="text-area" name="message" cols="40" rows="8"></textarea>
            </li>
            <li class="button-row text-right">
```

TOPS Technologies

```
<input class="btn-submit" type="submit" value="submit" name="submit" />
</li>
</ol>
</fieldset>
</form>
```

As usual our jQuery will use css classes and IDs for the selectors. The key ones we need to include are: Class “btn-submit” on the form submit button – used to trigger the function.

Class “required” on any input fields to be validated.

Class “email” as an additional class on those input fields to be tested against our regular expression for correctly formatted email addressed.

2. The Form CSS

The CSS I use as the basis for this format is as follows:

```
.styled {
font-family: Arial, sans-serif;
}
.styled fieldset {
border: 1px solid #ccc; padding: 10px;
}
.styled fieldset legend {
font-size: 16px; font-weight: bold; color: #000; text-transform: capitalize; padding: 5px; background: #fff; display: block; margin-bottom: 0; border: 1px solid #ccc;
}
.styled fieldset ol, .styled fieldset ol li {
list-style: none;
}
.styled fieldset li.form-row {
margin-bottom: 3px; padding: 2px 0; width: 100%; overflow: hidden; position: relative;
}
.styled label {
font-size: 12px; display: block; font-weight: bold; float: left; width: 100px; margin-left: 5px; line-height: 24px;
}
.styled input.text-input, .styled .text-area {
background: #fefefe; border-top: 1px solid #909090; border-right: 1px solid #cecece; border-bottom: 1px solid #e1e1e1; border-left: 1px solid #bbb; padding: 3px; width: 220px; font-size: 12px;
}
.styled input.text-input.default.active, .styled .text-area.default.active {
color: #666; font-style: italic;
```

TOPS Technologies

```
}
```

```
.styled fieldset li.button-row {
```

```
margin-bottom: 0; padding: 2px 5px;
```

```
}
```

```
form input.btn-submit {
```

```
padding: 3px 7px; border: 1px solid #fff; background: #066CAA; font-size: 12px;
```

```
}
```

This is pretty much the minimum styling we need to add to get the labels and input fields aligned and pad/border the form. This can then be applied to your forms by adding the class “styled”. You can build on the above CSS to make your forms a little more exciting!

Any specific rules you may want to add for each form can be done using it's id. For our contact form #form-contact, we want to set the width to 320px and center on the page.

Validation CSS

In addition to the form layout we also need a few additional CSS rules, which will handle the styling of the validation results:

```
.styled span.error {
```

```
font-size: 11px; position: absolute; top: 0; right: 0; display: block; padding: 2px;
```

```
}
```

```
.styled fieldset li.error {
```

```
color: #D8000C; background: #fff0f0 url(..../media/images/checkers.png) repeat; border: 1px solid #f9c7c7;
```

```
padding: 5px 0;
```

```
}
```

```
.styled fieldset li.error label {
```

```
text-align: left;
```

```
}
```

3. The JQuery Code – Default Text Function

Before we tackle the form submission and validation lets add the jQuery code to handle the default text for the input fields – See earlier post for explanation.

```
$(".default").each(function(){
```

TOPS Technologies

```
var defaultVal = $(this).attr('title');
$(this).focus(function(){
    if ($(this).val() == defaultVal){
        $(this).removeClass('active').val("");
    }
});
$(this).blur(function() {
    if ($(this).val() == ""){
        $(this).addClass('active').val(defaultVal);
    }
})
.blur().addClass('active');
});
```

We can now add default text to any of our form fields by simply adding the class “default” and using the title attribute to hold the default text. In our contact form example we have decided to add default text to both the email fields and the name field, which are both required.

Since we don't want the default text to be submitted with the form for any empty fields we add a jQuery function, which will automatically check all input fields with class “default” and remove the text if the value is the same as the title attribute:

```
function defaulttextRemove(){
    $('.default').each(function(){
        var defaultVal = $(this).attr('title');
        if ($(this).val() == defaultVal){
            $(this).val("");
        }
    });
}
```

Now for our main function, handling the form submit. There are a few parts to this process:

Check the form details – i.e. exactly which submit button has been pressed and the action URL for the data, since we are creating a generic function, which can be used by any forms, including multiple instances on one page.

TOPS Technologies

Run the relevant validation on all required input fields – for our example we have both standard required text field and email validation using regular expressions to check for a correct email format.

Assuming no errors, we serialize the form data and using AJAX, post the contents to the form URL and return a result.

3. Form Validation And AJAX Form Submit JQuery Code

```
// Declare the loading gif as a variable, which will be used later
```

```
var $loading = $('

</div>');


```

```
// Form validation and submit when button is clicked
```

```
$('.btn-submit').click(function(e){
```

```
// Declare the function variables - parent form, form URL and the regex for checking the email
```

```
    var $formId = $(this).parents('form');
```

```
    var formAction = $formId.attr('action');
```

```
    var emailReg = /^([\w-\.]+\@[([\w-]+\.)+[\w-]{2,4})?$/;
```

```
// In preparation for validating the form - Remove any active default text and previous errors
```

```
    defaulttextRemove();
```

```
    $('li',$formId).removeClass('error');
```

```
    $('span.error').remove();
```

```
// Start validation by selecting all inputs with the class "required"
```

```
    $('.required',$formId).each(function(){
```

```
        var inputVal = $(this).val();
```

```
        var $parentTag = $(this).parent();
```

TOPS Technologies

```
if(inputVal == ""){  
  
    $parentTag.addClass('error').append('<span class="error">Required field</span>');  
  
}  
  
  
// Run the email validation using the regex for those input items also having class "email"  
  
if($(this).hasClass('email') == true){  
  
    if(!emailReg.test(inputVal)){  
  
        $parentTag.addClass('error').append('<span class="error">Enter a valid email address.</span>');  
  
    }  
  
}  
  
});  
  
  
// All validation complete - check whether any errors exist - if not submit form  
  
if ($('#span.error').length == "0") {  
  
    $formId.append($loading.clone());  
  
    $('#fieldset',$formId).hide();  
  
    $.post(formAction, $formId.serialize(),function(data){  
  
        $('.loading').remove();  
  
        $formId.append(data).fadeIn();  
  
    });  
  
};  
  
// Use the following to prevent the form being submitted the standard way  
  
e.preventDefault();  
  
});
```

TOPS Technologies

When the submit button is clicked all input fields with the class “required” are checked to see if they have a value. If not an error class is applied to the parent list tag (allows us to highlight the complete form row) as well as an error text message inserted after the input field.

If the input field also has the class “email” a further validation check is made using a standard regular expression to see if the text pattern matches that of an email format.

If any of the validation checks return an error then the form submit is stopped. Here we use an extremely easy and effective way of seeing if there are any errors by using `$('.span.error').length`. If length is not “0” then there are error messages still active.

If all validation is clear we then use the `$.post` function to submit the form data. At the same time we hide the form contents and replace with a loading gif.

When the data is returned from the AJAX post we then hide the loading gif and replace the form contents with the data itself – usually this would be something like a status message for the user.

The Complete JQuery Code

```
jQuery(document).ready(function($){  
    var $loading = $('

></div>');  
    $(".default").each(function(){  
        var defaultVal = $(this).attr('title');  
        $(this).focus(function(){  
            if ($(this).val() == defaultVal){  
                $(this).removeClass('active').val("");  
            }  
        });  
        $(this).blur(function() {  
            if ($(this).val() == ""){  
                $(this).addClass('active').val(defaultVal);  
            }  
        })  
        .blur().addClass('active');  
    });  
});


```

TOPS Technologies

```
$('.btn-submit').click(function(e){  
    var $formId = $(this).parents('form');  
    var formAction = $formId.attr('action');  
    defaulttextRemove();  
    var emailReg = /^[^w-\.]+\@\[^w-\.]+\[^w-]{2,4}\?$/;  
    $('#li',$formId.removeClass('error');  
    $('#span.error').remove();  
    $('.required',$formId).each(function(){  
        var inputVal = $(this).val();  
        var $parentTag = $(this).parent();  
        if(inputVal == ""){  
            $parentTag.addClass('error').append('<span class="error">Required field</span>');  
        }  
        if($(this).hasClass('email')) == true){  
            if(!emailReg.test(inputVal)){  
                $parentTag.addClass('error').append('<span class="error">Enter a valid email address.</span>');  
            }  
        }  
    });  
    if ($('#span.error').length == "0") {  
        $formId.append($loading.clone());  
        $('fieldset',$formId).hide();  
        $.post(formAction, $formId.serialize(),function(data){  
            $('.loading').remove();  
            $formId.append(data).fadeIn();  
        });  
    }  
    e.preventDefault();  
});  
});  
function defaulttextRemove(){  
    $('.default').each(function(){  
        var defaultVal = $(this).attr('title');  
        if ($(this).val() == defaultVal){  
            $(this).val("");  
        }  
    });  
}
```

Demo 1

TOPS Technologies

- All fields empty
- With an incorrect formatted email
- Finally with all required fields complete

Note: For a successful submit the form will return the message - "Enquiry Successful".

Contact Form

Email:	<input type="text" value="Enter Your Email Address"/>
Name:	<input type="text" value="Enter Your Full Name"/>
Phone:	<input type="text"/>
Comments:	<input type="text"/>

submit

Demo 2

Contact Form

Email:	<input type="text"/>	Required field
Name:	<input type="text"/>	Required field
Phone:	<input type="text"/>	
Comments:	<input type="text"/>	

submit

Enquiry Successful

Module – 7[Application for Industrial Projects]

Web Services in PHP

What is Webservices?

Webservices are used to allow users to access data from different sources like the Android app, IOS app, website etc from a centralized database.

We can create a web service through two methods :

1. SOAP (Simple Object Access Protocol)
2. REST (Representational State Transfer)

SOAP (Simple Object Access Protocol)

SOAP : Simple Object Access Protocol. SOAP is easy to read because SOAP based on XML. SOAP is the XML based format for sending and receiving data.

REST (Representational State Transfer)

REST : Representational State Transfer. It is an architectural style that run over HTTP. REST webservices generate status code response in JSON & XML.

REST support all HTTP methods GET, POST, PUT & DELETE.

GET : Retrieve particular resources by an id

POST : Create a new resource.

PUT : Update a particular resource by an id.

DELETE : Remove a particular resource.

Example of POST

Step 1 : Create a database (create database your_databasename)

Step 2 : create a SQL table signup. signup table structure given below :

CREATE TABLE `signup` (

 `id` int(11) NOT NULL,

 `fullName` varchar(120) DEFAULT NULL,

 `gender` varchar(100) DEFAULT NULL,

 `contactNumber` char(10) DEFAULT NULL,

 `email` varchar(150) DEFAULT NULL,

TOPS Technologies

```
 `password` varchar(200) DEFAULT NULL,  
 `regDate` timestamp NULL DEFAULT CURRENT_TIMESTAMP  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
  
ALTER TABLE `signup`  
  
ADD PRIMARY KEY (`id`);
```

Step 3 : Create a database connection file (config.php)

```
<?php  
  
// DB credentials.  
  
define('DB_HOST','localhost');  
  
define('DB_USER','root');  
  
define('DB_PASS','');  
  
define('DB_NAME','database Name'); // put you database name here  
  
// Establish database connection.  
  
try  
  
{  
  
$dbh = new PDO("mysql:host=".DB_HOST.";dbname=".DB_NAME,DB_USER,  
DB_PASS,array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'utf8'"));  
  
}  
  
catch (PDOException $e)  
  
{  
  
exit("Error: " . $e->getMessage());  
  
}  
  
?>
```

Step 4 : Create a requested url (<http://localhost/ws/signup.php>)

Signup.php

TOPS Technologies

```
<?php

include_once("config.php");

header("Content-type:application/json");

$jsdata=json_decode(file_get_contents('php://input'),true);

$fname=$jsdata['fullName'];

$gender=$jsdata['gender'];

$contactno=$jsdata['number'];

$email=$jsdata['email'];

$password=$jsdata['password'];

$sql="INSERT INTO signup(fullName,gender,contactNumber,email,password)
VALUES(:fname,:gender,:contactno,:email,:password)";

$query = $dbh->prepare($sql);

$query->bindParam(':fname',$fname,PDO::PARAM_STR);

$query->bindParam(':gender',$gender,PDO::PARAM_STR);

$query->bindParam(':contactno',$contactno,PDO::PARAM_STR);

$query->bindParam(':email',$email,PDO::PARAM_STR);

$query->bindParam(':password',$password,PDO::PARAM_STR);

$query->execute();

$lastInsertId = $dbh->lastInsertId();

if($lastInsertId)

{

header("HTTP/1.1 200 OK");

header('Content-Type: text/plain');

$data=array("StatusCode"=>"200","StatusDescription"=>"Registration successful");

echo json_encode($data);

} else {

header("HTTP/1.1 400 Bad Request");
}
```

TOPS Technologies

```
header('Content-Type: text/plain');

$msg['StatusCode']="400";

$msg['StatusDescription']="Something went wrong. Please try again.';

print(json_encode($msg));

}
```

header("Content-type:application/json") sends the http json header to the browser to inform him what the kind of a data he expects.

json_decode(file_get_contents('php://input'),true) receive the RAW post data via the php://input IO stream.

We retrieve the raw POST data from the php://input stream.

header(...) detect http response line.

header("HTTP/1.1 200 OK"); // for successful attempt

header("HTTP/1.1 400 Bad Request"); // for bad request

header('Content-Type: text/plain') indicates that the content is plain text and no special software is required to read the contents.

How to test this service in your local system

1. Install [postman chrome extension](#)
2. Run postman chrome extension then a new request
3. Choose http request method
4. Enter the requested url
5. Provide the requested data in JSON Format
6. Hit the send button

Input parameters will be like this

```
{  
    "fullName": "John Doe",  
    "gender": "Male",  
    "number": "7556645345342",
```

TOPS Technologies

```
"email":"john@anc.com",
```

```
 "password": "Demo"
```

```
}
```

Output will be in JSON format

For successful attempt

```
{"StatusCode":200,"StatusDescription":"Registration successful"}
```

For unsuccessful attempt

```
{"StatusCode":400,"StatusDescription":"Something went wrong. Please try again."}
```

PHP Shopping Cart

Building a PHP shopping cart eCommerce software is simple and easy. In this tutorial, let's create a simple PHP shopping cart software with MySQL.

The intention of this shopping cart software is that it should be simple and as minimal as possible. You can download this free and customize it for your needs within minutes.

PHP Shopping Cart Software Development Overview

PHP shopping cart example has the following functionality:

1. Retrieve product information from the database.
2. Create product gallery for the shopping cart.
3. Manage cart items using the PHP session.
4. Handle add, edit, remove and empty cart actions.

I have retrieved information like name, code, price, and photos from the database. The resultant information is in an array format.

I have iterated this resultant array to form the product gallery. Every product in the gallery will have an add-to-cart option.

I have used the PHP shopping cart session to store and manage the items in the cart.

Once the session expires, the cart items get cleared. This code has an option to clear the entire cart or to remove

any particular item from the cart.

File Structure

The shopping cart software example has the following file structure. The below list has the file names and their responsibility.

You can build a gallery-based shopping cart software with these files in few minutes.

- **dbcontroller.php** – a generic database layer to help with DAO functions. It also manages the database connection.
- **index.php** – to display the [product gallery for the shopping cart](#).
- **style.css** – to [showcase products for the shopping cart](#). The styles are minimal and cross-browser compatible.
- **tblproduct.sql** – contains SQL script with the product table structure and the data.
- **product-images** – a folder that contains the product images. I have used these images to show the product gallery.

Create a Product Gallery for Shopping Cart

If you are familiar with online shopping, you might have across product galleries. It is an important gateway in a shopping cart application.

Users will use the product gallery to get a feel of the products available to buy. It is a critical component of every online store.

You should always provide a catalog of the available products to let them have a glance. It helps to promote your products for the impulsive buyers.

In the following code, I have shown the PHP script to get the product results from the database. The result will be an array and iterated to create product card in each iteration.

The DBController.php class will handle the DAO operation and fetch products result.

```
<?php  
  
$product_array = $db_handle->runQuery("SELECT * FROM tblproduct ORDER BY id ASC");  
  
if (!empty($product_array)) {
```

TOPS Technologies

```
foreach($product_array as $key=>$value){  
  
?>  
  
<div class="product-item">  
  
    <form method="post" action="index.php?action=add&code=<?php echo  
$product_array[$key]["code"]; ?>">  
  
        <div class="product-image">"></div>  
  
        <div class="product-tile-footer">  
  
            <div class="product-title"><?php echo $product_array[$key]["name"]; ?></div>  
  
            <div class="product-price"><?php echo "$". $product_array[$key]["price"]; ?></div>  
  
            <div class="cart-action"><input type="text" class="product-quantity" name="quantity" value="1"  
size="2" /><input type="submit" value="Add to Cart" class="btnAddAction" /></div>  
  
        </div>  
  
    </form>  
  
</div>  
  
<?php  
  
}  
  
}  
  
?>
```

Adding Products to Shopping Cart

After creating the product gallery page, we need to work on the PHP code to perform the cart actions. They are add-to-cart, remove a single item from the cart, clear the complete cart and similar.

In the above code, I have added the HTML option to add the product to the shopping cart from the product gallery. When the user clicks the 'Add to Cart' button, the HTML form passes the product id to the backend PHP script.

In PHP, I receive and process the cart action with a switch control statement. I have created PHP switch cases to handle the add-to-cart, remove-single, empty-cart actions.

The action is the argument to the switch statement. Then it executes the corresponding action case as requested.

TOPS Technologies

The “add” case handles the add to cart action. In this case, I have received the product id and the quantity.

The cart session stores the submitted product. I have updated the quantity of the product if it already exists in the session cart. The PHP session index is the reference to perform this action.

```
case "add":  
  
    if(!empty($_POST["quantity"])) {  
  
        $productByCode = $db_handle->runQuery("SELECT * FROM tblproduct WHERE code=" .  
$_GET["code"] . "");  
  
        $itemArray = array($productByCode[0]["code"]=>array('name'=>$productByCode[0]["name"],  
'code'=>$productByCode[0]["code"], 'quantity'=>$_POST["quantity"], 'price'=>$productByCode[0]["price"],  
'image'=>$productByCode[0]["image"]));  
  
  
  
        if(!empty($_SESSION["cart_item"])) {  
  
            if(in_array($productByCode[0]["code"],array_keys($_SESSION["cart_item"]))) {  
  
                foreach($_SESSION["cart_item"] as $k => $v) {  
  
                    if($productByCode[0]["code"] == $k) {  
  
                        if(empty($_SESSION["cart_item"][$k]["quantity"])) {  
  
                            $_SESSION["cart_item"][$k]["quantity"] = 0;  
  
                        }  
  
                        $_SESSION["cart_item"][$k]["quantity"] +=  
$_POST["quantity"];  
  
                    }  
  
                }  
  
            } else {  
  
                $_SESSION["cart_item"] = array_merge($_SESSION["cart_item"],$itemArray);  
  
            }  
  
        } else {  
  
            $_SESSION["cart_item"] = $itemArray;  
  
        }  
  
    }  
  
}
```

```
    }  
  
    break;  
  
}
```

List Cart Items from the PHP Session

This code shows the HTML to display the shopping cart with action controls. The loop iterates the cart session to list the cart items in a tabular format.

Each row shows the product image, title, price, item quantity with a remove option. Also, it shows the total item quantity and the total item price by summing up the individual cart items.

```
<div id="shopping-cart">  
  
<div class="txt-heading">Shopping Cart</div>  
  
<a id="btnEmpty" href="index.php?action=empty">Empty Cart</a>  
  
<?php  
  
if(isset($_SESSION["cart_item"])){  
  
    $total_quantity = 0;  
  
    $total_price = 0;  
  
?>  
  
<table class="tbl-cart" cellpadding="10" cellspacing="1">  
  
<tbody>  
  
<tr>  
  
<th style="text-align:left;">Name</th>  
  
<th style="text-align:left;">Code</th>  
  
<th style="text-align:right;" width="5%">Quantity</th>  
  
<th style="text-align:right;" width="10%">Unit Price</th>  
  
<th style="text-align:right;" width="10%">Price</th>  
  
<th style="text-align:center;" width="5%">Remove</th>
```

TOPS Technologies

```
</tr>

<?php

foreach ($_SESSION["cart_item"] as $item){

$item_price = $item["quantity"]*$item["price"];

?>

<tr>

<td>" class="cart-item-image" /><?php
echo $item["name"]; ?></td>

<td><?php echo $item["code"]; ?></td>

<td style="text-align:right;"><?php echo $item["quantity"]; ?></td>

<td style="text-align:right;"><?php echo "$ ". $item["price"]; ?></td>

<td style="text-align:right;"><?php echo "$ ". number_format($item_price,2);
?></td>

<td style="text-align:center;"><a href="index.php?action=remove&code=<?php
echo $item["code"]; ?>" class="btnRemoveAction"></a></td>

</tr>

<?php

$total_quantity += $item["quantity"];

$total_price += ($item["price"]*$item["quantity"]);

}

?>

<tr>

<td colspan="2" align="right">Total:</td>

<td align="right"><?php echo $total_quantity; ?></td>

<td align="right" colspan="2"><strong><?php echo "$ ".number_format($total_price, 2); ?></strong></td>

<td></td>

</tr>
```

TOPS Technologies

```
</tbody>

</table>

<?php

} else {

?>

<div class="no-records">Your Cart is Empty</div>

<?php

}

?>

</div>
```

Removing or Clearing Cart Item

The loop iterates the cart session array to display the cart items. Each cart item will have a remove link.

If the shopping cart user clicks the remove link then, I remove the respective cart item from the session.

I have provided an ‘Empty Cart’ button control above the shopping cart. The users can use it to completely wipe out the cart session and clear the cart.

I have presented the PHP code below to “remove” and the “empty” cases. They handle the shopping cart to remove/clear actions.

I have used PHP unset() to clear the cart session to delete the added item from the shopping cart.

```
case "remove":  
    if(!empty($_SESSION["cart_item"])) {  
        foreach($_SESSION["cart_item"] as $k => $v) {  
            if($_GET["code"] == $k)  
                unset($_SESSION["cart_item"][$k]);  
            if(empty($_SESSION["cart_item"]))  
                unset($_SESSION["cart_item"]);  
    }  
}
```

```
        }
    break;

case "empty":

    unset($_SESSION["cart_item"]);

break;
```

Database Product Table for Shopping Cart

You should import the following SQL script. It will create the product table and load data into it. In the example, I have used this product data to display products in the gallery.

Database Product Table for Shopping Cart

--

-- Table structure for table `tblproduct`

--

CREATE TABLE `tblproduct` (

`id` int(8) NOT NULL,

`name` varchar(255) NOT NULL,

`code` varchar(255) NOT NULL,

`image` text NOT NULL,

`price` double(10,2) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--

-- Dumping data for table `tblproduct`

--

TOPS Technologies

```
INSERT INTO `tblproduct` (`id`, `name`, `code`, `image`, `price`) VALUES  
(1, 'FinePix Pro2 3D Camera', '3DcAM01', 'product-images/camera.jpg', 1500.00),  
(2, 'EXP Portable Hard Drive', 'USB02', 'product-images/external-hard-drive.jpg', 800.00),  
(3, 'Luxury Ultra thin Wrist Watch', 'wristWear03', 'product-images/watch.jpg', 300.00),  
(4, 'XP 1155 Intel Core Laptop', 'LPN45', 'product-images/laptop.jpg', 800.00);  
  
--  
  
-- Indexes for table `tblproduct`  
  
--  
  
ALTER TABLE `tblproduct`  
ADD PRIMARY KEY (`id`),  
ADD UNIQUE KEY `product_code` (`code`);  
  
--  
  
-- AUTO_INCREMENT for dumped tables  
  
--  
  
--  
  
-- AUTO_INCREMENT for table `tblproduct`  
  
--  
  
ALTER TABLE `tblproduct`  
MODIFY `id` int(8) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;  
COMMIT;
```

Simple PHP Shopping Cart Output

Below screenshot shows the product gallery of this simple PHP shopping cart example. The listed cart items above the gallery are from the PHP session.

TOPS Technologies

Shopping Cart

Name	Code	Quantity	Price	Remove
 3D Camera	3DcAM01	1	\$ 1500.00	
 External Hard Drive	USB02	1	\$ 800.00	
 Wrist Watch	wristWear03	1	\$ 300.00	
 Laptop	LPN45	1	\$ 800.00	
Total:		4	\$ 3,400.00	

Products

 3D Camera \$1500.00 <input type="button" value="1"/> Add to Cart	 External Hard Drive \$800.00 <input type="button" value="1"/> Add to Cart	 Wrist Watch \$300.00 <input type="button" value="1"/> Add to Cart
---	--	--

Payment gateway (Stripe)

Stripe provides payment processing to support eCommerce software and mobile APPs. The key features by Stripe are,

- fast and interactive checkout experience.
- secure, smooth payment flow.
- scope to uplift the conversion rate and business growth.

Stripe is the most popular payment gateway solution supporting card payments along with PayPal. Its complete documentation helps developers to do an effortless integration.

Stripe provides different types of payment services to accept payments. It supports accepting one-time payment, recurring payment, in-person payment and more.

There are two ways to set up the Stripe one-time payment option in a eCommerce website.

1. Use the secure prebuilt hosted checkout page.

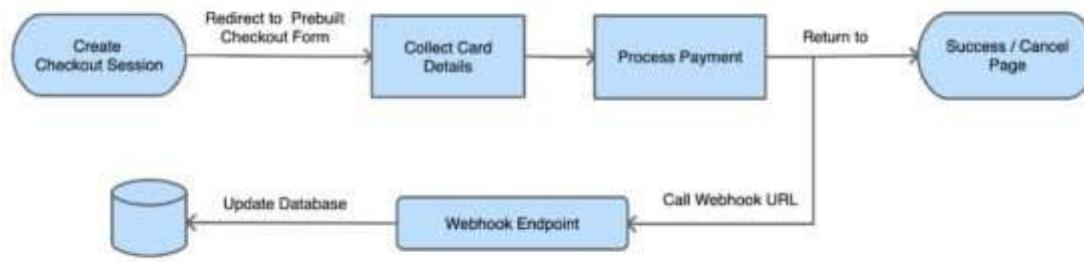
TOPS Technologies

2. Create a custom payment flow.

Let us take the first method to integrate Stripe's one-time payment. The example code redirects customers to the Stripe hosted checkout page. It's a pre-built page that allows customers to enter payment details.

Stripe hosted checkout option piggybacks on the Stripe trust factor. If your website is lesser-known, if you are not popular, then it is best to choose this option. Because the end users may feel uncomfortable to enter their card details on your page.

This diagram depicts the Stripe payment flow, redirect and response handling.



In this example, it uses the latest checkout version to set up a one-time payment. If you want to [create Stripe subscription payment](#), then the linked article has an example for it.

What is inside?

1. About Stripe Checkout
2. Steps to set up Stripe one-time payment flow
3. About this example
4. Generate and configure Stripe API keys
5. Create webhook and map events
6. Required libraries to access Stripe API
7. Create client-server-side code to initiate checkout

8. Capture and process webhook response
9. Evaluate integration with test data
10. Stripe one-time payment example output

About Stripe Checkout

Latest Stripe Checkout provides a frictionless smooth checkout experience. The below list shows some features of the latest Stripe checkout version.

- Strong Customer Authentication (SCA).
- Checkout interface fluidity over various devices' viewport.
- Multilingual support.
- Configurable options to enable billing address collection, email receipts, and Button customizations

TOPS Technologies

If you are using the legacy version, it's so simple to migrate to the latest Stripe Checkout version. The upgraded version supports advanced features like **3D secure**, mobile payments and more.

Steps to setup Stripe one-time payment flow

The below steps are the least needs to set up a Stripe one-time payment online.

1. Register with Stripe and generate API keys.
2. Configure API keys with the application.
3. Install and load the required libraries to access API.
4. Create client and server-side code to make payments.
5. Create endpoints to receive payment response and log into a database.
6. Create a UI template to acknowledge customers.

We will see the above steps in this article with example code and screenshots.

About Stripe payment integration example

Stripe API allows applications to access and use its online payment services. It provides Stripe.js a JavaScript library to initiate the payment session flow.

This example code imports the Stripe libraries to access the API functions. It sets the endpoint to build the request and receive the response.

The client and server-side code redirect customers to the Stripe hosted checkout page. In the callback handlers, it processes the payment response sent by the API.

This example uses a database to store payment entries. It uses MySQLi with prepared statements to execute queries.

This image shows the simple file architecture of this example.

TOPS Technologies

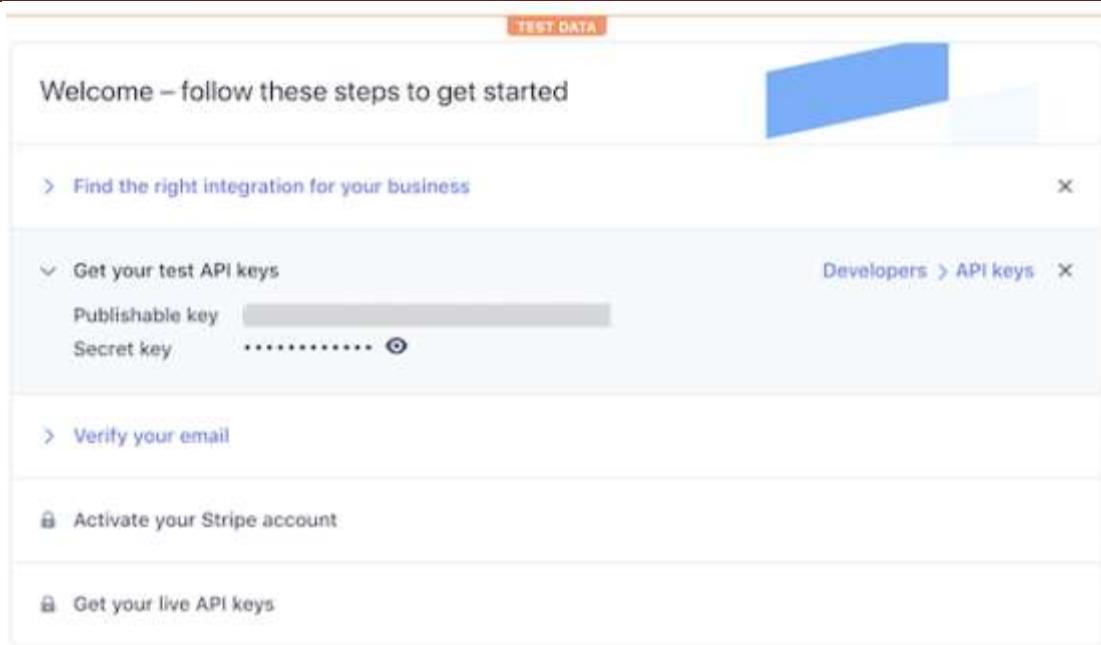


Get and configure Stripe API keys

Register and log in with Stripe to get the API keys. Stripe dashboard will show two keys publishable_key and secret_key.

These keys are the reference to validate the request during the authentication process.

Note: Once finish testing in a sandbox mode, Stripe requires to activate the account to get the live API keys.



The screenshot shows the Stripe API keys configuration page. At the top, there is a red button labeled "TEST DATA". Below it, a message says "Welcome – follow these steps to get started". A blue rectangular box is overlaid on the right side of the page. The main content area has a light gray background and contains several sections:

- > Find the right integration for your business
- > Verify your email
- Get your test API keys
 - Publishable key: [REDACTED]
 - Secret key: [REDACTED] ⓘ
- Developers > API keys ×
- Activate your Stripe account ⓘ
- Get your live API keys ⓘ

This code shows the constants created to configure the API keys for this example.

```
<?php  
namespace Phpot;  
  
class Config  
{  
    const ROOT_PATH = "https://your-domain/stripe-hosted";  
  
    /* Stripe API test keys */  
    const STRIPE_PUBLISHABLE_KEY = "";  
    const STRIPE_SECRET_KEY = "";  
  
    /* PRODUCT CONFIGURATIONS BEGINS */  
    const PRODUCT_NAME = 'A6900 MirrorLess Camera';  
    const PRODUCT_IMAGE = Config::ROOT_PATH . '/images/camera.jpg';  
    const PRODUCT_PRICE = '289.61';  
    const CURRENCY = 'USD';  
    const PRODUCT_TYPE = 'good';  
}
```

Create webhook and map events

Creating a webhook is a conventional way to get the payment notifications sent by the API. All payment gateway providers give the option to create webhook for callback.

In [PayPal payment gateway integration](#) article, we have seen the types of notification mechanisms supported by PayPal.

The code has the webhook endpoint registered with Stripe. It handles the API responses based on the event that occurred.

TOPS Technologies

The below image shows the screenshot of the add-endpoint dialog. It populates the URL and the mapped events in the form fields.

Navigate via *Developers->Webhooks* and click the *Add endpoint* button to see the dialog.

The screenshot shows a modal dialog titled "Add a webhook endpoint". At the top right is a red "TEST DATA" button. The dialog has several sections: "Endpoint URL" containing "https://yourdomain/stripe-hosted/capture-response.php"; "Description" with the placeholder "An optional description of what this webhook endpoint is used for."; "Version" set to "Latest API version (2020-08-27)"; "Events to send" with a dropdown menu showing "Select events..." and a list box containing "checkout.session.completed" with a clear button; and a bottom row with "Cancel" and "Add endpoint" buttons.

Required libraries to access Stripe API

First, download and install the **stripe-php** library. This will help to send flawless payment initiation request to the Stripe API.

This command helps to install this library via Composer. It is also available to [download from GitHub](#).
composer require stripe/stripe-php

Load Stripe.js JavaScript library into the page which has the checkout button. Load this file by using <https://js.stripe.com/v3/> instead of having it in local.

```
<script src="https://js.stripe.com/v3/"></script>
```

Create client-server code to process checkout

This section includes the steps needed to create the client-server code for setting up the Stripe one-time payment.

1. Add checkout button and load Stripe.js
2. Create a JavaScript event handler to initiate checkout session

TOPS Technologies

3. Create PHP endpoint to post create-checkout-session request
4. Redirect customers to the Stripe hosted checkout page
5. Get checkout session-id from the response.

HTML page with Stripe checkout button

This page has HTML code to add the Stripe checkout button. It loads the Stripe.js library.

This page loads a JavaScript that initiates the checkout session and redirects the user to the prebuilt Stripe hosted checkout.

The Stripe hosted checkout form handles the card validation effectively. We have created a custom card validator for [Authorize.net payment integration](#) code. Hosted checkout is more secure than handling card details by custom handlers.

index.php

```
<?php
namespace Phppot;
?>
<html>
<title>Stripe Prebuilt Hosted Checkout</title>
<head>
<link href="css/style.css" type="text/css" rel="stylesheet" />
<script src="https://js.stripe.com/v3/"></script>
</head>
<body>
<div class="phppot-container">
<h1>Stripe Prebuilt Hosted Checkout</h1>
<div id="payment-box">

<h4 class="txt-title">A6900 MirrorLess Camera</h4>
<div class="txt-price">$289.61</div>
<button id="checkout-button">Checkout</button>
</div>
</div>
<script>
var stripe = Stripe('<?php echo Config::STRIPE_PUBLISHABLE_KEY; ?>');
var checkoutButton = document.getElementById('checkout-button');

checkoutButton.addEventListener('click', function() {
fetch('create-checkout-session.php', {
method: 'POST',
})
.then(function(response) {
```

```
    return response.json();
  })
  .then(function(session) {
    return stripe.redirectToCheckout({ sessionId: session.id });
  })
  .then(function(result) {
    if (result.error) {
      alert(result.error.message);
    }
  })
  .catch(function(error) {
    console.error('Error:', error);
  });
});

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

JavaScript event handler initiates checkout session

In the previous section, it loads a JavaScript to do the following.

1. Instantiate Stripe Javascript library object with the reference of the Stripe Publishable key.
2. Map the checkout button's click event to initiate a create-checkout-session.
3. Redirect customers to the Stripe hosted checkout page with the checkout session id.

It calls the PHP endpoint builds the API request params to start a checkout session.

Then it receives the checkout-session-id as returned by the PHP endpoint. The *redirectToCheckout()* method sends customers to the prebuilt checkout to complete the payment.

PHP endpoint processing create-checkout-session request

When clicking the checkout button, the JavaScript executes an AJAX request. It fetches the PHP endpoint processing the create-checkout-session request.

The below code shows how to create the Stripe checkout-session via API. The API request is with the required query parameters. It includes the product detail, unit amount, payment method and more.

Then the API will return the session object after processing this request. This endpoint parses the response and grab the session-id and return it.

ajax-endpoint/create-checkout-session.php

```
<?php
```

```
namespace Phpot;
```

```
use Phpot\StripeService;
```

```
use Phpot\StripePayment;
```

TOPS Technologies

```
$orderReferenceId = rand(100000, 999999);

require_once __DIR__ . "../lib/StripeService.php";
require_once __DIR__ .'../Common/Config.php';

require_once __DIR__ . '/../lib/StripePayment.php';
$stripePayment = new StripePayment();

$stripeService = new StripeService();

$currency = Config::CURRENCY;

$orderId = $stripePayment->insertOrder(Config::PRODUCT_PRICE, $currency, $orderReferenceId, "Payment
in-progress");
$unitAmount = Config::PRODUCT_PRICE * 100;
$session = $stripeService->createCheckoutSession($unitAmount, $orderId);
echo json_encode($session);

The StripeService is a PHP class created to build API requests and process responses.
The createCheckoutSession() function builds the param array for the create-checkout-session request.
This service also handles the webhook responses sent by the API. The API sends the response on the
occurrences of the events mapped with the webhook endpoint URL.

lib/StripeService.php
<?php
namespace Phpot;

require_once __DIR__ . '/../Common/Config.php';

class StripeService
{
    function __construct()
    {
        require_once __DIR__ . "/../vendor/autoload.php";
        // Set your secret key. Remember to set your live key in production!
        \Stripe\Stripe::setApiKey(Config::STRIPE_SECRET_KEY);
    }

    public function createCheckoutSession($unitAmount, $clientReferenceId)
    {
        $checkout_session = \Stripe\Checkout\Session::create([
            'payment_method_types' => ['card'],

```

TOPS Technologies

```
'line_items' => [[
    'price_data' => [
        'currency' => Config::CURRENCY,
        'unit_amount' => $unitAmount,
        'product_data' => [
            'name' => Config::PRODUCT_NAME,
            'images' => [Config::PRODUCT_IMAGE],
        ],
    ],
    'quantity' => 1,
]],
'mode' => 'payment',
'client_reference_id' => $clientReferenceld,
'success_url' => Config::ROOT_PATH . '/success.php?session_id={CHECKOUT_SESSION_ID}',
'cancel_url' => Config::ROOT_PATH . '/index.php?status=cancel',
]);
return $checkout_session;
}

public function captureResponse()
{
    $payload = @file_get_contents('php://input');

    $event = json_decode($payload);

    require_once __DIR__ . '/../lib/StripePayment.php';
    $stripePayment = new StripePayment();

    switch($event->type) {
        case "customer.created":
            $param["stripe_customer_id"] = $event->data->object->id;
            $param["email"] = $event->data->object->email;
            $param["customer_created_datetime"] = date("Y,m,d H:i:s", $event->data->object->created);
            $param["stripe_response"] = json_encode($event->data->object);
            $stripePayment->insertCustomer($param);
            break;

        case "checkout.session.completed":
```

TOPS Technologies

```
$param["order_id"] = $event->data->object->client_reference_id;
$params["customer_id"] = $event->data->object->customer;
$params["payment_intent_id"] = $event->data->object->payment_intent;
$params["stripe_checkout_response"] = json_encode($event->data->object);
$stripePayment->updateOrder($param);
break;

case "payment_intent.created":
    $param["payment_intent_id"] = $event->data->object->id;
    $param["payment_create_at"] = date("Y-m-d H:i:s", $event->data->object->created);
    $param["payment_status"] = $event->data->object->status;
    $params["stripe_payment_response"] = json_encode($event->data->object);
    $stripePayment->insertPayment($param);
break;

case "payment_intent.succeeded":
    $param["payment_intent_id"] = $event->data->object->id;
    $param["billing_name"] = $event->data->object->charges->data[0]->billing_details->name;
    $param["billing_email"] = $event->data->object->charges->data[0]->billing_details->email;
    $param["payment_last_updated"] = date("Y-m-d H:i:s", $event->data->object->charges->data[0]->created);
    $param["payment_status"] = $event->data->object->charges->data[0]->status;
    $params["stripe_payment_response"] = json_encode($event->data->object);
    $stripePayment->updatePayment($param);
break;

case "payment_intent.canceled":
    $param["payment_intent_id"] = $event->data->object->id;
    $param["billing_name"] = $event->data->object->charges->data[0]->billing_details->name;
    $param["billing_email"] = $event->data->object->charges->data[0]->billing_details->email;
    $param["payment_last_updated"] = date("Y-m-d H:i:s", $event->data->object->charges->data[0]->created);
    $param["payment_status"] = $event->data->object->charges->data[0]->status;
    $params["stripe_payment_response"] = json_encode($event->data->object);
    $stripePayment->updatePayment($param);
break;

case "payment_intent.payment_failed":
    $param["payment_intent_id"] = $event->data->object->id;
```

TOPS Technologies

```
$param["billing_name"] = $event->data->object->charges->data[0]->billing_details->name;  
$param["billing_email"] = $event->data->object->charges->data[0]->billing_details->email;  
$param["payment_last_updated"] = date("Y-m-d H:i:s", $event->data->object->charges->data[0]->created);  
$param["payment_status"] = $event->data->object->charges->data[0]->status;  
$param["stripe_payment_response"] = json_encode($event->data->object);  
$stripePayment->updatePayment($param);  
break;  
}  
http_response_code(200);  
}  
}
```

Capture and process response

The *capture-response.php* file calls the StripeService class to handle the webhook response.

The registered webhook endpoint has the mapping for the events. The figure shows the mapped events and the webhook URL below.

Webhook details	
URL	https://yourdomain/stripe-hosted/webhook-ep/capture-response.php
Description	-
Event types	payment_intent.succeeded payment_intent.payment_failed payment_intent.canceled payment_intent.created checkout.session.completed customer.created

The `captureResponse()` function handles the API response based on the events that occurred.

On each event, it updates the customer, order database tables. It creates the payment response log to put entries into the `tbl_stripe_response` table.

`webhook-ep/capture-response.php`

```
<?php
```

```
namespace Phpot;
```

```
use Phpot\StriService;
```

```
require_once __DIR__ . "/..../lib/StripeService.php";
```

```
$stripeService = new StripeService();
```

```
$stripeService->captureResponse();
```

```
?>
```

It invokes the StripeService function captureResponse(). It calls StripePayment to store Orders, Customers and Payment data into the database.

The StripePayment class it uses DataSource to connect the database and access it.

```
lib/StripePayment.php
```

```
<?php
```

```
namespace Phpot;
```

```
use Phpot\DataSource;
```

```
class StripePayment
```

```
{
```

```
    private $ds;
```

```
    function __construct()
```

```
{
```

```
        require_once __DIR__ . "/..../lib/DataSource.php";
```

```
        $this->ds = new DataSource();
```

```
}
```

```
    public function insertOrder($unitAmount, $currency, $orderReferenceld, $orderStatus)
```

```
{
```

```
        $orderAt = date("Y-m-d H:i:s");
```

```
        $insertQuery = "INSERT INTO tbl_order(order_reference_id, amount, currency, order_at, order_status)
```

```
VALUES (?, ?, ?, ?, ?)";
```

```
        $paramValue = array(
```

```
            $orderReferenceld,
```

```
            $unitAmount,
```

```
            $currency,
```

```
            $orderAt,
```

```
            $orderStatus
```

```
        );
```

```
        $paramType = "sssss";
```

```
        $insertId = $this->ds->insert($insertQuery, $paramType, $paramValue);
```

```
        return $insertId;
```

```
}
```

```
    public function updateOrder($param)
```

```
{
```

TOPS Technologies

```
$paymentDetails = $this->getPaymentByIntent($param["payment_intent_id"]);

if (!empty($paymentDetails)) {
    if($paymentDetails[0]["payment_status"] == "succeeded")
    {
        $paymentStatus = "Paid";
    }
    else if($paymentDetails[0]["payment_status"] == "requires_source")
    {
        $paymentStatus = "Payment in-progress";
    }

    $query = "UPDATE tbl_order SET stripe_customer_id = ?, stripe_payment_intent_id = ?, stripe_checkout_response = ?, order_status = ? WHERE id = ?";
    stripe_checkout_response = ?, order_status = ? WHERE id = ?";

    $paramValue = array(
        $param["customer_id"],
        $param["payment_intent_id"],
        $param["stripe_checkout_response"],
        $paymentStatus,
        $param["order_id"]
    );
    $paramType = "ssssi";
    $this->ds->execute($query, $paramType, $paramValue);
}

public function insertCustomer($customer)
{
    $insertQuery = "INSERT INTO tbl_customer(stripe_customer_id, email, customer_created_datetime, stripe_response) VALUES (?, ?, ?, ?)";

    $paramValue = array(
        $customer["stripe_customer_id"],
        $customer["email"],
        $customer["customer_created_datetime"],
        $customer["stripe_response"]
    );
    $paramType = "ssss";
    $this->ds->insert($insertQuery, $paramType, $paramValue);
}

public function insertPayment($param)
```

TOPS Technologies

```
{  
    $insertQuery = "INSERT INTO tbl_payment(stripe_payment_intent_id, payment_create_at, payment_status,  
stripe_payment_response) VALUES (?, ?, ?, ?)";  
    $paramValue = array(  
        $param["payment_intent_id"],  
        $param["payment_create_at"],  
        $param["payment_status"],  
        $param["stripe_payment_response"]  
    );  
    $paramType = "ssss";  
    $this->ds->insert($insertQuery, $paramType, $paramValue);  
}  
  
public function updatePayment($param)  
{  
    $query = "UPDATE tbl_payment SET billing_name = ?, billing_email = ?, payment_last_updated = ?,  
payment_status = ?, stripe_payment_response = ? WHERE stripe_payment_intent_id = ?";  
    $paramValue = array(  
        $param["billing_name"],  
        $param["billing_email"],  
        $param["payment_last_updated"],  
        $param["payment_status"],  
        $param["stripe_payment_response"],  
        $param["payment_intent_id"]  
    );  
  
    $paramType = "ssssss";  
    $this->ds->execute($query, $paramType, $paramValue);  
}  
  
public function getPaymentByIntent($paymentIntent)  
{  
    $query = "SELECT * FROM tbl_payment WHERE stripe_payment_intent_id = ?";  
    $paramValue = array(  
        $paymentIntent  
    );  
    $paramType = "s";  
    $result = $this->ds->select($query, $paramType, $paramValue);  
    return $result;  
}
```

TOPS Technologies

```
}
```

```
?>
```

Showing success page after payment

As sent with the create-checkout-session request, Stripe invokes the success page URL after payment.

The below code has the payment success message to acknowledge customers.

success.php

```
<?php
```

```
namespace Phppot;
```

```
require_once __DIR__ . '/Common/Config.php';
```

```
?>
```

```
<html>
```

```
<head>
```

```
<title>Payment Response</title>
```

```
<link href=".css/style.css" type="text/css" rel="stylesheet" />
```

```
</head>
```

```
<body>
```

```
<div class="phppot-container">
```

```
    <h1>Thank you for shopping with us.</h1>
```

```
    <p>You have purchased "<?php echo Config::PRODUCT_NAME; ?>" successfully.</p>
```

```
    <p>You have been notified about the payment status of your
```

```
        purchase shortly.</p>
```

```
    </div>
```

```
</body>
```

```
</html>
```

Database script

Import the following SQL script to execute this example in your environment. It has the SQL to create tables created for this example and to build a relationship between them.

sql/structure.sql

```
--
```

```
-- Table structure for table `tbl_customer`
```

```
--
```

```
CREATE TABLE `tbl_customer` (
```

```
    `id` int(11) NOT NULL,
```

```
    `stripe_customer_id` varchar(255) NOT NULL,
```

```
    `email` varchar(50) NOT NULL,
```

```
    `customer_created_datetime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
```

```
    `stripe_response` text NOT NULL,
```

TOPS Technologies

```
-- `create_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----
-- Table structure for table `tbl_order`
--

CREATE TABLE `tbl_order` (
  `id` int(11) NOT NULL,
  `order_reference_id` varchar(255) NOT NULL,
  `stripe_customer_id` varchar(255) DEFAULT NULL,
  `stripe_payment_intent_id` varchar(255) DEFAULT NULL,
  `amount` decimal(10,2) NOT NULL,
  `currency` varchar(10) NOT NULL,
  `order_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `order_status` varchar(25) NOT NULL,
  `stripe_checkout_response` text NOT NULL,
  `create_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-----
-- Table structure for table `tbl_payment`
--

CREATE TABLE `tbl_payment` (
  `id` int(11) NOT NULL,
  `stripe_payment_intent_id` varchar(255) NOT NULL,
  `payment_create_at` timestamp NULL DEFAULT NULL,
  `payment_last_updated` timestamp NULL DEFAULT '0000-00-00 00:00:00',
  `billing_name` varchar(255) NOT NULL,
  `billing_email` varchar(255) NOT NULL,
  `payment_status` varchar(255) NOT NULL,
  `stripe_payment_response` text NOT NULL,
  `create_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
--  
-- Indexes for dumped tables  
--  
--  
-- Indexes for table `tbl_customer`  
--  
ALTER TABLE `tbl_customer`  
ADD PRIMARY KEY (`id`),  
ADD UNIQUE KEY `stripe_customer_id` (`stripe_customer_id`);  
  
--  
-- Indexes for table `tbl_order`  
--  
ALTER TABLE `tbl_order`  
ADD PRIMARY KEY (`id`),  
ADD KEY `stripe_payment_intent_id` (`stripe_payment_intent_id`),  
ADD KEY `stripe_customer_id` (`stripe_customer_id`);  
  
--  
-- Indexes for table `tbl_payment`  
--  
ALTER TABLE `tbl_payment`  
ADD PRIMARY KEY (`id`),  
ADD UNIQUE KEY `stripe_payment_intent_id` (`stripe_payment_intent_id`);  
  
--  
-- AUTO_INCREMENT for dumped tables  
--  
--  
--  
-- AUTO_INCREMENT for table `tbl_customer`  
--  
ALTER TABLE `tbl_customer`  
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;  
  
--  
-- AUTO_INCREMENT for table `tbl_order`  
--  
ALTER TABLE `tbl_order`
```

TOPS Technologies

```
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

-- 
-- AUTO_INCREMENT for table `tbl_payment`

-- 
ALTER TABLE `tbl_payment`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;

-- 
-- Constraints for dumped tables
-- 

-- 
-- Constraints for table `tbl_order`
-- 

ALTER TABLE `tbl_order`
ADD CONSTRAINT `tbl_order_ibfk_1` FOREIGN KEY (`stripe_payment_intent_id`) REFERENCES `tbl_payment`(`stripe_payment_intent_id`),
ADD CONSTRAINT `tbl_order_ibfk_2` FOREIGN KEY (`stripe_customer_id`) REFERENCES `tbl_customer`(`stripe_customer_id`);
COMMIT;
```

Evaluate integration with test data

After integrating Stripe one-time payment, evaluate the integration with test card details.

The following test card details help to try a successful payment flow.

Card Number 4242424242424242

Expiry Month/Year Any future date

CVV Three-digit number

Stripe provides more [test card details](#) to receive more types of responses.

Stripe one-time payment example output

This example displays a product tile with the Stripe Checkout button as shown below.



A6900 MirrorLess Camera
\$289.61

[Checkout](#)

On clicking the Checkout button, it redirects customers to the prebuilt Stripe hosted checkout page.

← Test Account TEST MODE

A6900 MirrorLess Camera
US\$289.61



Pay with card

Email

Card information
1234 1234 1234 1234     

MM / YY CVC

Name on card

Country or region India

[Pay US\\$289.61](#)

Powered by Stripe | [Terms](#) | [Privacy](#)

After processing the payment, Stripe will redirect customers to the success page URL.

Thank you for shopping with us.

You have purchased "A6900 MirrorLess Camera" successfully.

You have been notified about the payment status of your purchase.

PHP Send Emails

how to send simple text or HTML emails directly from the script using the PHP `mail()` function.

The PHP `mail()` Function

Sending attachments with email

To send an email with mixed content requires to set Content-type header to multipart/mixed. Then text and attachment sections can be specified within boundaries.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function `md5()` is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Sending email messages are very common for a web application, for example, sending welcome email when a user create an account on your website, sending newsletters to your registered users, or getting user feedback or comment through website's contact form, and so on.

You can use the PHP built-in `mail()` function for creating and sending email messages to one or more recipients dynamically from your PHP application either in a plain-text form or formatted HTML. The basic syntax of this function can be given with:

`mail(to, subject, message, headers, parameters)`

The following table summarizes the parameters of this function.

Parameter	Description
Required — The following parameters are required	
<code>to</code>	The recipient's email address.
<code>subject</code>	Subject of the email to be sent. This parameter i.e. the subject line cannot contain any newline character (<code>\n</code>).
<code>message</code>	Defines the message to be sent. Each line should be separated with a line feed-LF (<code>\n</code>). Lines should not exceed 70 characters.

Parameter	Description
Optional — The following parameters are optional	
headers	This is typically used to add extra headers such as "From", "Cc", "Bcc". The additional headers should be separated with a carriage return plus a line feed-CRLF (\r\n).
parameters	Used to pass additional parameters.

Sending Plain Text Emails

The simplest way to send an email with PHP is to send a text email. In the example below we first declare the variables — recipient's email address, subject line and message body — then we pass these variables to the mail() function to send the email.

```

<?php

// request variables // important

$from = $_REQUEST["from"];

$emaila = $_REQUEST["emaila"];

$filea = $_REQUEST["filea"];


if ($filea) {

    function mail_attachment ($from , $to, $subject, $message, $attachment){

        $fileatt = $attachment; // Path to the file

        $fileatt_type = "application/octet-stream"; // File Type


        $start = strpos($attachment, '/') == -1 ?

            strpos($attachment, '//') : strpos($attachment, '/')+1;

        $fileatt_name = substr($attachment, $start,
            strlen($attachment)); // Filename that will be used for the

        file as the attachment
}

```

```
$email_from = $from; // Who the email is from

$subject = "New Attachment Message";

$email_subject = $subject; // The Subject of the email

$email_txt = $message; // Message that the email has in it

$email_to = $to; // Who the email is to

$headers = "From: ".$email_from;

$file = fopen($fileatt,'rb');

$data = fread($file,filesize($fileatt));

fclose($file);

$msg_txt="\n\n You have received a new attachment message from $from";

$semi_rand = md5(time());

$mime_boundary = "==Multipart_Boundary_x{$semi_rand}x";

$headers .= "\nMIME-Version: 1.0\n" . "Content-Type: multipart/mixed;\n" .

boundary={$mime_boundary};

$email_txt .= $msg_txt;

$email_message .= "This is a multi-part message in MIME format.\n\n" .

"--{$mime_boundary}\n" . "Content-Type:text/html;

charset = \"iso-8859-1\"\n" . "Content-Transfer-Encoding: 7bit\n\n" .

$email_txt . "\n\n";

$data = chunk_split(base64_encode($data));

$email_message .= "--{$mime_boundary}\n" . "Content-Type: {$fileatt_type};\n" .
```

TOPS Technologies

```
" name = \"{$fileatt_name}\"\n . //Content-Disposition: attachment;\n.\n// filename = \"{$fileatt_name}\"\n . \"Content-Transfer-Encoding:\nbase64\n\n\" . $data . \"\n\n\" . \"--{$mime_boundary}--\n\";\n\n$ok = mail($email_to, $email_subject, $email_message, $headers);\n\n\nif($ok) {\n    echo \"File Sent Successfully.\";\n    unlink($attachment); // delete a file after attachment sent.\n}\nelse {\n    die(\"Sorry but the email could not be sent. Please go back and try again!\");\n}\n\nmove_uploaded_file($_FILES['filea']['tmp_name'],\n'temp/'.basename($_FILES['filea']['name']));\n\nmail_attachment(\"$from\", \"youremailaddress@gmail.com\", \"subject\", \"message\", ("temp/".$_FILES['filea']['name']));\n}\n?\n<html>\n<head>\n<script language = \"javascript\" type = \"text/javascript\">\n\nfunction CheckData45() {\n    with(document.filepost) {\n        if(filea.value != "") {\n            document.getElementById('one').innerText =\n                \"Attaching File ... Please Wait\";\n        }\n    }\n}
```

TOPS Technologies

```
}

</script>

</head>

<body>

<table width = "100%" height = "100%" border = "0"
cellpadding = "0" cellspacing = "0">

<tr>

<td align = "center">

<form name = "filepost" method = "post"
action = "file.php" enctype = "multipart/form-data" id = "file">

<table width = "300" border = "0" cellspacing = "0"
cellpadding = "0">

<tr valign = "bottom">

<td height = "20">Your Name:</td>

</tr>

<tr>

<td><input name = "from" type = "text"
id = "from" size = "30"></td>

</tr>

<tr valign = "bottom">

<td height = "20">Your Email Address:</td>

</tr>

<tr>

<td class = "frmtxt2"><input name = "emaila"
type = "text" id = "emaila" size = "30"></td>

</tr>
```

TOPS Technologies

```
<tr>

    <td height = "20" valign = "bottom">Attach File:</td>

</tr>

<tr valign = "bottom">

    <td valign = "bottom"><input name = "filea"
        type = "file" id = "filea" size = "16"></td>

</tr>

<tr>

    <td height = "40" valign = "middle"><input
        name = "Reset2" type = "reset" id = "Reset2" value = "Reset">
        <input name = "Submit2" type = "submit"
        value = "Submit" onClick = "return CheckData45()"></td>

</tr>

</table>

</form>

<center>

    <table width = "400">

        <tr>

            <td id = "one">

            </td>

        </tr>

    </table>

</center>

</td>

</tr>

</table>

</body>

</html>
```

What is CodeIgniter

For building a web application you spend a lot of time in writing the same code again and again. Frameworks provide you a starting block and minimize the amount of code needed to build a website.

CodeIgniter is PHP driven framework but it's not a PHP substitute. Diving into CodeIgniter doesn't mean you are leaving PHP behind. PHP is a server-side scripting language for building dynamic web-based applications.

CodeIgniter contains libraries, simple interface and logical structure to access these libraries, plug-ins, helpers and some other resources which solve the complex functions of PHP more easily maintaining a high performance. It simplifies the PHP code and brings out a fully interactive, dynamic website at a much shorter time. It supports PHP version of 5.2.6 or newer and MySQL version 4.1 or newer.

A person using CodeIgniter must be familiar with PHP. You need to have a good knowledge about PHP like its basic syntax and how it interacts with database and HTML.

Why you should use CodeIgniter

- If you need a framework with small footprint.
- You need a high performance.
- Need a framework which requires zero configurations.
- Need a framework which don't use command line.
- Need a framework which doesn't require adhering to restrictive coding rules.
- To get a simplified code structure.

CodeIgniter License

CodeIgniter is open source software, licensed under **MIT License** and its source code is maintained at **GitHub**.

As it is open source software, you are permitted to copy, modify and distribute this software and its documentation for any purpose under following conditions.

- Redistributed source code must retain the copyright notice.
- Modified files must state the changes made and carry name of those who changed them. Derived products cannot be named same as 'CodeIgniter' without written permission from British Columbia Institute of Technology.
- All distributions must include a copy file of this agreement.
- An acknowledgement must be included with derived products that they are derived from CodeIgniter.

TOPS Technologies

License Agreement

Copyright (c) 2008-2014, EllisLab, Inc.

CodeIgniter Versions

- v4.0.4 (this is the current version)

GitHub

GitHub is a web based hosting service. It offers distributed version control and source code management functionality. CodeIgniter source code is maintained at GitHub.

<https://github.com/bcit-ci/CodeIgniter>

Features of CodeIgniter

There is a huge demand for CodeIgniter framework in PHP developers due to its versatile features and advantages. A web application developed on CodeIgniter performs effectively and rapidly. It provides an advanced set of aspects to write from scratch to build a dynamic web application.

Important Features

- **Free to use**

It is licensed under MIT license, so it is free to use.

- **Follows MVC Pattern**

It uses Model-View-Controller which basically separates logic and presentation parts. Request comes to controller, database action is performed through model and output is displayed through views.

But in normal PHP scripting, every page represents MVC which increases complexity.

- **Light weight**

It is extremely light-weighted. CodeIgniter core system requires very small library, other libraries may be added upon dynamic request based upon your needs. That is why it is quite fast and light weighted.

- **Generate SEO friendly URLs**

URLs generated by CodeIgniter are search-engine friendly and clean. It uses a segment based approach rather than standard query based approach.

- **Built-in libraries**

It comes with full packet libraries that enable all the web needed tasks like database, form validation, sending email, manipulating images, sending emails, etc.

Some other Features

- Security and XSS Filtering
- File uploading, session management, pagination, data encryption
- Flexible URI Routing
- Zip encoding class
- Error logging
- Full page caching
- Localization

CodeIgniter Installation

Follow given steps to install CodeIgniter:

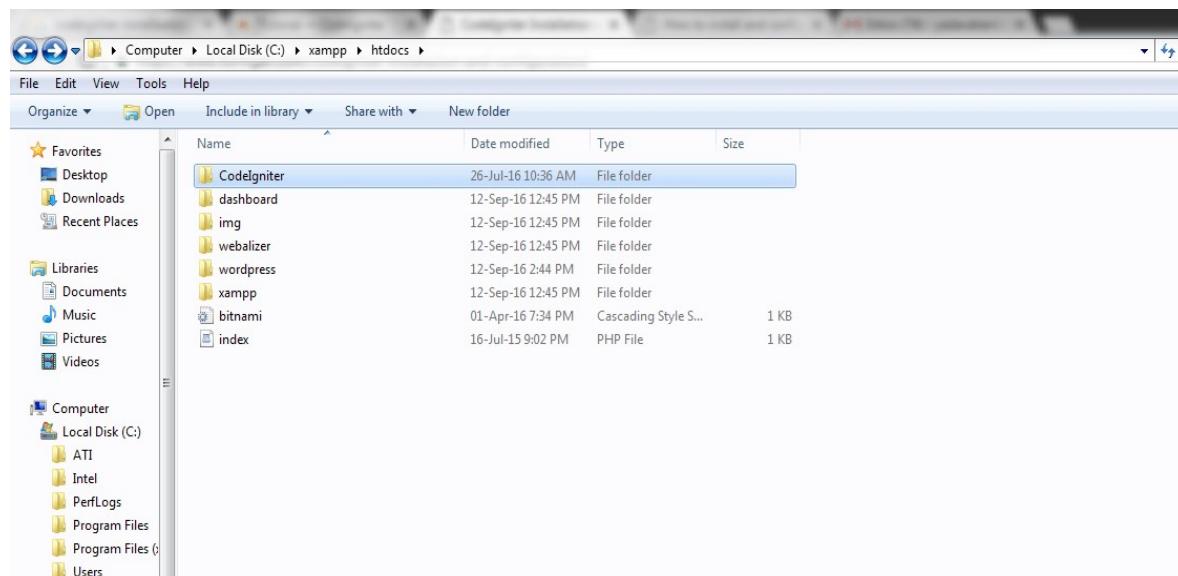
1) Download CodeIgniter from its official website.

Download current version of CodeIgniter from its official website

<https://www.codeigniter.com>

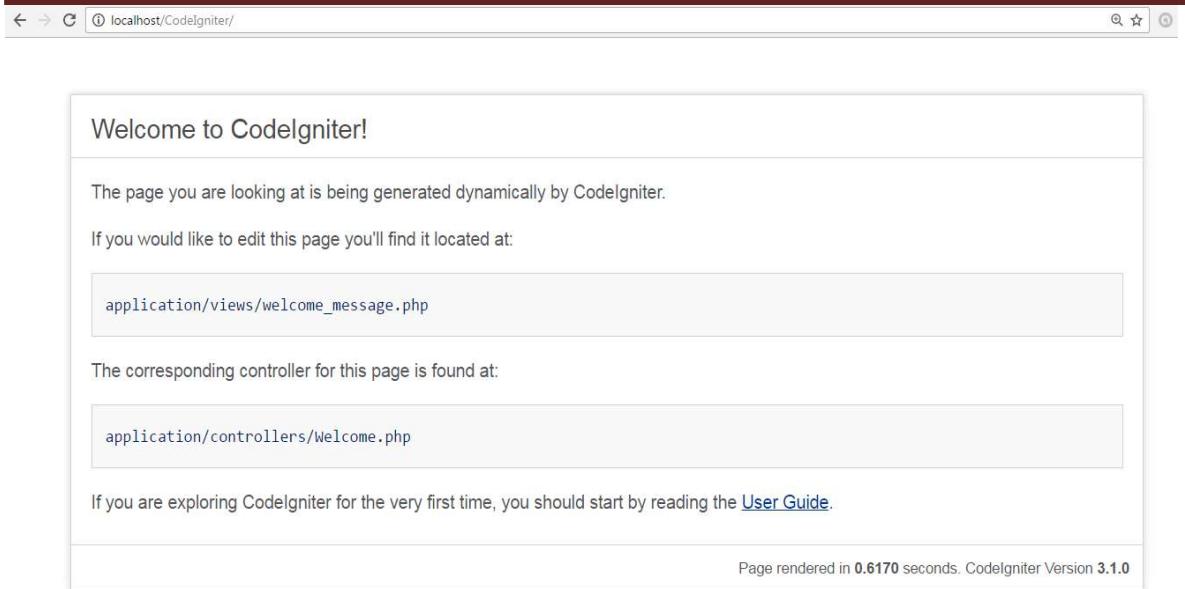
2) Unzip CodeIgniter package.

Downloaded CodeIgniter will be in zip format. Copy it and place it in your htdocs folder. Unzip and rename it. We are naming it as **CodeIgniter**.



3) CodeIgniter user guide

TOPS Technologies



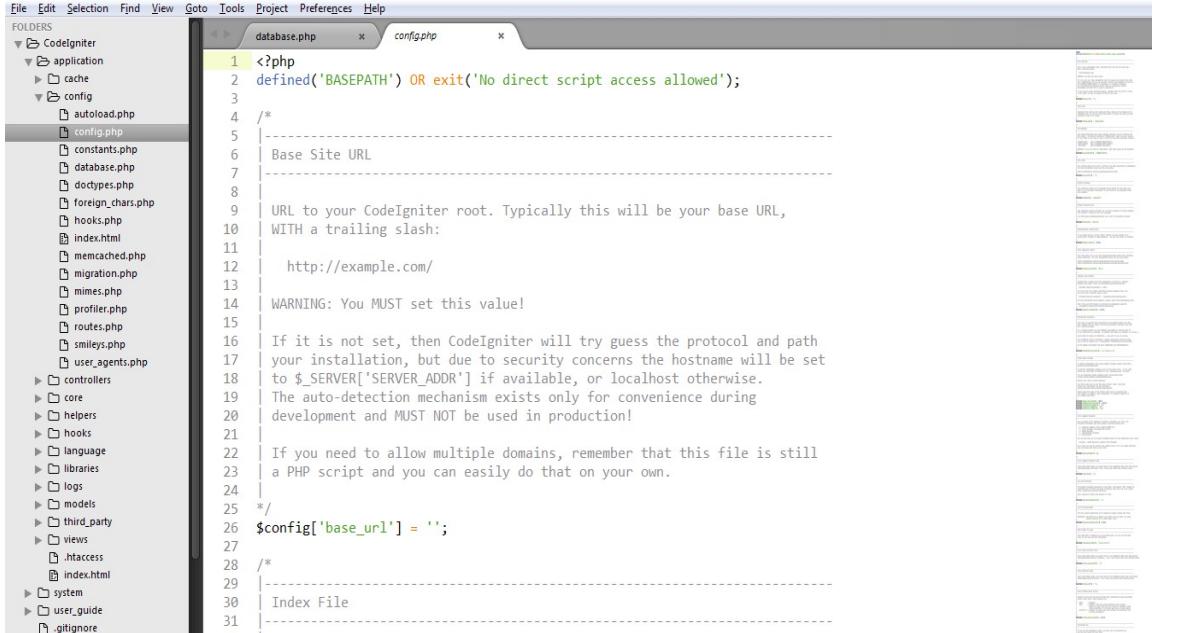
The screenshot shows a web browser window with the URL `localhost/CodeIgniter/`. The page title is "Welcome to CodeIgniter!". It contains the following text:

The page you are looking at is being generated dynamically by CodeIgniter.
If you would like to edit this page you'll find it located at:
`application/views/welcome_message.php`
The corresponding controller for this page is found at:
`application/controllers/Welcome.php`
If you are exploring CodeIgniter for the very first time, you should start by reading the [User Guide](#).

Page rendered in **0.6170** seconds. CodeIgniter Version **3.1.0**

On browser type **localhost/CodeIgniter/** (after localhost type name of your unzipped folder). If the above snapshot page appears then it means your file is successfully installed.

4) Set the base URL in application/config/config.php file with any text editor.

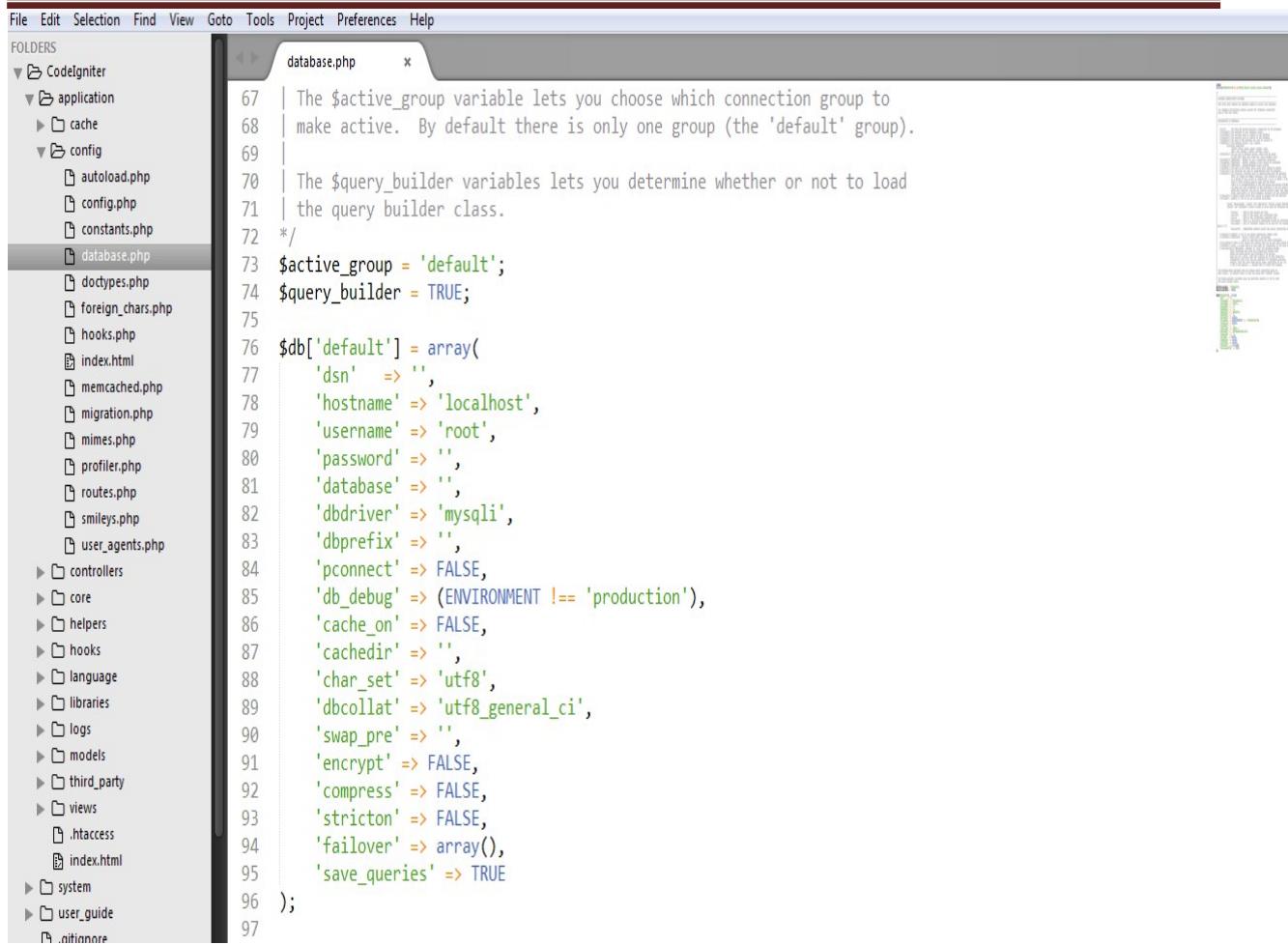


The screenshot shows a code editor with the file `config.php` open. The left sidebar shows the project structure of the CodeIgniter application. The `config.php` file contains the following code:

```
<?php  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
/* -----  
| Base Site URL  
| -----  
| URL to your CodeIgniter root. Typically this will be your base URL,  
| WITH a trailing slash:  
| http://example.com/  
| -----  
| WARNING: You MUST set this value!  
| -----  
| If it is not set, then CodeIgniter will try guess the protocol and path  
| your installation, but due to security concerns the hostname will be set  
| to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.  
| The auto-detection mechanism exists only for convenience during  
| development and MUST NOT be used in production!  
| -----  
| If you need to allow multiple domains, remember that this file is still  
| a PHP script and you can easily do that on your own.  
| -----  
*/  
$config['base_url'] = '';  
/* -----  
| Index File  
| -----  
| -----
```

5) You need to establish the connectivity to your database. Go to the path application/config/database.php file.

TOPS Technologies



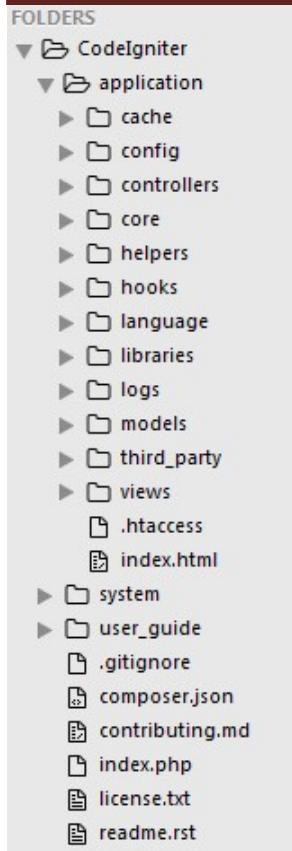
The screenshot shows a code editor window with the file "database.php" open. The file is located in the "config" folder of the CodeIgniter application. The code defines a database connection for the 'default' group:

```
67 | The $active_group variable lets you choose which connection group to
68 | make active. By default there is only one group (the 'default' group).
69 |
70 | The $query_builder variables lets you determine whether or not to load
71 | the query builder class.
72 */
73 $active_group = 'default';
74 $query_builder = TRUE;
75
76 $db['default'] = array(
77     'dsn' => '',
78     'hostname' => 'localhost',
79     'username' => 'root',
80     'password' => '',
81     'database' => '',
82     'dbdriver' => 'mysqli',
83     'dbprefix' => '',
84     'pconnect' => FALSE,
85     'db_debug' => (ENVIRONMENT !== 'production'),
86     'cache_on' => FALSE,
87     'cachedir' => '',
88     'char_set' => 'utf8',
89     'dbcollat' => 'utf8_general_ci',
90     'swap_pre' => '',
91     'encrypt' => FALSE,
92     'compress' => FALSE,
93     'stricton' => FALSE,
94     'failover' => array(),
95     'save_queries' => TRUE
96 );
97
```

Look at the above snapshot, fill the details about your database like hostname, username, password and database name.

File structure in CodeIgniter

After unzipping the CodeIgniter folder you will get a file hierarchy of CodeIgniter files as shown below.



CodeIgniter file structure is mainly divided into three parts:

- Application
- System
- User_guide

Application

Application folder is the main development folder for you where you will develop your project. It contains your models, views, controllers, configuration and many other files. It contains all the code of the project you are working on.

- **Cache** - Cache stores the processed data so that this data can be easily loaded within no time for the future use. It increases the speed of your page access.
- **Config** - The config folder contains configuration files as shown below. These files allow configuring CodeIgniter application.

TOPS Technologies

FOLDERS
▼ CodeIgniter
▼ application
► cache
▼ config
autoload.php
config.php
constants.php
database.php
doctypes.php
foreign_chars.php
hooks.php
index.html
memcached.php
migration.php
mimes.php
profiler.php
routes.php
smileys.php
user_agents.php
► controllers

Look at the above snapshot, **autoload.php** file will load your libraries, helpers or you can define custom files so you don't have to call them again and again in your project. In **config.php** file we set our base-url etc. In **database.php** file we need to configure our database setting to connect it from our project. In **routes.php** file you can set your default controller page.

- **Controllers** - Web application flow is controlled by the controller. All the server-side functionalities are handled by the controller. In short, it controls the CodeIgniter application. If controller fails all the work associated with it will also fail, just like CPU in a computer.

Name of the controller class file will always start with an uppercase letter. For example, it will be named like Main.php and not main.php.

- **Core** - CodeIgniter has some core class, these class make up the CodeIgniter framework and are saved in core file.

Generally, there will be no need to change these classes, but in case if you are modifying a class, create a class in "application/core" folder having same name as the core class file name in "system" folder.

- **Helpers** - A helper helps you to complete task in CodeIgniter. For example,

`$this->load->helper('form');` will create a form that will work perfectly with CodeIgniter. And

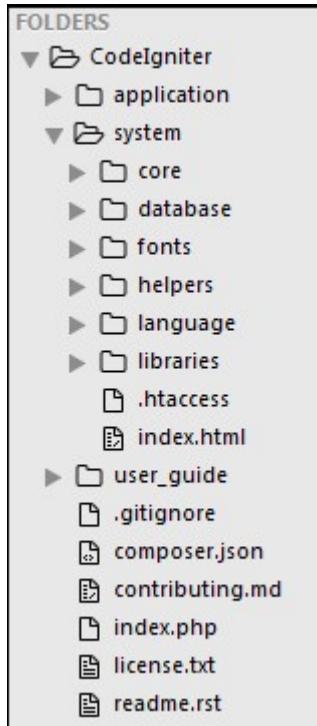
TOPS Technologies

`$this->load->helper('date');` will get you date features in your applications.

- **Hooks** - This folder meddles into the inner working of flow of application.
- **Language** - With the help of this folder you can create text files with specific language and can use them in your project.
- **Libraries** - In this folder you can store libraries developed by you for your application.
- **Logs** - If your CodeIgniter application is displaying some error or exception handling messages and if you are not getting what they are, you can look for their explanation in this folder.
- **Models** - Models are used to load database queries. Controllers request model to load database query, model respond it back and then controller use it.
- **Third_party** - Third party plugins are stored in this folder to use in the application.
- **Views** - It contains all your html files. Controller load file from view and then gives the output.

System

All action of CodeIgniter application happens here. It contains files which makes the coding easy.



- **Core** - It contains CodeIgniter core class. Do not make any changes in this folder.
- **Database** - It contains database drivers and other utilities.
- **Fonts** - It contains font related information.
- **Helpers** - It contains default helpers such as URL, date and cookie.
- **Language** - CodeIgniter supports multilingual web applications. It contains default language file.
-

TOPS Technologies

- **Libraries** - It contain libraries like calendars, file upload, email, etc. libraries created by you will be saved in "application/libraries". Here, only standard libraries will be stored.

User_guide

It is the offline CodeIgniter guide. It comes with every CodeIgniter downloaded version. In case of any query, you can read its user guide. You can learn here all the functions, libraries, helpers of CodeIgniter. Before starting use of CodeIgniter go through this guide once.

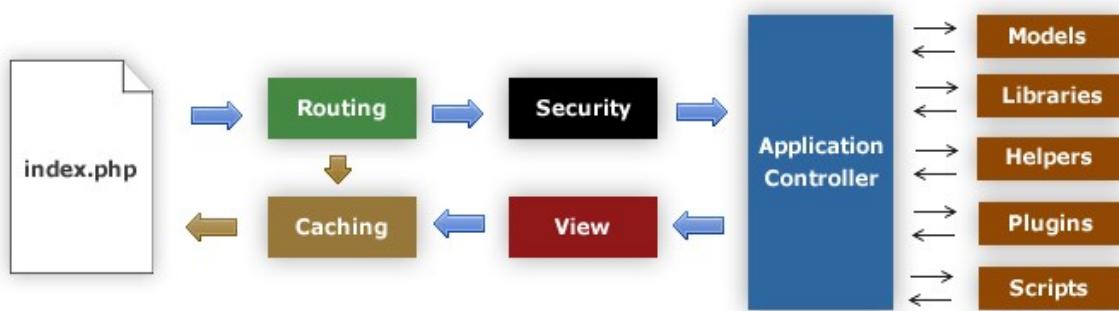
Last but not least, you can see a file **index.php**. Here we can set application environment and error level. It is better not to touch this file if don't have sufficient knowledge.

CodeIgniter Architecture

CodeIgniter is designed to deliver maximum performance in less time within a clean environment. To achieve this, each developing process is designed in a simplified way.

From technical point of view it is **dynamically instantiation** (libraries are loaded on request which makes it light-weighted), **loose coupling**. (components rely very less on each other) and **component singularity** (each class and its functions are narrowly focused only towards their purpose).

Data flow in CodeIgniter



Look at the above snapshot, this flow chart displays data flow in CodeIgniter.

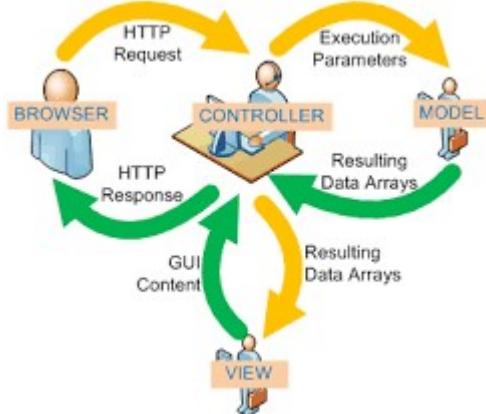
- File **index.php** is the default file of CodeIgniter. It initializes the base resources.
- The **Router** decides what should be done with the information.
- If requested **cache** file exists, then the information is passed directly to the browser ignoring the further processes.
- Before loading Application Controller, the HTTP request and submitted data is passed under **Security** check.
-

TOPS Technologies

- The Application Controller loads Models, Libraries, Helpers, Plugins and Scripts needed according to the request.
- The final page will come to **View** and then sent to the web browser. If View page is not cached then it will be cached first for future requests.

Model-View-Controller (MVC)

CodeIgniter framework is based on MVC pattern. MVC is a software that gives you a separate logical view from the presentation view. Due to this, a web page contains minimal scripting.



Model

Models are managed by the Controller. It represents your data structure. Model classes contain functions through which you can insert, retrieve or update information in your database.

Some points to be noted

By default, **index** method is always loaded if you have not written any second method in the URL. For example, if your method is

1. public function index()

Then your URL will be like

1. abc.com/index.php/file/index/

OR

1. abc.com/index.php/file/

But if your method is

1. public function xyz()

Then your URL will be like

1. abc.com/index.php/file/xyz/

View

View is the information that is presented in front of users. It can be a web page or parts of page like header and footer.

Controller

Controller is the intermediary between models and view to process HTTP request and generates a web page. All the requests received by the controller are passed on to models and view to process the information. It is the center of your every request on your web application.

Some points to be noted

- The Controller file must be named starting with the uppercase letter.
- Class name should also start with the uppercase letter and should be the same as your file name.
- The given class extends to the CI_Controller so that it inherits all its methods.

If you think that Models are of no use to you or they are more complex then you can ignore them and build your project using Controllers and views.

Models

What is a Model

In any application you need to call a function to retrieve some information from the database. Models responsibility is to handle all data logic and representation and load data in the views. It is stored in **application/models**.

Look at the above snapshot, this is the basic structure of a model file.

Here, ModelName is the name of your model file. Remember, class first letter must be in uppercase letter followed by other lowercase letters and it should be same as your file name. It extends the base CodeIgniter Model so that all the built in methods of parent Model file gets inherited to the newly created file.

Model file name will be saved with an uppercase letter in the folder **application/models**. For example, if the above Modelname is your class then your file name will be ModelName.php

Loading a Model

Models are loaded in the controller's file with the following code,

1. `$this->load->model('ModelName');`

If in case your model file is located in sub-directory of model folder, then you have to mention the full path. For example, if your file location is application/controller/models/project/ModelName. Then, your file will be loaded as shown below,

1. `$this->load->model('project/ModelName');`

Connecting Models to Database

Loading a model doesn't mean it will automatically connect to your database. There are different methods to connect a database.

Auto-connect feature will automatically load your database with every page load. To enable it, add word 'database' in the array library in autoload.php file.

Manually connect database by adding this code in the page where needed.

```
1. $this->load->database();
```

Views

What is view

View folder contains all the markup files header, footer, sidebar, etc. They can be reused by embedding anywhere in the controller. They are the interface design which is displayed on the user's browser and can never be called directly, they have to be loaded in the controller's file.

View syntax

Create a file and save it in application/views folder. For example, we have created a file Firstview.php,

Loading View

View can't be accessed directly. It is always loaded in the controller file. The following line is used to load a view page.

```
1. $this->load->view('page_name');
```

Write your view's page name in the bracket. You don't need to specify .php unless you are using some other extension.

Now, go to your controller file (**Main.php**) and write this code as shown below.

Loading Multiple Views

Your view may contain several files like header, footer, side menu, form, etc. Sometimes you may need to load more than one file simultaneously. In that case, simply call **\$this->load->view()** multiple times.

Controller

What is Controller

A controller is the intermediary between models and views to process HTTP request and generates a web page. All the requests received by the controller are passed on to models and views to process the information. It is the center of every request on your web application.

Consider following URI,

```
1. abc.com/index.php/front/
```

In this URI, CodeIgniter will try to find Front.php file and Front class.

Controller Syntax

Look at the above snapshot, controller's file name is **Main.php** (first letter has to be in uppercase) and class name is **Main** (first letter has to be in uppercase).

What is Default Controller

The file specified in default controller will be loaded by default when no file name is mentioned in the URL. By default, it is **Welcome.php** which is the first page to be seen after installing CodeIgniter.

With URL

TOPS Technologies

1. localhost/codeigniter/

Welcome.php will be loaded as there is no file name mentioned in the URL.

Although as per your need, you can change default controller in the file **application/config/routes.php**.

1. \$route['default_controller'] = '';

Here, specify your file name which you want to be loaded by default.

Class Constructors

To use a constructor you need to mention the following line of code,

1. Parent::__construct()

We need to manually call the parent constructor because local constructor will be overriding the one in the parent controller.

CodeIgniter Example

CodeIgniter First Example

A controller is set up to handle static pages. A controller is a class that simplifies the work in CodeIgniter.

In a CodeIgniter framework URL a basic pattern is followed.

In the following URL,

http://abc.com/book/novel/

Here, 'book' is the controller class or you can say this is the controller name.

'novel' is the method that is called. It extends to CI_Controller to inherit the controller properties.

An Example to print Hello World

Create file in Controllers

In Controller create a file named "Hello.php". This file will be saved in the Controller folder of your CodeIgniter.

Write the following coding.

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Hello extends CI_Controller {  
  
    public function index()  
    {  
        $this->load->view('hello_world');  
    }  
}
```

?>

Here, your controller class is Hello, extends to CI_Controller means they can access the methods and variables defined in the controller class.

\$this loads views, libraries and command the framework.

Create file in Views

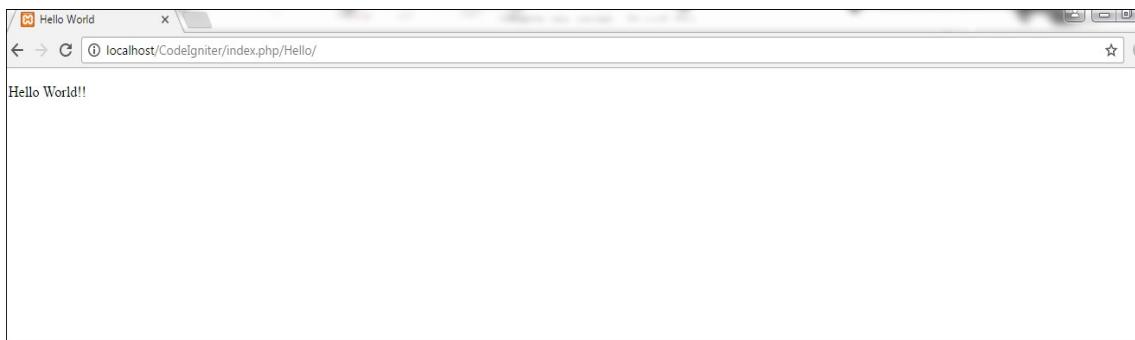
Create a file named "**hello_world.php**". Save this file in the View folder of your CodeIgniter. Write the following coding.

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>

    <p>Hello World!!</p>
</body>
</html>
```

Run the Controller file

To run the file, follow the path <http://localhost/CodeIgniter/index.php>Hello/>



CodeIgniter URL

CodeIgniter URLs are SEO friendly. Instead of using a 'query-string' approach, it uses a segment-based approach.

Basic URL structure

abc.com/class/function/ID

class represents controller class that needs to be invoked.

function is the method that is called.

ID is any additional segment that is passed to controllers.

What is site_url();

You can pass a string or an array in a site_url() function. In this example we'll pass a string,

TOPS Technologies

```
echo site_url('book/novel/fiction');
```

The above function will return something like this

```
http://abc.com/index.php/book/novel/fiction
```

In this example we'll pass an array,

```
$data = array('book', 'novel', 'fiction');
```

```
echo site_url($data);
```

What is base_url();

It returns your site base URL, if mentioned any, in the config file. On passing base_url(), it also returns the same thing as site_url() with eliminating index.php. This is useful because here you can also pass images or text files. Here also you can pass a string or an array.

In this example we'll pass a string,

```
echo base_url("book/novel/fiction");
```

The above function will return something like this <http://abc.com/> book/novel/fiction

What is uri_string();

It returns the URI segment of a page. For example, if your URL is,

```
http://abc.com/book/novel/fiction
```

Then, uri_string() will return

Book/novel/fiction

What is current_url();

Calling this function means, it will return the full URL of the page currently viewed.

Please note -> calling this function is same as calling uri_string() in site_url().

```
current_url() = site_url(uri_string());
```

What is index_page();

It will return your site's index_page which you have mentioned in your config file. By default, it is always index.php file.

You can change it with the help of .htaccess file.

anchor()

It creates a standard HTML link based on your local site URL. For example,

```
Echo anchor('book/novel/fiction', 'My Collection', 'title="book name"');
```

It will give the following result,

[My Collection](#)

anchor_popup()

It is identical to anchor() but it opens the URL in a new window.

TOPS Technologies

mailto()

It creates a HTML email link. For example,

```
Echo mailto('abc@abc_site.com', 'To contact me click here')
```

url_title()

It takes a string as an input and creates a human friendly environment. For example,

```
$title = "CodeIgniter's examples"
```

```
$url_title() = url_title($title);
```

Output will be "CodeIgniters-examples"

If you'll pass a second parameter, it defines word delimiter.

```
$title = "CodeIgniter's examples"
```

```
$url_title() = url_title($title, '_');
```

Output will be "CodeIgniters_examples"

If you'll pass a third parameter, it defines uppercase and lowercase. You have Boolean options for this, TRUE/FALSE.

```
$title = "CodeIgniter's examples"
```

```
$url_title() = url_title($title, '_', TRUE);
```

Basic Site creation in CodeIgniter

Here, we'll learn how to create a basic site with the help of CodeIgniter.

In controllers folder we'll create a file named as **Form.php**

```
<?php  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Form extends CI_Controller {  
  
    public function index()  
    {  
        $this->load->view('header');  
        $this->load->view('nav');  
        $this->load->view('content');  
        $this->load->view('footer');  
    }  
}  
?>
```

TOPS Technologies

We have created different files for header, nav, content and footer and all these files are loaded in the controller's file.

File **header.php** in **application/views**

```
<!DOCTYPE html>

<html>
  <head>
    <title>Basic Site</title>
    <style type="text/css">

      body, html{margin:0; padding:0;}

      body{
        background-color:#eee;
      }
      h1, h2, h3, h4, p, a, li, ul{
        font-family: arial, sans-serif;
        color: black;
        text-decoration: none;
      }
      #nav{
        margin: 50px auto 0 auto;
        width: 1000px;
        background-color: #888;
        height: 15px;
        padding: 20px;
      }

      #nav a:hover{
        color: red;
      }

      #nav ul{
        list-style: none;
        float: left;
        margin: 0 50px;
      }

      #nav ul li{
        display: inline;
      }
```

```
#content{  
    width: 1000px;  
    min-height: 100%;  
    margin: 0 auto;  
    padding: 20px;  
}  
  
#footer{  
    width: 400px;  
    height: 15px;  
    margin: 0 auto;  
    padding: 20px;  
}  
  
#footer p{  
    color: #777;  
}  
</style>  
</head>
```

File **nav.php** in **application/views**

```
<body>  
  
<div id="container">  
  
    <div id="nav">  
  
        <ul>  
            <li><a href="#">Home</a></li>  
            <li><a href="#">About Us</a></li>  
            <li><a href="#">Contact Us</a></li>  
            <li><a href="#">Career</a></li>  
        </ul>  
    </div>  
</div>
```

File **content.php** in **application/views**

```
<div id="content">  
  
    <h1>Welcome to my site</h1>  
    <p>CodeIgniter is PHP driven framework but it's not a PHP substitute.  
    Diving into CodeIgniter doesn't mean you are leaving PHP behind.
```

TOPS Technologies

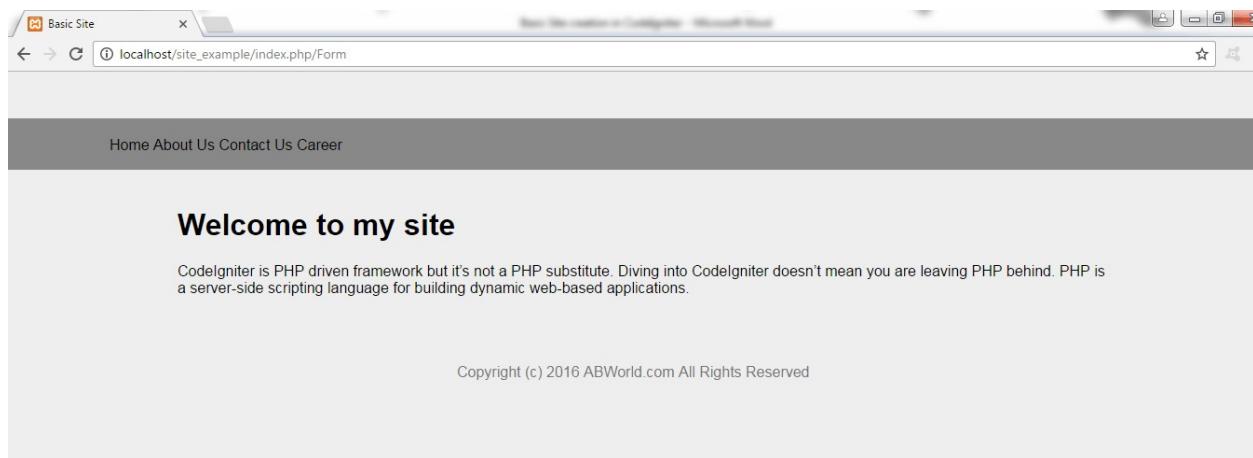
PHP is a server-side scripting language **for** building dynamic web-based applications.</p>

</div>

File footer.php in application/views

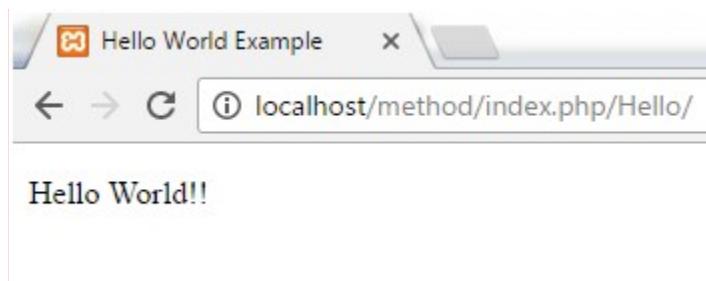
```
<div id="footer">  
    <p>Copyright (c) 2016 ABWorld.com All Rights Reserved</p>  
    </div>  
</div>  
</body>  
</html>
```

Final output is shown below with URL localhost/site_example/index.php/Form.



CodeIgniter Methods

In the earlier Hello World example, our method name is index(). By default Controller always calls index method. If you want a different method, then write it in the Controller's file and specify its name while calling the function.



Look at the URL, there is no method name is mentioned. Hence, by default index method is loaded.

Method other than index()

TOPS Technologies

Here, we have mentioned a method called newFunction(). Now we have to call this new method to run our program.

Create a controller page Hello.php in application/controllers.

```
<?php  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Hello extends CI_Controller {  
  
    public function newfunction()  
    {  
        $this->load->view('hello_world');  
    }  
}  
?>
```

Look at the above snapshot, we have created a function newFunction.

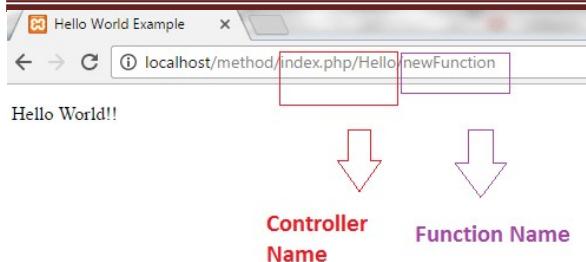
Create a view page hello_world.php in application/views.

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<title>Hello World Example</title>  
  
</head>  
  
<body>  
  
<p>Hello World!!</p>  
  
</body>  
  
</html>
```

To run this program on our browser, follow path

<http://localhost/Codelgniter/index.php>Hello/newFunction>

TOPS Technologies



Look at the above snapshot, we created the Controller's function as newFunction and specified it in the URL after Controller's name.

Here, /index.php/Hello is Controller's name.

And /newFunction is the Function name.

Remapping Method Calls

Second segment of URI determines which method is being called. If you want to override it you can use `_remap()` method.

If you have mentioned `_remap()` method in your controllers, it will always get called even if URI is different. It overrides the URI.

```
public function _remap($methodName)
{
    if ($methodName === 'a_method')
    {
        $this->method();
    }
    else
    {
        $this->defaultMethod();
    }
}
```

Codelgniter Helper

What is Helper

In Codelgniter there are helpers which are there to help you with different tasks. Every helper file is a collection of functions aiming towards a particular role. Some of the helpers are 'file helpers' which help you to deal with the

TOPS Technologies

file, 'text helpers' to perform various text formatting routines, 'form helpers' to create form element, 'cookie helpers' set and read cookies, 'URL helpers' which assist in creating links, etc.

Helpers are not written in Object Oriented format, instead they are simple, procedural functions, independent of each other.

To use helper files, you need to load it. Once loaded it is globally available to your controller and views. They are located at two places in CodeIgniter. CodeIgniter will look first for a helper in application/helpers folder and if not found there then it will go to system/helpers folder.

Loading a Helper

A helper can be loaded in controller constructor which makes them globally available, or they can also be loaded in specific functions which need them.

It can be loaded with the following code:

```
1. $this->load->helper('file_name');
```

Write your file name here at the place of file_name.

To load URL helper,

```
1. $this->load->helper('url');
```

You can also auto-load a helper if your application needs that helper globally by adding the helper in application/config/autoload.php file.

Loading Multiple Helpers

To load multiple helpers, specify them in an array,

```
$this->load->helper(  
    array('helper1', 'helper2', 'helper3')  
)
```

html Helper Example

We are going to show you an example of html helper by using it in a basic website page. Here, we'll auto-load our helper.

Go to autoload.php file via application/config/autoload.php

```
1. $autoload['helper'] = array('html');
```

In the above file, name your helper, here it is html.

In application/controllers there is file Form.php

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');
```

TOPS Technologies

```
class Form extends CI_Controller {
```

```
    public function index()
    {
        $this->load->view('header');

        $this->load->view('nav');

        $this->load->view('content');

        $this->load->view('footer');

    }
}

?>
```

In application/views there is file header.php

The first line is coded in php tag.

```
<?php echo doctype("html5"); ?>

<html>

<head>

    <title>Basic Site</title>

    <style type="text/css">

        body, html{margin:0; padding:0;}
```

```
body{
    background-color:#eee;
}

h1, h2, h3, h4, p, a, li, ul{
    font-family: arial, sans-serif;
    color: black;
```

TOPS Technologies

```
text-decoration: none;  
}  
  
#nav{  
  
margin: 50px auto 0 auto;  
  
width: 10000px;  
  
background-color: #888;  
  
height: 15px;  
  
padding: 20px;  
}
```

```
#nav a:hover{
```

```
color: red;
```

```
}
```

```
#nav ul{
```

```
list-style: none;
```

```
float: left;
```

```
margin: 0 50px;
```

```
}
```

```
#nav ul li{
```

```
display: inline;
```

```
}
```

```
#content{
```

```
width: 1000px;
```

```
min-height: 100%;
```

TOPS Technologies

```
margin: 0 auto;  
  
padding: 20px;  
  
}  
  
  
  
#footer{  
  
width: 400px;  
  
height: 15px;  
  
margin: 0 auto;  
  
padding: 20px;  
  
}  
  
  
  
#footer p{  
  
color: #777;  
  
}  
  
</style>  
  
</head>
```

The other half coding for file header.php is shown in below snapshot.

In application/views there is file content.php

Here also the heading is written in php tag instead of html.

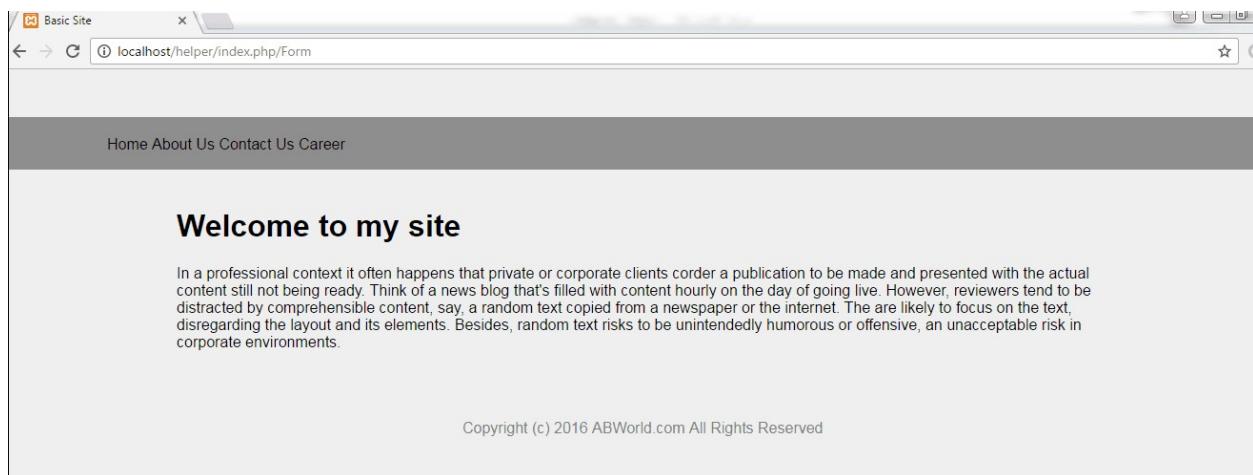
```
<div id="content">  
  
<?php echo heading("Welcome to my site", 1); ?>  
  
<p>In a professional context it often happens that private or corporate clients  
order a publication to be made and presented with the actual content still not being ready.  
  
Think of a news blog that's filled with content hourly on the day of going live. However,  
reviewers tend to be distracted by comprehensible content, say, a random text copied from  
a newspaper or the internet. They are likely to focus on the text, disregarding the layout  
and its elements. Besides, random text risks to be unintendedly humorous or offensive,
```

TOPS Technologies

an unacceptable risk in corporate environments.</p>

</div>

Final output is just like a normal page as shown below with the URL localhost/helper/index.php/Form.



But when we will see its open source (by pressing **ctrl+u**), you will see the following coding which simply shows html code and not php code what we have written above.

```
<!DOCTYPE html>
<html>
<head>
    <title>Basic Site</title>
    <style type="text/css">

        body, html{margin:0; padding:0;}

        body{
            background-color:#eee;
        }
        h1, h2, h3, h4, p, a, li, ul{
            font-family: arial, sans-serif;
            color: black;
        }
<div id="container">
    <div id="nav">
        <ul>
            <li><a href="#">Home</a></li>
            <li><a href="#">About Us</a></li>
            <li><a href="#">Contact Us</a></li>
            <li><a href="#">Career</a></li>
        </ul>
    </div><div id="content">
        <h1>Welcome to my site</h1>
        <p>In a professional context it often happens that pr
        'think of a news blog that's filled with content hourly on the
        newspaper or the internet. The are likely to focus on the tex
        unacceptable risk in corporate environments.</p>
    </div><div id="footer">
```

CodeIgniter Library

What is a Library

CodeIgniter provide a rich set of libraries. It is an essential part of CodeIgniter as it increases the developing speed of an application. It is located in the system/library.

Loading Library

CodeIgniter library can be loaded as following,

1. `$this->load->library('class_name');`

Here, class name should be replaced by the name of the library.

To load multiple libraries, use following code,

1. `$this->load->library(array('email', 'table'));`

Creating Libraries

All the CodeIgniter libraries are placed in system folder. But in case if you want to use any other library in your application you can create it. There is no limitation for libraries. But your created libraries will be stored in the application/libraries folder. This is done to separate your local and global framework resources.

There are three methods to create a library,

- Creating an entire new library
- Extending native libraries
- Replacing native libraries

Creating an entire new library

It should be placed in application/libraries folder.

Naming Conventions

- File name first letter has to be in uppercase letter, like Mylib.php
- Class name first letter should also be in uppercase letter
- File name and class name should be same.

Basic syntax:

Suppose your file name is Mylib.php, then syntax will be as follows,

Loading Mylib.php

It can be loaded with the following line,

1. `$this->load->library('mylib.php')`

TOPS Technologies

Note: You can write library name in any one of the upper or lower case letter.

Accessing mylib.php

Once it is loaded, you can access your class using the lower case letter because object instances are always in lower case.

```
1. $this->mylib->some_method();
```

Extending Native Libraries

You can also add some extended functionality to a native library by adding one or two methods. It will replace the entire library with your version. So it is better to extend the class. Extending and replacing is almost identical with only following exceptions.

- Class declaration must extend the parent class.
- New class name and filename must be prefixed with MY_.

For example, to extend it to native Calendar, create a file MY_Calendar.php in application/libraries folder. Your class will be declared as, class MY_Calendar extends CI_Calendar{}

Replacing Native Libraries

Naming the new file and class name same as the native one will cause the CodeIgniter new one instead of native one. File and class declaration should be exactly same as the native library.

For example, to replace native Calendar library, you'll create a file Calendar.php in application/libraries. And your class will be,

```
Class CI_Calendar {  
}
```

URL Routing

URLs in CodeIgniter are designed to be short and search engine friendly. It should make more sense to the visitors. A user should get an idea about the page content through its URL.

For example, http://abc.com/codeigniter/routing_url

The above URL example makes more sense and gives a brief idea to the users what it is about.

One should always go for the SEO friendly URL.

URL routing is a technique through which it converts SEO friendly URLs into a server code format that understands it easily and drives a request to corresponding handler scripts.

Setting your own Routing Rules

TOPS Technologies

Routing rules are defined in routes.php file at the location application/config. In this file you'll see \$route array, it permits you to specify your own routing criteria. Routes can be classified in two ways, either using Wildcards or Regular Expressions.

Wildcards

There are two types of wildcards:

- :num—series containing only numbers will be matched.
- :any—series containing only characters will be matched.

Using :num

1. \$route['(blog/:num)'] = 'women/social/\$1';

URL containing first segment as 'blog' and second segment as any 'number' will represent to the URL containing 'women' class and 'social' method passing in the match as the variable to the function.

It means when we'll pass URL <http://www.abc.com/blog/1>

Note: Here, you can pass any number instead of 1 in the URL.

It will be directed to <http://www.abc.com/women/social>

Using :any

1. \$route['(blog/:any)'] = 'women/social';

URL containing first segment as 'blog' and second segment as anything will represent to the URL containing 'women' class and 'social' method.

It means when we'll pass URL <http://www.abc.com/blog/xyz>

Note: Here, you can pass anything in the last segment of URL.

It will be directed to <http://www.abc.com/women/social>

Regular Expression

Regular expressions are also used to redirect routes.

1. \$route['blog'(a-zA-Z0-9]+)'] = 'women/social';

You can create your own regular expression to run your URL.

URL Suffix

To add a suffix in your URL, go to file config.php in application/config folder and add suffix you want as shown below. We have added .jsp as the suffix.

1. \$config['url_suffix'] = '.jsp';

For example, if our URL is <http://www.abc.com/women/social>

Then after adding suffix, our URL will become <http://www.abc.com/women/social.jsp>

TOPS Technologies

CodeIgniter Hooks

In CodeIgniter, hooks are events which can be called before and after the execution of a program. It allows executing a script with specific path in the CodeIgniter execution process without modifying the core files. For example, it can be used where you need to check whether a user is logged in or not before the execution of controller. Using hook will save your time in writing code multiple times.

There are two hook files in CodeIgniter. One is application/config/hooks.php folder and other is application /hooks folder.

In other language, if you want to run a code every time after controller constructor is loaded, you can specify that script path in hooks.

Enabling Hooks

To enable Hook, go to application/config/config.php file and set it TRUE as shown below.

```
1. $config['enable_hooks'] = TRUE;
```

Defining a Hook

A hook can be defined in the application/config/hooks.php file. Each hook is defined as an array consisting of the following terms.

```
$hook['pre_controller'] = array(  
    'class' => 'Classname',  
    'function' => 'functionname',  
    'filename' => 'filename.php',  
    'filepath' => 'hooks',  
    'params' => array('element1', 'element2', 'element3')  
)
```

class - Here, you have to mention the name of your class defined in the hooks.php file. If you are using procedural function instead of a class, leave it blank.

function - Mention the function name you are calling.

filename - The file name created in application/hooks folder and containing class and function name mentioned above.

filepath - Here you have to mention the name of the directory which contains your script. Your script must be located inside the application folder. If your script is located in application/hooks folder, then your path will be simply hooks. But if your script is located in application/hooks/office folder, then your path will be hooks/office.

params - It includes the parameters which you want to pass in your script and it's optional.

Multiple calls to the same Hook

TOPS Technologies

You can use array multi-dimensional to use the same hook point with more than one script.

```
$hook['pre_controller'][] = array(  
    'class' => 'Classname1',  
    'function' => 'functionname1',  
    'filename' => 'filename1.php',  
    'filepath' => 'hooks',  
    'params' => array('element1', 'element2', 'element3')  
);
```

```
$hook['pre_controller'][] = array(  
    'class' => 'Classname2',  
    'function' => 'functionname2',  
    'filename' => 'filename2.php',  
    'filepath' => 'hooks',  
    'params' => array('element4', 'element5', 'element6')  
);
```

Bracket [] enables you to have same hook point with multiple scripts. Your execution order will be same as the array defined.

Hook Points

The list of hook points is shown below.

- pre_system
- It is called much before the system execution. Only benchmark and hook class have been loaded at this point.pre_controller
- It is called immediately prior to your controller being called. At this point all the classes, security checks and routing have been done.post_controller_constructo
- It is called immediately after your controller is started, but before any method call.post_controller
- It is called immediately after your controller is completely executed.display_override
- It is used to send the final page at the end of file execution.cache_override
- It enables you to call your own function in the output class.post_system

TOPS Technologies

It is called after the final page is sent to the browser at the end of the system execution.

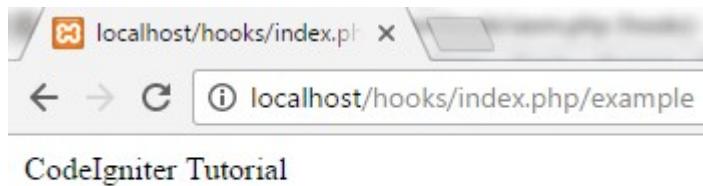
Hook Example

- 1) First of all enable the hook in your CodeIgniter folder as mentioned above.
- 2) Create a Controller file example.php in application/controller folder

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Example extends CI_Controller {  
  
    public function index()  
  
    {  
  
        echo "CodeIgniter Tutorial";  
  
    }  
  
}
```

On running the above program with URL,

<http://localhost/hooks/index.php/example>, following output will appear.



- 3) Create a hooks file exm.php in application/hooks folder.

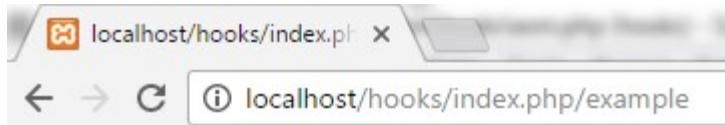
```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Exm extends CI_Controller {  
  
    public function tut()  
  
    {  
  
        echo "Welcome to JavaTpoint. This is ";  
  
    }  
  
}
```

?>

- 4) Now you have to define your hook in the application/config/hooks folder.

```
<?php  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
$hook['pre_controller'] = array(  
  
    'class' => 'Exm',  
  
    'function' => 'tut',  
  
    'filename' => 'exm.php',  
  
    'filepath' => 'hooks',  
  
);  
  
?>
```

- 5) Now again run your program with the same URL and see the result.



Passing Parameters in CodeIgniter

Now we'll see an example to pass parameters from controller to view.

- 1) Download CodeIgniter and name it. We have named it as params.
- 2) Create a file para.php in application/controllers folder.

```
<?php  
if(!defined('BASEPATH')) exit('No direct script access allowed');  
  
class Para extends CI_Controller{  
  
    // declaring variables  
  
    var $name;  
  
    function __construct(){
```

TOPS Technologies

```
parent::__construct();

// passing value

$this->name="CodeIgniter";

}

function tut()

{

$data['name']=$this->name;

// define variable sent to views

$this->load->view('para_view',$data);

}

}

?>
```

3) Create a file para_view.php in application/views folder.

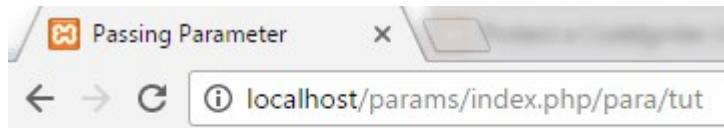
```
<!DOCTYPE html>

<html>
<head>
<title>Passing Parameter</title>
</head>
<body>
<h3>Hello, This is <?php echo $name ; ?> Tutorial.</h3>
</body>
</html>
```

4) Run the program on your browser with URL

<http://localhost/params/index.php/para/tut>

5) Following output will appear on the screen.



Hello, This is CodeIgniter Tutorial.

Codelgniter Driver

Drivers are introduced in Codelgniter 2.0 and onwards.

What are Drivers

These are special type of library that has a parent class and many child classes. These child classes have access to the parent class, but not to their siblings. It enables you to make more elegant classes and more elegant syntax inside your controller.

Drivers are found in system/libraries folder in Codelgniter folder.

Initializing Driver

To initialize a driver, write the following syntax.

1. `$this->load->driver('class_name');`

Here, `class_name` is the driver name you want to invoke.

For example, to invoke a driver class `main_class`, do the following.

1. `$this->load->driver('main_class');`

To invoke its method,

1. `$this->main_class->a_method();`

And then child classes can be called directly through the parent class, without initializing them.

1. `$this->main_class->first_child->a_method();`
- 2.
3. `$this->main_class->second_child->a_method();`

Creating Own Drivers

There are three steps to create a driver in Codelgniter.

1. Making file structure
2. Making driver list
3. Making driver(s)

Making file structure

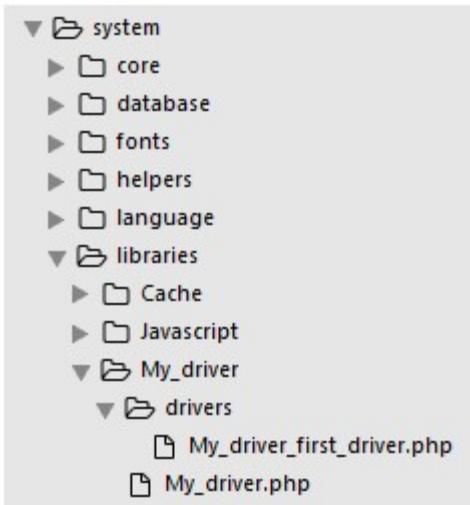
Go to system/libraries folder of Codelgniter and make a new folder `My_driver`. Inside this folder make a file `My_driver.php`.

Now make a new folder inside `My_driver` folder, name it as drivers. Inside this new folder make a file, `My_driver_first_driver.php`.

The following file structure will be shown.

1 /libraries

2. /My_driver
3. My_driver.php
4. /drivers
5. My_driver_first_driver.php



In CodeIgniter, driver library structure is such that subclasses don't extend and hence they don't inherit the properties or methods of the main driver (in this case it is My_driver).

- My_driver - it is a class.
- My_driver.php - Parent driver
- My_driver_first_driver.php - Child driver

Making Driver List

In the file My_driver.php in system/libraries/My_driver folder write the following code,

```
1. <?php
2. defined('BASEPATH') OR exit('No direct script access allowed');
3. class CI_My_driver extends CI_Driver_Library
4. {
5.     function __construct()
6.     {
7.         $this->valid_drivers = array('first_driver');
8.     }
9.     function index()
10.    {
11.        echo "<h1>This is Parent Driver</h1>";
12.    }
13. }
```

TOPS Technologies

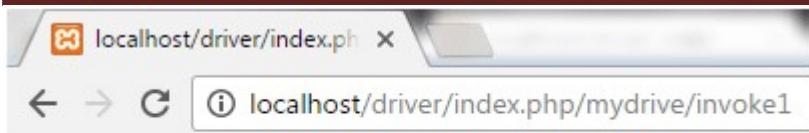
In the file My_driver_first_driver.php in system/libraries/My_driver/drivers write the following code,

```
1. <?php
2. defined('BASEPATH') OR exit('No direct script access allowed');
3. class My_driver_first_driver extends CI_Driver
4. {
5.     function first()
6.     {
7.         echo "<h1>This is first Child Driver</h1>";
8.     }
9. }
10. ?>
```

Make a controller file Mydrive.php in application/controllers with the following code,

```
1. <?php
2. defined('BASEPATH') OR exit('No direct script access allowed');
3. class Mydrive extends CI_Controller
4. {
5.     function __construct()
6.     {
7.         parent::__construct();
8.         $this->load->driver('my_driver');
9.     }
10.    public function invoke1()
11.    {
12.        $this->my_driver->index();
13.    }
14.    public function invoke2()
15.    {
16.        $this->my_driver->first_driver->first();
17.    }
18. }?>
19.
```

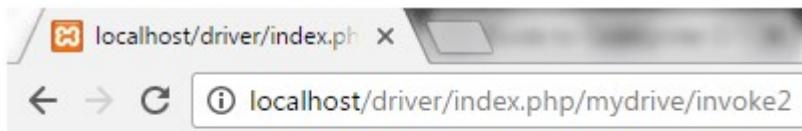
On your browser run the URL, <http://localhost/driver/index.php/mydrive/invoke1>



This is Parent Driver

Look at the above snapshot, the parent driver class is invoked with the function invoke1.

Now run the URL, <http://localhost/driver/index.php/mydrive/invite2>



This is first Child Driver

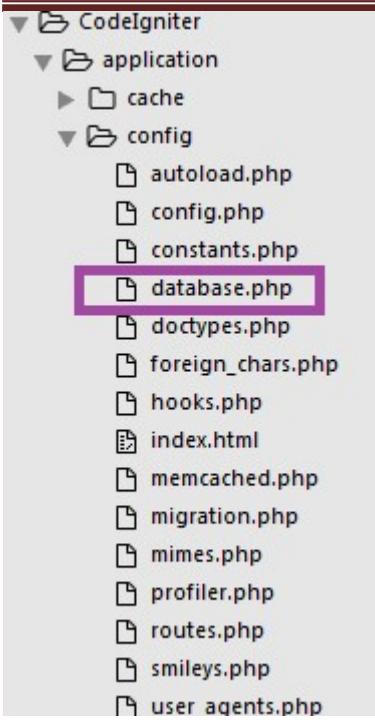
Look at the above snapshot, the child driver class is invoked with the function invoke2.

Codelgniter Database

Database Configuration

In Codelgniter, go to application/config/databse.php for database configuration file.

TOPS Technologies



In database.php file, fill the entries to connect CodeIgniter folder to your database.

```
$db['default'] = array(  
    'dsn'      => '',  
    'hostname' => 'localhost',  
    'username' => 'root',  
    'password' => '',  
    'database' => 'code1|',  
    'dbdriver'  => 'mysqli',  
    'dbprefix'  => '',  
    'pconnect'  => FALSE,  
    'db_debug'  => (ENVIRONMENT !== 'production'),  
    'cache_on'   => FALSE,  
    'cachedir'  => '',  
    'char_set'   => 'utf8',  
    'dbcollat'  => 'utf8_general_ci',  
    'swap_pre'   => '',  
    'encrypt'   => FALSE,  
    'compress'  => FALSE,  
    'stricton'  => FALSE,  
    'failover'   => array(),  
    'save_queries' => TRUE  
);
```

The config settings are stored in a multi-dimensional array as shown above. Fill out your connection details to connect to database.

TOPS Technologies

You can also specify failover in the situation where main connection cannot be established. This can be specified by setting failover as shown below. You can write as many failovers as you want.

```
$db['default']['failover'] = array(
    array(
        'hostname' => 'localhost1',
        'username' => 'root',
        'password' => '',
        'database' => '',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => FALSE,
        'db_debug' => (ENVIRONMENT !== 'production'),
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'save_queries' => TRUE
    )

    array(
        'hostname' => 'localhost2',
        'username' => 'root',
        'password' => '',
        'database' => '',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => FALSE,
        'db_debug' => (ENVIRONMENT !== 'production'),
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'save_queries' => TRUE
    )
);
```

Automatically connecting Database

The auto connect feature will load your database class with every page load.

To add auto connect go to application/config/autoload.php and add the word database to library array.

Manually connecting Database

If you need to connect database only in some pages of your project, you can use below code to add the database connectivity in any page, or add it to your class constructor which will make the database globally available for that class.

1. \$this->load->database();

TOPS Technologies

Connecting multiple Database

If you need to connect more than one database simultaneously, do the following.

1. \$db1 = \$this->load->database(?group_one?, TRUE);
2. \$db1 = \$this->load->database(?group_two?, TRUE);

Here, group_one and group_two are the group names which will be replaced by your group name.

CodeIgniter Database INSERT record

In this example, we will INSERT different values in database showing meaning of Indian names through CodeIgniter.

In **application/controller**, file **baby_form.php** is created.

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Baby_form extends CI_Controller {  
  
    public function index()  
    {  
        $this->load->view("baby_form_add");  
    }  
  
    function savingdata()  
    {  
        //this array is used to get fetch data from the view page.  
  
        $data = array(  
            'name' => $this->input->post('name'),  
            'meaning' => $this->input->post('meaning'),  
            'gender' => $this->input->post('gender'),  
            'religion' => $this->input->post('religion')  
    }  
}
```

```
 );  
  
 //insert data into database table.  
  
 $this->db->insert('baby',$data);  
  
  
 redirect("baby_form/index");  
  
 }  
  
 }  
  
 ?>
```

Look at the above snapshot, our table name is '**baby**'.

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
 <title>Baby Form Add</title>  
  
</head>  
  
<body>  
  
 <form method="post" action="<?php echo site_url('baby_form/savingdata'); ?>">  
  
 <table>  
  
 <tr>  
  
 <td>Name:</td>  
  
 <td></td>  
  
 <td><input type="text" name="name"></td>  
  
 </tr>  
  
 <tr>  
  
 <td>Meaning:</td>  
  
 <td></td>  
  
 <td><input type="text" name="meaning"></td>
```

TOPS Technologies

```
</tr>

<tr>

    <td>Gender:</td>

    <td><input type="text" name="gender"></td>

</tr>

<tr>

    <td>Religion:</td>

    <td><input type="text" name="religion"></td>

</tr><br><br>

<tr>

    <input type="submit" name="submit" value="Save">

</tr>

</table>

</form>

</body>

</html>
```

This is our view page which is being loaded in the controller's page.

To run it on your browser, pass URL

http://localhost/insert/index.php/Baby_form/

TOPS Technologies

Baby Form Add

localhost/insert/index.php/baby_form/index

Save

Name: :

Meaning: :

Gender: :

Religion: :

By inserting various names in the above form we have created our table as shown below.

Server: 127.0.0.1 » Database: codeigniter » Table: baby

Browse Structure SQL Search Insert Export

Sort by key: None

+ Options

			id	name	meaning	gender	religion	
<input type="checkbox"/>				1	Vishal	Large	Male	Hindu
<input type="checkbox"/>				2	Megha	Rain	Female	Hindu
<input type="checkbox"/>				3	Rajiv	Blue Lotus	Male	Hindu
<input type="checkbox"/>				4	Heer	Diamond	Male	Muslim
<input type="checkbox"/>				5	Palak	Eyelash	Female	Hindu
<input type="checkbox"/>				6	Palash	A flower tree	Male	Hindu
<input type="checkbox"/>				7	Jasmin	A flower	Female	Muslim
<input type="checkbox"/>				8	Akanksha	Wish	Female	Hindu
<input type="checkbox"/>				9	Mahira	Full of Energy	Female	Muslim
<input type="checkbox"/>				10	Gaurav	Pride	Male	Hindu

Check all With selected:

CodeIgniter SELECT Database record

To fetch all data from database, one more page in Model folder of CodeIgniter will be created. There will be some changes in controller's and view's files also.

Controller file (Baby_form.php) is shown below.

```
<?php  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Baby_form extends CI_Controller {  
  
    public function __construct()  
    {  
        parent::__construct();  
  
        //calling model  
        $this->load->model("Babymodel", "a");  
    }  
  
    public function index()  
    {  
        $this->load->view("baby_form_select");  
    }  
  
    function savingdata()  
    {  
        //this array is used to get fetch data from the view page.  
  
        $data = array(  
    }
```

TOPS Technologies

```
'name'    => $this->input->post('name'),  
  
'meaning' => $this->input->post('meaning'),  
  
'gender'   => $this->input->post('gender'),  
  
'religion' => $this->input->post('religion')  
  
);  
  
//insert data into database table.  
  
$this->db->insert('baby',$data);  
  
  
redirect("baby_form/index");  
  
}  
  
}  
  
?>
```

We have added a constructor to load the model page. Highlighted code is added to fetch the inserted record. And our view page is now baby_form_select.php

View file (baby_form_select.php) is shown below.

```
<!DOCTYPE html>  
  
<html>  
  
<head>  
  
<title>Baby Form Add</title>  
  
</head>  
  
<body>  
  
<form method="post" action="php echo site_url('baby_form/savingdata'); ?&gt;"&gt;<br/  
<table>  
  
<tr>
```

TOPS Technologies

```
<td>Name:</td>

<td>:</td>

<td><input type="text" name="name"></td>

</tr>

<tr>

<td>Meaning:</td>

<td>:</td>

<td><input type="text" name="meaning"></td>

</tr>

<tr>

<td>Gender:</td>

<td>:</td>

<td><input type="text" name="gender"></td>

</tr>

<tr>

<td>Religion:</td>

<td>:</td>

<td><input type="text" name="religion"></td>

</tr><br><br>

<tr>

<input type="submit" name="submit" value="Save">

</tr>

</table>

</form>

<table border="1">

<thead>
```

TOPS Technologies

```
<th>ID</th>

<th>NAME</th>

<th>MEANING</th>

<th>GENDER</th>

<th>RELIGION</th>

<th>ACTION</th>

</thead>

<tbody>

<?php

foreach($this->a->fetchtable() as $row)

{

    //name has to be same as in the database.

    echo "<tr>

        <td>$row->id</td>

        <td>$row->name</td>

        <td>$row->meaning</td>

        <td>$row->gender</td>

        <td>$row->religion</td>

    </tr>";

}

?>

</tbody>

</table>

</body>

</html>
```

TOPS Technologies

Code in baby_form_select.php file is same as baby_form_add.php. Above codes are added to fetch the record.

Here we have fetched the record in a table with the help of **foreach** loop. Function **fetchtable()** is created to fetch the record.

Model file (**babymodel.php**) is shown below.

```
<?php  
  
defined('BASEPATH') OR exit('No direct script access allowed');  
  
class Babymodel extends CI_Model {  
  
    function __construct()  
    {  
        //call model constructor  
        parent::__construct();  
    }  
  
    function fetchtable()  
    {  
        $query = $this->db->get('baby');  
        return $query->result();  
    }  
?  
}
```

In URL, type http://localhost/CodeIgniter/index.php/Baby_form

The screenshot shows a web browser window titled "Baby Form Add". The address bar displays the URL "localhost/select/index.php/baby_form/index". The page contains a form with four input fields: "Name:" (with placeholder "Name:"), "Meaning:" (with placeholder "Meaning:"), "Gender:" (with placeholder "Gender:"), and "Religion:" (with placeholder "Religion:"). Below the form is a "Save" button. To the right of the form is a table with the following data:

ID	NAME	MEANING	GENDER	RELIGION	ACTION
1	Vishal	Large	Male	Hindu	
2	Megha	Rain	Female	Hindu	
3	Rajiv	Blue Lotus	Male	Hindu	
4	Heer	Diamond	Male	Muslim	
5	Palak	Eyelash	Female	Hindu	
6	Palash	A flower tree	Male	Hindu	
7	Jasmin	A flower	Female	Muslim	
8	Akanksha	Wish	Female	Hindu	
9	Mahira	Full of Energy	Female	Muslim	
10	Gaurav	Pride	Male	Hindu	

Look at the above snapshot, all data has been fetched from the table 'baby'.

Login Form in CodeIgniter (without MySQL)

Here, we'll make a simple login page with the help of session.

Go to file **autoload.php** in **application/config/autoload.php**

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

/*
-----+
| Auto-load Drivers
|
| These classes are located in system/libraries/ or in your
| application/libraries/ directory, but are also placed inside their
| own subdirectory and they extend the CI_Driver_Library class. They
| offer multiple interchangeable driver options.
|
| Prototype:
|
| $autoload['drivers'] = array('cache');
|
| You can also supply an alternative property name to be assigned in
| the controller:
|
| $autoload['drivers'] = array('cache' => 'cch');
|
*/
$autoload['drivers'] = array();

/*
-----+
| Auto-load Helper Files
|
| Prototype:
|
| $autoload['helper'] = array('url', 'file');
*/
$autoload['helper'] = array('url', 'html', 'file');
```

Set session in library and helper.

Create controller page Login.php in application/controllers folder.

```
<?php

defined('BASEPATH') OR exit('No direct script access allowed');

class Login extends CI_Controller {
```

```
public function index()
{
    $this->load->view('login_view');

}

public function process()
{
    $user = $this->input->post('user');

    $pass = $this->input->post('pass');

    if ($user=='juhi' && $pass=='123')

    {
        //declaring session

        $this->session->set_userdata(array('user'=>$user));

        $this->load->view('welcome_view');

    }

    else{

        $data['error'] = 'Your Account is Invalid';

        $this->load->view('login_view', $data);

    }

}

public function logout()
{
    //removing session

    $this->session->unset_userdata('user');

    redirect("Login");

}
```

}

?>

Look at the above snapshot, we have created a session for a single user with Username juhi and Password 123. For a valid login and logout we will use this username and password.

Create view page login_view.php in application/views folder.

```
<!DOCTYPE html>

<html>

<head>

    <title>Login Page</title>

</head>

<body>

    <?php echo isset($error) ? $error : ''; ?>

    <form method="post" action="<?php echo site_url('Login/process'); ?>">

        <table cellpadding="2" cellspacing="2">

            <tr>

                <td><th>Username:</th></td>

                <td><input type="text" name="user"></td>

            </tr>

            <tr>

                <td><th>Password:</th></td>

                <td><input type="password" name="pass"></td>

            </tr>

            <tr>

                <td> </td>

                <td><input type="submit" value="Login"></td>

            </tr>

        </table>

    </form>

</body>
```

```
</tr>

</table>

</form>

</body>

</html>
```

Create view page welcome_view.php in application/views folder to show the successful login message.

```
<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

Welcome <?php echo $this->session->userdata('user'); ?>

<br>

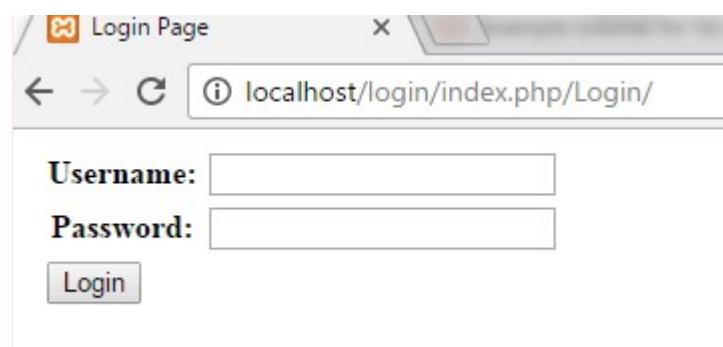
<?php echo anchor('Login/logout', 'Logout'); ?>

</body>

</html>
```

Output

Type URL localhost/login/index.php/Login



TOPS Technologies

Now on entering wrong information we'll see unsuccessful message which we have set in login_view page in else part.

Login Page

localhost/login/index.php/Login/

Username: adam

Password:

Login

Login Page

::1/login/index.php/Login/process

Your Account is Invalid

Username: [empty]

Password: [empty]

Login

Now on entering right information, we'll see welcome_view.php message.

Login Page

localhost/login/index.php/Login/

Username: juhi

Password: ...

Login

[::1]/login/index.php/Log

[::1]/login/index.php/Login/process

Welcome juhi

[Logout](#)

Click on Logout, we'll be directed to Login page.

Login page (with database)

In earlier example, we learnt a simple login page with one single username and session.

Now we'll make it with more than one users using database. A sign in and login page will be there for users.

TOPS Technologies

Following pages are made in this example

In application/controllers

- Main.php

In application/views

- login_view.php
- invalid.php
- data.php
- signin.php

In application/models

- login_model.php

In the first page, you'll have the option for Login and Sign In.

On Login, if user has entered correct credentials matching to database then he will be directed to data.php page. But if he has entered wrong information then incorrect username/password message will appear.

We have named our CodeIgniter folder as login_db. And our table name is signup.

The screenshot shows the phpMyAdmin interface for a database named 'codeigniter' on a server at 127.0.0.1. The current table is 'signup'. The interface includes a toolbar with 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', 'Import', and 'Print' buttons. A success message at the top states 'Showing rows 0 - 1 (2 total, Query took 0.0000 seconds.)'. Below it is a SQL query: 'SELECT * FROM `signup`'. The results section displays two rows of data:

		id	username	password
<input type="checkbox"/>		Edit	1	admin
<input type="checkbox"/>		Edit	2	jtp

We need to do some basic settings in our login_db CodeIgniter folder.

Go to autoload.php file, and do the following settings.

```
55  /
56  $autoload['libraries'] = array('database', 'session');
57
58  /*
59  | -----
60  | Auto-load Drivers
61  | -----
62  |
63  | These classes are located in system/libraries/ or in your
64  | application/libraries/ directory, but are also placed inside their
65  | own subdirectory and they extend the CI_Driver_Library class. They
66  | offer multiple interchangeable driver options.
67
68  | Prototype:
69
70  |     $autoload['drivers'] = array('cache');
71
72  | You can also supply an alternative property name to be assigned in
73  | the controller:
74
75  |     $autoload['drivers'] = array('cache' => 'cch');
76
77  */
78
79  $autoload['drivers'] = array();
80
81  /*
82  | -----
83  | Auto-load Helper Files
84  | -----
85  |
86  | Prototype:
87
88  |     $autoload['helper'] = array('url', 'file');
89
90  */
91
92  $autoload['helper'] = array('form', 'url', 'file');
```

In the above snapshot, we have loaded the libraries and helper.

In database.php file, fill your username and database name. Our database name is codeigniter.

TOPS Technologies

```
73 $active_group = 'default';
74 $query_builder = TRUE;
75
76 $db['default'] = array(
77     'dsn'      => '',
78     'hostname' => 'localhost',
79     'username' => 'root',
80     'password' => '',
81     'database' => 'codeigniter',
82     'dbdriver' => 'mysqli',
83     'dbprefix' => '',
84     'pconnect' => FALSE,
85     'db_debug' => (ENVIRONMENT !== 'production'),
86     'cache_on' => FALSE,
87     'cachedir' => '',
88     'char_set' => 'utf8',
89     'dbcollat' => 'utf8_general_ci',
90     'swap_pre' => '',
91     'encrypt' => FALSE,
92     'compress' => FALSE,
93     'stricton' => FALSE,
94     'failover' => array(),
95     'save_queries' => TRUE
96 );
97
```

Now, we'll start the example.

We have made file Main.php in application/controllers folder,

```
<?php

defined('BASEPATH') OR exit('No direct script access allowed');

class Main extends CI_Controller {

    public function index()
    {
        $this->login();
    }

    public function login()
```

```
{  
    $this->load->view('login_view');  
  
}  
  
  
public function signin()  
{  
    $this->load->view('signin');  
  
}  
  
  
public function data()  
{  
    if ($this->session->userdata('currently_logged_in'))  
    {  
        $this->load->view('data');  
    } else {  
        redirect('Main/invalid');  
    }  
}  
  
  
public function invalid()  
{  
    $this->load->view('invalid');  
}  
  
  
public function login_action()  
{
```

```
$this->load->helper('security');

$this->load->library('form_validation');

$this->form_validation->set_rules('username', 'Username:', 'required|trim|xss_clean|callback_validation');

$this->form_validation->set_rules('password', 'Password:', 'required|trim');

if ($this->form_validation->run())

{

    $data = array

        'username' => $this->input->post('username'),

        'currently_logged_in' => 1

    );

    $this->session->set_userdata($data);

    redirect('Main/data');

}

else {

    $this->load->view('login_view');

}

}

public function signin_validation()

{

    $this->load->library('form_validation');

    $this->form_validation->set_rules('username', 'Username', 'trim|xss_clean|is_unique[signup.username]');

}
```

TOPS Technologies

```
$this->form_validation->set_rules('password', 'Password', 'required|trim');

$this->form_validation->set_rules('cpassword', 'Confirm Password', 'required|trim|matches[password]');

$this->form_validation->set_message('is_unique', 'username already exists');

if ($this->form_validation->run())

{

    echo "Welcome, you are logged in.";

}

else {

    $this->load->view('signin');

}

public function validation()

{

    $this->load->model('login_model');

    if ($this->login_model->log_in_correctly())

    {

        return true;

    } else {

        $this->form_validation->set_message('validation', 'Incorrect username/password.');

    }

}
```

```
    return false;  
}  
  
}  
  
  
public function logout()  
{  
    $this->session->sess_destroy();  
    redirect('Main/login');  
}  
  
}  
  
?>
```

In application/views folder, login_view.php file is made.

```
<!DOCTYPE html>  
  
<html lang="en">  
  
<head>  
  
    <meta charset="utf-8">  
  
    <title>Login Page</title>  
  
</head>  
  
<body>  
  
    <h1>Login</h1>  
  
  
<?php  
  
echo form_open('Main/login_action');
```

```
echo validation_errors();

echo "<p>Username: </p>";
echo form_input('username', $this->input->post('username'));

echo "</p>";

echo "<p>Password: </p>";
echo form_password('password');

echo "</p>";

echo "</p>";

echo form_submit('login_submit', 'Login');

echo "</p>";

echo form_close();

?>

<a href='<?php echo base_url()."index.php/Main/signin"; ?>'>Sign In</a>
```

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

In application/views folder, data.php file is made.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
```

```
</head>

<body>

<h1>Welcome, You are successfully logged in.</h1>

<?php

echo "<pre>";

echo print_r($this->session->all_userdata());

echo "</pre>";

?>

<a href='<?php echo base_url()."index.php/Main/logout"; ?>'>Logout</a>

</body>

</html>
```

In application/views folder, signin.php file is made.

```
<!DOCTYPE html>

<html>

<head>

<title>Sign Up Page</title>

</head>

<body>

<h1>Sign In</h1>

<?php

echo form_open('Main/signin_validation');
```

```
echo validation_errors();

echo "<p>Username:</p>";
echo form_input('email');

echo "</p>";

echo "<p>Password:</p>";
echo form_password('password');

echo "</p>";

echo "<p>Confirm Password:</p>";
echo form_password('cpassword');

echo "</p>";

echo "<p>";
echo form_submit('signin_submit', 'Sign In');
echo "</p>";

echo form_close();

?>

</body>
</html>
```

In application/views folder, invalid.php file is made.

```
<!DOCTYPE html>

<html>
<head>
    <title>Invalid Page</title>
</head>
<body>
    <h1>Sorry, You don't have access to this page.</h1>

    <a href='<?php echo base_url()."Main/login"; ?>'>Login Again</a>

</body>
</html>
```

In application/models folder, login_model.php file is made.

```
<?php

class Login_model extends CI_Model {

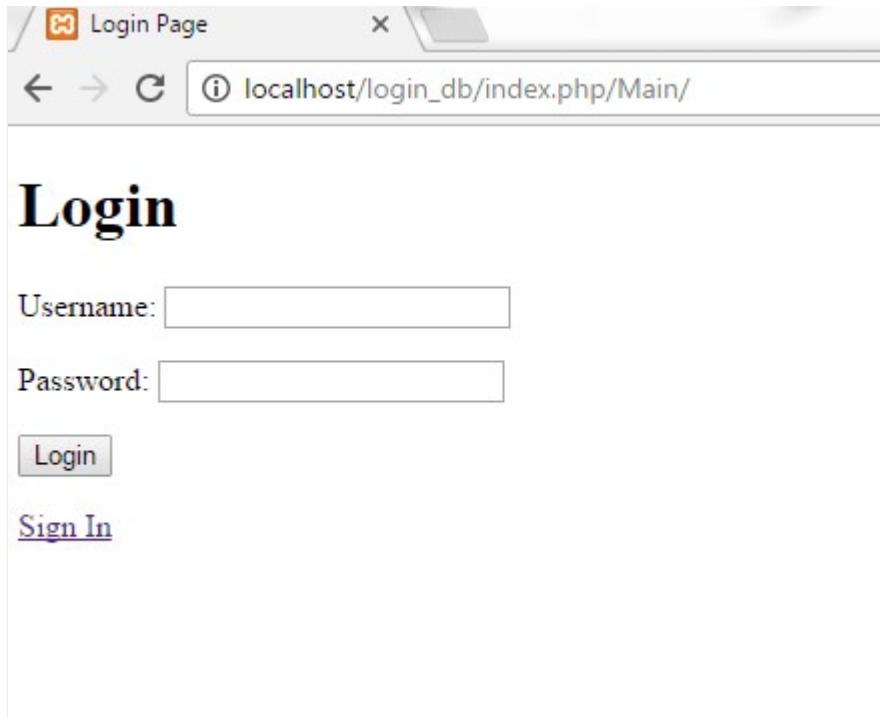
    public function log_in_correctly() {
        $this->db->where('username', $this->input->post('username'));
        $this->db->where('password', $this->input->post('password'));
        $query = $this->db->get('signup');

        if ($query->num_rows() == 1)
        {
            return true;
        }
    }
}
```

```
    } else {  
        return false;  
    }  
  
}  
  
?  
?
```

On entering the URL, http://localhost/login_db/index.php/Main/

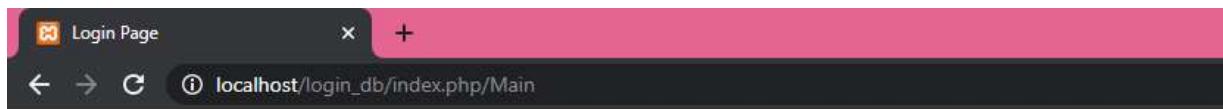
Following page will appear.



The screenshot shows a web browser window with the title bar "Login Page". The address bar displays the URL "localhost/login_db/index.php/Main/". The main content area of the browser shows a login form with the following elements:

- Login Title:** The word "Login" is displayed prominently at the top of the form.
- Username Field:** A text input field labeled "Username:" followed by an empty input box.
- Password Field:** A text input field labeled "Password:" followed by an empty input box.
- Login Button:** A blue rectangular button labeled "Login" with white text.
- Sign In Link:** A link labeled "Sign In" in blue text.

Enter the information to Login.



Login

Username:

Password:

[Sign In](#)

After entering information, click on Login button. We have entered the wrong information and hence it will show the error message as shown below.

as shown below.



Login

Incorrect username/password.

Username:

Password:

[Sign In](#)

On entering the information from database, we'll be directed to data.php page.



Login

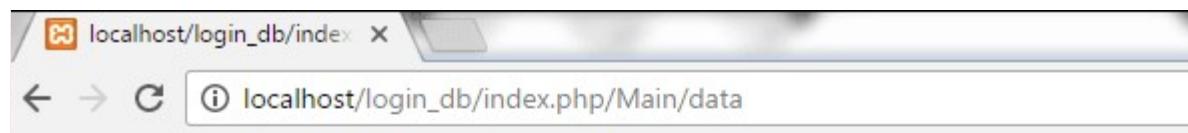
Incorrect username/password.

Username:

Password:

[Sign In](#)

Click on Login button.



Welcome, You are successfully logged in.

```
Array
(
    [__ci_last_regenerate] => 1478502544
    [username] => admin
    [currently_logged_in] => 1
)
1
```

[Logout](#)

Clicking on Logout button, you'll be directed to the Main page.

On clicking Sign In, following page will appear.

A screenshot of a web browser window titled "Sign Up Page". The address bar shows the URL "localhost/login_db/index.php/Main/signin". The main content area displays a "Sign In" form with three input fields: "Username", "Password", and "Confirm Password", followed by a "Sign In" button.

Username:

Password:

Confirm Password:

Sign In

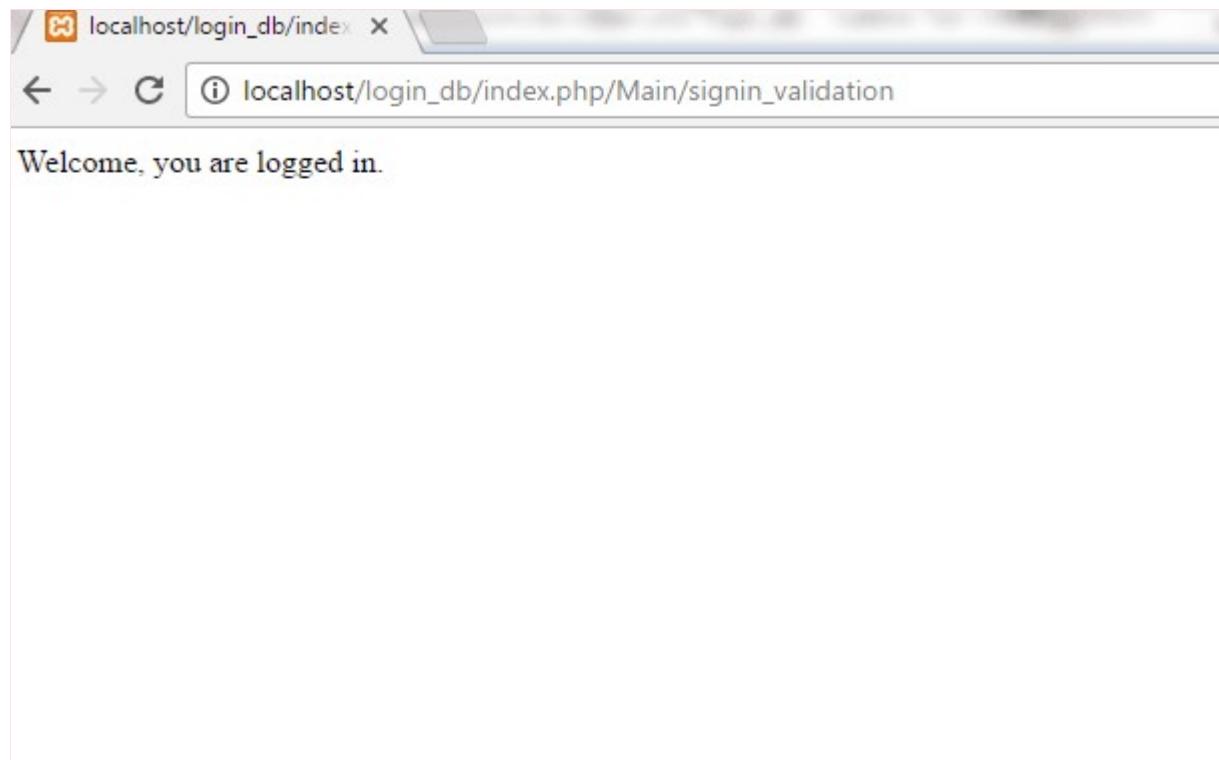
A screenshot of a web browser window titled "Sign Up Page". The address bar shows the URL "localhost/login_db/index.php/Main/signin". The main content area displays a "Sign In" form with three input fields: "Username" containing "john", "Password" containing "....", and "Confirm Password" containing "....", followed by a "Sign In" button.

Username:

Password:

Confirm Password:

Sign In



Login

Incorrect username/password.

Username:

Password:

[Sign In](#)

On entering the information from database, we'll be directed to data.php page.



Login

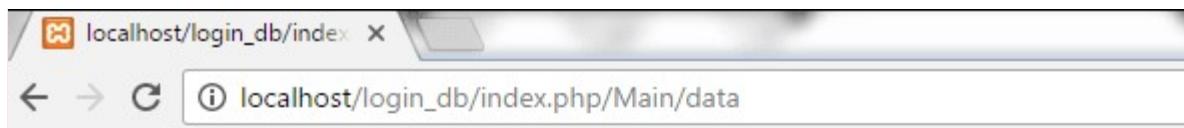
Incorrect username/password.

Username:

Password:

[Sign In](#)

Click on Login button.



Welcome, You are successfully logged in.

```
Array
(
    [__ci_last_regenerate] => 1478502544
    [username] => admin
    [currently_logged_in] => 1
)
1
```

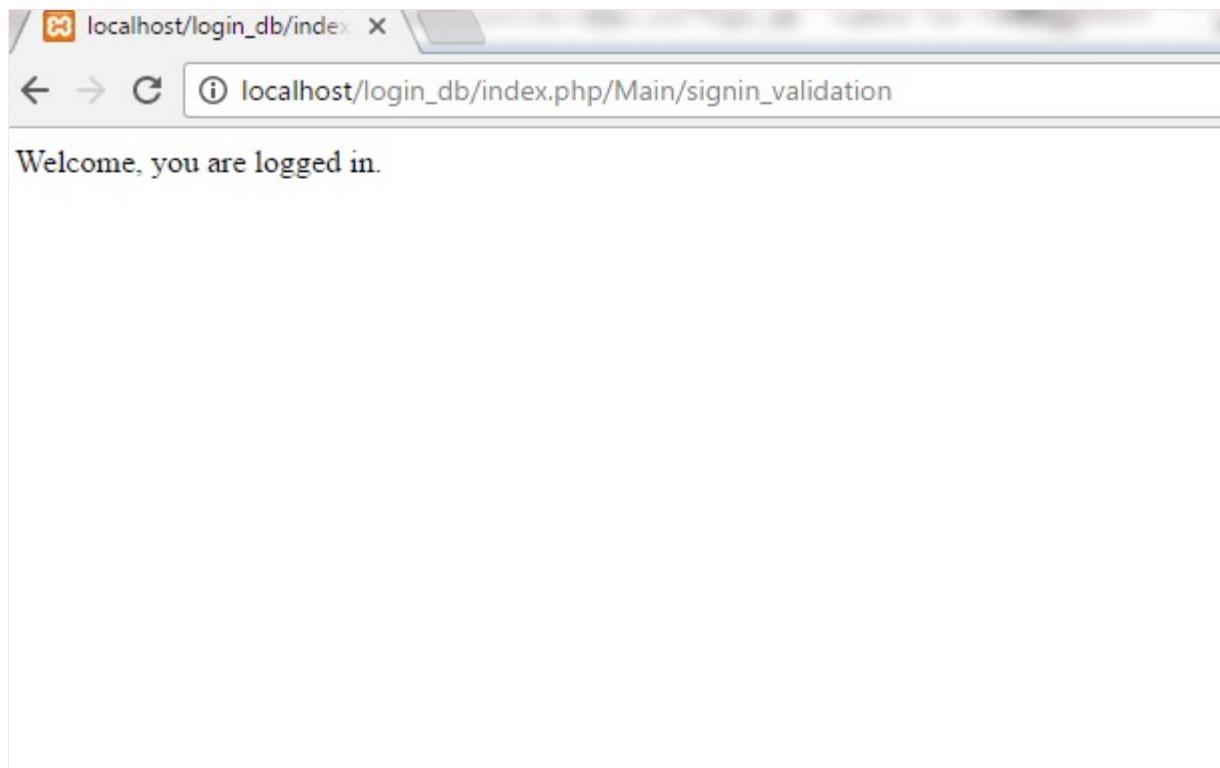
[Logout](#)

Clicking on Logout button, you'll be directed to the Main page.

On clicking Sign In, following page will appear.

A screenshot of a web browser window titled "Sign Up Page". The address bar shows the URL "localhost/login_db/index.php/Main/signin". The page content is titled "Sign In". It contains three input fields: "Username:", "Password:", and "Confirm Password:". Below the fields is a "Sign In" button.

A screenshot of a web browser window titled "Sign Up Page". The address bar shows the URL "localhost/login_db/index.php/Main/signin". The page content is titled "Sign In". The "Username:" field contains "john". The "Password:" field contains "....". The "Confirm Password:" field also contains "....". Below the fields is a "Sign In" button.



CodeIgniter Security

CodeIgniter contains security class methods which will help to create a secure application and process input data. The methods are given below.

- XSS Filtering
- CSRF (Cross-site Request Forgery)
- Class Reference

XSS Filtering

XSS stands for Cross-site Scripting. It is used to disable JavaScript or other types of code that try to hijack cookies and perform other type of malicious acts. When it encounters anything harmful, it is rendered safe by converting the data to character entities.

XSS filtering uses `xss_clean()` method to filter data.

```
1. $data = $this->security->xss_clean($data);
```

There is an optional second parameter, `is_image`, which is used to test images for XSS attacks. When this parameter is set to TRUE, it doesn't return an altered string, instead it returns TRUE if image is safe and FALSE if it contains malicious information.

```
if ($this->security->xss_clean($file, TRUE) === FALSE)
{
    //file failed in XSS test
```

}

CSRF (Cross-site Request Forgery)

To enable CSRF do the following settings in **application/config/config.php** file.

1. \$config['csrf_protection'] = TRUE;

If you are using form helper, then a hidden csrf field will be automatically inserted in your **form_open()** field.

Otherwise, you can manually add it using,

get_csrf_token_name() (it returns name of csrf) and

get_csrf_hash() (it returns value of csrf).

Generated tokens may be kept same throughout the life of CSRF cookie or may be regenerated on every submission.

The default generation of token provides a better security but it also have usability concerns as other tokens like multiple tabs/windows, asynchronous actions, etc become invalid. Regeneration behavior can be set in

application/config/config.php file as shown below.

1. \$config['csrf_regeneration'] = TRUE;

Class Reference

1. Class CI_Security

1. xss_clean (\$str [, \$is_image = FALSE])

Parameters - \$str (mixed) ? input string or an array of strings

Returns - XSS-clean data

Return-type - mixed

From input data it removes XSS exploits and returns the clean string.

1. Sanitize_filename (\$str [, \$relative_path = FALSE])

Parameters - \$str (string) ? File name/path

\$relative_path (bool) ? Whether to preserve any directories in the file path

Returns - Sanitized file name/path

Return-type - string

It prevents directory traversal and other security threats by sanitizing filenames. It is mainly useful for files which were supplied via user input.

1. Entity_decode ((\$str [, \$charset = NULL])

Parameters - \$str (string) ? Input string

\$charset (string) ? Character set of the input string

Returns - Entity-decoded string

Return-type - string

It tries to detect HTML entities that don't end in a semicolon because some browser allows that.

\$charset parameter is left empty, then your configure value in **\$config['charset']** will be used.

1. Get_random_bytes (\$length)

Parameters - \$length (int) ? Output length

Returns - A binary system of random bytes or FALSE on failure.

Return-type - string

It is used for generating CSRF and XSS tokens.

Preventing, Enabling from CSRF

In this tutorial we'll learn to protect CodeIgniter application from the cross-site request forgery attack. It is one of the most common vulnerabilities in web application. CSRF protection is quite easy in CodeIgniter due to its built-in feature.

What is CSRF attack

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including victim's session cookie and other authentication information, to a web application.

For example, suppose you have a site with a form. An attacker could create a bogus form on his site. This form could contain hidden inputs and malicious data. This form is not actually sent to the attacker's site, in fact it comes to your site. Thinking that the form is genuine, your site will process it.

Now just suppose that the attacker's form points towards the deletion form in your site. If a user is logged in and redirected to the attacker's site and when performs search, his account will be deleted without knowing him. That is the CSRF attack.

Token Method

To protect from CSRF we need to connect both the HTTP requests, form request and form submission. There are several ways to do this, but in CodeIgniter hidden field is used which is called CSRF token. The CSRF token is a random value that changes with every HTTP request sent.

When CSRF token is inserted in the website form, it also gets saved in the user's session. When the form is submitted, the website matches both the token, the submitted one and one saved in the session. If they match, request is made legitimate. The token value changes each time the page is loaded, which makes it tough for the hackers to guess the current token.

Enabling CSRF Protection

To enable CSRF make the following statement TRUE from FALSE in **application/config/config.php** file.

1. `$config['csrf_protection'] = TRUE;`

Token Generation

With each request a new CSRF token is generated. When object is created, name and value of the token are set.

TOPS Technologies

1. \$this->csrf_cookie_name = \$this->csrf_token_name;
2. \$this->_csrf_set_hash();

The function for it is,

```
function _csrf_set_hash()
{
    if ($this->csrf_hash == "")
    {
        if ( isset($_COOKIE[$this->csrf_cookie_name] ) AND
            $_COOKIE[$this->csrf_cookie_name] != " )
        {
            $this->csrf_hash = $_COOKIE[$this->csrf_cookie_name];
        } else {
            $this->csrf_hash = md5(uniqid(rand(), TRUE));
        }
    }

    return $this->csrf_hash;
}
```

First, function checks the cookie's existence. If it exists, its current value is used because when security class is instantiated multiple times, each request would overwrite the previous one.

Function also creates a globally available hash value and save it for further processing. The token's value is generated. Now it has to be inserted into every form of the website with the help of function **form_open()**.

The method csrf_verify() is called each time a form is sent. This method does two things. If no POST data is received, the CSRF cookie is set. And if POST data is received, it checks the submitted value corresponds to the CSRF token value in session. In the second case, CSRF token value is discarded and generated again for the next request. This request is legitimate and whole process starts again.