

Words of Concern

Point to be As Allan Bloom has said "Education is the movement from darkness to light". Through this handbook, I have tried to illuminate what might otherwise appear as black boxes to some. In doing so, I have used references from several other authors to synthesize or simplify or elaborate information. This is not possible without omitting details that I deem trivial while dilating the data that I consider relevant to topic. Every effort has been made to avoid errors. In spite of this, some errors might have crept in. Any errors or discrepancies noted maybe brought to my notice which I shall rectify in my next revision.

This handbook is solely for educational purpose and is not for sale. This handbook shall not be reproduced or distributed or used for commercial purposes in any form or by any means.

Thanks,
Deeksha.V.
Interface College of Computer Applications (ICCA)
Davangere



Bibliography

1. PHP & MySQL for Dynamic web sites- Fourth Edition By Larry Ullman.
2. Learning PHP,MySQL and Javascript By Robin Nixon -O "REILLY publications.
3. Programming PHP By Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre.
4. SAMS Teach Yourself PHP in 24 hours, Author: Matt Zandstra, Sams Publishing.

Interface College of Computer Applications (ICCA)

Syllabus:**Course Code: DSC17****Course Title: PHP and MySQL****Credits - 04****Sem - VI****Hours- 52****Formative Assessment Marks: 40****Summative Assessment Marks: 60****Duration of SEA/Exam: 02 hrs.**

Course Outcomes (COs): After the successful completion of the course, the student will be able to:

- 1.Design dynamic and interactive web pages and websites.
- 2.Run PHP scripts on the server and retrieve results.
- 3.Handle databases like MySQL using PHP in websites.

Contents**52 Hrs****Unit I****10 Hrs**

Introduction: -Introduction to PHP, History and Features of PHP, Installation & Configuration of PHP, Embedding PHP code in your web pages, Understanding PHP,HTML and white space, Writing comments in PHP, Sending data to the web browser, Data types in PHP, Keywords in PHP, using variables, constants in PHP, expressions in PHP, operators in PHP.

Unit II**12 Hrs**

Programming with PHP: - Conditional statements: If, if-else, switch, The? Operator, Looping statements: While Loop, do-while Loop, for Loop, Arrays in PHP: Introduction-What is array?, Creating Arrays, Accessing array elements, Types of arrays: Indexed v/s Associative arrays, Multidimensional arrays, Creating array, Accessing array, Manipulating Arrays, Displaying array, using array functions, Including and requiring files-use of Include() and Require()),Implicit and Explicit casting in PHP.

Unit III**10 Hrs**

Using Functions, class- objects, Forms in PHP: - Functions in PHP, Function definition, Creating and invoking user-defined functions, Formal parameters versus actual parameters, Function and variable scope, Recursion, Library functions, Date and time functions, strings in PHP: What is string?, Creating And declaring string, String functions.

Unit IV**10 Hrs**

Class & objects in PHP: - What is class & object, Creating and accessing a class & object, Object properties, Object methods, Overloading, inheritance, constructor and destructor, form handling: Creating HTML form, Handling HTML form data in PHP.

Unit V**10 Hrs**

Database Handling using PHP with MySQL: - Introduction to MySQL: Database terms, Data types, Accessing MySQL- Using MySQL Client and using PHP MyAdmin, MySQL commands, using PHP with MySQL: PHP MySQL functions, connecting to MySQL and selecting the database, Executing simple Queries, Retrieving query results, Counting returned records, Updating records with PHP.

Unit 1

Introduction to PHP

PHP

- PHP is the Web development language written by and for Web developers.
- PHP stands for PHP: Hypertext Preprocessor. The product was originally named Personal Home Page Tools.
- PHP is a server-side scripting language, which can be embedded in HTML or used as a standalone binary.

Syntax:-

```
<?php  
    body of the program  
?>
```

History and Features of PHP

History:

1. Creation (1994): Rasmus Lerdorf created PHP as a set of Perl scripts for his personal use to manage his online resume and track web visits.
2. PHP/FI (1995): Lerdorf extended his scripts and released the PHP/FI (Personal Home Page/Forms Interpreter) version, which added the ability to create simple dynamic web applications.
3. PHP 3 (1998): The first formal version of PHP was released, introducing a more robust and modular architecture.
4. PHP 4 (2000): This version brought significant improvements and features such as better support for object-oriented programming (OOP) and introduced the Zend Engine, which greatly enhanced PHP's performance.
5. PHP 5 (2004): PHP 5 further improved support for OOP with features like interfaces and exceptions. It also introduced the Standard PHP Library (SPL) and enhanced XML support.
6. PHP 6 (Development halted): Work began on PHP 6 to add native Unicode support. However, due to technical challenges and delays, the project was eventually abandoned.
7. PHP 7 (2015): PHP 7 was a major release known for its significant performance improvements, with up to twice the speed of PHP 5. PHP 7 also introduced many new features and enhancements, including scalar type declarations, return type declarations, anonymous classes, and more.
8. PHP 8 (2020): PHP 8 brought several new features and improvements, including Just-In-Time (JIT) compilation for improved performance, union types, named arguments, attributes, and more.

Features:

1. Server-Side Scripting: PHP is primarily used for server-side scripting, meaning that it runs on a web server and processes code before sending the output to the client's browser. This allows for dynamic content generation.
2. Dynamic Typing: PHP is dynamically typed, meaning you don't need to declare the data type of a variable explicitly.
3. Ease of Use: PHP syntax is relatively easy to learn and understand, especially for beginners. It is similar to C, Java, and Perl, making it accessible to developers from various programming backgrounds.

4. Wide Compatibility: PHP is compatible with most web servers and operating systems, including Linux, Unix, Windows, macOS, and more. It also supports a wide range of databases, including MySQL, PostgreSQL, SQLite, and Oracle.
5. Support for Various Protocols: PHP supports various network protocols, such as HTTP, SMTP, LDAP, IMAP, POP3, and more, making it versatile for web development and beyond.
6. Extensive Library Support: PHP has a vast ecosystem of libraries and frameworks, such as Laravel, Symfony, CodeIgniter, and WordPress, which help developers build web applications more efficiently.
7. Embeddable: PHP code can be embedded directly into HTML, allowing developers to mix dynamic content with static content seamlessly.
8. Open Source: PHP is open-source software, meaning it is freely available for anyone to use, modify, and distribute.
9. Active Community: PHP has a large and active community of developers who contribute to its development, provide support, and share resources and knowledge.

Installation and Configuration of PHP

- To start working with PHP, you need a local development environment.
- This environment simulates a web server on your computer, allowing you to test your PHP code before deploying it to a live server.
- Popular choices for local server setups include XAMPP, WAMP, and MAMP.
- These software packages come bundled with Apache (the web server), MySQL (the database), and PHP, forming the core components for web development.

Installing a Local Server

- XAMPP: Download XAMPP from the official website, and follow the installation instructions for your operating system. Once installed, you can start the Apache and MySQL services from the control panel.
- WAMP: Similarly, for Windows users, WAMP provides an easy installation process. After installation, launch the WAMP control panel to start the Apache and MySQL services.
- MAMP: Mac users can opt for MAMP. After installation, you can configure the Apache and MySQL settings through the MAMP application.

Detailed guide on installing XAMPP for Windows

1. Downloading XAMPP

- Visit the official XAMPP download page: <https://www.apachefriends.org/download.html>
- Select the download corresponding to your operating system (Windows, macOS, or Linux).
- Choose either the installer or the archive (.zip) file.

2. Installing XAMPP

- For Installer: Double-click the downloaded .exe file.
- Follow the on-screen instructions in the XAMPP Setup Wizard.
- Choose an installation directory (it's recommended to keep the default suggested location, typically C:\xampp).

- You can optionally select which components to install (Apache, MySQL, PHP, etc.). By default, all components are selected.
- Click "Next" and follow any further prompts until the installation is complete.

3. Starting and Stopping XAMPP Services

- Open the XAMPP Control Panel by going to Start -> All Programs -> XAMPP -> XAMPP Control Panel.

4. Verifying Installation

- Open a web browser and navigate to <http://localhost>. You should see the default XAMPP welcome page if everything is functioning correctly.

Embedding PHP Code in your Web Pages

- Embedding PHP code in web pages allows you to create dynamic content by executing server-side scripts before sending the HTML to the client's browser. Here's how you can embed PHP code in your web pages:

1. Using PHP Opening and Closing Tags:

- PHP code is enclosed within opening `<?php` and closing `?>` tags.
- Any code within these tags will be interpreted and executed by the server.

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

2. Embedding PHP within HTML:

- You can embed PHP code within HTML tags or anywhere in an HTML document

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP Example</title>
</head>
<body>

<h1>Today's Date</h1>

<p>
    <?php
        // PHP code to display today's date
        echo "Today is " . date("Y-m-d") . ".";
    ?>
</p>

</body>
</html>
```

3. Using Short Tags

- Short tags (`<? ... ?>`) can also be used to embed PHP code, but they are not recommended for compatibility reasons and may be disabled in some PHP configurations.

```
<?
    // PHP code goes here
?>
```

4. Outputting HTML within PHP:

- You can output HTML content within PHP using echo or print statements.

```
<?php
    echo "<h1> ICCA </h1>";
?>
```

5. Embedding PHP in External Files:

- PHP code can also be embedded in external files with a .php extension, which are then included or required in other PHP scripts.

external.php

```
<?php
$message = "Hello from external file!";
?>
```

index.php

```
<!DOCTYPE html>
<html>
<head>
    <title>External PHP Example</title>
</head>
<body>
    <?php include 'external.php'; ?>
    <p><?php echo $message; ?></p>
</body>
</html>
```

Understanding PHP,HTML and White Space

1. PHP and HTML Interaction:

- PHP dynamically generates HTML content for web pages.
- PHP generates HTML source code, which is then rendered by the web browser.
- This interaction allows for the creation of dynamic and interactive web pages.

2. White Space Insensitivity:

- PHP and HTML are generally insensitive to white space, allowing developers to format code for better readability without affecting functionality or rendering.
- In PHP, developers can space out code and use blank lines for better organization.
- In HTML, extraneous white spacing doesn't affect the appearance of the rendered page, but it can improve the readability of the source code.

3. Examples:

- In PHP scripts, developers can format code across multiple lines without affecting functionality.

```
<?php
    echo "ICCA"; // Whitespaces can be used better readability
?>
```

- In HTML source, different formatting styles will render the same page, as HTML is mostly white space insensitive.

```
<html>
    <head><title>HTML Code</title></head>
    <body>
        <?php echo "<h1>Welcome to ICCA</h1>";?>
    </body>
</html>
```

4. Creating White Space:

- Developers can use HTML tags like `
` (line break) and `<p></p>` (paragraph) to control spacing in the rendered web page.
- In PHP, developers can use `echo()` or `print()` statements over multiple lines or use the newline character (`\n`) within double quotation marks to control white space in the generated HTML source.

Writing Comments in PHP

- Adding comments to your PHP code is crucial for improving readability, maintainability, and collaboration
- Single-line comments: Use two forward slashes (`//`) followed by your comment text. Everything after `//` on that line is ignored by the PHP interpreter.

`// This is a single-line comment explaining the code above.`

- Multi-line comments: Use the `/*` and `*/` symbols to enclose your comment text. This allows comments to span multiple lines.

```
/*
This is a multi-line comment.
It can explain a complex code block
or provide additional information.
*/
```

Sending Data to the web browser

- In PHP, there are several ways to send data to the web browser. Here's an overview of the common methods:

1. Using echo or print statements:

- These are the simplest methods for sending data to the browser. They directly output the provided string or variable value.

```
<?php
echo "ICCA";
print "<br> Interface College";
?>
```

2. HTML Output:

- When working with PHP to generate dynamic web pages, you'll often be sending HTML content. You can directly embed HTML code within your PHP script using quotation marks.

```
<?php
echo "<h1>ICCA</h1>";
print "<h2> Interface College</h2>";
?>
```

3. Sending Arrays:

- You can send arrays to the browser using `print_r` or `var_dump` for debugging purposes.

```
<?php
$arrayele = array("Interface","College","of","Computer","Applications");
print_r($arrayele);
echo "<br>";
print_r( $arrayele[0] );
echo "<br>";
var_dump( $arrayele );
echo "<br>";
```



```
var_dump( $arrayele[1] );
?>
```

4. Sending File Content:

- You can send the content of a file to the browser using functions like `readfile` or `file_get_contents`.

```
<?php
readfile("example.txt");
// or
echo file_get_contents("example.txt");
?>
```

5. Sending HTTP Headers:

- You can send HTTP headers using the `header` function to control various aspects of the response, such as content type or caching.

```
header("Content-Type: text/plain"); // Informs browser the content is plain text.
echo "This is plain text data.";
```

Data Types in PHP

- Different types of data take up different amounts of memory and may be treated differently when they are manipulated by a script.
- PHP is loosely typed, meaning that it automatically determines the data type at the time data is assigned to each variable.

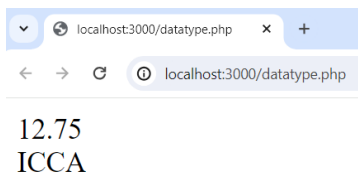
Standard Data Types

| Type | Description | Example |
|-----------------|---|-------------|
| Boolean | One of the special values true or false | True/False |
| Integer | A whole number | 5,10 |
| Float or double | A floating-point number | 3.142,5.510 |
| String | A collection of characters | "Hello" |
| Object | An instance of a class | |
| Array | An ordered set of keys and values | |
| Resource | Reference to a third-party resource (a database, for example) | |
| NULL | An uninitialized variable | \$a; |

Example

```
<?php
$a = 25; //Integer
$b = 12.25; //Float
$c = $a-$b;
$d = "ICCA";//String
echo "$c<br/>";
echo "$d";
?>
```

Output



```
12.75
ICCA
```


Note:

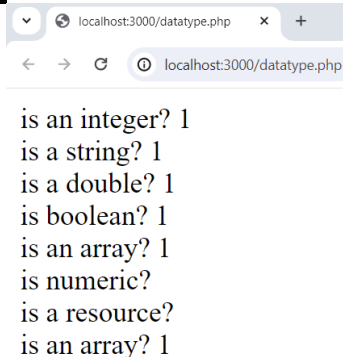
- 1) Resource types are often returned by functions that deal with external applications or files
- 2) The NULL type is reserved for variables that have been declared but no value has been assigned to them.

Testing the Type of a Variable

- PHP has several functions available to test the validity of a particular type of variable one for each type
- The is_* family of functions, such as
 1. is_null() function is used to test whether a variable is NULL or not.
 2. is_int() function is used to test whether the type of the specified variable is an integer or not.
 3. is_string() function is used to find whether a variable is a string or not
 4. is_double() function is used to test whether a variable is a float/double or not. The function
 5. is is an alias of is_float().
 6. is_bool() function is used to find whether a variable is an a boolean or not.
 7. is_array() function is used to find whether a variable is an array or not.
 8. is_numeric() function is used to check whether a variable is numeric or not.
 9. is_resource() function is used to find whether a variable is a resource or not.

Example

```
<?php
$testing = 5;
echo "is an integer? ".is_int($testing) . "<br/>";
$testing = "five";
echo "is a string? ".is_string($testing) . "<br/>";
$testing = 5.024;
echo "is a double? ".is_double($testing) . "<br/>";
$testing = true;
echo "is boolean? ".is_bool($testing) . "<br/>";
$testing = array('apple', 'orange', 'pear');
echo "is an array? ".is_array($testing) . "<br/>";
echo "is numeric? ".is_numeric($testing) . "<br/>";
echo "is a resource? ".is_resource($testing) . "<br/>";
echo "is an array? ".is_array($testing) . "<br/>";
?>
```

Output


```
is an integer? 1
is a string? 1
is a double? 1
is boolean? 1
is an array? 1
is numeric?
is a resource?
is an array? 1
```

Keywords in PHP

- PHP has a set of keywords that are reserved words which cannot be used as function names, class names or method names.

Example: - class, function, new, global, goto, break, continue, include, extends etc....

Variables in PHP

A variable is a special container, which “holds” a value, such as a number, string, object, array, or a Boolean.

Rules of for naming variable

- A variable consists of a name of your choosing, preceded by a dollar sign(\$).
- Variable names can include letters, numbers, and the underscore character(_), but they cannot include spaces.
- Names must begin with a letter or an underscore.

```
$a;
$a_longish_variable_name;
$2453;
$sleepyZZZZ;
```

Example

```
<?php
$a = 25; //Integer
$b = 12.25; //Float
$c = $a-$b;
$d = "ICCA"; //String
echo "$c<br/>";
echo "$d";
?>
```

Globals and Superglobals Variables

1. Global Variables

- In addition to the rules for naming variables, there are rules regarding the availability of variables.
- In general, the assigned value of a variable is present only within the function or script where it resides.
- For example, if you have scriptA.php that holds a variable called \$name with a value of joe, and you want to create scriptB.php that also uses a \$name variable, you can assign to that second \$name variable a value of john without affecting the variable in scriptA.php.
- The value of the \$name variable is local to each script, and the assigned values are independent of each other.
- However, you can also define the \$name variable as global within a script or function. If the \$name variable is defined as a global variable in both scriptA.php and scriptB.php, and these scripts are connected to each other (that is, one script calls the other or includes the other), there will be just one value for the now- shared \$name variable.

2. Superglobal Variables

- In addition to global variables of your own creation, PHP has several predefined variables called superglobals.
- These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:
 - \$_GET contains any variables provided to a script through the GET method.

- `$_POST` contains any variables provided to a script through the POST method. `$_COOKIE` contains any variables provided to a script through a cookie.
- `$_FILES` contains any variables provided to a script through file uploads.
- `$_SERVER` contains information such as headers, file paths, and script locations. `$_ENV` contains any variables provided to a script as part of the server environment. `$_REQUEST` contains any variables provided to a script via GET, POST, or COOKIE input mechanisms.
- `$_SESSION` contains any variables that are currently registered in a session.

Constant in PHP

- The value does not change after the execution of the program is called constant.
- You must use PHP's built-in `define()` function to create a constant, which subsequently cannot be changed unless you specifically `define()` it again.
- To use the `define()` function, place the name of the constant and the value you want to give it within parentheses and separated by a comma:

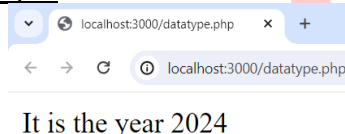
Syntax

```
define("YOUR_CONSTANT_NAME", Constantvalue);
```

Example

```
<?php
    define("THE_YEAR", "2024");
    echo "It is the year ".THE_YEAR;
?>
```

Output



It is the year 2024

- The `define()` function can also accept a third Boolean argument that determines whether the constant name should be case sensitive.
- By default, constant names are case sensitive.
- However, by passing `true` to the `define()` function, you can change this behavior. you could access the "THE_YEAR" constant without worrying about case:
eg: `echo the_year;`
`echo ThE_YeAr;`
`echo THE_YEAR;`

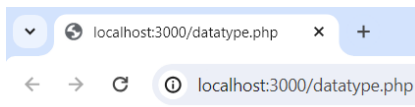
constant() function

- As indicated by the name, this function will return the value of the constant.
- This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

Example

```
<?php
    define("THE_YEAR", "2024");
    echo "It is the year ".THE_YEAR."<br/>";
    echo constant("THE_YEAR");
?>
```

Output



It is the year 2024
2024

Predefined Constants

- PHP automatically provides some built-in constants for you.
- For example, the constant FILE returns the name of the file that the PHP engine is currently reading. The constant LINE returns the current line number of the file.

Expression

- An expression is any combination of functions, values, and operators that resolves to a value

(OR)

- The combination of operands with an operator to produce a result is called an expression

Eg: $a + b$; Where: a & b are operands $+$ is a operator

Operators in PHP

- Operators are symbols used to manipulate data stored in variables, to make it possible to use one or more values to produce a new value, or to check the validity of data to determine the next step in a condition, and so forth

1. Arithmetic Operators: - There are following arithmetic operators supported by PHP language.

| Operator | Name | Example | Sample Result |
|----------|----------------|---------|-----------------|
| + | Addition | $10+3$ | 13 |
| - | Subtraction | $10-3$ | 7 |
| / | Division | $10/3$ | 3.3333333333333 |
| * | Multiplication | $10*3$ | 30 |
| % | Modulus | $10\%3$ | 1 |

2. The Assignment Operator

- The assignment operator consists of the single character: $=$.
- The assignment operator takes the value of the right-side operand and assigns it to the left-side operand
- Example: $\$name = "Jimbo";$

3. Combined Assignment Operators

- A combined assignment operator consists of a standard operator symbol followed by an equal sign.
- Each arithmetic operator, as well as the concatenation operator, also has a corresponding combination assignment operator.

| Operator | Example | Equivalent To |
|-----------------|-----------------------------|----------------------------------|
| <code>+=</code> | <code>\$x += 5</code> | <code>\$x = \$x + 5</code> |
| <code>-=</code> | <code>\$x -= 5</code> | <code>\$x = \$x - 5</code> |
| <code>/=</code> | <code>\$x /= 5</code> | <code>\$x = \$x / 5</code> |
| <code>*=</code> | <code>\$x *= 5</code> | <code>\$x = \$x * 5</code> |
| <code>%=</code> | <code>\$x %= 5</code> | <code>\$x = \$x % 5</code> |
| <code>.=</code> | <code>\$x .= " test"</code> | <code>\$x = \$x . " test"</code> |

4. Comparison Operators

- Comparison operators perform comparative tests using their operands and return the Boolean value true if the test is successful or false if the test fails.

Example:

(1) `$x=5;`
`$y=5.0;`
`$x == $y // returns true`

(2) `$x=5;`
`$y=5.0;`
`$x === $y // returns false`

| Operator | Name | Returns True If... | Example (\$x is 4) | Result |
|--------------------|--------------------------|--|--------------------------|--------|
| <code>==</code> | Equivalence | Left is equivalent to right. | <code>\$x == 5</code> | false |
| <code>!=</code> | Nonequivalence | Left is not equivalent to right. | <code>\$x != 5</code> | true |
| <code>===</code> | Identical | Left is equivalent to right, and they are the same type. | <code>\$x === 4</code> | true |
| | Nonequivalence | Left is equivalent to right, but they are not the same type. | <code>\$x === "4"</code> | false |
| <code>></code> | Greater than | Left is greater than right. | <code>\$x > 4</code> | false |
| <code>>=</code> | Greater than or equal to | Left is greater than or equal to right. | <code>\$x >= 4</code> | true |
| <code><</code> | Less than | Left is less than right. | <code>\$x < 4</code> | false |
| <code><=</code> | Less than or equal to | Left is less than or equal to right. | <code>\$x <= 4</code> | true |

5. Increment & Decrement Operator

- PHP provides some special operators that allow you to add or subtract the integer constant 1 from an integer variable, assigning the result to the variable itself.

1. Post- increment & decrement operator

`$x++`; `$x` is incremented by 1

`$x--`; `$x` is decremented by 1

2. Pre- increment and decrement operator

++\$x; \$x is incremented by 1
 -\$x ; \$x is decremented by 1

6. Logical Operator

- Logical operators test combinations of Boolean values.
- For example, the or operator, which is indicated by two pipe characters (||) or simply the word or, returns the Boolean value

| Operator | Name | Returns True If... | Example | Result |
|----------|------|--------------------------------------|----------------|--------|
| | Or | Left or right is true. | true false | true |
| or | Or | Left or right is true. | true or false | true |
| xor | Xor | Left or right is true, but not both. | true xor true | false |
| && | And | Left and right are true. | true && false | false |
| and | And | Left and right are true. | true and false | false |
| ! | Not | The single operand is not true. | ! true | false |

7. The Concatenation Operator

- The concatenation operator is represented by a single period (.).
- Treating both operands as strings, this operator appends the right-side operand to the left- side operand.
- So "hello"." world" returns "hello world"

Operator precedence

- Operator precedence determines the grouping of terms in an expression.
- This affects how an expression is evaluated.
- Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator

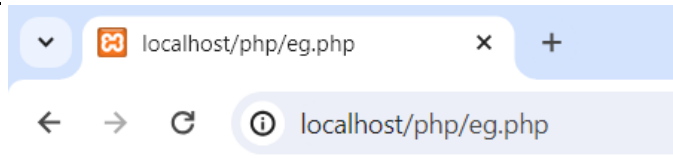
| Category | Operator | Associativity |
|----------------|------------------|---------------|
| Unary | ! ++ -- | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Logical AND | && | Left to right |
| Logical OR | | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %= | Right to left |

Example

```
<?php
$num1 = 100;
$num2 = 200;
$num3 = 90;
if(( $num1 > $num2 ) && ($num1>$num3))
{
    echo "$num1 is greater";
}
```

```
}  
elseif( $num2 > $num3 )  
{  
    echo"$num2 is greater";  
}  
else  
{  
    echo"$num3 is greater";  
}  
?>
```

Output



200 is greater



Interface College of Computer Applications (ICCA)

Unit 2: - Programming with PHP

Conditional Statements

It is common for scripts to evaluate conditions and change their behavior accordingly.

The if Statement

- The if statement is a way of controlling the execution of a statement that follows it.
- The if statement evaluates an expression found between parentheses.
- If this expression results in a true value, the statement is executed. Otherwise, the statement is skipped entirely

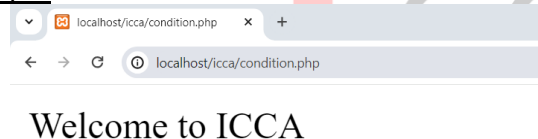
Syntax:

```
if (expression)
{
    // code to execute if the expression evaluates to true
}
```

Example

```
<?php
    $var = "ICCA";
    if ($var == "ICCA")
    {
        echo "Welcome to ICCA" ;
    }
?>
```

Output



Welcome to ICCA

Using the else Clause with the if Statement

- When working with an if statement, you might want to define an alternative block of code that should be executed if the expression you are testing evaluates to false.
- You can do this by adding else to the if statement followed by a further block of code.

Syntax:

```
if (expression)
{
    // code to execute if the expression evaluates to true
}
else
{
    // code to execute in all other cases
}
```

Example

```
<?php
    $var = "DVG";
    if ($var == "ICCA")
```

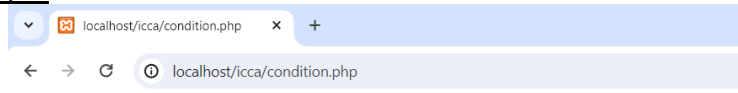
```

{
    echo "Welcome to ICCA" ;
}
else
{
    echo "Welcome to Davangere";
}

```

?>

Output



Welcome to Davangere

Using the elseif Clause

- We can use an if...elseif...else clause to test multiple expressions.

Syntax:

```

if (expression)
{
    // code to execute if the expression evaluates to true
}
elseif (another expression)
{
    // code to execute if the previous expression failed
    // and this one evaluates to true
}
else

```

Example

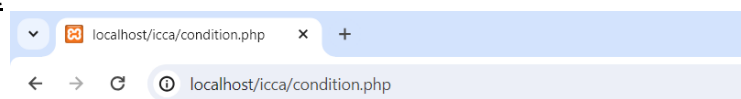
```

<?php
    $var = "Interface";
    if ($var == "ICCA")
    {
        echo "Welcome to ICCA" ;
    }
    elseif($var == "Interface")
    {
        echo "Welcome to Interface" ;
    }
    else
    {
        echo "Welcome to Davangere";
    }

```

?>

Output



Welcome to Interface

The switch Statement

- The switch statement is an alternative way of changing flow, based on the evaluation of an expression.
- A switch statement evaluates only one expression in a list of expressions, selecting the correct one based on a specific bit of matching code.

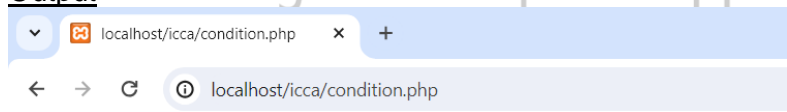
Syntax

```
switch (expression)
{
    case result1:
        // execute this if expression results in result1
        break;
    case result2:
        // execute this if expression results in result2
        break;
    default:
        // the default code block
}
```

Example

```
<?php
$var = "ICCA";
switch ($var)
{
    case "ICCA":
        echo "Welcome to ICCA";
        break;
    case "ISAT":
        echo "Welcome to ISAT";
        break;
    default:
        echo "Welcome to Interface";
        break;
}
```

Output



Welcome to ICCA

Using the ?: Operator

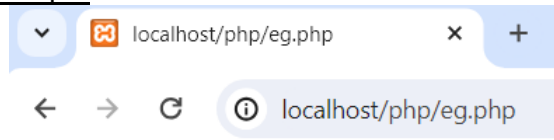
- The ?: or ternary operator is similar to the if statement, except that it returns a value derived from one of two expressions separated by a colon.
- This construct provides you with three parts of the whole, hence the name ternary.
- The expression used to generate the returned value depends on the result of a test expression.

Syntax:

(expression) ? returned_if_expression_is_true : returned_if_expression_is_false;

Example

```
<?php
$num = 10;
$result = ($num % 2 == 0) ? "Even" : "Odd";
echo "The number $num is $result.";
?>
```

Output

The number 10 is Even.

Looping Statement

- Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.
 - for – loops through a block of code a specified number of times.
 - while – loops through a block of code if and as long as a specified condition is true.
 - do...while – loops through a block of code once, and then repeats the loop as long as a special condition is true.
 - foreach – loops through a block of code for each element in an array.

While loop

- The while statement will execute a block of code if and as long as a test expression is true.
- If the test expression is true then the code block will be executed.
- After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

```
while (expression)
{
    // do something
}
```

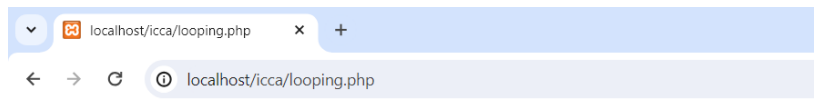
Example

```
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
    $num--;
    $i++;
}

echo ("Loop stopped at i = $i and num = $num" );
?>
```

Output



Loop stopped at $i = 10$ and $num = 40$

The do...while Statement

- The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

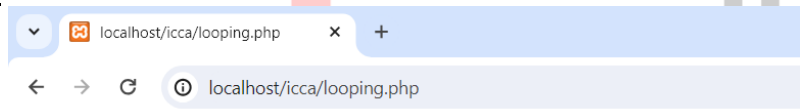
```
do
{
    // code to be executed
} while (expression);
```

Example

```
<?php
    $i = 0;
    $num = 0;

    do {
        $i++;
        echo "$i ";
    } while( $i < 10 );
    echo "<br/> Loop stopped at i = $i";
?>
```

Output



1 2 3 4 5 6 7 8 9 10

Loop stopped at $i = 10$

Interface College of Computer Applications (ICCA)

The for loop statement

- The for statement is used when you know how many times you want to execute a statement or a block of statements.
- The PHP for loop allows the user to put all the loop-related statements in one place.

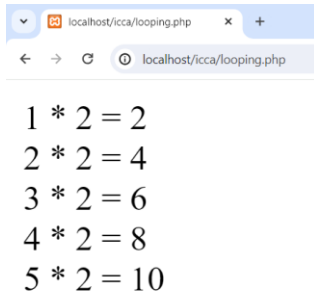
Syntax

```
for (initialization expression; test expression; modification expression)
{
    // code to be executed
}
```

Example

```
<?php
    for ($counter=1; $counter<=5; $counter++)
    {
        echo "$counter * 2 = ". ($counter * 2). "<br/>";
    }
?>
```

Output



```

1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10

```

The foreach loop statement

- The foreach statement is used to loop through arrays.
- For each pass the value of the current array element is assigned to \$value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```

foreach (array as value)
{
    code to be executed;
}

```

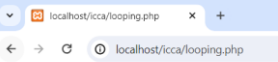
Example

```

<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
    echo "Value is $value <br />";
}
?>

```

Output



```

Value is 1
Value is 2
Value is 3
Value is 4
Value is 5

```

Break Statement

- Both while and for statements incorporate a built in test expression with which you can end a loop.
- The break statement enables you to break out of a loop based on the results of additional tests.

Example

```

<?php
for($a=0;$a<=10;$a++)
{
    if($a==5)
    {
        break;
    }
}

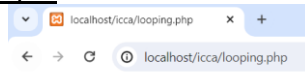
```

```

    else
    {
        echo "$a ";
    }
}
?>

```

Output



0 1 2 3 4

Continue Statement

- The continue statement ends execution of the current iteration but doesn't cause the loop as a whole to end.

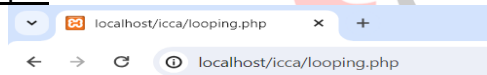
Example

```

<?php
for($a=0;$a<=10;$a++)
{
    if($a==5)
    {
        continue;
    }
    else
    {
        echo "$a ";
    }
}
?>

```

Output



0 1 2 3 4 6 7 8 9 10

Arrays

- Array is collection; it might be homogenous or heterogeneous in nature.
- Accessing array elements is carried by index and index starts from 0 and its max length is maxsize – 1.
- Memory allocation of the array is mostly sequential.

In PHP arrays are classified as below.

- Numeric array(Indexed Array) :-
 - An array with a numeric index.
 - Values are stored and accessed in linear fashion.
- Associative array
 - An array with strings as index.
 - This stores element values in association with key values rather than in a strict linear index order.
- Multidimensional array
 - An array containing one or more arrays and values are accessed using multiple indices.

1. Numeric array (Indexed array)

- These arrays can store numbers, strings and any object but their index will be represented by numbers.
- By default array index starts from zero.

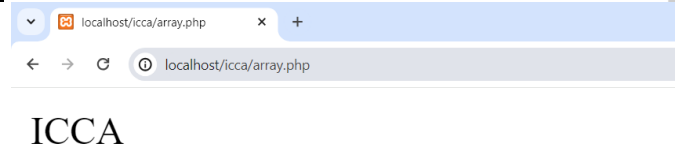
Creating array

- You can create an array using either the array() function or the array operator [].
- The array() function is usually used when you want to create a new array and populate it with more than one element, all in one fell swoop.
- The array operator is often used when you want to create a new array with just one element at the outset, or when you want to add to an existing array element.

Example for creating array using array() function

```
<?php
$clgname = array("I", "C", "C", "A");
foreach($clgname as $name)
{
    echo $name;
}
?>
```

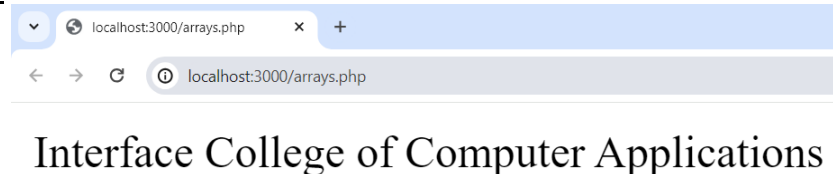
Output



Example for creating array using array operator([])

```
<?php
$clgname[] = "Interface";
$clgname[] = "College";
$clgname[] = "of";
$clgname[] = "Computer";
$clgname[] = "Applications";
foreach($clgname as $name)
{
    echo "$name ";
}
?>
```

Output



Accessing Array

- Array can be accessed using index and foreach statement

Example

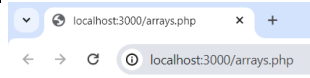
```
<?php
$clgname = array("I", "C", "C", "A");
$clgname[] = "Interface";
```

```

echo $clgname[2]."<br/>";
echo $clgname[0]."<br/>";
echo $clgname[4]."<br/>";
?>

```

Output



C
I
Interface

2. Associative Arrays

- The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index.
- Associative array will have their index as string so that you can establish a strong association between key and values.

Creating Associative Arrays

- Whereas numerically indexed arrays use an index position as the key—0, 1, 2, and so forth.
- Associative arrays use actual named keys.

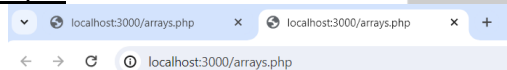
Example

```

<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
foreach ($character as $key => $value)
{
    echo "'". $key . "->". $value . "<br/>";
}
?>

```

Output



name->Rama
clg->ICCA
age->20
course->BCA

Accessing Associative Array

You can reference specific elements of an associative array using the specific key and foreach statement.

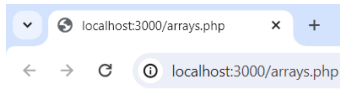
Example

```

<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
echo $character['name']. "<br/>";
echo $character['clg'];
?>

```

Output



Rama
ICCA

Note 1: Don't keep associative array inside double quote while printing otherwise it would not return any value.

Note 2: The only difference between an associative array and a numerically indexed array is the key name.

3. Multidimensional Array

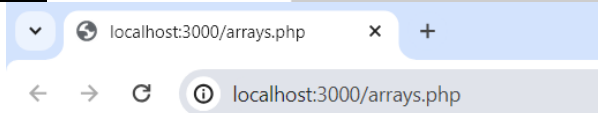
- The first two types of arrays hold strings and integers, whereas this third type holds other arrays.
- In a multi-dimensional array each element in the main array can also be an array.
- And each element in the sub-array can be an array, and so on.

Examples for creating Multidimensional Array

Example 1:-

```
<?php
$characters = array(
    array("name" => "Rama", "age" => 20, "Course" => "BCA" ),
    array("name" => "Shama", "age" => 20, "Course" => "Bsc" ),
    array("name" => "Bhama", "age" => 20, "Course" => "BBA")
);
echo $characters[0]['name'];
?>
```

Output

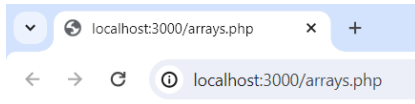


Rama

Example 2

```
<?php
$characters = array(
    array(1,2,3,4,5 ),
    array( 6,7,8,9,10),
    array(11,12,13)
);
echo "characters[0][1]= ".$characters[0][1]."<br/>";
echo "characters[1][1]= ".$characters[1][1]."<br/>";
?>
```

Output

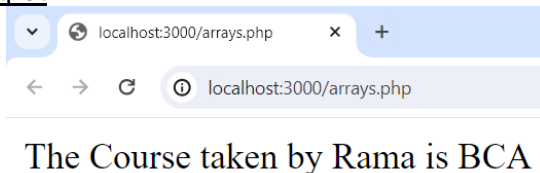


```
characters[0][1]= 2
characters[1][1]= 7
```

Example 3

```
<?php
$characters = array(
    "Rama" => array("age" => 20, "Course" => "BCA" ),
    "Shama" => array("age" => 20, "Course" => "Bsc" ),
    "Bhama" => array("age" => 20, "Course" => "BBA" )
);
echo "The Course taken by Rama is ".$characters['Rama']['Course'];
?>
```

Output



Accessing Multidimensional array

- Values in the multi-dimensional array are accessed using multiple index.
- Refer above examples

Manipulating Arrays using Built-in array functions

- Arrays can be manipulated by using several built-in functions
- Some of the more common (and useful) functions are described briefly in this section

1. count() and sizeof() :- Each of these functions counts the number of elements in an array; they are aliases of each other.

Example:-

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Displaying the number of elements in an array using count function =
".count($clgname);
echo "<br/>Displaying the number of elements in an array using sizeof
function = ".sizeof($clgname);

?>
```

Output



Displaying the number of elements in an array using count function = 4
Displaying the number of elements in an array using sizeof function = 4

2. array_push() :- This function adds one or more elements to the end of an existing array.

Syntax

```
int array_push(array &$array, mixed $value1 [, mixed $value2, ...])
```

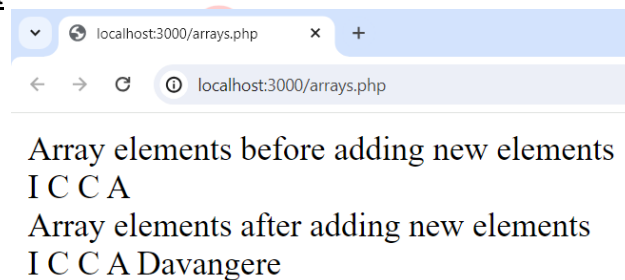
Where,

- `array &$array`: This is a reference to the array to which you want to add elements. The ampersand (&) ensures the changes are reflected in the original array.
- `mixed $value1, [, mixed $value2, ...]`: These represent the values you want to append to the end of the array. You can specify one or more values to be pushed.

Example

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Array elements before adding new elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
array_push($clgname,"Davangere");
echo "<br/>Array elements after adding new elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
```

Output



Array elements before adding new elements
I C C A
Array elements after adding new elements
I C C A Davangere

3. `array_pop()`: - This function removes the last element of an existing array and returns the remaining elements of an array.

Syntax

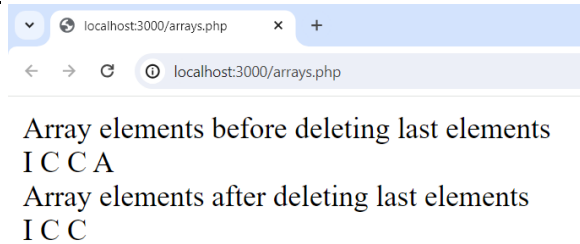
`mixed array_pop(array &$array)`

Where,

- `array &$array`: This is a reference to the array from which you want to remove the last element. The ampersand (&) ensures the modification happens on the original array itself.

Example

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Array elements before deleting last elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
array_pop($clgname);
echo "<br/>Array elements after deleting last elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
?>
```

Output

4. array_unshift(): - This function adds one or more elements to the beginning of an existing array.

Syntax

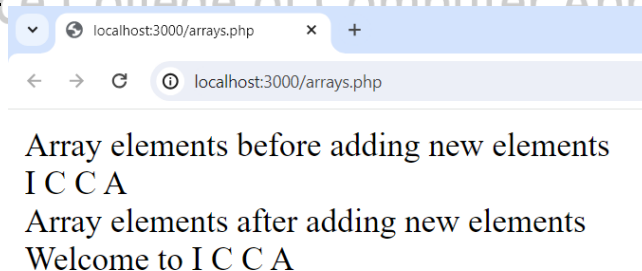
int array_unshift(array &\$array, mixed \$value1 [, mixed \$value2, ...])

Where,

- array &\$array: This is a reference to the array you want to prepend elements to. The ampersand (&) ensures the changes are made on the original array.
- mixed \$value1, [, mixed \$value2, ...]: These represent the values you want to prepend to the beginning of the array. You can specify one or more values to be prepended.

Example

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Array elements before adding new elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
array_unshift($clgname,"Welcome","to");
echo "<br/>Array elements after adding new elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
?>
```

Output

5. array_shift(): - This function removes (and returns) the first element of an existing array.

Syntax

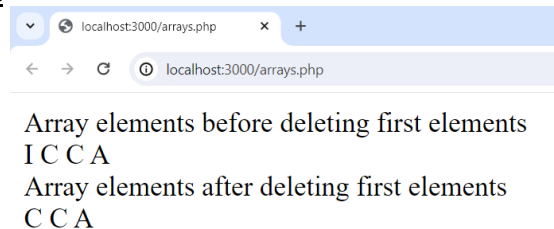
mixed array_shift(array &\$array)

Where,

- array &\$array: This is a reference to the array you want to remove the first element from. The ampersand (&) ensures that the modification happens on the original array itself.

Example

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Array elements before deleting last elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
array_shift($clgname);
echo "<br/>Array elements after deleting last elements<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
?>
```

Output

6. array_merge(): -This function combines two or more existing arrays.

Syntax

array array_merge(array \$array1, array \$array2, ..., array \$arrayN)

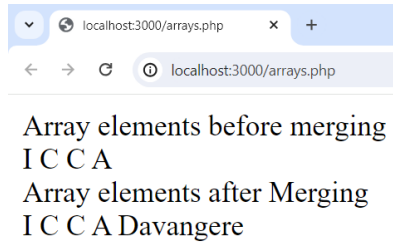
Where,

- array \$array1, array \$array2, ..., array \$arrayN: These represent the arrays you want to merge. You can pass any number of arrays as arguments to array_merge

Example

```
<?php
$clgname = array("I", "C", "C", "A");
$city[] = "Davangere";
echo "Array elements before merging<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
$ele_merge = array_merge($clgname,$city);
echo "<br/>Array elements after Merging<br/>";
foreach($ele_merge as $name)
{
    echo "$name ";
}
?>
```

Output



Array elements before merging
I C C A
Array elements after Merging
I C C A Davangere

7. array_keys(): -This function returns an array containing all the key names within a given array.

Syntax

array array_keys(array \$input, mixed \$filter_value = null, bool \$strict = false)

Where,

- array \$input: The array from which you want to extract the keys.
- mixed \$filter_value (optional): If provided, it filters the keys based on their values (using strict comparison by default).
- bool \$strict (optional): When set to TRUE, uses strict comparison during value filtering with \$filter_value

Example

```
<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
foreach ($character as $key => $value)
{
    echo " ". $key . "-> ". $value . "<br/>";
}
$keyarray = array_keys($character);
foreach ($keyarray as $key)
{
    echo "<br/> ". $key . " ";
}
?>
```

Output



name->Rama
clg->ICCA
age->20
course->BCA

name
clg
age
course

8. array_values(): -This function returns an array containing all the values within a given array.

Syntax

array array_values(array \$input)

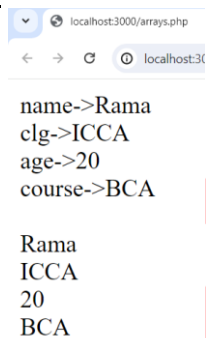
Where,

array \$input: This is the array from which you want to extract the values.

Example

```
<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
foreach ($character as $key => $value)
{
    echo "" . $key . "->" . $value . "<br/>";
}
$keyarray = array_values($character);
foreach ($keyarray as $key)
{
    echo "<br/>" . $key . "";
}

?>
```

Output


```
name->Rama
clg->ICCA
age->20
course->BCA

Rama
ICCA
20
BCA
```

9. **shuffle()**: - This function randomizes the elements of a given array. The syntax of this function is simply as follows:

Syntax

bool shuffle(array &\$array)

Where,

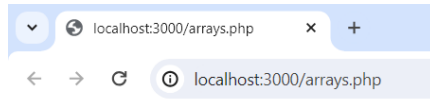
- array &\$array: This is a reference to the array you want to shuffle. The ampersand (&) ensures the shuffling happens on the original array.

Example

```
<?php
$clgname = array("I", "C", "C", "A");
echo "Array elements before Shuffle<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}
shuffle($clgname);
echo "<br/>Array elements after Shuffle<br/>";
foreach($clgname as $name)
{
    echo "$name ";
}

?>
```

Output



Array elements before Shuffle

I C C A

Array elements after Shuffle

I C A C

10. **reset()**: - This function rewinds the pointer to the beginning of an array

Syntax

mixed reset(array &\$array)

Where,

- **array &\$array**: This is a reference to the array for which you want to reset the internal pointer. The ampersand (&) ensures changes are reflected in the original array.

Eg:--`arrayreset($character)`; This function proves useful when you are performing multiple manipulations on an array, such as sorting, extracting values, and so forth.

11. **sort()**: Sorts an array in ascending order based on the values. It reassigns numerical keys, starting from 0.

Syntax

sort(array &\$array, int \$sort_flags = SORT_REGULAR)

Where,

- **array &\$array**: This is a reference to the associative array you want to sort. The ampersand (&) ensures the sorting happens on the original array.
- **int \$sort_flags = SORT_REGULAR (optional)**: This parameter specifies how the keys in the array should be compared during sorting. It can be set to the same constants as the `asort` function:
 - **SORT_REGULAR (default)**: Sorts keys using a normal comparison (equivalent to comparing two strings).
 - **SORT_NUMERIC**: Sorts keys numerically (assuming they are valid numbers).
 - **SORT_STRING**: Sorts keys as strings.
 - **SORT_LOCALE_STRING**: Sorts keys as strings, considering the current locale.
 - **SORT_NATURAL**: Sorts keys as strings using a "natural order" algorithm that handles numbers correctly.
 - **SORT_FLAG_CASE**: Can be combined with other flags to control case-sensitivity (e.g., `SORT_NUMERIC | SORT_FLAG_CASE` for case-insensitive numeric sort).

Example

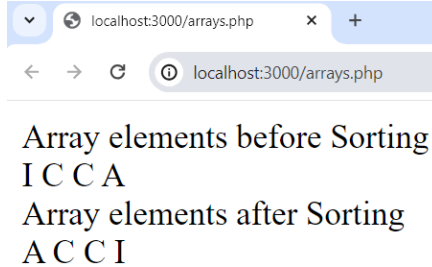
```
<?php
$clname = array("I", "C", "C", "A");
echo "Array elements before Sorting<br/>";
foreach($clname as $name)
{
    echo "$name ";
}
sort($clname);
echo "<br/>Array elements after Sorting<br/>";
foreach($clname as $name)
```

```

    {
        echo "$name ";
    }
?>

```

Output



Array elements before Sorting
I C C A
Array elements after Sorting
A C C I

12. **rsort()**: - Sorts an array in descending order based on the values. Similar to **sort()**, it reassigns numerical keys.

Syntax

rsort(array &\$array, int \$sort_flags = SORT_REGULAR)

Where,

- **array &\$array**: This is a reference to the associative array you want to sort. The ampersand (&) ensures the sorting happens on the original array.
- **int \$sort_flags = SORT_REGULAR** (optional): This parameter specifies how the keys in the array should be compared during sorting. It can be set to the same constants as the **asort** function:
 - **SORT_REGULAR** (default): Sorts keys using a normal comparison (equivalent to comparing two strings).
 - **SORT_NUMERIC**: Sorts keys numerically (assuming they are valid numbers).
 - **SORT_STRING**: Sorts keys as strings.
 - **SORT_LOCALE_STRING**: Sorts keys as strings, considering the current locale.
 - **SORT_NATURAL**: Sorts keys as strings using a "natural order" algorithm that handles numbers correctly.
 - **SORT_FLAG_CASE**: Can be combined with other flags to control case-sensitivity (e.g., **SORT_NUMERIC | SORT_FLAG_CASE** for case-insensitive numeric sort).

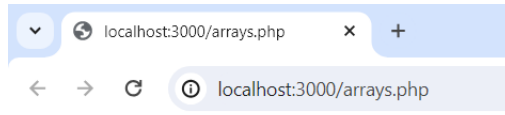
Example

```

<?php
    $clname = array(5,100,20,48,26,30,78);
    echo "Array elements before Sorting<br/>";
    foreach($clname as $name)
    {
        echo "$name ";
    }
    rsort($clname);
    echo "<br/>Array elements after Sorting<br/>";
    foreach($clname as $name)
    {
        echo "$name ";
    }
?>

```

Output



Array elements before Sorting

5 100 20 48 26 30 78

Array elements after Sorting

100 78 48 30 26 20 5

13. **ksort()**: - Sorts an array in ascending order based on the keys. It maintains the original association between keys and values.

Syntax

ksort(array &\$array, int \$sort_flags = SORT_REGULAR)

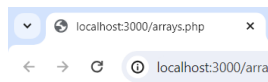
Where,

- array &\$array: This is a reference to the associative array you want to sort. The ampersand (&) ensures the sorting happens on the original array.
- int \$sort_flags = SORT_REGULAR (optional): This parameter specifies how the keys in the array should be compared during sorting. It can be set to the same constants as the asort function:
 - SORT_REGULAR (default): Sorts keys using a normal comparison (equivalent to comparing two strings).
 - SORT_NUMERIC: Sorts keys numerically (assuming they are valid numbers).
 - SORT_STRING: Sorts keys as strings.
 - SORT_LOCALE_STRING: Sorts keys as strings, considering the current locale.
 - SORT_NATURAL: Sorts keys as strings using a "natural order" algorithm that handles numbers correctly.
 - SORT_FLAG_CASE: Can be combined with other flags to control case-sensitivity (e.g., SORT_NUMERIC | SORT_FLAG_CASE for case-insensitive numeric sort).

Example

```
<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
echo "Before sorting<br/>";
foreach ($character as $key => $value)
{
    echo "'". $key . "->". $value . "<br/>";
}
ksort($character);
echo "</br>";
echo "After sorting<br/>";
foreach ($character as $key => $value)
{
    echo "'". $key . "->". $value . "<br/>";
}
?>
```

Output



Before sorting
 name->Rama
 clg->ICCA
 age->20
 course->BCA

After sorting
 age->20
 clg->ICCA
 course->BCA
 name->Rama

Note: - `krsort()`: Sorts an array in descending order based on the keys. It maintains the original association between keys and values.

14. `asort()`: - Sorts an associative array in ascending order based on the values. It maintains the original association between keys and values.

Syntax

`asort(array &$array, int $sort_flags = SORT_REGULAR)`

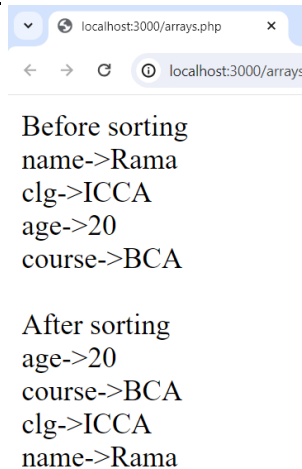
Where,

- `array &$array`: This is a reference to the associative array you want to sort. The ampersand (&) ensures the sorting happens on the original array.
- `int $sort_flags = SORT_REGULAR` (optional): This parameter specifies how the values in the array should be compared during sorting. It can be set to one of the following constants:
 - `SORT_REGULAR` (default): Sorts elements using a normal comparison (equivalent to comparing two strings).
 - `SORT_NUMERIC`: Sorts elements numerically.
 - `SORT_STRING`: Sorts elements as strings.
 - `SORT_LOCALE_STRING`: Sorts elements as strings, considering the current locale.
 - `SORT_NATURAL`: Sorts elements as strings using a "natural order" algorithm that handles numbers correctly.
 - `SORT_FLAG_CASE`: Can be combined with other flags to control case-sensitivity (e.g., `SORT_NUMERIC | SORT_FLAG_CASE` for case-insensitive numeric sort).

Example

```
<?php
$character = array("name" => "Rama", "clg" => "ICCA", "age" => 20,
                  "course" => "BCA");
echo "Before sorting<br/>";
foreach ($character as $key => $value)
{
    echo "'". $key . "->". $value . "<br/>";
}
asort($character);
echo "</br>";
echo "After sorting<br/>";
foreach ($character as $key => $value)
{
    echo "'". $key . "->". $value . "<br/>";
}
```

```
}
?>
Output
```



Note: - `arsort()`: Sorts an associative array in descending order based on the values. It maintains the original association between keys and values.

15. `array_splice()`: - The `array_splice()` function in PHP is used to remove a portion of an array and replace it with something else.

Syntax

`array_splice(array &$input, int $start, int $length = 0, mixed $replace = array())`

Where,

- `array &$input`: This is a reference to the array you want to modify. The ampersand (&) makes changes persistent in the original array.
- `int $start`: This is the numeric index where you want to start inserting or removing elements. Negative values can be used to count from the end of the array.
- `int $length = 0` (optional): This specifies the number of elements to remove from the `$start` position. If set to 0, no elements are removed, and the insertion happens at the specified position.
- `mixed $replace = array()` (optional): This is either a single value or an array of values to be inserted at the `$start` position. If omitted, elements are simply removed starting from `$start`.

Example

```
<?php
$original_array = array(1,2,4,5,6);
echo "Before inserting new item, the array is: ";
foreach ($original_array as $x)
{
    echo "$x ";
}
// Inserted value and position
$inserted_value = 3;
$position = 2;

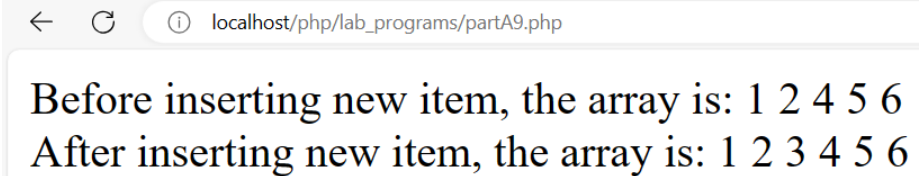
if ($position >= 0 && $position <= count($original_array)) {
    // Insert the value into the array at the specified position
    array_splice($original_array, $position, 0, $inserted_value);
    echo "<br>After inserting new item, the array is: ";
    foreach ($original_array as $x) {
        echo "$x ";
    }
}
```



```

    }
} else
{
    echo "<br>Invalid position specified. The position should be between 0 and " .
    count($original_array);
}
?>

```

Output


Before inserting new item, the array is: 1 2 4 5 6
 After inserting new item, the array is: 1 2 3 4 5 6

Including and Requiring files

- Both include and require are functions in PHP used to incorporate the contents of another file into the current script at runtime.
- They offer a way to organize and reuse code across multiple files, promoting code maintainability and efficiency.

The primary difference lies in how they handle errors:

- **require():**
 - If the specified file is not found or there's an error including the file, it throws a fatal error (E_COMPILE_ERROR) and halts the script's execution entirely. This ensures your script doesn't proceed with potentially broken code due to a missing file.
- **include():**
 - On the other hand, if include() encounters an error including the file, it only generates a warning (E_WARNING) and the script execution continues. This might lead to unexpected behaviour if the missing file contains essential code.

Including Files with include()

- The include() statement enables you to incorporate other files (usually other PHP files) into your documents.
- PHP code in these included files will then be executed as if it were part of the main document.
- This is useful for including code libraries within multiple pages.
- The include() statement requires a single argument: a relative path to the file to include.
- **Syntax :-** include('filename');

Example:-**includedfile.php**

```

<?php
function factorial($number) {
    if ($number < 0) {
        throw new Exception("Factorial is not defined for negative numbers.");
    }
    $factorial = 1;
    for ($i = 1; $i <= $number; $i++) {
        $factorial *= $i;
    }
}

```

```

    }
    return $factorial;
}
?>

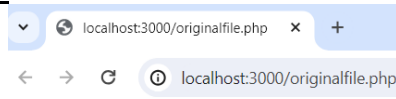
```

originalfile.php

```

<?php
include("includedfile.php");
echo"The factorial of 5 is: -".factorial(5);
?>

```

Output:

The factorial of 5 is: -120

Including files with require()

- The require() statement enables you to incorporate other files (usually other PHP files) into your documents.
- PHP code in these included files will then be executed as if it were part of the main document.
- This is useful for including code libraries within multiple pages.
- The require() statement requires a single argument: a relative path to the file to include.

Example**includedfile.php**

```

<?php
function factorial($number) {
    if ($number < 0) {
        throw new Exception("Factorial is not defined for negative numbers.");
    }
    $factorial = 1;
    for ($i = 1; $i <= $number; $i++) {
        $factorial *= $i;
    }
    return $factorial;
}
?>

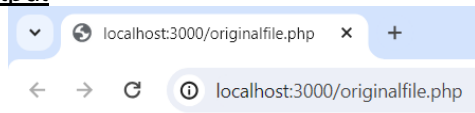
```

originalfile.php

```

<?php
require("includedfile.php");
echo"The factorial of 6 is: -".factorial(6);
?>

```

Output

The factorial of 6 is: -720

Implicit and Explicit Casting

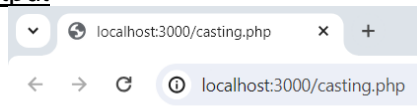
- This is the process of changing a variable's data type to another data type automatically or manually. This process is known as typecasting and it can be done in two ways.
 - Implicit Casting (Automatic)
 - Explicit Casting (Manual)

Implicit Casting

- Implicit data typecasting can be done by PHP automatically.
- For example the divide operation by two integer numbers then in this case result might be either an integer or float number.

```
<?php
$num = 4;
$num1 = 8;
echo $num/$num1;
?>
```

Output



0.5

Explicit Casting

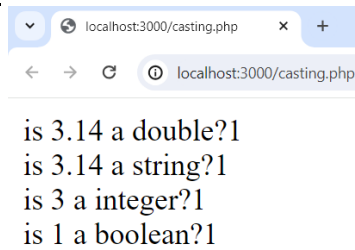
- Explicit casting can be customized by the user. Users can change variable types from one data type to another data type with the help of PHP methods.

| Cast | Description |
|-------------------------------|-----------------------|
| (int) or (integer) | : Cast to an integer. |
| (float) or (double) or (real) | :Cast to a float. |
| (bool) or (boolean) | : Cast to a boolean. |
| (string): | Cast to a string. |
| (array): | Cast to an array. |
| (object): | Cast to an object. |

Example

```
<?php
$a = 3.14;
echo "is $a a double?".is_double($a)."<br/>";
$b = (string) $a;
echo "is $b a string?".is_string($b)."<br/>";
$b = (integer) $a;
echo "is $b a integer?".is_integer($b)."<br/>";
$b = (boolean) $a;
echo "is $b a boolean?".is_bool($b)."<br/>";
```

Output ?>



```
is 3.14 a double?1
is 3.14 a string?1
is 3 a integer?1
is 1 a boolean?1
```

Changing Type with settype()

- PHP provides the function settype(), which is used to change the type of a variable.

Syntax:

```
settype($variabletochange, 'new type');
```

Where:--

settype():-- it is a function

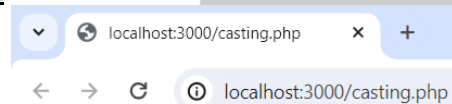
\$variabletochange:-- name of the variable being converted.

new type:-- Type of the variable to which you want to convert

Example

```
<?php
$a = 3.14;
echo "is $a a double?".is_double($a)."<br/>";
settype($a, 'string');
echo "is $a a string?".is_string($a)."<br/>";
settype($a, 'Integer');
echo "is $a a integer?".is_integer($a)."<br/>";
?>
```

Output



```
is 3.14 a double?1
is 3.14 a string?1
is 3 a integer?1
```

Interface BCA College, DVG (Site: www.iccadvg.org), Phone no: 8884768574

Unit III: - Using Functions, Class-Objects, Forms in PHP

Function

- A function is a self-contained block of code that can be called by your scripts.
- When called, the function's code is executed and performs a particular task.
- Functions are of two types
 - Built-in functions
 - User defined functions

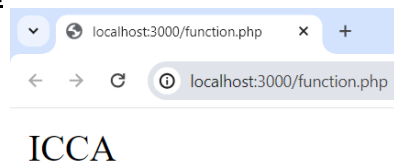
Built-in functions

- PHP has hundreds of built-in function.
- Example :- strtoupper(), abs(), count(), sizeof() etc...

Calling Built-in strtoupper() function

```
<?php
    $newsrtring= strtoupper("icca");
    print($newsrtring);
?>
```

Output



- This function calls the strtoupper() function, passing it the string "Hello Web!!".
- The function then goes about its business of changing the contents of the string to upper case letters.
- A function call consists of the function name followed by parentheses, between those parentheses we can pass the information.
- A piece of information passed to a function is called arguments.

User defined function

- We can define our own function using the function statement.

Defining a Function

Syntax:--

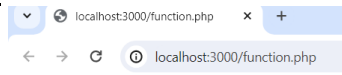
```
function some_function($argument1, $argument2)
{
    //function code here
}
```

Where,

function:- keyword
function_name : - name of the function
arg:- arguments

Declaring and calling a function

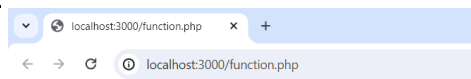
```
<?php
    function display()
    {
        echo "<h1> ICCA </h1>";
    }
    display();
?>
```

Output

ICCA

Function with argument

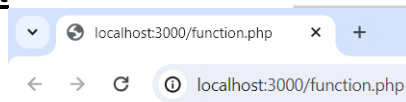
```
<?php
function display($txt)
{
    echo "<h1> $txt </h1>";
}
display("Welcome to ICCA");
?>
```

Output

Welcome to ICCA

A Function with an Optional Argument

```
<?php
function return_eg($a,$b = 5)
{
    return $a + $b;
}
echo return_eg(5);
?>
```

Output

Interface BCA College V. Computer Applications (ICCA)

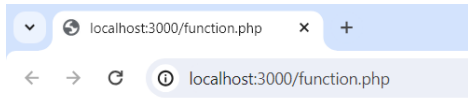
Returning Values from User-Defined Functions

- A function can return a value using the return statement in conjunction with a value.
- The return statement stops the execution of the function and sends the value back to the calling code.

Example:-

```
<?php
function return_eg($a,$b)
{
    return $a + $b;
}
echo return_eg(5,7);
?>
```

Output



12

Actual vs Formal Parameters:

- **Formal Parameters:** Formal parameters are the variables defined in the function declaration. They act as placeholders for the values that will be passed to the function when it is called.
- **Actual Parameters:** Actual parameters, also known as arguments, are the specific values that are supplied to function call.

Example

```
<?php
function return_eg($a,$b) //Formal parameters
{
    return $a + $b;
}
echo return_eg(5, 7); //Actual parameters
?>
```

Function and Variable Scope

- Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types
 - Local variables
 - Global variables

Local Variable

- A variable declared in a function is considered local; that is, it can be referenced solely in that function.
- Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function.

Example

```
<?php
$x = 4;
function assignx ()
{
    $x = 0;
    print "<h2>\$x inside function is $x. </h2>";
}
assignx();
print "<h1>\$x outside of function is $x. </h2>";
?>
```

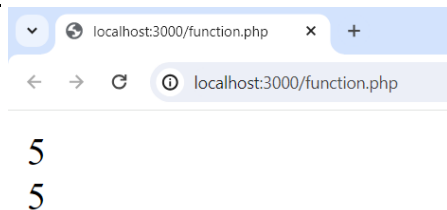
Output**\$x inside function is 0.****\$x outside of function is 4.**

Global Variable

- In PHP, variables declared outside of functions are not directly available within functions by default. These variables are considered local to the global scope and separate from the function's local scope.
- So, in order to access and modify the global variable within function, use the global keyword followed by the variable name.

Example

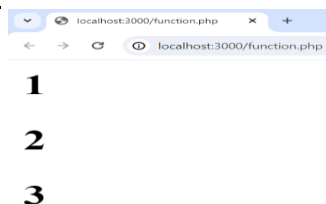
```
<?php
    $x=4;
    function eg()
    {
        global $x;
        $x++;
        echo "$x";
    }
    eg();
    echo "<br/> $x <br/>";
?>
```

Output**Saving State Between Function Calls with the static Statement**

- Local variables within functions have a short but happy life—they come into being when the function is called and die when execution is finished.
- If you declare a variable within a function in conjunction with the static statement, the variable remains local to the function, and the function “remembers” the value of the variable from execution to execution.

Example

```
<?php
    function keep_track()
    {
        STATIC $count = 0;
        $count++;
        print "<h2>$count</h2>";
    }
    keep_track();
    keep_track();
    keep_track();
?>
```

Output

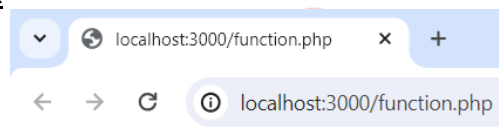
Recursion

- Recursion is a technique where a function calls itself directly or indirectly. This creates a loop where the function keeps executing until a certain condition is met, at which point it stops calling itself and the loop unwinds.

Example

```
<?php
function factorial($n) {
    if ($n < 0) {
        return -1; // Handle negative input (optional)
    } else if ($n == 0) {
        return 1; // Base case: factorial of 0 is 1
    } else {
        return $n * factorial($n - 1); // Recursive call
    }
}

$number = 5;
$result = factorial($number);
echo "$number factorial is: $result";
?>
```

Output

5 factorial is: 120

Library functions

- PHP offers a rich set of built-in library functions that cover various functionalities, saving you time and effort from writing code from scratch. Here's an overview of some key categories:

1. String Manipulation:

- `strlen()`: Get the length of a string.
- `strpos()`: Find the first occurrence of a substring within a string.
- `str_replace()`: Replace occurrences of a substring with another string.
- `strtolower()`: Convert a string to lowercase.
- `strtoupper()`: Convert a string to uppercase.
- `trim()`: Remove whitespace from the beginning and end of a string.

2. Arrays:

- `count()`: Get the number of elements in an array.
- `array_push()`: Add one or more elements to the end of an array.
- `array_pop()`: Remove and return the last element from an array.
- `array_merge()`: Merge two or more arrays.
- `in_array()`: Check if a value exists in an array.
- `array_key_exists()`: Check if a specific key exists in an array.

3. Numbers and Math:

- `abs()`: Get the absolute value of a number.
- `round()`: Round a number to a specified precision.
- `ceil()`: Round a number up to the nearest integer.

- floor(): Round a number down to the nearest integer.
- rand(): Generate a random integer.
- mt_rand(): Generate a better-quality random integer (using the Mersenne Twister algorithm).
- Math functions (e.g., sqrt(), pow(), sin(), cos())

4. Dates and Times:

- date(): Format a date according to a specified format string.
- time(): Get the current timestamp as a Unix timestamp.
- strtotime(): Convert a human-readable date/time string to a Unix timestamp.
- Functions for manipulating dates and times (e.g., mktime(), date_add(), date_diff())

Date and Time Functions

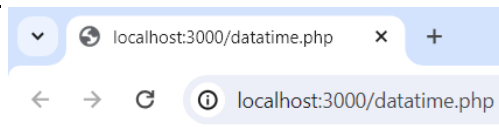
1. Getting the Date with time()

- PHP's time() function gives you all the information you need about the current date and time.
- It requires no arguments and returns an integer.

Example

```
<?php
echo time();
?>
```

Output



1710150937

// the output represents March 11, 2024 at 03:19PM.

- The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch
- And the number of seconds that have elapsed since then is referred to as a timestamp.

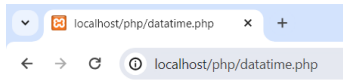
2. Converting a Timestamp with getdate()

- Now that you have a timestamp to work with, you must convert it before you present it to the user.
- getdate() optionally accepts a timestamp and returns an associative array containing information about the date.
- If you omit the timestamp, getdate() works with the current timestamp as returned by time().

Example

```
<?php
$a = getdate();
foreach($a as $key => $val)
{
    echo "<br/>". $key . "--". $val . "" ;
}
?>
```

Output

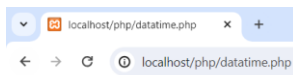


```
seconds--44
minutes--13
hours--10
mday--12
wday--2
mon--3
year--2024
yday--71
weekday--Tuesday
month--March
0--1710234824
```

Example 2

```
<?php
    $date_array = getdate(1710150937);
    foreach($date_array as $key => $value)
    {
        echo "<br/>". $key . "--". $value . " ";
    }
?>
```

Output



```
seconds--37
minutes--55
hours--10
mday--11
wday--1
mon--3
year--2024
yday--70
weekday--Monday
month--March
0--1710150937
```

Interface College of Computer Applications (ICCA)

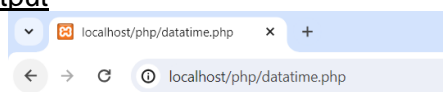
3. Creating Timestamps with mktime()

- mktime() returns a timestamp that you can then use with date() or getdate().
- mktime() accepts up to six integer arguments in the following order: Hour, Minute, Second, Month, Day of month, and Year.

Example

```
<?php
    $ts = mktime(14, 57, 0, 3, 12, 2024);
    echo $ts."<br>";
    echo date("m/d/y G:i:s e", $ts);
?>
```

Output



```
1710251820
03/12/24 14:57:00 Europe/Berlin
```

4. Converting a Timestamp with date()

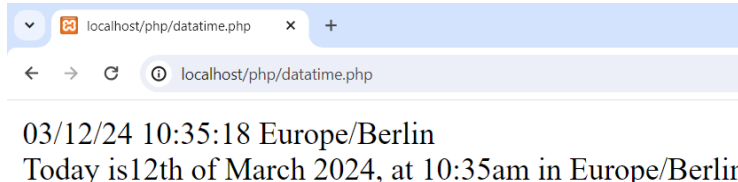
- The date() function returns a formatted string that represents a date.
- date() optionally accepts a timestamp.

| Format | Description | Example |
|--------|--|------------------------------------|
| a | am or pm (lowercase) | am |
| A | AM or PM (uppercase) | AM |
| d | Day of month (number with leading zeros) | 01 |
| D | Day of week (three letters) | Tue |
| e | Timezone identifier | America/New_York |
| F | Month name | January |
| h | Hour (12-hour format—leading zeros) | 09 |
| H | Hour (24-hour format—leading zeros) | 21 |
| g | Hour (12-hour format—no leading zeros) | 9 |
| G | Hour (24-hour format—no leading zeros) | 21 |
| i | Minutes | 21 |
| j | Day of the month (no leading zeros) | 17 |
| l | Day of the week (name) | Tuesday |
| L | Leap year (1 for yes, 0 for no) | 1 |
| m | Month of year (number—leading zeros) | 01 |
| M | Month of year (three letters) | Jan |
| n | Month of year (number—no leading zeros) | 1 |
| s | Seconds of hour | 11 |
| S | Ordinal suffix for the day of the month | th |
| r | Full date standardized to RFC 822 (http://www.faqs.org/rfcs/rfc822.html) | Tue, 17 Jan 2012 09:21:11 -0500 |
| U | Timestamp | 1326853271 |
| y | Year (two digits) | 12 |
| Y | Year (four digits) | 2012 |
| z | Day of year (0–365) | 17 |
| Z | Offset in seconds from GMT | -18000 |

Example

```
<?php
$time = time(); //stores the exact timestamp to use in this script
echo date("m/d/y G:i:s e", $time);
echo "<br/>";
echo "Today is";
echo date("jS \of F Y, \a\t g:ia \i\l\n e", $time);
?>
```

Output



```
03/12/24 10:35:18 Europe/Berlin
Today is12th of March 2024, at 10:35am in Europe/Berlin
```

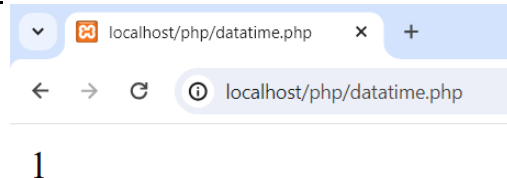
5. Testing a Date with checkdate()

- You might need to accept date information from user input. Before you work with a user-entered date or store it in a database, make sure that the date is valid.

- checkdate() accepts three integers: month, day, and year.
- checkdate() returns true if the month is between 1 and 12, the day is acceptable for the given month and year (accounting for leap years), and the year is between 0 and 32767.

Example

```
<?php
    echo checkdate(4, 32, 2089);
    echo checkdate(13, 32, 2089);
    echo checkdate(4, 12, 2089);
?>
```

OutputStrings in PHP

- There are four primary ways to create strings in PHP:
1. Single-quoted strings: This is the simplest and most common way. Enclose the text within single quotes ('). Any single quotes within the string must be escaped with a backslash (\).
- Example:-
- ```
$message = 'ICCA';
echo $message;
```
2. Double-quoted strings: Similar to single-quoted strings, but double quotes (") are used. However, double-quoted strings allow variable interpolation and interpretation of escape sequences.
- Example:-
- ```
$message = 'ICCA';
echo $message;
```
3. Heredoc: Heredoc is a way to create strings in PHP that span multiple lines and may contain variables without using string concatenation. It's particularly useful for creating large blocks of text or HTML markup within PHP code

Syntax

```
<?php
    $str = <<<IDENTIFIER
    Multiple lines of strings
    IDENTIFIER;
?>
```

- Three opening heredoc delimiters (<<<) followed by an identifier (chosen name without quotes) initiate the heredoc syntax.
- The actual string content goes on the next lines, indented at the same level (indentation is not strictly required, but improves readability).
- The string ends with the same identifier on a new line, but without any leading or trailing whitespace. This closes the heredoc block.
- Heredoc does interpret variable values within the string content when using double quotes or single quotes.

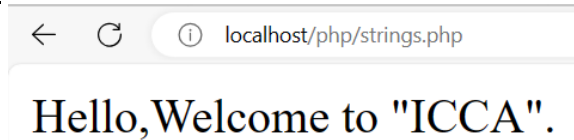
Example

```
<?php
```

```
$name = "ICCA";
$message = <<<MESSAGE
Hello,Welcome to "$name".
MESSAGE;
echo $message;
```

?>

Output



4. Nowdoc

- Nowdoc is similar to Heredoc but behaves like a single-quoted string. It does not parse variables or special characters within the string.

Syntax

```
<?php
    $str = <<<'IDENTIFIER'
    Multiple lines of strings
    IDENTIFIER;
```

?>

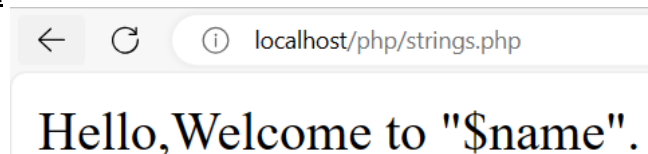
- Start with three less-than signs (<<<) followed by a single quote (').
- Follow it with an identifier (a word consisting of letters, numbers, and underscores). This identifier is unique within the script and acts as a marker for the nowdoc string.
- Type your multi-line string content. You can indent the content for better readability, but the indentation level must be consistent throughout the string.
- To terminate the nowdoc string, type the same identifier again on a new line, followed by a semicolon (;)
- Unlike heredoc, nowdoc strings don't allow variable interpolation using \${variable_name} syntax. Any variable names within the string will be treated literally.

Example

```
<?php
    $name = "ICCA";
    $message = <<<'MESSAGE'
    Hello,Welcome to "$name".
    MESSAGE;
    echo $message;
```

?>

Output



String Functions

There are number of built-in string functions some are explained below

1. strtoupper()

- To get an uppercase version of a string, use the strtoupper() function. This function requires only the string that you want to convert and returns the converted string:

Syntax:

strtoupper(string)

Example

```
<?php
$var = "interface college of computer applications";
echo "Before converting to uppercase:--".$var;
echo "<br/>After converting to uppercase:- ".strtoupper($var);
?>
```

Output



Before converting to uppercase:--interface college of computer applications
After converting to uppercase:- INTERFACE COLLEGE OF COMPUTER APPLICATIONS

2. strtolower

- To convert a string to lowercase characters, use the strtolower() function.

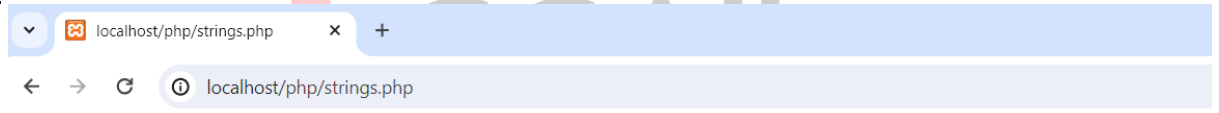
Syntax:

strtolower(string)

Example

```
<?php
$var = "INTERFACE COLLEGE OF COMPUTER APPLICATIONS ";
echo "Before converting to lowercase:--".$var;
echo "<br/>After converting to lowercase:- ".strtolower($var);
?>
```

Output



Before converting to lowercase:--INTERFACE COLLEGE OF COMPUTER APPLICATIONS
After converting to lowercase:- interface college of computer applications

3. ucwords

- The ucwords() function makes the first letter of every word in a string uppercase.

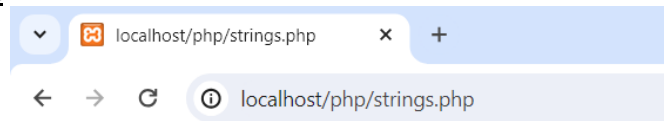
Syntax:

ucwords(string)

Example

```
<?php
$var = "interface college of computer application ";
echo $var;
echo "<br/>".ucwords($var);
?>
```

Output



interface college of computer application
Interface College Of Computer Application

4. ucfirst()

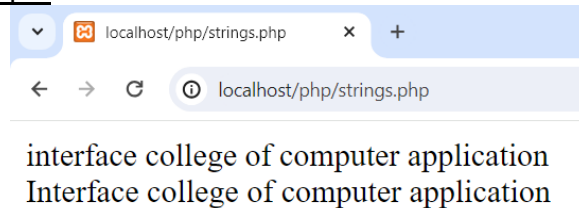
- The ucfirst() function capitalizes only the first letter in a string

Syntax:

```
ucfirst(string)
```

Example

```
<?php
$var = "interface college of computer application ";
echo $var;
echo "<br/>".ucfirst($var);
?>
```

Output5. trim()

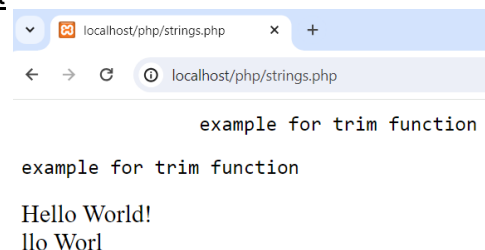
- The trim() function shaves any whitespace characters, including newlines, tabs, and spaces, from both the start and end of a string.
- It accepts the string to modify and returns the cleaned-up version.

Syntax

```
trim($string, $charlist)
```

Example

```
<?php
$text = "\t\texample for trim function";
echo "<pre>$text</pre>";
$text1 = trim($text);
echo "<pre>$text1</pre>";
$str = "Hello World!";
echo $str . "<br>";
echo trim($str, "Hed!");
?>
```

Output

Note: - The <pre> tag in HTML stands for "preformatted text." It's used to display text exactly as it appears in the HTML code, preserving spaces, line breaks, and other formatting.

6. ltrim()

PHP provides the ltrim() function to strip whitespace from the beginning of a string.

Example

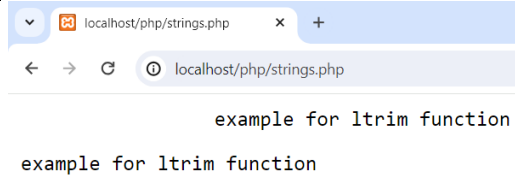
```
<?php
```



```
$text = "\t\t example for ltrim function ";
echo "<pre>$text<pre>";
$text = ltrim($text);
echo "<pre>$text</pre>";
```

?>

Output



7. rtrim()

- rtrim() removes whitespace only at the end of the string argument:

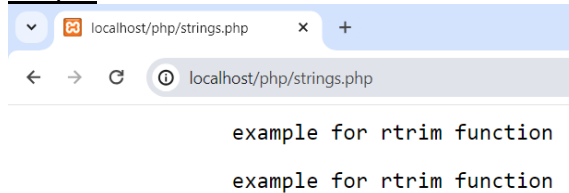
Example

<?php

```
$text = "\t\t example for rtrim function ";
echo "<pre>$text<pre>";
$text = rtrim($text);
echo "<pre>$text</pre>";
```

?>

Output



8. substr()

- Extracts a portion of the string based on starting index and length.

Syntax

substr(\$str, start, length)

Where,

- \$str: The original string from which you want to extract the substring.
- \$start: (Integer) The starting position of the substring within the original string. Indexing starts from 0, so the first character is at position 0.
- \$length (Optional, Integer): The length of the substring to extract. If omitted, the function extracts characters from the \$start position till the end of the string.

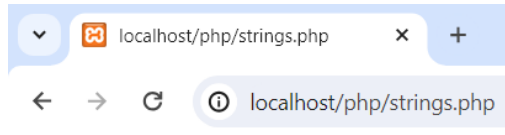
Example

<?php

```
$text = "INTERFACE COLLEGE OF COMPUTER APPLICATIONS";
$substr = substr($text,0,9);
$substr1 = substr($text,10,7);
echo $substr;
echo " $substr1";
```

?>

Output



INTERFACE COLLEGE

9. `str_repeat()`
Repeats a string a specified number of times.
Syntax

`str_repeat($str, multiplier)`

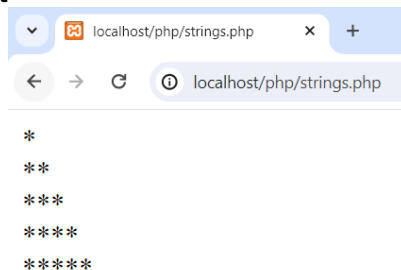
Where,

- `$str`: The string you want to repeat.
- `$multiplier`: (Integer) The number of times you want to repeat the string.

Example

```
<?php
$symbol = "*";
for($i=1;$i<=5;$i++)
{
    $asterisk_line = str_repeat($symbol,$i);
    echo $asterisk_line."<br/>";
}
?>
```

Output



10. `substr()`
- The `substr()` function returns a string based on the start index and length of the characters.
 - This function requires two arguments: a source string and the starting index.
 - Using these arguments, the function returns all the characters from the starting index to the end of the string.
 - You can also (optionally) provide a third argument—an integer representing the length of the string you want to return.
 - If this third argument is present, `substr()` returns only that number of characters, from the start index.

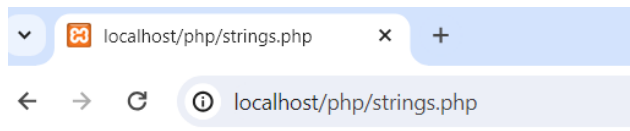
Syntax:

`substr(source string, starting index, [length];`

Example

```
<?php
$sentence = "Interface college of computer applications";
echo substr($sentence,0)."<br/>";
echo substr($sentence,3,2)."<br/>";
echo substr($sentence,-2);
?>
```

Output



Interface college of computer applications
er
ns

11. substr_replace()

- The `substr_replace()` function works similarly to the `substr()` function, except it enables you to replace the portion of the string that you extract.
- The function requires three arguments: the string to transform, the text to add to it, and the starting index; it also accepts an optional length argument.
- The `substr_replace()` function finds the portion of a string specified by the starting index and length arguments, replaces this portion with the string provided, and returns the entire transformed string.

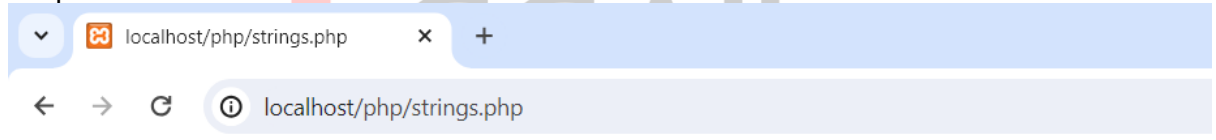
Syntax

`substr_replace(source string, replace string, index, [length])`

Example

```
<?php
    $sentence = "Interface college of computer applications";
    echo "text before replacing : ".$sentence."<br>";
    echo "text after replacing string : 
    ".substr_replace($sentence,"INTERFACE",0,9);
?>
```

Output



text before replacing : Interface college of computer applications
text after replacing string : INTERFACE college of computer applications

Interface College of Computer Applications (ICCA)

12. str_replace()

- The `str_replace()` function is used to replace all instances of a given string within another string.
- It requires three arguments: the search string, the replacement string, and the master string. The function returns the transformed string.

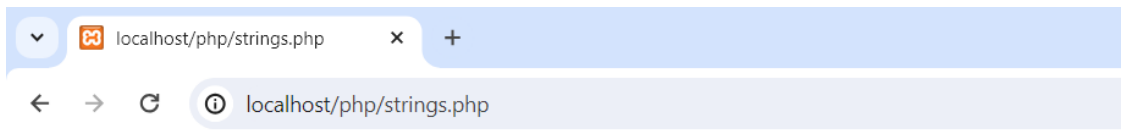
Syntax

`str_replace(search string, replace string, source string)`

Example

```
<?php
    $sentence = "Interface college!!,interface,Interface";
    echo "text before replacing : ".$sentence."<br>";
    echo "text after replacing string : 
    ".str_replace("Interface","INTERFACE",$sentence);
?>
```

Output



text before replacing : Interface college!!,interface,Interface
 text after replacing string : INTERFACE college!!,interface,INTERFACE

13. strcmp()

- The strcmp() function in PHP is used for string comparison. It takes two strings as arguments and returns an integer value indicating the relationship between them.

Syntax

strcmp(\$str1, \$str2)

Return values,

- 0: If the two strings are identical.
- Negative value (< 0): If the first string (\$str1) is less than the second string (\$str2). This happens when the first difference between corresponding characters in the strings has a lower ASCII value in \$str1 compared to \$str2.
- Positive value (> 0): If the first string (\$str1) is greater than the second string (\$str2). This occurs when the first difference between corresponding characters has a higher ASCII value in \$str1 compared to \$str2.

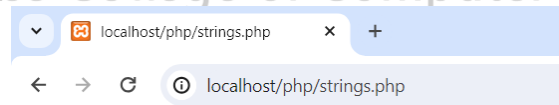
Example

```
<?php
$str1 = "apple";
$str2 = "banana";
$str3 = "Apple";

$result1 = strcmp($str1, $str2);
$result2 = strcmp($str1, $str3);
$result3 = strcmp($str3, $str1);

echo "apple compared to banana: $result1 <br/>";
echo "apple compared to Apple (case sensitive): $result2 <br/>";
echo "Apple compared to apple: $result3 <br/>";
?>
```

Output



apple compared to banana: -1
 apple compared to Apple (case sensitive): 1
 Apple compared to apple: -1

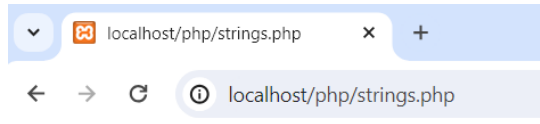
14. strlen()

- strlen(\$str): Returns the length of the string (number of characters).

Example

```
<?php
$text = "Interface College";
echo "Number of characters in a string=".strlen($text);
?>
```

Output



Number of characters in a string=17

15. str_word_count()

- Counts the number of words or returns an array of words based on the specified type.

Syntax

str_word_count(string \$string [, int \$format = 0 [, string \$charlist]])

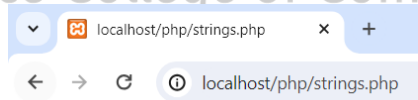
Where,

- \$string: The input string to be analyzed.
- \$format (optional): Specifies the return value. It can take three possible values:
 - 0: Returns the total number of words found in the string (integer).
 - 1: Returns an array containing all the individual words identified within the string.
 - 2: Returns an associative array where:
 - The key is the numeric position (0-based index) of the word within the original string.
 - The value is the actual word itself.
 - \$charlist (optional): A list of additional characters to consider as words.

Example

```
<?php
$sentence = "Interface college of computer applications !!!";
$total_words = str_word_count($sentence);
echo "Total words: $total_words <br/>";
$words_array = str_word_count($sentence, 1, ",!");
print_r($words_array);
echo "<br/>";
$words_with_positions = str_word_count($sentence, 2);
print_r($words_with_positions);
?>
```

Output



Total words: 5

Array ([0] => Interface [1] => college [2] => of [3] => computer [4] => applications [5] => !!!)

Array ([0] => Interface [10] => college [18] => of [21] => computer [30] => applications)

- \$total_words returns the total number of words found in the string.
- \$words_array returns an array containing all the words found in the string.
- \$words_with_position returns an associative array where the keys represent the positions of the words in the string, and the values are the words themselves.

Unit 4 :- Class and Objects in PHP

Class and Object

1. Class:

- Defines the blueprint or template for objects.
- Contains properties (also called attributes or fields) to store data.
- Contains methods (also called functions or procedures) to perform actions or operations related to the class.
- Provides encapsulation, allowing related properties and methods to be grouped together.
- Can be used to create multiple objects with similar characteristics.

2. Object:

- An instance of a class.
- Represents a specific entity or concept in the program.
- Possesses its own state, defined by the values of its properties.
- Exhibits behavior through its methods, which manipulate its state or interact with other objects.
- Each object created from a class is independent, with its own set of property values.

Creating and accessing a class and object

1. Define a class

- To define a class, specify the class keyword followed by a name

Syntax

```
class ClassName
{
    //code
}
```

Example

```
class BankAccount
{
    //code
}
```

2. Define a Object

- We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.
- Objects of a class are created using the new keyword.

Syntax

```
class ClassName
{
    //code
}
ObjectName = new ClassName();
```

Example

```
class BankAccount
{
    //code
}
acc1 = new BankAccount();
```

- To access a property, you use the object operator (->)
<?php
\$object->property;

Object Properties

- Properties are essentially variables associated with an object. They represent the data or state of the object.
- You define properties within a class definition.
- Access modifiers (public, private, protected) control access to properties from outside the class.

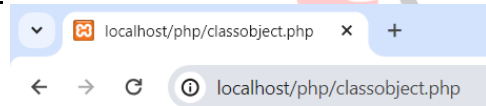
Types of Object Properties:

- **Public Properties:** These can be accessed and modified from anywhere in your code using the object's name and the arrow operator (->).
- **Private Properties:** Accessible only within the class definition itself (and extending classes) using the \$this keyword. This promotes encapsulation and data hiding.
- **Protected Properties:** Similar to private properties, but also accessible within child classes extending the parent class.

Example

```
<?php
class DavangereUniversity
{
    public $clgname;          // Object Properties
    public $course;
    public function display() // Object Methods
    {
        echo "I am studing $this->course in $this->clgname College";
    }
}
$college = new DavangereUniversity();
$college->clgname = "ICCA";
$college->course = "BCA";
$college->display();
```

Output



I am studing BCA in ICCA College

Interface College of Computer Applications (ICCA)

Assigning values to Object Properties

- In PHP, there are several ways to assign values to object properties.
1. Directly Inside the Class: Assign default values directly inside the class definition using property declarations


```
class DavangereUniversity
{
    public $clgname = "ICCA";
    public $course = "BCA";
}
```
 2. Directly Accessing Properties: We can also directly assign values to properties after the object has been created


```
class DavangereUniversity
{
    public $clgname;
    public $course;
```

```

}
$college = new DavangereUniversity();
$college->clgname = "ICCA";
$college->course = "BCA";

```

3. Using Setter Methods: We can define setter methods within the class to assign values to properties after the object has been created:

```

class DavangereUniversity
{
    public $clgname;
    public $course;
    public function setclgname($clgname)
    {
        $this->clgname = $clgname;
    }
    public function setcourse($course)
    {
        $this->course = $course;
    }
}
$college = new DavangereUniversity();
$college->setclgname("ICCA");
$college->setcourse("BCA");

```

4. Inside the Constructor: We can assign values to properties when creating an object by passing them as arguments to the constructor:

```

class DavangereUniversity
{
    public $clgname;
    public $course;
    public function __construct($clgname, $course)
    {
        $this->clgname = $clgname;
        $this->course = $course;
    }
}
$college = new DavangereUniversity("ICCA","BCA");

```

Object Methods

- Methods are functions defined within a class that represent the actions or behaviors an object can perform.
- They can access and manipulate the object's properties, as well as interact with other objects.
- Similar to properties, methods can also have access modifiers (public, private, protected).

Example

```

<?php
class DavangereUniversity
{
    public $clgname;           // Object Properties
    public $course;
    public function display()  // Object Methods
    {
        echo "I am studying $this->course in $this->clgname College";
    }
}

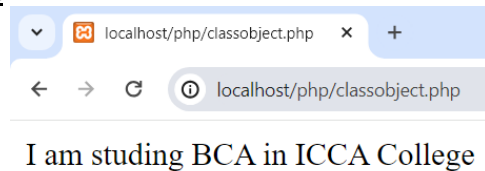
```



```

    }
}
$college = new DavangereUniversity();
$college->clgname = "ICCA";
$college->course = "BCA";
$college->display();

```

Output1. Accessing a Property from Within a Method

- In PHP, we can access properties from within a method of the same class using the \$this keyword.

Example :- Refer above example

2. Changing the Value of a Property from Within a Method

- In PHP, we can change the value of a property within a method using the \$this keyword and assigning a new value to the current object's property

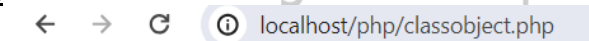
Example

```

<?php
// Changing the Value of a Property from Within a Method
class College
{
    public $clgname = "ICCA";

    function display($clg)
    {
        $this->clgname = $clg;
        echo "Welcome to, $this->clgname !!";
    }
}
$obj1 = new College();
$obj1->display("Interface College");

```

Output

Welcome to, Interface College !!

Overloading

- Unlike some other object-oriented languages, PHP doesn't support overloading in the traditional sense.
- Traditional overloading refers to having multiple methods with the same name but different argument lists.
- However, PHP offers a mechanism to achieve similar behavior using magic methods. These are special methods that allow you to define custom logic for how PHP interacts with object properties and methods that aren't explicitly declared within the class.

Property Overloading

- Property overloading in PHP refers to the ability to create dynamic properties for an object at runtime.
- PHP utilizes special methods prefixed and suffixed with double underscores (__) for property overloading.
- The key methods involved are:
 - `__get($propertyName)`: This method is invoked whenever you try to access a property that isn't explicitly declared within the class. You can define custom logic here to handle property retrieval or perform actions based on the accessed property.
 - `__set($propertyName, $value)`: This method is called when you attempt to set a value to a non-existent property. It allows you to define how the object should handle setting properties that aren't formally defined.

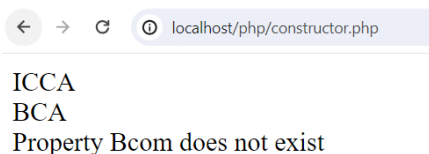
Example

```
<?php
class MyClass {
    private $data = [];

    public function __get($name)
    {
        if (array_key_exists($name, $this->data))
        {
            return $this->data[$name];
        }
        else
        {
            // Handle undefined property access
            echo "Property $name does not exist\n";
        }
    }
    public function __set($name, $value)
    {
        $this->data[$name] = $value;
    }
}

$obj = new MyClass();
// Setting properties dynamically
$obj->clname = 'ICCA';
$obj->course = 'BCA';
// Getting properties dynamically
echo $obj->clname . "<br>";
echo $obj->course . "<br>";
//Accessing undefined property
echo $obj->Bcom . "<br>";
?>
```

Output



```
localhost/php/constructor.php

ICCA
BCA
Property Bcom does not exist
```

Method Overloading

- In PHP, method overloading allows you to dynamically create methods that are not explicitly declared within a class.
- These dynamic methods are processed using magic methods
- Here's how method overloading works in PHP:
 - Using `__call()` and `__callStatic()`:
 - When an inaccessible or undefined method is called within a class instance or statically, PHP automatically invokes the magic methods `__call()` and `__callStatic()`.
 - These methods allow you to handle method calls that haven't been explicitly defined in your class.

Example

```
<?php
class example
{
    public function __call($methodName, $arguments)
    {
        $method = [$this, $methodName.count($arguments)];
        if (is_callable($method))
        {
            return call_user_func_array($method, $arguments);
        }
    }
    private function multiply1($argument1)
    {
        echo $argument1;
    }
    private function multiply2($argument1, $argument2)
    {
        echo $argument1 * $argument2;
    }
}
$class = new example;
echo $class->multiply(2). "<br>";
$class->multiply(5, 7);
?>
```

Output

← → ↻ ⓘ localhost/php/constructor.php

2
35

Explanation of above program

- **__call Magic Method:**
 - This is a special method that gets invoked automatically whenever you try to call a method on the object that doesn't exist explicitly within the class.
 - It takes two arguments:
 - \$methodName: The name of the method being called (e.g., "multiply" in this case).
 - \$arguments: An array containing the arguments passed to the method call.
- **Creating a Dynamic Method Name:**
 - Inside `__call`, the code constructs a new method name dynamically.

- It concatenates the original method name (\$methodName) with the number of arguments (count(\$arguments)) provided during the call.
- For example, if you call \$class->multiply(2), the constructed method name would be multiply1.
- **Checking Callability:**
 - The code uses is_callable(\$method) to check if the dynamically constructed method name (\$method) actually exists as a method within the class and can be called.
- **Calling the Method:**
 - If the method is callable (exists and can be called), the code uses call_user_func_array(\$method, \$arguments).
 - \$method: This is the dynamically constructed method name (e.g., multiply1).
 - \$arguments: This is the original array of arguments passed during the method call.
 - This essentially calls the appropriate private method (multiply1 or multiply2) based on the number of arguments provided.

Inheritance

- Inheritance in PHP is a fundamental concept in object-oriented programming (OOP) that allows classes to inherit properties and methods from other classes.
- Inheritance is implemented using the extends keyword in PHP, where a child class extends a parent class.
- The child class inherits all the non-private properties and methods defined in the parent class, enabling it to utilize and build upon the functionality provided by the parent class.
- By inheriting from a parent class, the child class gains access to the public and protected properties and methods defined in the parent class.
- It can use them as if they were defined within the child class itself.
- This promotes code reusability, as common functionality can be encapsulated in a base class and inherited by multiple derived classes.
- In addition to inheriting properties and methods, the child class has the flexibility to override or extend inherited methods by providing its implementation.
- This allows the child class to customize the behavior inherited from the parent class and adapt it to its specific needs.

Syntax for Inheriting a Class in PHP

```
class ChildClass extends ParentClass
{
    // Class definition
}
```

Where.

- ChildClass is the name of the child class that is going to inherit from the ParentClass.
- ParentClass is the name of the parent class that contains the properties and methods to be inherited.

Types of Inheritance in PHP

- Inheritance is a core concept in object-oriented programming (OOP) that allows classes to inherit properties and methods from other classes.
- In PHP, there are several types of inheritance that developers can use to create class hierarchies and achieve code reusability.

1. Single Inheritance:

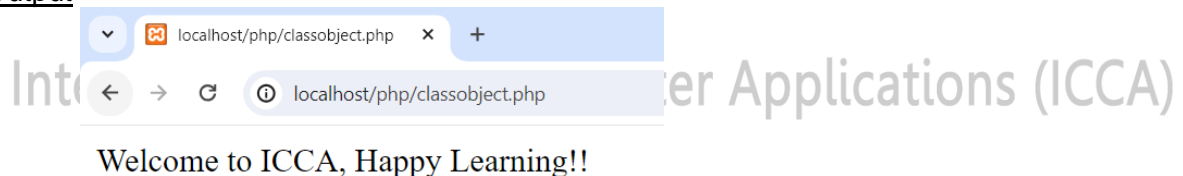
- Single inheritance refers to a scenario where a class inherits from a single parent class.
- In PHP, single inheritance is the most commonly used type of inheritance.
- It allows a class to inherit the properties and methods of one parent class, also known as the superclass or base class.

Syntax:

```
class ParentClass
{
    //code;
}
class ChildClass extends ParentClass
{
    // code;
}
```

Example

```
<?php
class College
{
    function display()
    {
        echo "Welcome to ICCA";
    }
}
class ChildClass extends College
{
    function display1()
    {
        echo ", Happy Learning!!";
    }
}
$obj1 = new ChildClass();
$obj1->display();
$obj1-> display1();
```

Output**2. Multilevel Inheritance:**

- Multilevel inheritance involves a class inheriting from a parent class, and that parent class itself inheriting from another parent class.
- This forms a hierarchy or chain of classes.
- In PHP, multilevel inheritance allows classes to inherit properties and methods from multiple levels up the class hierarchy.

Syntax

```
class ParentClass
{
    //code;
}
class ChildClass extends ParentClass
{
    //code;
```

```

}
class GrandChild extends ChildClass
{
    //code;
}

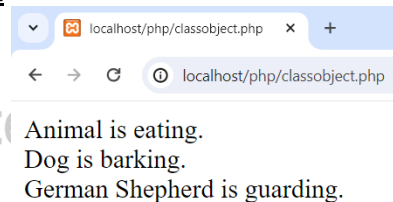
```

Example

```

class Animal
{
    public function eat()
    {
        echo "Animal is eating.<br>";
    }
}
class Dog extends Animal
{
    public function bark()
    {
        echo "Dog is barking.<br>";
    }
}
class GermanShepherd extends Dog
{
    public function guard()
    {
        echo "German Shepherd is guarding.<br>";
    }
}
$germanShepherd = new GermanShepherd();
$germanShepherd->eat();
$germanShepherd->bark();
$germanShepherd->guard();

```

Output


Animal is eating.
Dog is barking.
German Shepherd is guarding.

3. Hierarchical Inheritance

- In hierarchical inheritance, more than one child class is inherited from the single parent class. The hierarchy forms a tree-like structure.

Syntax

```

class ParentClass
{
    //code;
}
class ChildClass1 extends ParentClass
{
    //code;
}
class ChildClass2 extends ParentClass
{

```

```

    //code;
}

```

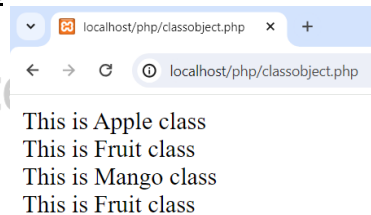
Example

```

<?php
class Fruit
{
    public function fruit()
    {
        echo "This is Fruit class<br>";
    }
}
class Apple extends Fruit
{
    public function apple()
    {
        echo "This is Apple class<br>";
        echo "".$this->fruit();
    }
}
class Mango extends Fruit
{
    public function mango()
    {
        echo "This is Mango class<br>";
        echo "".$this->fruit();
    }
}
$a = new Apple();
$a->apple();
$m = new Mango();
$m->mango();
?>

```

Output



```

This is Apple class
This is Fruit class
This is Mango class
This is Fruit class

```

4. Multiple Inheritance

- In multiple inheritance, there can be more than one parent class for a single child class.
- Multiple inheritance is not supported in PHP, but we can achieve this by using interfaces or traits.

a. By Using Traits Along with Classes:

- A trait in PHP is similar to a class, except that traits contain methods that can be used in multiple classes.
- This helps in achieving multiple inheritance in PHP.
- To declare a trait, we use the keyword “trait,” and to use it in the class, we have the keyword “use.”

- Traits are similar to classes, but they cannot be instantiated on their own. Instead, they are intended to be used by classes to incorporate their methods.

The syntax for declaring a trait in PHP is as follows:

```
<?php
trait TraitName
{
    // required code
}
?>
```

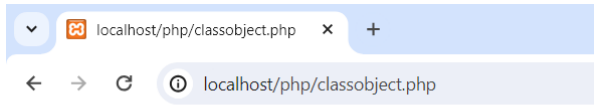
To use a trait in a class, the syntax is as follows:

```
<?php
class MyClass
{
    use TraitName;
}
?>
```

Example

```
<?php
// trait 1
trait t1
{
    public function sayhello()
    {
        echo "Hello! Welcome to";
    }
}
// trait 2
trait t2
{
    public function sayfor()
    {
        echo " ICCA.";
    }
}
class Child
{
    //Include trait in class
    use t1;
    use t2;
    public function fun()
    {
        echo "\nHappy Learning!";
    }
}
$obj = new Child();
$obj->sayhello();
$obj->sayfor();
$obj->fun();
?>
```

Output



Hello! Welcome to ICCA. Happy Learning!

b. By using Interfaces along with Classes

- An interface contains public methods that the class must implement.
- Moreover, interfaces only specify the method signature, i.e., the name of the method and argument list, without the actual implementation.

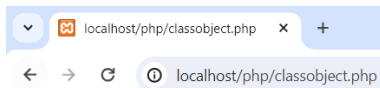
The syntax of an interface in PHP is as follows:

class child_class_name implements interface_name1, interface_name2...

Example

```
<?php
interface I1
{
    public function insideI1();
}
interface I2
{
    public function insideI2();
}
class Child implements I1, I2
{
    function insideI2()                // Function of the interface B
    {
        echo "Inside interface I2<br>";
    }
    function insideI1()                // Function of the interface C
    {
        echo "Inside interface I1<br>";
    }
    public function insideChild()
    {
        echo "\nInside Child class";
    }
}
$obj = new Child();
$obj->insideI1();
$obj->insideI2();
$obj->insideChild();
?>
```

Output



Inside interface I1
Inside interface I2
Inside Child class

Overriding

- Overriding refers to the ability to define a method in a child class that is already defined in the parent class.

- This allows the child class to provide its own implementation of the method, potentially with different behavior or functionality, while still maintaining the same method signature (name and parameters) as the parent class.

Example

```
<?php
class ParentClass
{
    public function sayHello()
    {
        echo "Hello from ParentClass!<br>";
    }
}
class ChildClass extends ParentClass
{
    public function sayHello()
    {
        echo "Hello from ChildClass!<br>";
    }
}
$parent = new ParentClass();
$child = new ChildClass();
$parent->sayHello();
$child->sayHello();
?>
```

Output



```

Hello from ParentClass!
Hello from ChildClass!
```

- In this example, both ParentClass and ChildClass have a method named sayHello(). When you call sayHello() on an instance of ParentClass, it executes the method defined in ParentClass. However, when you call sayHello() on an instance of ChildClass, it executes the method defined in ChildClass, overriding the behavior of the parent class method.

Constructor and destructor

Constructor

- A constructor in PHP is a special method within a class that gets automatically executed when an object of the class is created.
- It allows initialization tasks, such as setting default property values or performing necessary setups, to be performed seamlessly during object instantiation.

Types of Constructor

1. Default Constructor:

- This is the most basic type and is automatically provided by PHP if you don't define any constructor in your class.
- It has no arguments (public function __construct() {}) and performs no specific initialization tasks.
- While it works, it's often recommended to define a custom constructor to ensure proper object initialization.

Example 1

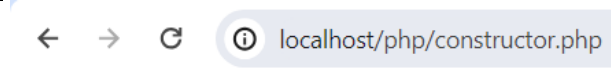
```
<?php
class College
{

}
$obj1 = new College();
?>
```

Example 2

```
<?php
class College
{
    public function __construct()
    {
        echo "Welcome to ICCA";
    }
}
$obj1 = new College();
?>
```

Output



Welcome to ICCA

Note:- In older versions of PHP (before PHP 5), the way classes were constructed differed from the more modern approach shown in the previous code example. In the old-style constructors, you would define a method with the same name as the class itself, and this method would serve as the constructor.

2. Parameterized Constructor:

- This is the most commonly used type. It's explicitly defined within a class using the `__construct` method and accepts arguments (parameters).
- These arguments allow you to pass values during object creation, which are then used to set the object's properties or perform other initialization tasks.
- Parameterized constructors provide more control over the initial state of your objects.

Example

```
<?php
class College
{
    public $clname;
    public function __construct($clname)
    {
        $this->clname = $clname;
        echo "Welcome to $this->clname ";
    }
    public function display()
    {
        echo "<br>Welcome to $this->clname ";
    }
}
$obj1 = new College("ICCA");
$obj1->display();
```

?>
Output

localhost/php/constructor.php

Welcome to ICCA
Welcome to ICCA

3. Copy Constructor

- PHP doesn't have built-in support for copy constructors, which are used to create a new object as a copy of an existing one.
- However, you can achieve similar functionality using a few different approaches:

i. Cloning:

- The clone keyword is the most common way to create a copy of an object in PHP. It performs a shallow copy, meaning it copies the values of the object's properties but references remain references. This can be problematic if the object holds references to other objects.

Example

```
<?php
class MyClass
{
    private $value;
    public function __construct($value)
    {
        $this->value = $value;
    }
    // Method to get the value
    public function getValue()
    {
        return $this->value;
    }
}
class CopyClass
{
    // Method to create a copy of MyClass instance
    public static function copyInstance(MyClass $original) {
        return clone $original;
    }
}
// Creating an instance of MyClass
$instance1 = new MyClass(10);
// Creating a copy using the CopyClass method
$instance2 = CopyClass::copyInstance($instance1);
// Checking the values of both instances
echo "Instance 1 Value: " . $instance1->getValue() . "<br>";
echo "Instance 2 Value: " . $instance2->getValue() . "<br>";
```

Output

localhost/php/copy_constructor.php

Instance 1 Value: 10
Instance 2 Value: 10

ii. Named Constructor (Factory Method):

- Instead of a dedicated copy constructor, you can define a static method that takes an existing object as input and returns a new object with the same properties. This approach offers more flexibility and can be used for various object creation scenarios.

Example

```
<?php
class MyClass
{
    private $value;
    public function __construct($value)
    {
        $this->value = $value;
    }
    // Method to get the value
    public function getValue()
    {
        return $this->value;
    }
}

class CopyClass
{
    // Method to create a copy of MyClass instance
    public static function copyInstance(MyClass $original)
    {
        return new MyClass($original->getValue());
    }
}

// Creating an instance of MyClass
$instance1 = new MyClass(10);
// Creating a copy using the CopyClass method
$instance2 = CopyClass::copyInstance($instance1);
// Checking the values of both instances
echo "Instance 1 Value: " . $instance1->getValue() . "<br>";
echo "Instance 2 Value: " . $instance2->getValue() . "<br>";

?>
```

- MyClass is the original class, and CopyClass contains the method copyInstance() which is responsible for creating a copy of MyClass instances.

Output

localhost/php/copy_constructor.php

```
Instance 1 Value: 10
Instance 2 Value: 10
```

Constructor in inheritance

- Unlike methods, a child class constructor doesn't automatically call the parent class constructor.
- If the child class defines its own constructor, it needs to explicitly call the parent constructor using parent::__construct(\$arguments) syntax within its own constructor body.
- This ensures the parent class's initialization logic is executed before the child class's specific initialization.

Example

```
<?php
class ParentClass
{
```

```

    public function __construct()
    {
        echo "Parent constructor called.<br>";
    }
}
class ChildClass extends ParentClass
{
    public function __construct()
    {
        // Call parent class constructor explicitly
        parent::__construct();
        echo "Child constructor called.\n";
    }
}
// Instantiate ChildClass
$obj = new ChildClass();
?>

```

Output

← → ↺ ⓘ localhost/php/constructor.php

Parent constructor called.
Child constructor called.

Destructor

- In PHP, a destructor is a special method that gets called when an object is destroyed or when the script finishes execution. The destructor method is defined using the `__destruct()` magic method..

Example

```

<?php
class MyClass
{
    public function __construct()
    {
        echo "Constructor called.<br>";
    }
    public function __destruct()
    {
        echo "Destructor called.<br>";
    }
}
// Create an instance of MyClass
$obj = new MyClass();
// Object is destroyed when script finishes execution
echo "Script finished.<br>";
?>

```

Output

← → ↺ ⓘ localhost/php/constructor.php

Constructor called.
Script finished.
Destructor called.

Form Handling:

Creating HTML Form

- HTML forms are essential elements for creating interactive web pages. They allow users to submit information to a web server for processing.
- Here's a detailed explanation of HTML form creation:

1. The <form> Tag:

- The foundation of an HTML form is the <form> tag.
- It defines the start and end of the form element.

Syntax:

```
<form action="your_script.php" method="post">
</form>
```

Attributes of the <form> Tag:

- action: This attribute specifies the URL of the PHP script that will process the form data upon submission. If omitted, the form data is submitted to the same page (itself).
- method: This attribute defines the HTTP method used to send the form data to the server. The two common methods are:
 - GET: Appends the form data to the URL as key-value pairs (visible in the address bar). Use it for non-sensitive data like search queries.
 - POST: Sends the form data as a separate entity from the URL, offering more security for sensitive data like passwords.
- Other attributes like id, class, and name can be used for styling and scripting purposes.

2. Form Elements:

- Within the <form> tag, you place various form elements to collect user input. Here are some common elements:
 - <input>: This versatile element comes with various subtypes for different input types:
 - text: Single-line text input (e.g., name, email).
 - password: Hidden character password input.
 - radio: Radio buttons for mutually exclusive choices (e.g., gender).
 - checkbox: Checkboxes for multiple selections (e.g., interests).
 - submit: Creates a submit button to trigger form submission.
 - Many more subtypes exist for specific data types like dates, numbers, URLs, etc.
 - <textarea>: Creates a multi-line text input field for longer messages or descriptions.
 - <select>: Defines a dropdown menu for selecting options from a list.

3. Essential Attributes for Form Elements:

- name: This attribute is crucial for each form element. It assigns a unique identifier to the element's submitted data. The server-side script (e.g., PHP) uses this name to access the corresponding data.
- type: This attribute specifies the type of input for the element (as discussed with <input> subtypes).
- value: This attribute sets the default value displayed in the element (e.g., pre-filling text boxes or selecting options).
- required: This attribute ensures a field must be filled before submitting the form.

Example

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="UTF-8">
  <title>GET Form Example</title>
</head>
<body>
  <h1>Enter your name and age</h1>
  <form action="partB2.php" method="post">

    <table>
      <tr><td><label for="name">Name:</label></td>
      <td><input type="text" id="name" name="name" required><br></td></tr>
      <tr><td><label for="age">Age:</label></td>
      <td><input type="number" id="age" name="age" min="1" required></td></tr>
    </table>
    <input type="submit" value="Submit">
  </form>
</body>
</html>

```

Handling HTML Form data in PHP

- In PHP, handling data submitted through HTML forms involves two main aspects:

1. Accessing Submitted Data:

- Superglobals: PHP provides superglobal variables that are accessible throughout your script regardless of scope. The two primary ones for form handling are:
 - `$_GET`: Used for forms submitted with the `method="GET"` attribute. It stores key-value pairs where the key is the element's name attribute and the value is the submitted data. The data becomes part of the URL (visible in the address bar).
 - `$_POST`: Used for forms submitted with the `method="POST"` attribute. It also stores key-value pairs but offers a more secure way to handle sensitive data like passwords since the data isn't appended to the URL.

2. Processing and Validation:

- Once you have access to the form data using `$_GET` or `$_POST`, you can perform various actions:
 - Displaying Data: Echo or print the data for confirmation (e.g., "Hello, \$name! Thanks for contacting us.").
 - Database Storage: Use libraries like PDO or mysqli to connect to a database and store the data in a secure manner.
 - Calculations or Operations: Perform calculations or operations based on the submitted values (e.g., processing an order or sending an email).
 - Validation: It's essential to validate the submitted data to ensure its accuracy and security. Here are some common validation techniques:
 - Required Fields: Check if required fields are filled using if `(empty($_POST['name']))` or similar logic.
 - Data Types: Validate if the data matches the expected type (e.g., checking if an email is a valid email address).
 - Length or Range: Ensure the data falls within a specific length or range (e.g., minimum password length).
 - Server-side Validation: Always validate on the server-side (PHP) even if you have client-side (JavaScript) validation for better security.

Example eg.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>GET Form Example</title>
</head>
<body>
  <h1>Enter your name and age</h1>
  <form action="eg.php" method="post">

    <table>
      <tr><td><label for="name">Name:</label></td>
      <td><input type="text" id="name" name="name" required><br></td></tr>
      <tr><td><label for="age">Age:</label></td>
      <td><input type="number" id="age" name="age" min="1" required></td></tr>
    </table>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

eg.php

```
<?php
if (isset($_POST['name']))
{
  // Access form data
  $name = $_POST["name"];
  $age = $_POST["age"];
}

// Simple validation (check for non-alphanumeric characters)
if (!preg_match('/^[a-zA-Z .]+$/', $name)) {
  echo "Invalid name! Please enter only letters.";
  exit; // Stop script execution
}
echo "Hello, <b>$name</b> You are <b>$age</b> years old.";
?>
```

Output

Enter your name and age

Name:

Age:

Hello, **Seetha** You are **29** years old.

Unit 5: - Database Handling Using PHP with MySQL

Introduction to MySQL :

Database terms

- **Data:** - Data is a collection of a distinct small unit of information. It can be used in a variety of forms like text, numbers, media, bytes, etc. it can be stored in pieces of paper or electronic memory, etc.
- **Database (DB):** A collection of structured data organized for easy access, storage, and manipulation.
- **Database Management System (DBMS):** Software that allows users to interact with databases, including creating, managing, and querying data. Popular DBMS examples include MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, and SQLite.
- **Schema:** The overall structure of a database, defining tables, columns, data types, keys, and relationships between them.
- **Table:** A fundamental unit of organization within a database, similar to a spreadsheet with rows and columns. Each table stores data about a specific subject or entity (e.g., customers, products, orders).
- **Row (Record):** A single horizontal entry in a table, representing a complete instance of an entity (e.g., a customer record).
- **Column (Field):** A vertical section within a table that holds a specific type of data for each row (e.g., customer name, email address).
- **Data Type:** Defines the kind of data a column can store (e.g., text, numbers, dates, booleans).
- **SQL (Structured Query Language):** A standardized language used to interact with relational databases. It allows you to perform tasks like creating tables, inserting data, querying data, and updating data.
- **Query:** A specific request made to a database to retrieve or manipulate data. SQL queries use SELECT, INSERT, UPDATE, and DELETE statements for various operations.
- **Join:** A way to combine data from multiple tables based on a shared relationship, allowing you to retrieve data from connected tables in a single query.
- **Primary Key:** A unique identifier for each row in a table, ensuring no duplicate records exist. A table can only have one primary key.
- **Foreign Key:** A column in one table that references the primary key of another table, establishing a link between related data.
- **Data Integrity:** Maintaining the accuracy and consistency of data within a database. Constraints (rules) can be applied to enforce data integrity, such as requiring specific data types or preventing invalid entries.

Data Types

- In programming languages data types specifies the size and type of the value, which an variable can hold.
- In DBMS data types specifies the size and type of domain values which for a attribute's in table or relation.
- Choosing correct data types results in proper computer memory utilization and minimize the wastage of computer memory.
- MySQL uses many different data types, broken into three categories
 - Numeric
 - Date and time
 - String types

Numeric Data Types

- MySQL uses all the standard ANSI SQL numeric data types.
 - The following list shows the common numeric data types and their descriptions.
1. INT
 - A normal-sized integer that can be signed or unsigned.
 - If signed, the allowable range is from –2147483648 to 2147483647.
 - If unsigned, the allowable range is from 0 to 4294967295.
 - You can specify a width of up to 11 digits.
 2. TINYINT
 - A small integer that can be signed or unsigned.
 - If signed, the allowable range is from –128 to 127. If unsigned, the allowable range is from 0 to 255.
 - You can specify a width of up to 4 digits.
 3. SMALLINT
 - A small integer that can be signed or unsigned.
 - If signed, the allowable range is from 32768 to 32767.
 - If unsigned, the allowable range is from 0 to 65535.
 - You can specify a width of up to 5 digits.
 4. MEDIUMINT
 - A medium-sized integer that can be signed or unsigned.
 - If signed, the allowable range is from –8388608 to 8388607.
 - If unsigned, the allowable range is from 0 to 16777215.
 - You can specify a width of up to 9 digits.
 5. BIGINT
 - A large integer that can be signed or unsigned.
 - If signed, the allowable range is from –9223372036854775808 to 9223372036854775807.
 - If unsigned, the allowable range is from 0 to 18446744073709551615.
 - You can specify a width of up to 11 digits.
 6. FLOAT(M,D)
 - A floating-point number that cannot be unsigned.
 - You can define the display length (M) and the number of decimals (D).
 - This is not required and defaults to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals).
 - Decimal precision can go to 24 places for a FLOAT.
 7. DOUBLE(M,D)
 - A double-precision floating-point number that cannot be unsigned.
 - You can define the display length (M) and the number of decimals (D).
 - This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE.
 - REAL is a synonym for DOUBLE.
 8. DECIMAL(M,D)
 - An unpacked floating-point number that cannot be unsigned.
 - In unpacked decimals, each decimal corresponds to 1 byte.

- Defining the display length (M) and the number of decimals (D) is required.
- NUMERIC is a synonym for DECIMAL.

Date and Time Types

- The MySQL date and time data types are as follows:
1. DATE
 - A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31.
 - For example, December 30, 1973, is stored as 1973-12-30.
 2. DATETIME
 - A date and time combination in YYYY-MM-DD HH:MM:SS format, between 1000-01-01 00:00:00 and 9999-12-31 23:59:59.
 - For example, 3:30 in the afternoon on December 30, 1973, is stored as 1973-12-30 15:30:00.
 3. TIMESTAMP
 - A timestamp between midnight, January 1, 1970, and sometime in 2037.
 - You can define multiple lengths to the TIMESTAMP field, which directly correlates to what is stored in it.
 - The default length for TIMESTAMP is 14, which stores YYYYMMDDHHMMSS.
 - This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30, 1973, is stored as 19731230153000.
 - Other definitions of TIMESTAMP are 12 (YYMMDDHHMMSS), 8 (YYYYMMDD), and 6 (YYMMDD).
 - TIME Stores the time in HH:MM:SS format. data type.
 4. YEAR(M)
 - Stores a year in two-digit or four-digit format.
 - If the length is specified as 2 (for example, YEAR(2)), YEAR can be 1970 to 2069 (70 to 69). If the length is specified as 4, YEAR can be 1901 to 2155.
 - The default length is 4.
 - You will likely use DATETIME or DATE more often than any other date- or time-related data type

String Types

- Although numeric and date types are fun, most data you'll store will be in string format.
 - This list describes the common string data types in MySQL
1. CHAR(M)
 - A fixed-length string between 1 and 255 characters in length (for example, CHAR(5)), right-padded with spaces to the specified length when stored.
 - Defining a length is not required, but the default is 1.
 2. VARCHAR(M)
 - A variable-length string between 1 and 255 characters in length;
 - For example, VARCHAR(25).
 - You must define a length when creating a VARCHAR field.
 3. BLOB or TEXT
 - A field with a maximum length of 65,535 characters.

- BLOBs are Binary Large Objects and are used to store large amounts of binary data, such as images or other types of files.
 - Fields defined as TEXT also hold large amounts of data; the difference between the two is that sorts and comparisons on stored data are case sensitive on BLOBs and are not case sensitive in TEXT fields.
 - You do not specify a length with BLOB or TEXT.
4. TINYBLOB or TINYTEXT
 - A BLOB or TEXT column with a maximum length of 255 characters.
 - You do not specify a length with TINYBLOB or TINYTEXT.
 5. MEDIUMBLOB or MEDIUMTEXT
 - A BLOB or TEXT column with a maximum length of 16,777,215 characters.
 - You do not specify a length with MEDIUMBLOB or MEDIUMTEXT.
 6. LONGBLOB or LONGTEXT
 - A BLOB or TEXT column with a maximum length of 4,294,967,295 characters.
 - You do not specify a length with LONGBLOB or LONGTEXT.
 7. ENUM
 - An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected (or it can be NULL).
 - For example, if you want your field to contain A or B or C, you would define your ENUM as ENUM ('A', 'B', 'C'), and only those values (or NULL) could ever populate that field. ENUMs can have 65,535 different values.
 - ENUMs use an index for storing items.

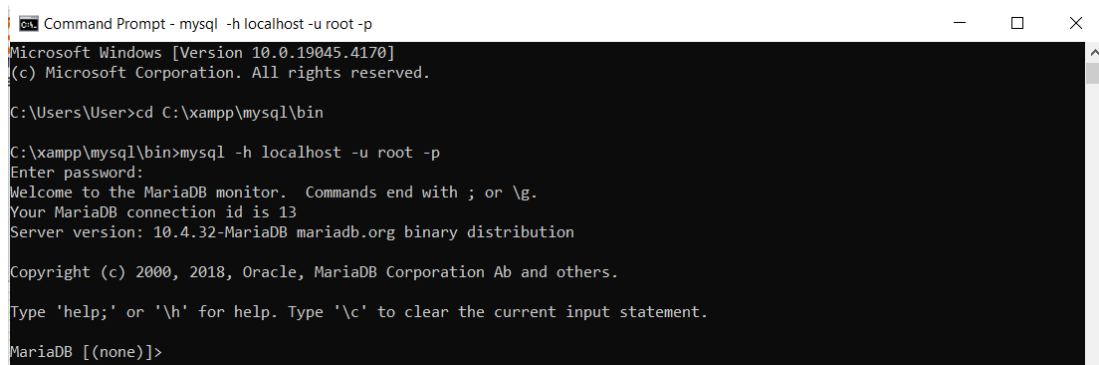
Accessing MySQL

- There are two main ways to access and interact with a MySQL database:
 1. Using MySQL Client
 2. Using php MyAdmin

Accessing Using MySQL Client

- This method involves using a command-line interface to connect to the MySQL server and execute SQL queries directly.
- Text-based interface for direct interaction with the MySQL server
- Here are the steps to connect to a MySQL server using the MySQL command-line client:
 1. Ensure XAMPP is installed and running on your system.
 2. By default, XAMPP configures MySQL to listen on port 3306.
 3. You'll need your XAMPP MySQL root user credentials (username and password). These are typically set during XAMPP installation, and the default username is often "root" and the password is left blank
 4. Press the Windows key, type "cmd", and press Enter.
 5. By default, XAMPP installs MySQL binaries in the \xampp\mysql\bin directory. Use the cd command to navigate to this location:
cd C:\xampp\mysql\bin
 6. Once you entered into bin folder, connect to the MySQL server using the following command
C:\xampp\mysql\bin>mysql -h localhost -u root -p Press Enter.
 7. If prompted, enter your password when requested.

8. If the connection is successful, you'll see the MySQL welcome message and the mysql> prompt, indicating you're connected.
9. From here, you can execute various SQL statements at the mysql> prompt



```

Command Prompt - mysql -h localhost -u root -p
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd C:\xampp\mysql\bin

C:\xampp\mysql\bin>mysql -h localhost -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 13
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>

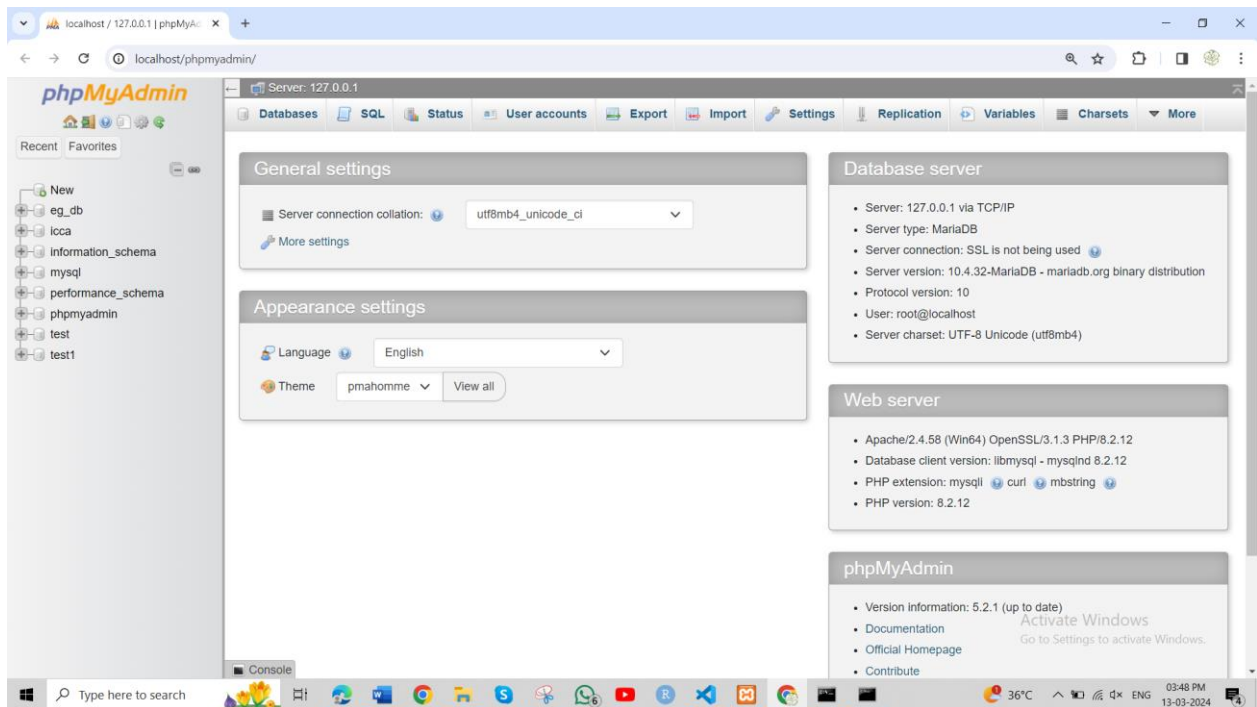
```

Accessing using php MyAdmin

- Web-based administration tool with a graphical user interface (GUI).
- Here are the steps to access MySQL using phpMyAdmin with the XAMPP server application:
 1. Ensure XAMPP is installed and running on your system. You can verify this by opening your web browser and going to <http://localhost/>. You should see the XAMPP welcome page
 2. In your web browser, navigate to the following URL: <http://localhost/phpmyadmin/>. This will launch the phpMyAdmin interface for managing your MySQL databases within XAMPP.
 3. You might be prompted for a username and password. By default, XAMPP uses the following credentials for phpMyAdmin access:
 - Username: root
 - Password: (Blank by default - Not recommended for production)
 4. Once logged in, you'll see the phpMyAdmin dashboard. This interface provides various functionalities for managing MySQL databases:
 - Left panel: Lists available databases and allows you to create new ones.
 - Central panel: Displays information about the currently selected database (tables, users, etc.).
 - Right panel: Provides options for working with the selected database object (e.g., creating/editing tables, browsing data, running queries).

Basic Operations:

- Browse Databases: Click on a database name in the left panel to see its tables.
- Browse Tables: Click on a table name to see its structure (columns and data types) and existing data entries.
- Create Databases: Use the "New" option in the left panel to create a new database.
- Create Tables: Within a database, use the "Create table" option to define a new table structure with columns and data types.
- Insert/Update Data: You can directly insert or update data within tables using the provided forms.
- Run SQL Queries: phpMyAdmin also allows you to execute SQL queries in the "SQL" tab for more advanced tasks.



MySQL Commands

1. CREATE Command

- The table-creation command requires
 - Name of the table
 - Names of fields
 - Definitions for each field.
- The generic table-creation syntax is

CREATE TABLE table_name (column_name column_type);

Example

```
CREATE TABLE IF NOT EXISTS icca_student(
    Regno varchar(10) NOT NULL PRIMARY KEY,
    Name varchar(20) NOT NULL,
    DOB date NOT NULL,
    Branch varchar(10) not null);
```

2. INSERT Command

- After you have created some tables, you use the SQL command INSERT for adding new records to these tables.
- The basic syntax of INSERT is

INSERT INTO table_name (column list) VALUES (column values)

Example

- A statement with all columns named:

```
INSERT INTO icca_student(Regno,Name,DOB,Branch) VALUES
('BCA202101','Balaji','2002-05-05','BCA');
```

- A statement that uses all columns but does not explicitly name them

```
INSERT INTO icca_student VALUES
```

('BCA202101','Balaji','2002-05-05','BCA');

3. SELECT Command

- SELECT is the SQL command used to retrieve records from your tables
- The basic SELECT Syntax is

SELECT expressions_and_columns FROM table_name [WHERE some_condition_is_true] [ORDER BY some_column [ASC | DESC]] [LIMIT offset, rows]

- If you want to order results a specific way, such as by date, ID, name, and so on, specify your requirements using the ORDER BY clause.
- You can use the LIMIT clause to return only a certain number of records from your SELECT query result
- The WHERE Clause is used to specify a particular condition

Example

SELECT *from icca_student;

4. DELETE Command

- If you want to delete a record from any MySQL table then you can use the SQL command DELETE
- The basix of DELETE is

DELETE FROM table_name [WHERE some_condition_is_true] [LIMIT rows]

Example

DELETE FROM icca_student WHERE Name = 'Balaji';

5. UPDATE Command

- UPDATE is the SQL command used to modify the contents of one or more columns in an existing record or set of records.
- The most basic UPDATE syntax looks like this

UPDATE table_name SET column1='new value', column2='new value2' [WHERE some_condition_is_true]

- Making a conditional UPDATE means that you are using WHERE clauses to match specific records.
- Using a WHERE clause in an UPDATE statement is just like using a WHERE clause in a SELECT statement. All the same comparison and logical operators can be used, such as equal to, greater than, OR, and AND.

Example

UPDATE icca_student SET Branch ="MCA";

Using PHP with MySQL:

PHP MySQL Functions

PHP provides several built-in functions for interacting with MySQL databases. Here's a list of commonly used PHP MySQL functions:

1. Connection Functions:

- mysqli_connect(): Opens a new connection to the MySQL server.
- mysqli_close(): Closes a previously opened database connection.

2. Query Execution Functions:

- `mysqli_query()`: Performs a query on the database.
- `mysqli_multi_query()`: Performs one or multiple queries which are concatenated by a semicolon.
- `mysqli_real_query()`: Executes a single query against the database.
- `mysqli_prepare()`: Prepares an SQL statement for execution.

3. Result Functions:

- `mysqli_fetch_assoc()`: Fetches a result row as an associative array.
- `mysqli_fetch_row()`: Fetches a result row as a numeric array.
- `mysqli_fetch_array()`: Fetches a result row as an associative array, a numeric array, or both.
- `mysqli_fetch_object()`: Fetches a result row as an object.

4. Error Handling Functions:

- `mysqli_error()`: Returns a string description of the last error occurred during the database operation.
- `mysqli_errno()`: Returns the error code for the most recent function call.

5. Transaction Management Functions:

- `mysqli_begin_transaction()`: Initiates a transaction.
- `mysqli_commit()`: Commits the current transaction.
- `mysqli_rollback()`: Rolls back the current transaction.

6. Prepared Statements Functions:

- `mysqli_stmt_prepare()`: Prepares an SQL statement for execution.
- `mysqli_stmt_bind_param()`: Binds variables to a prepared statement as parameters.
- `mysqli_stmt_execute()`: Executes a prepared query.
- `mysqli_stmt_bind_result()`: Binds variables to a prepared statement for result storage.

7. Other Functions:

- `mysqli_real_escape_string()`: Escapes special characters in a string for use in an SQL statement.
- `mysqli_insert_id()`: Returns the auto-generated id used in the last query.
- `mysqli_num_rows()`: Returns the number of rows in a result set.

Connecting to MySQL and Selecting the database

1. `mysqli_connect()`:

- Establishes a connection to a MySQL server.

Syntax: -

`mysqli_connect(servername, username, password, [database]);`

Where,

- `servername`: The hostname or IP address of the MySQL server. (e.g., "localhost" if it's on the same machine as your PHP script)
- `username`: A valid MySQL username with access to the database you want to connect to.
- `password`: The password associated with the username.
- `database` (optional): The specific database you want to connect to initially.

2. `mysqli_select_db()`

- Selects the specific database you want to work with within a connection established by `mysqli_connect`.

Syntax: -

mysqli_select_db(connection, database);

Where,

- connection: The link identifier resource returned by mysqli_connect.
- database: The name of the database you want to use within the connection

Note:- mysqli_close()

- Closes a previously opened connection to a MySQL database using the MySQLi extension.

Syntax: -

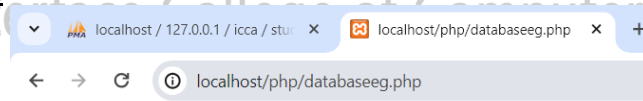
mysqli_close(connection)

Where,

- connection: The MySQLi connection object that you want to close.

Example

```
<?php
$conn = mysqli_connect("localhost","root","");
if($conn)
{
    echo "Successfully connected to localhost";
}
else
{
    die("Could not connect");
}
$newdb = mysqli_select_db($conn,"icca");
if($newdb)
{
    echo "<br/>Successfully Selected Specified Database";
}
else
{
    die("Sorry! Database could not be selected");
}
mysqli_close($conn);
?>
```

Output

Successfully connected to localhost
Successfully Selected Specified Database

Executing Simple Queries

- Inserting, updating, deleting, and retrieving data all revolve around the use of the mysqli_query() function.

Creating a table with PHP

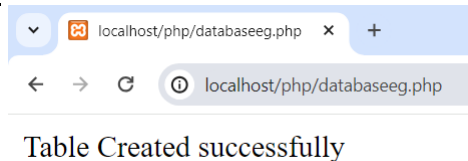
- Steps to create a table with PHP
1. Establish a database connection using PHP's built-in functions like mysqli_connect().
 2. Construct a SQL query using appropriate CREATE TABLE statement to define the structure of the table including column names, data types, constraints, etc.
 3. Execute the SQL query using PHP's database functions such as mysqli_query() to create the table in the database.

4. Handle any errors that may occur during the database interaction using error handling mechanisms provided by PHP, such as `mysqli_error()`.
5. Close the connection using `mysqli_close()` function.

Example

```
<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "CREATE TABLE IF NOT EXISTS icca_student(
    Regno varchar(10) NOT NULL PRIMARY KEY,
    Name varchar(20) NOT NULL,
    DOB date NOT NULL,
    Branch varchar(10) not null)";
$result = mysqli_query($conn, $query);
if($result)
{
    echo "Table Created successfully";
}
else
{
    echo "Something went wrong";
}
mysqli_close($conn);
?>
```

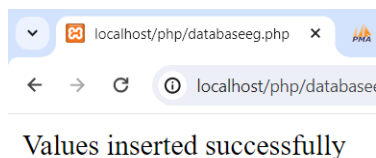
Output



Inserting Data with PHP

```
<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "INSERT INTO icca_student(Regno,Name,DOB,Branch) VALUES
    ('BCA202101','Balaji','2002-05-05','BCA'),
    ('BCA202102','Dhanush','2002-01-08','BCA'),
    ('BCA202103','Guru','2002-02-02','BCA')";
$result = mysqli_query($conn, $query);
if($result)
{
    echo "Values inserted successfully";
}
else
{
    echo("Failed to insert values");
}
mysqli_close($conn);
?>
```

Output



Retrieving Query Results

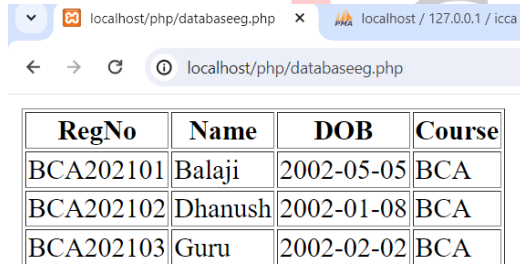
1. mysqli_fetch_row()

- This function fetches a row from the result set returned by a database query as a numeric array only.
- It returns a numeric array where each element corresponds to the value of a column in the order they appear in the SELECT statement.

Example

```
<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "SELECT *from icca_student";
$result = mysqli_query($conn, $query) or die("Could not fetch result");
if($result)
{
    echo "<table border=1>";
    echo "<tr><th>RegNo</th><th>Name</th><th>DOB</th><th>Course</th>";
    echo "<tr>";
    while($row = mysqli_fetch_row($result))
    {
        foreach($row as $display)
        {
            echo "<td>$display</td>";
        }
        echo "</tr>";
    }
    mysqli_close($conn);
}
?>
```

Output



| RegNo | Name | DOB | Course |
|-----------|---------|------------|--------|
| BCA202101 | Balaji | 2002-05-05 | BCA |
| BCA202102 | Dhanush | 2002-01-08 | BCA |
| BCA202103 | Guru | 2002-02-02 | BCA |

2. mysqli_fetch_array

- The `mysqli_fetch_array()` function fetches a result row as an associative array, a numeric array or both.

Syntax:-

`mysqli_fetch_array(result,resulttype);`

Where,

result -> Specifies a result set.

resulttype -> Specifies what type of array that should be produced. Can be one of the following values: MYSQLI_ASSOC, MYSQLI_NUM or MYSQL_BOTH.

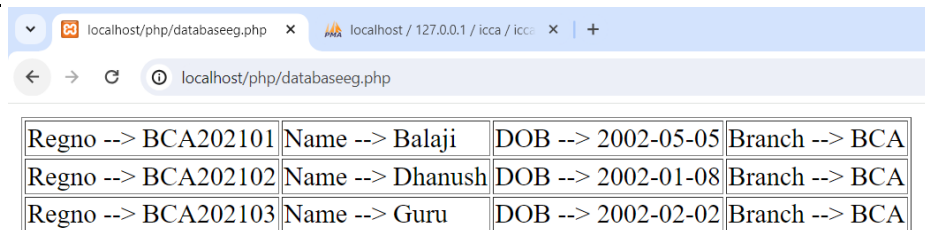
Example

```
<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "SELECT *from icca_student";
$result = mysqli_query($conn, $query) or die("Could not fetch result");
if($result)
{
```

```

        echo "<table border=1>";
        echo "<tr>";
        while($row = mysqli_fetch_array($result, MYSQLI_ASSOC))
        {
            foreach($row as $key => $value)
            {
                echo "<td>$key --> $value</td>";
            }
            echo "</tr>";
        }
        mysqli_close($conn);
    }
    ?>

```

Output


| | | | |
|---------------------|------------------|--------------------|----------------|
| Regno --> BCA202101 | Name --> Balaji | DOB --> 2002-05-05 | Branch --> BCA |
| Regno --> BCA202102 | Name --> Dhanush | DOB --> 2002-01-08 | Branch --> BCA |
| Regno --> BCA202103 | Name --> Guru | DOB --> 2002-02-02 | Branch --> BCA |

Counting Returned Records

- There are two primary methods to count the number of records returned by a query in PHP:

1. Using mysqli_num_rows()

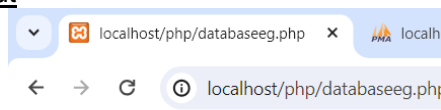
- This method is specifically designed for counting rows in a MySQLi result set.
- Syntax: - **mysqli_num_rows(result)**

Example

```

<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "SELECT *from icca_student";
$result = mysqli_query($conn, $query) or die("Could not fetch result");
if($result)
{
    $no_of_rows = mysqli_num_rows($result);
    echo "Result set has $no_of_rows number of rows";
}
mysqli_close($conn);
?>

```

Output

Result set has 3 number of rows

2. Using Count(*) in Query

- Using COUNT(*) directly in SQL is another way to count the number of records returned by a query.

Example

```

<?php

```

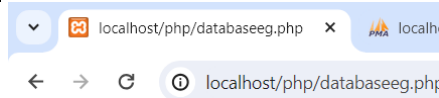
```

$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
$query = "SELECT COUNT(*) as total from icca_student";
$result = mysqli_query($conn, $query) or die("Could not fetch");
$row = mysqli_fetch_assoc($result);
$row_count = $row['total'];
echo "Result set has $row_count number of records";
mysqli_close($conn);

```

?>

Output



Result set has 3 number of rows

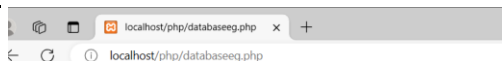
Updating records with PHP

```

<?php
$conn = mysqli_connect("localhost","root","","icca") or die("Could not connect");
echo "Updating the branch details from BCA to MCA<br/>";
$query = "UPDATE icca_student set Branch='MCA'";
$result = mysqli_query($conn, $query);
if($result)
{
    echo "Record Updated Successfully<br/>";
}
else
{
    echo "Sorry! could not update records";
}
$query = "SELECT *from icca_student";
$result = mysqli_query($conn, $query) or die("Could not fetch result");
if($result)
{
    echo "<table border=1>";
    echo "<tr><th>RegNo</th><th>Name</th><th>DOB</th><th>Course</th>";
    echo "<tr>";
    while($row = mysqli_fetch_row($result))
    {
        foreach($row as $display)
        {
            echo "<td>$display</td>";
        }
        echo "</tr>";
    }
}

```

Output



Updating the branch details from BCA to MCA
Record Updated Successfully

| RegNo | Name | DOB | Course |
|-----------|---------|------------|--------|
| BCA202101 | Balaji | 2002-05-05 | MCA |
| BCA202102 | Dhanush | 2002-01-08 | MCA |
| BCA202103 | Guru | 2002-02-02 | MCA |