Unit 1

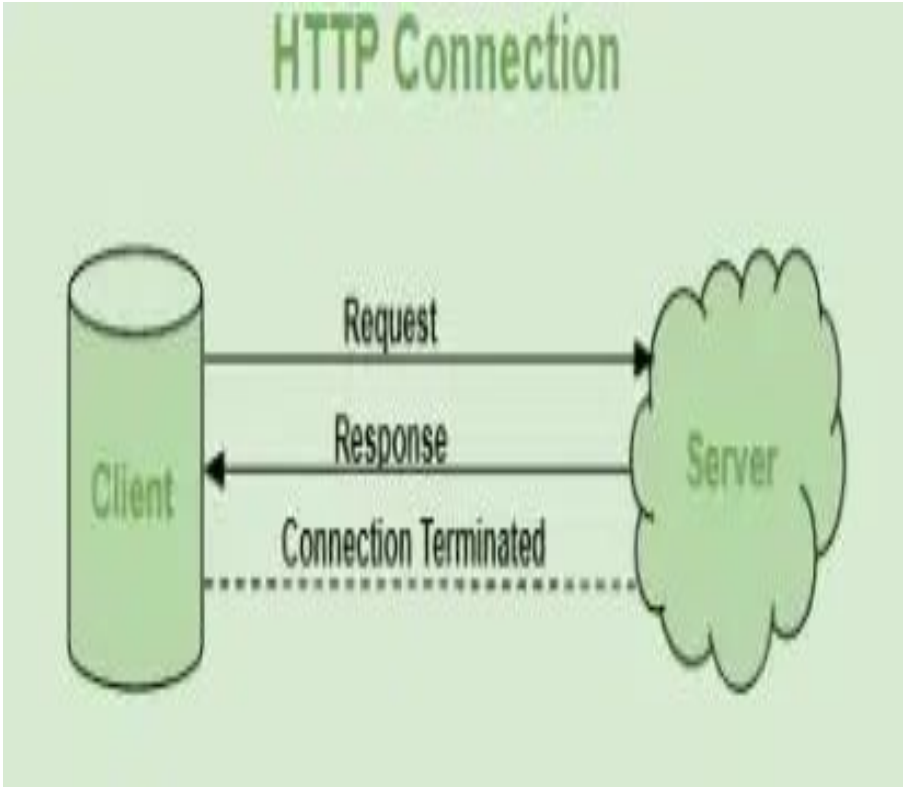
Introduction to PHP

1.1 HTTP Basics

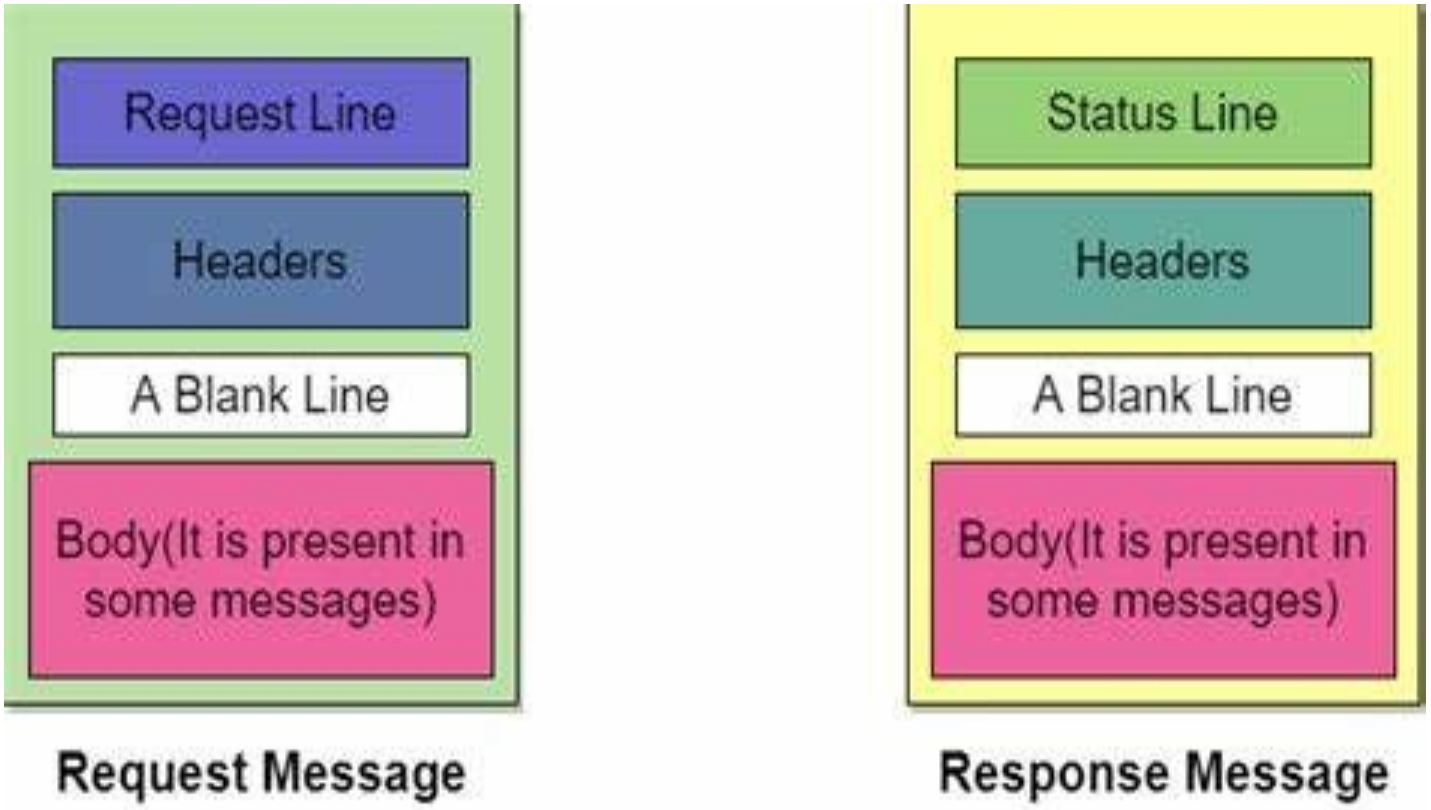
- **What is HTTP ?**
- HTTP (Hypertext Transfer Protocol) is a fundamental protocol of the Internet. It is enabling the transfer of data between a client and a server. It is the foundation of data communication for the World Wide Web.
- HTTP provides a standard between a web browser and a web server to establish communication. It is a set of rules for transferring data from one computer to another. Data such as text, images, and other multimedia files are shared on the World Wide Web. Whenever a web user opens their web browser, the user indirectly uses HTTP. It is an application protocol that is used for distributed, collaborative, hypermedia information systems.

- **Features of HTTP:**

1. **Connectionless protocol**: HTTP is a connectionless protocol. HTTP client initiates a request and waits for a response from the server. When the server receives the request, the server processes the request and sends back the response to the HTTP client after which the client disconnects the connection. The connection between client and server exist only during the current request and response time only.
2. **Media independent:** HTTP protocol is a media independent as data can be sent as long as both the client and server know how to handle the data content. It is required for both the client and server to specify the content type in MIME-type header.
3. **Stateless:** HTTP is a stateless protocol as both the client and server know each other only during the current request. Due to this nature of the protocol, both the client and server do not retain the information between various requests of the web pages.


HTTP Connection
Request
Response
Connection Terminated
Client
Server

1.2 HTTP Request and Response



Format of HTTP request Messages is

Start-line: Describes the request to be implemented or the status of the response.

For example GET/abc.html HTTP/1.1

Headers: Provide additional information about the request.

Example header_name:value1,value3,---

Host:www.abc.org

Connection:keep_alive

Blank line: Indicates the end of the headers.

Body: Contains the data associated with the request or response (optional).

Format of HTTP response massage is

HTTP Response Format:

Status Line: Provides the status of the HTTP request.

HTTP Version: The version of the HTTP protocol used (e.g., HTTP/1.1).

Status Code: A numeric value indicating the result of the request (e.g., 200, 404).

Headers: Contains meta-information about the response. Headers are key-value pairs separated by a colon.

Common Headers:

Content-Type: The MIME type of the body (e.g., text/html).

Content-Length: The length of the response body in bytes.

Date: The date and time when the response was generated.

Server: Information about the server.

Set-Cookie: Cookie data to be stored on the client side.

Blank Line: A blank line indicating the end of the header section.

Body: The main content of the response, which could be HTML, JSON, an image, etc. The body is optional and depends on the type of response.

1.3 Introduction to web server and web browser

What is server?

A server is a computer system or a software application that provides services or resources to other computers, known as clients, over a network. Servers play a critical role in managing, storing, and delivering data, applications, and services to users across various networks, including the internet and local area networks (LANs).

- **Types of Servers:**

1.Web Server: Hosts websites and serves web pages to clients via HTTP or HTTPS.

Examples: Apache, Nginx, Microsoft Internet Information Services (IIS).

2.Application Server: It connects database server and end users.

Examples: IBM WebSphere, Apache Tomcat, JBoss.

3.Database Server: Manages and provides access to a database for storing and retrieving data.

Examples: MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database.

4.File Server: Stores and manages files, allowing clients to access and share them.

5.Mail Server: Manages and stores email for users, enabling the sending, receiving, and storing of email messages.

Examples: Microsoft Exchange, Postfix, Sendmail.

6.FTP Server: Transfers files between computers using the File Transfer Protocol (FTP).

Examples: FileZilla Server, vsftpd, ProFTPD.

7.Proxy Server: It works in between client and an external server to filter request.**Examples:** Squid, HAProxy.

Web Server

- A web server is a software application or hardware device that stores, processes, and serves web content to users over the internet. It plays a critical role in the client-server model of the World Wide Web, where clients (typically web browsers) request web pages and resources, and servers respond to these requests by delivering the requested content.

- Web servers operate on the Hypertext Transfer Protocol (HTTP), which is the foundation of data communication on the World Wide Web. When you enter a website's URL into your browser, it sends an HTTP request to the web server hosting that website, which then sends back the web page you requested, allowing you to view it in your browser.
- **Web Browser**
- A web browser is a software application that allows users to access and interact with content on the World Wide Web. It fetches and displays web pages from the internet, making it possible for users to browse websites, view multimedia content, use web applications, and more.

How a Web Browser Works:

1. **User Enters URL:** The user types a web address (URL) into the browser's address bar.
2. **DNS Lookup:** The browser contacts a Domain Name System (DNS) server to translate the domain name into an IP address.
3. **HTTP Request:** The browser sends an HTTP or HTTPS request to the web server at the retrieved IP address.
4. **Server Response:** The web server processes the request and sends back the requested HTML content along with resources like CSS, JavaScript, and images.
5. **Rendering:** The browser's rendering engine parses and processes the HTML, CSS, and JavaScript to display the web page visually on the screen.
6. **Interactive Elements:** JavaScript and other interactive elements enable dynamic content and user interactions on the web page.

1.4 Introduction To PHP

The term PHP is an acronym for – ***Hypertext Preprocessor***. PHP is a server-side scripting language designed specifically for web development. It is an open-source which means it is free to download and use. It is very simple to learn and use. The file extension of PHP is ".php".

What is PHP?

PHP is a server-side scripting language created primarily for web development but it is also used as a general-purpose programming language. Unlike client-side languages like JavaScript, which are executed on the user's browser, PHP scripts run on the server. The results are then sent to the client's web browser as plain HTML.

1.4.1 History of PHP

PHP was introduced by **Rasmus Lerdorf** in **1994,** the first version and participated in the later versions. It is an interpreted language and it does not require a compiler. The language quickly evolved and was given the name "PHP," which initially named was "Personal Home Page."

PHP 3 (1998): The first version considered suitable for widespread use.

PHP 4 (2000): Improved performance and the introduction of the Zend Engine.

PHP 5 (2004): Added object-oriented programming features.

PHP 7 (2015): Significant performance improvements and reduced memory usage.

PHP 8 (2020): Introduction of Just-In-Time (JIT) compilation, further enhancing

1.4.2 What does PHP Do?

- PHP, which stands for "Hypertext Preprocessor," is a widely-used open-source scripting language especially suited for web development. Here's a breakdown of what PHP does and its main features:
- **Key Functions of PHP:**

1. **Server-Side Scripting:** PHP is executed on the server, and the results are sent to the client's browser as plain HTML. This allows for dynamic content generation.
2. **Database Interaction:** PHP can connect to various databases, including MySQL, PostgreSQL, and SQLite, to retrieve, store, and manipulate data.
3. **Form Handling:** PHP can collect data from forms (e.g., registration or login forms) and process it, making it essential for user input validation and processing.
4. **Session Management:** PHP manages user sessions, allowing developers to store user information between different pages of a website.
5. **File Handling:** PHP can create, read, write, upload, and manage files on the server, making it useful for content management systems.
6. **Built-in Functions and Libraries:** PHP comes with a rich set of built-in functions and libraries, which simplifies many common tasks in web development, such as string manipulation, date handling, and more.
7. **Integration with HTML and CSS:** PHP can be embedded directly into HTML code, allowing for the seamless integration of server-side logic with front-end design.
8. **Security:** PHP provides several features to help secure web applications, such as data encryption, secure password hashing, and input validation.

1.4.3 Why PHP is Better

1. It's free and open-source

Being a free server-side language, it is a budget-friendly option that can be used to create any type of web solution be it a website, eCommerce store, or web application.

2. Compatible with all OS

The next advantage of PHP is that it is compatible with different operating systems including Mac, Windows, Linux, and Unix as well as its interfaces perfectly with MySQL and Apache server.

3. Dynamic and flexible

Websites or web applications that are created using PHP are very secure because they offer fool-proof encryption. The capability of encryption and scalability add value to this amazing server-side language and make it a dynamic and flexible technology. Without any manual intervention, PHP-based web apps and websites can load automatically.

4. Database Flexibility

PHP is a very flexible language also in terms of database connectivity. It supports varied databases including MySQL, MongoDB, PostgreSQL, and more.

5. Works well with CMSs

An excellent benefit of choosing PHP as the web development technology for your business is that it is widely used in several content management systems. Many topmost CMSs including WordPress, Drupal, Magento, PrestaShop, Joomla, and more are written in PHP or are powered by a PHP-based framework.

1.5 PHP Lexical Structure

- The lexical structure of a programming language is the set of basic rules that governs how you write programs in that language. It is the lowest-level syntax of the language and specifies such things as what variable names look like, what characters are used for comments, and how program statements are separated from each other.

1 Case Sensitivity

- The names of user-defined classes and functions, as well as built-in constructs and keywords such as echo, while, class, etc., are case-insensitive. Thus, these three lines are equivalent:
- echo("hello, world"); ECHO("hello, world"); EcHo("hello, world");Variables, on the other hand, are case-sensitive. That is, $name, $NAME, and $NaME are three different variables.

2 Statements and Semicolons

- A statement is a collection of PHP code that does something. It can be as simple as a variable assignment or as complicated as a loop with multiple exit points. Here is a small sample of PHP statements, including function calls, assignment, and an if test:
- echo "Hello, world"; myfunc(42, "O'Reilly"); $a = 1; $name = "Elphaba"; $b = $a / 25.0; if ($a == $b) { echo "Rhyme? And Reason?"; }PHP uses semicolons to

3 Whitespace and Line Breaks

- In general, whitespace doesn't matter in a PHP program. You can spread a statement across any number of lines, or lump a bunch of statements together on a single line. For example, this statement:
- raise_prices($inventory, $inflation, $cost_of_living, $greed);

4 Comments

Comments give information to people who read your code.

There are 3 types of way to write comments.

1.Multiline comments:

/* In this section, we take a bunch of variables and assign numbers to them. There is no real reason to do this, we're just having fun. */

2.Singleline comments:

// create an HTML form

3. Shell-style comments:

create an HTML form requesting that the user confirm the action

5.Identifiers

- An identifier is simply a name. In PHP, identifiers are used to name variables, functions, constants, and classes. The first character of an identifier must be either an ASCII letter (uppercase or lowercase), the underscore character (_), or any of the characters between ASCII 0x7F and ASCII 0xFF. After the initial character, these characters and the digits 0-9 are valid.

- Variable names always begin with a dollar sign ($) and are case-sensitive. Here are some valid variable names:
- $bill $head_count $MaximumForce $I_HEART_PHP $_underscore $_intHere are some illegal variable names

6.Literals

- A literal is a data value that appears directly in a program. The following are all literals in PHP:
- 2001 ,0xFE ,1.4142 ,"Hello World" 'Hi' true null

7.Keywords

- A keyword is a word reserved by the language for its core functionality you cannot give a variable, function, class, or constant the same name as a keyword.
- Ex:

Abstract, and ,array, as ,break ,callable, case, catch, class , const ,continue, declare, default, die do ,echo else elseif empty

1.6 Language Basics

1.6.1 Data Types

PHP data types are a fundamental concept to defines how variables store and manipulate data. PHP is a loosely typed language, which means variables do not need to be declared with a specific data type. PHP allows eight different types of data types. All of them are discussed below.

- Predefined Data Types(Single value)
  - Integer
  - Float (Double)
  - String
  - Boolean
- User-Defined (compound) Data Types
  - Array
  - Objects
- Special Data Types
  - NULL
  - Resources

1.Predefined Data Types(Single value)

Integers hold only whole numbers including positive and negative numbers, i.e., numbers without fractional part or decimal point. They can be decimal (base 10), octal (base 8), or hexadecimal (base 16). The default base is decimal (base 10). The octal integers can be declared with leading 0 and the hexadecimal can be declared with leading 0x.

```
<?php  // decimal base integers

$deci1 = 50;

$deci2 = 654;

// octal base integers

$octal1 = 07;
```

```
// hexadecimal base integers

$octal = 0x45; $sum = $deci1 + $deci2;echo $sum;

echo "\n\n";

//returns data type and value
var_dump($sum)
 ?>
```

2. Float (Double)

Can hold numbers containing fractional or decimal parts including positive and negative numbers or a number in exponential form. By default, the variables add a minimum number of decimal places. The Double data type is the same as a float as floating-point numbers or real numbers.

```php
<?php

$val1 = 50.85;
$val2 = 654.26;

$sum = $val1 + $val2;

echo $sum;
echo "\n\n";

//returns data type and value
var_dump($sum)

?>
```

output

```
705.11

  float(705.11)
```

3. String

Hold letters or any alphabets, even numbers are included. These are written within double quotes during declaration. The strings can also be written within single quotes, but they will be treated differently while printing variables. To clarify this look at the example below.

```
<?php
$name = "Krishna";
echo "The name of the Geek is $name \n";
echo 'The name of the geek is $name ';
echo "\n\n";
//returns data type, size and value
var_dump($name)
echo gettype($name)//string
?>
```

output

```
The name of the Geek is Krishna
The name of the geek is $name


  string(7) "Krishna"
```

4. Boolean

Boolean data types are used in conditional testing. Hold only two values, either TRUE(1) or FALSE(0). Successful events will return *true* and unsuccessful events return *false.* NULL type values are also treated as *false* in Boolean. Apart from NULL, 0 is also considered false in boolean. If a string is empty then it is also considered false in boolean data type.

ex:
```
<?php
    if(TRUE)
      echo "This condition is TRUE";
    if(FALSE)
      echo "This condition is not TRUE";
    ?>
```
output

```
This condition is TRUE
```

2 User-Defined (compound) Data Types

1. Array

Array is a compound data type that can store multiple values of the same data type. Below is an example of an array of integers. It combines a series of data that are related together.

```
<?php
$intArray = array( 10, 20 , 30);
echo "First Element: $intArray[0]\n";
echo "Second Element: $intArray[1]\n";
echo "Third Element: $intArray[2]\n\n";
?>
```

Output

```
First Element: 10
Second Element: 20
Third Element: 30
```

2.Objects – Instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.

```php
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar = new foo;
$bar->do_foo();
?>
```

Special Data Types

1. NULL

These are special types of variables that can hold only one value i.e., NULL. We follow the convention of writing it in capital form, but it's case-insensitive NULL, Null, and nul are treated the same. If a variable is created without a value or no value, it is automaticall assigned a value of NULL. It is written in capital letters.

```
<?php
    $nm = NULL;
    echo $nm;     // this will return no output
?>
```

2.resource

A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions.

1.6.2 Variables

PHP Variables

A variable can have a short name (like `$x` and `$y`) or a more descriptive name (`$age`, `$carname`, `$total_volume`).

Rules for PHP variables:

- A variable starts with the `$` sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (`$age` and `$AGE` are two different variables)

1.6.1 Variable variables

In PHP, the name of a variable can be derived from the value of another variable. For example:

```
<?php

$my_var= 'title';
$$my_var = 'PHP variable variables';

echo $title;
?>
```

Output:

```
PHP variable variables
```

How it works.

- First, define a variable $my_var that holds the string 'title'.
- Second, define a variable variable that holds the string 'PHP variable variables'. Note that we use double $ signs instead of one. By doing this, we technically create another variable with the name $title.
- Third, display the value of the $title variable

1.6.3.Variable References

In PHP, **References** enable accessing the same variable content by different names. They are not like pointers in C/C++ as it is not possible to perform arithmetic oprations using them. In C/C++, they are actual memory addresses. In PHP in contrast, they are symbol table aliases. In PHP, variable name and variable content are different, so the same content can have different names. A reference variable is created by prefixing **&** sign to original variable. Hence **b=&a** will mean that b is a reference variable of a.

Example

```
<?php
$var1=10;
$var2=&$var1;
echo "$var1 $var2\n";
$var2=20;
echo "$var1 $var2\n";
?>
```

output-

```
10 10
20 20
```

1.6.3 PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

i.Local variable

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.

A variable declaration outside the function with the same name is completely different from the variable declared inside the function. Let's understand the local variables with the help of an example:

```
<?php
    function local_var()
    {  $num = 45;  //local variable
        echo "Local variable declared inside the function is: ". $num;
    }  local_var();
?>
```

Output:

Local variable declared inside the function is: 45

ii.Global variable

The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program. To access the global variable within a function, use the GLOBAL keyword before the variable. However, these variables can be directly accessed or used outside the function without any keyword Therefore there is no need to use any keyword to access a global variable outside the function.

Let's understand the global variables with the help of an example:

```php
<?php
    $name = "Sanaya Sharma";        //Global Variable
    function global_var()
    {
        global $name;
        echo "Variable inside the function: ". $name;
        echo "</br>";
    }  global_var();
    echo "Variable outside the function: ". $name;
?>
```

Output:

Variable inside the function: Sanaya Sharma

Variable outside the function: Sanaya Sharma

iii.Static variable

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**. Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

```
<?php
    function static_var()
    {   static $num1 = 3;       //static variable
        $num2 = 6;          //Non-static variable
$num1++;   $num2++;
        echo "Static: " .$num1 ."</br>";
        echo "Non-static: " .$num2 ."</br>";
    }
```

```
//first function call
    static_var();

    //second function call
    static_var();
?>
```

Output:

Static: 4

Non-static: 7

Static: 5

Non-static: 7

1.7 Expression and Operators

- Expressions are the most important building blocks of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".
- The most basic forms of expressions are constants and variables. When you type $a = 5, you're assigning 5 into *$a*. 5, obviously, has the value 5, or in other words 5 is an expression with the value of 5 (in this case, 5 is an integer constant).
- After this assignment, you'd expect *$a*'s value to be 5 as well, so if you wrote $b = $a, you'd expect it to behave just as if you wrote $b = 5. In other words, *$a* is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.
- Example
- <?php
  function foo ()
  {
  return 5;
  }
  ?>

Operators

- **Unary Operators:** works on single operands such as ++, -- etc.
- **Binary Operators:** works on two operands such as binary +, -, *, / etc.
- **Ternary Operators:** works on three operands such as "?:".

Operator Type	Operator	Description	Example
Arithmetic Operators	`+`	Addition	`$a + $b`
	`-`	Subtraction	`$a - $b`
	`*`	Multiplication	`$a * $b`
	`/`	Division	`$a / $b`
	`%`	Modulus (remainder of division)	`$a % $b`
Assignment Operators	`=`	Assigns a value to a variable	`$a = 5`
	`+=`	Add and assign	`$a += 3` (equivalent to `$a = $a + 3`)
	`-=`	Subtract and assign	`$a -= 2`
	`*=`	Multiply and assign	`$a *= 4`
	`/=`	Divide and assign	`$a /= 2`
	`%=`	Modulus and assign	`$a %= 2`

Comparison Operators	`==`	Equal	`$a == $b`
	`===`	Identical (equal and same type)	`$a === $b`
	`!=` or `<>`	Not equal	`$a != $b`
	`!==`	Not identical (not equal or not same type)	`$a !== $b`
	`>`	Greater than	`$a > $b`
	`<`	Less than	`$a < $b`
	`>=`	Greater than or equal to	`$a >= $b`
	`<=`	Less than or equal to	`$a <= $b`
Logical Operators	`&&`	Logical AND	`$a && $b`
	`		`
	`!`	Logical NOT	`!$a`
Increment/ Decrement	`++`	Increment (increase by 1)	`$a++` or `++$a`
	`--`	Decreme[illegible]ecrease by 1)	`$a--` or `--$a`

String Operators	`.`	Concatenation (joins two strings)	`$a . $b`
	`.=`	Concatenation assignment (appends one string to another)	`$a .= $b`
Array Operators	`+`	Union (combines two arrays)	`$a + $b`
	`==`	Equality (checks if two arrays are equal)	`$a == $b`
	`===`	Identical (checks if arrays are equal and of the same type)	`$a === $b`
	`!=` or `<>`	Inequality (checks if two arrays are not equal)	`$a != $b`
	`!==`	Non-identical (checks if arrays are not identical)	`$a !== $b`
Conditional (Ternary)	`? :`	Ternary operator (shorthand for `if-else`)	`$a > 5 ? 'Greater' : 'Smaller'`
Null Coalescing	`??`	Checks if variable is set and not null, returns default value if it is null	`$a ?? $b`

Error Control Operator	`@`	Suppresses error messages from expressions	`@fopen("file.txt", "r")`
Type Comparison	`is_array()`	Checks if a variable is an array	`is_array($a)`
	`is_bool()`	Checks if a variable is a boolean	`is_bool($a)`
	`is_int()`	Checks if a variable is an integer	`is_int($a)`
	`is_float()`	Checks if a variable is a float	`is_float($a)`
	`is_null()`	Checks if a variable is null	`is_null($a)`
	`is_object()`	Checks if a variable is an object	`is_object($a)`
	`is_string()`	Checks if a variable is a string	`is_string($a)`
	`is_numeric()`	Checks if a variable is numeric (integer or float)	`is_numeric($a)`
Array Access	`[]`	Used to access or set values in an array by index or key	`$array[0]`

1.8 Type juggling and type casting with example

1.Type Juggling

- Type juggling in PHP refers to the automatic conversion of one data type to another based on the context in which the variable or expression is used. PHP is a loosely typed language, which means it performs type juggling automatically.

Example of Type Juggling:

```
<?php
$a = "10"; // string
$b = 20;   // integer$result =
    $a + $b; // PHP converts $a
    to integer for arithmetic
    operationecho
$result; ?>
```

Output 30

2.Type Casting

- Type casting in PHP is the explicit conversion of a variable from one data type to another. This is done using a cast operator in parentheses before the variable or value.

- Example of Type Casting:

```
<?php
$a = "10"; // string
$b = 20;   // integer
$result = (int)$a + $b;
 // Explicitly cast $a to integer
echo $result; ?>
```

Output: 30

1.8 Control statement in php

1.Conditional/Decision Control Statements in Php:

- Conditional statements allow the execution of specific code blocks based on certain conditions.
- The most commonly used conditional statements in PhP are if, if-else, if-else-elseif, and switch.

i.If Statement

- Syntax:
  - if (condition) {
    // codes for the condition is true
    }

ii.If-else Statement

Syntax:

```
if (condition) {
        // codes for the condition is true
    } else {
        // codes for the condition is false
    }
```

iii.If – else – else if Statement

Syntax:

```
if (condition1) {
        // codes for condition1 is true
    } elseif (condition2) {
        // codes for condition2 is true
    }
elseif (condition n) {
        // codes for condition n is true
    } else {
    // codes for false condition
    }
```

- **Switch Statement**
  - Syntax:

```
switch (expression/variable) {
        case value1:
            // code for expression equals value1
            break;
        case value2:
            // code for expression equals value2
            break;
    case value n:
            // code for expression equals value n
            break;
        default:
            // code for false/does not match any of the values
        of the above case
     }
```

2. **Looping Control Statements in Php:** Looping statements are used to execute a block of code repeatedly until a certain condition is met/condition is true.The most commonly used looping statements in PhP are for, while, do-while, and for-each.

i.For Loop Statement

Syntax:

```
for (initialization; terminate_condition; increment/decrement) {
        // code to be executed repeatedly while the condition is true
    }
```

ii.While Loop Statement

Syntax:

```
while (condition) {// codes for the true condition
    }
```

iii. Do-While Loop Statement

Syntax:

```
do {// code to be executed at least once and repeatedly while the condition is true
    } while (condition);
```

iv.For Each Loop Statement (Used with Array mainly)

Syntax:

```
foreach ($array_name as $variable_name {
        // code to be executed for each element in the array
    }
```

3. Jumping Control Statements in Php:

Jump/jumping statements allow the programmer to change the normal flow of execution of a program suddenly.

The most commonly used jump statements in PHP are break, continue, and goto.

i.Break Statement

used to exit a loop prematurely/forcibly.

Syntax:

```
while (condition) {
        // code to be executed repeatedly while the condition is true
        if (some condition) {
            break;
        }
    }
```

ii.Continue Statement

used to skip the current iteration and move to the next one

```
while (condition) {
        // code to be executed repeatedly while the condition is true
        if (some condition) {
            continue;
        }
        // code after the continue statement will not be executed for that iteration
    }
```

4. Error handling statements

Error handling statements allow the programmer to handle errors and exceptions that occur during the execution of a program.

The most commonly used error-handling statements in PHP are try, catch, and throw.

1.9 Garbage Collection

- Garbage collection in PHP is the process of automatically identifying and reclaiming memory that is no longer in use by the application. This helps in managing memory efficiently and preventing memory leaks. PHP primarily uses a reference counting mechanism for memory management.
- **Key Concepts in PHP Garbage Collection**
- **Reference Counting**:
  - Each variable in PHP has a reference count that tracks how many references exist to that variable.
  - When a variable's reference count drops to zero (i.e., no references to the variable), the memory occupied by that variable is automatically freed.
- **Cyclic Garbage Collection**:
  - PHP's reference counting mechanism can fail in cases where there are reference cycles (two or more objects reference each other).
  - To handle this, PHP has a cyclic garbage collector that periodically checks for these reference cycles and frees the memory occupied by them.

Unit 2

Functions And Arrays

2.1 Introduction

- **PHP Functions**
- PHP function is a piece of code that can be reused many times. It can take input as argument list and return value. There are thousands of built-in functions in PHP.
- **Advantage of PHP Functions**
- **Code Reusability**: PHP functions are defined only once and can be invoked many times, like in other programming languages.
- **Less Code**: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.
- **Easy to understand**: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

2.2 Defining and calling a function

- **Steps:**
- **Function Declaration**: Use the function keyword followed by the function name and a block of code enclosed within curly braces {} to define a function.
- **Function Parameters**: Define any parameters the function may accept within the parentheses ().
- **Function Body**: Write the code that the function will execute within the curly braces {}.
- **Function Call**: Invoke the function by its name followed by parentheses () containing any required arguments.
- **Syntax:**
- // Function declaration
  function functionName($param1, $param2, ...) {
  // Function body
  // Perform specific tasks here
  }
  // Function call
  functionName($arg1, $arg2, ...);

2.2.1 Function without parameters in PHP

- As we discussed above, the function without parameters does not contain any variables as arguments. The normal function is known as function without parameter. i.e

Ex:

```
<?php
 function sum()
{ $no=20+20;
 echo $no; }
sum(); ?>
```

Output

40

2.2.2 Function with parameters

PHP Parameterized functions are the functions with parameters.
You can pass any number of parameters inside a function.
These passed parameters act as variables inside your function.
They are specified inside the parentheses, after the function name.
Example

```
<html>  <head>
   <title>Parameter Addition</title>
</head>  <body>
<?php
  function add($x, $y) {
         $sum = $x + $y;   echo "Sum of two numbers is = $sum <br><br>";
       }   add(467, 943);  ?>
</body>  </html>
```

PHP Call By Reference

In case of PHP call by reference, actual value is modified if it is modified inside the function. In such case, you need to use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

Example

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'This is ';
adder($str);
echo $str;
?>
```

2.2.3 Function Returning the Result

Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called.

Example

```
<?php
 function square($num)
 {
 return $num * $num;
 }
 echo square(4); // outputs '16'.
 ?>
```

2.3 Default parameter values

- A function may define default values for parameters using syntax similar to assigning a variable. The default is used only when the parameter's argument is not passed.
- Note that passing null does *not* assign the default value.

Example:

```
<?php
    function makecoffee($type = "cappuccino")
    {
    return "Making a cup of $type.\n";
    }
    echo makecoffee();
    echo makecoffee(null);
    echo makecoffee("espresso");
    ?>
```

2.4 Variable Parameter and Missing Parameters

- When Number of parameters are not fixed then Variable Parameter function can be used. for e.g. to find out area of different shapes.
- PHP Provide three functions

1.func_get_args() It returns an array of the all parameter provided to the Function.

2.func_num_args() It returns the number of parameters provided to the Function.

3.func_get_arg() It returns a specific argument from the parameters.

- **Missing Parameters**
- When you call a function you can pass any number of argument to the function. Any parameters the function expects that are not passed to it remain unset, and a warning is issued for each of missing parameter.

Ex:

```
<?php
Function Display($a,$b)
If(isset($a))
{
Echo”first value set”;
}
If(isset($b))
{
Echo”second value set”;
}}
Display(12,34);
Display(10);
Display();
?>
```

2.5 variable function and anonymous function

Variable Functions: In PHP, variable functions allow you to call a function using a variable. If the variable's value is the name of a function, you can call the function using that variable. This can be particularly useful for creating dynamic and flexible code.

Example:

php

function sayHello()

{ echo "Hello, World!"; }

$func = 'sayHello'; $func(); // Calls the sayHello function and outputs "Hello, World!"

In this example, $func holds the name of the sayHello function, and calling $func() executes sayHello.

- **Anonymous Functions**: Anonymous functions, also known as closures, are functions without a specified name. They are often used as values for variables, and they can be passed as arguments to other functions or returned from functions. Anonymous functions are useful for defining small functions on the fly.

Example:

php

```
$greet = function($name)
{ return "Hello, $name!"; };
 echo $greet("Alice"); // Outputs "Hello, Alice!"
```

In this example, an anonymous function is assigned to the variable $greet, and calling $greet("Alice") executes the function.

difference between anonymous function and normal function

anonymous function	Normal function
There is no name to a function.	Every function has a name
Function definition end with semicolon(;)	Function definition doesn't end with semicolon(;)
It is assigned to a variable and then called using variable name followed by parenthesis.	It is called using function name following by parenthesis.
Function will not be in the global scope.	Function will remain in the global scope.

2.6 Indexed vs Associative Arrays

Indexed array: Indexed array is an array with a numeric key. It is basically an array where in each of the keys is associated with its own specific value.

Example 1:

```
<?php
   // Declaring an array
$arr = array();
  // Assigning values
$arr[0] = 5;
$arr[1] = 6;
  print("Array : ");
print_r($arr); //it outputs the contents of the array or object in a readable format.
  ?>
```

Output

```
Array : Array
(
 [0] => 5
[1] => 6
)
```

Associative array: An associative array is stored in the form of (index)key-value pair. This type of array is where the key is stored in the numeric or string format.

Example 1:

PHP

```
<?php
   // Declaring an array
$arr = array(
     "Java" => "Spring Boot",
     "Python" => "go",
     "PHP" => “xmapp"
); print("Array : ");
print_r($arr);
  ?>
```

Output

```
Array : Array ( [Java] => Spring Boot
 [Python] => go
 [PHP] => xmapp)
```

difference between an indexed array and an associative array

Indexed Array	Associative Array
The keys of an indexed array are integers which start at 0.	Keys may be strings in the case of an associative array.
They are like single-column tables.	They are like two-column tables.
They are not maps.	They are known as maps.

2.7 Identifying Element of an array

- To access a values from an array in PHP script use the array name and element or index.
- The key can be either an integer or a string.
- Example $abc=array(“abcd”,”xyz”);

with $abc indexed array $abc[0] will refer to an array element “abcd”.

Add Values at end without using function

You can add element in the existing array after the last element without knowing the index of last element.

Example $number=array(10,4,67,80);

$number[]=20;

In associative array if element is added and index is not mentioned the PHP will assign numeric index.

Example

$a=array[‘one=>1’,’two=>4’];

$a[]=10;

Here value of $a[0] is 10.It assigns index values starting with 0.

Range of values

The range() function used to create an array containing a range of elements.

- **Syntax:**
- range(low_value, high_value, step)
- low value and high value both value are must be required and step is optional It is used as increment between elements. The default value is 1.
- *Mixed: Mixed indicates multiple (but not necessarily all) types.

```
<?php
$number_list = range(11,19);
print_r ($number_list);
echo "<br /> ";
$number_list_step = range(11,19,3);
print_r ($number_list_step);
echo "<br /> ";
$letter_list = range("u","z");
print_r ($letter_list);
?>
```

output

```
Array ( [0] => 11 [1] => 12 [2] => 13 [3] =>
    14 [4] => 15 [5] => 16 [6] => 17 [7] =>
    18 [8] => 19 )
    Array ( [0] => 11 [1] => 14 [2] => 17 )
    Array ( [0] => u [1] => v [2] => w [3] =>
    x [4] => y [5] => z )
```

Size of an Array

- **count()**: This is the primary function used to count elements in an array or properties in an object that implements the Countable interface.
- **sizeof()**: This is an alias of count(), meaning it performs the exact same operation as count().
- Example:

```
$array = [1, 2, 3, 4, 5];
 echo count($array); // Outputs: 5
echo sizeof($array); // Outputs: 5
```

Padding an array

- The array_pad() function inserts a specified number of elements, with a specified value, to an array.
- If you assign a negative size parameter, the function will insert new elements BEFORE the original elements (See example below).
- **Note:** This function will not delete any elements if the size parameter is less than the size of the original array.

array_pad(*array, size, value*)

example

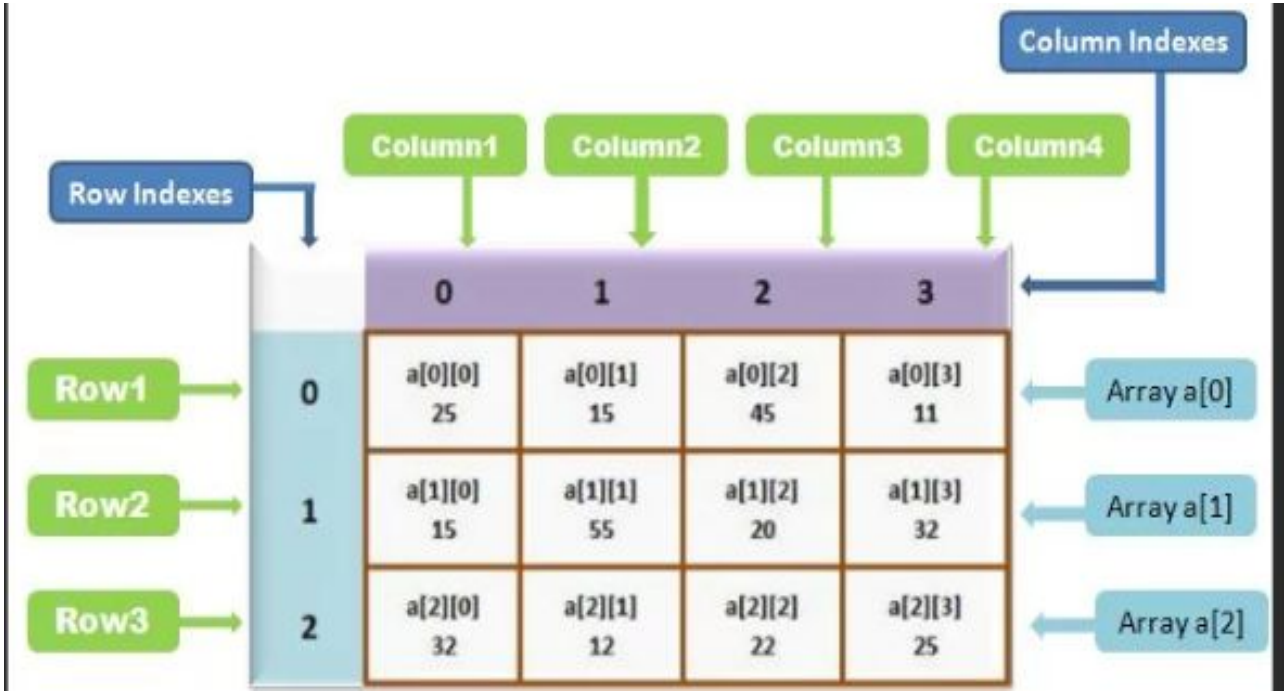
```
<?php
$a=array("red","green");
print_r(array_pad($a,6,"blue"));
?>
```

Output

```
Array ( [0] => red [1] => green [2] => blue [3] => blue
    [4] => blue [5] => blue )
```

2.8 Multidimensional Arrays

- A multidimensional array is an array containing one or more arrays.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep.
- They can be used to store data in a tabular format, such as a matrix or a table.



Example:<?php

```
// Creating a multidimensional array
$multiArray = array( array(1, 2, 3),
  array(4, 5, 6),
  array(7, 8, 9) ); // Accessing elements
echo $multiArray[0][1]; // Outputs: 2
echo $multiArray[2][2]; // Outputs: 9
?>
```

2.9 Extracting multiple values

The list() function is used to extract multiple values from an array and assign them to separate variables. It's especially helpful when you have an array with indexed values and you want to get each value into a different variable.

Ex:

```
<?php
 $array = [10, 20, 30, 40];
list($a, $b, $c, $d) = $array; // Extract values from array
echo $a; // 10
echo $b; // 20
echo $c; // 30
echo $d; // 40 ?>
```

2.9.1 Slicing an array

The array_slice() function is an inbuilt function of PHP.

The array_slice() function is used to extract a slice of an array.

Example

```
<?php  $input = [1, 2, 3, 4, 5, 6, 7];
$slice = array_slice($input, 2,4);
print_r($slice);?>
Array ( [0] => 3
 [1] => 4
[2] => 5
[3] => 6 )
```

2.9.2 Spliting an array into chunks

The array_chunk() method splits an array into chunks and returns a multidimensional array, each array containing length elements.

Syntax

array_chunk($array, $length, $preserve_keys)

The array_chunk() function has two required parameters and one optional parameter.

- $array: Specifies the input array.
- $length: An integer that specifies the number of elements in a chunk.
- $preserve_keys: When set to true keys will be preserved. The default is false which will reindex each chunk.

The array_chunk() function returns a multidimensional array, with each dimension containing $length elements.

Ex:

```
<?php
$students = ["abc","pqr","stq","mno","xyz","ijk","bpr"];
    $chunkedArray = array_chunk($students, 3, true);
    print_r($chunkedArray)
?>
```

Output:Array ([0] => Array ([0] => abc [1] => pqr [2] => stq)
[1] => Array ([3] => mno [4] => xyz [5] => ijk)

2.9.3 Key and values

The array_keys() function is used to get all the keys or a subset of the keys of an array. It returns the keys, numeric and string, from the input array.

Syntax: $keys=array_keys(arrayname)

Ex:<?php

$person = ["name" => "John", "age" => 25, "city" => "New York"];

$keys = array_keys($person);

print_r($keys); ?>

// Output: // Array ([0] => name [1] => age [2] => city)

Array_values() This function return an array containing only the values from the array which is passed as parameter.

Syntax:$values=array_values(array);

Ex: <?php $person = ["name" => "John", "age" => 25, "city" => "New York"];

$val1= array_values($person);

print_r($val1);?>output//Array ([0] => John[1] => 25 [2] => New York)

2.9.4 Checking whether an elements exists

The **array_key_exists() function** in PHP is a built-in function that checks whether a specified key exists in an array. This function is commonly used when working with associative arrays.

Ex:

```
<?php
$array = array(
   "first" => 1,
   "second" => 2,
   "third" => 3
);
// Key to check
$key = "second";
// Check if the key exists in the array
if (array_key_exists($key, $array)) {
   echo "The key '$key' exists in the array.";
} else {
   echo "The key '$key' does not exist in the array.";
}
?>
```

2.9.5 Removing and inserting element

The array_splice() function removes selected elements from an array and replaces it with new elements. The function also returns an array with the removed elements.

Syntax array_splice(*array, start, length, array*)

```
<?php
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("a"=>"purple","b"=>"orange");
print_r(array_splice($a2,0,3,$a1));
?>
```

output Array ([a] => purple [b] => orange)

2.10 Traversing Array

Traversing an array means iterating over the array elements to access and manipulate the values. You can traverse an array in several ways depending on your requirements. Traversing an array in PHP means going through each element in the array one by one.

1. Using foreach loop

The foreach loop is the easiest way to go through an array and access each element.

Example:

```
$array = [1, 2, 3, 4, 5];

foreach ($array as $value) {
   echo $value . "\n"; // Prints each number
}
```

2. Using for loop

If you know the indexes of the array, you can use a for loop.

Example:

```
$array = [1, 2, 3, 4, 5];
for ($i = 0; $i < count($array); $i++)
{ echo $array[$i] . "\n"; // Prints each number by index }
```

3. Using while loop

You can also use a while loop to traverse an array, but it's less common than foreach.

Example:

```
$array = [1, 2, 3, 4, 5]; $i = 0;
while ($i < count($array))
 { echo $array[$i] . "\n";
$i++; }
```

Difference between for loop and for each loop

Feature	`for` Loop	`foreach` Loop
Use Case	Best for when you need to access array elements by index or know the number of iterations.	Best for iterating over all elements of an array without worrying about indexes.
Syntax	`for ($i = 0; $i < count($array); $i++) {}`	`foreach ($array as $value) {}`
Access to Index	Requires manually managing the index.	Automatically gives access to the value or both key and value.
Array Type	Can be used for both indexed and associative arrays, but requires specific handling for associative arrays.	Primarily designed for arrays, works naturally with both indexed and associative arrays.
Index or Key	You can only access the index using `$i`.	You can access both the key and value directly (e.g., `foreach ($array as $key => $value)`).
Performance	Slightly slower for associative arrays since you manually handle keys.	Faster and simpler for associative arrays because PHP handles it automatically.
Control	Allows more control over the loop iteration, such as skipping or changing steps.	Simple and concise for most array traversal needs, but less control over iteration.
Common Use Case	Iterating with a known number of iterations (e.g., when using an index to a array elements).	Iterating over all elements in an array without needing to worry about the number of elements.

2.10 Iterator

- Any object that implements the Iterator interface can be used as an argument of a function that requires an iterable.
- An iterator contains a list of items and provides methods to loop through them. It keeps a pointer to one of the elements in the list. Each item in the list should have a key which can be used to find the item.
- An iterator must have these methods:

1. **current()** - Returns the element that the pointer is currently pointing to. It can be any data type
2. **key()** Returns the key associated with the current element in the list. It can only be an integer, float, boolean or string
3. **next()** Moves the pointer to the next element in the list
4. **rewind()** Moves the pointer to the first element in the list
5. **valid()** If the internal pointer is not pointing to any element (for example, if next() was called at the end of the list), this should return false. It returns true in any other case

2.11calling a function for each array elements

The **array_walk()** function allows you to apply a user-defined function to each element of an array. It modifies the array in place. The function takes two parameters:

The array you want to operate on.

The name of the callback function that will be applied to each element.

Ex:

```
<?php
$numbers = [1, 2, 3, 4, 5];
// Define a callback function to modify each element
function squareElement(&$value, $key) {
    // Square the value of each element
    $value = $value * $value;
}
// Apply the function to each element of the array
array_walk($numbers, 'squareElement');

// Print the modified array
print_r($numbers);
?>
```

Output

```
Array ( [0] => 1 [1] => 4 [2] => 9 [3] => 16 [4] => 25 )
```

2.12 Reduce an array

- The array_reduce() function in PHP is used to reduce an array to a single value by applying a custom function to its elements. Here's a simple explanation and example of how it works:

Syntax:

- array_reduce($array, $callback);
- $array: The array to reduce.
- $callback: A function that takes two arguments

Ex:

```
<?php
$numbers = [1, 2, 3, 4, 5];
// Define a callback function to sum the elements
function sumElements($carry, $item) {
return $carry + $item;
}
$sum = array_reduce($numbers, 'sumElements', 0);
// Print the result
echo "Sum: " . $sum;
?>
```

Output

Sum: 15

2.13 Search a value in an array

The in_array() function in PHP is used to check if a value exists in an array. It returns true if the value is found and false otherwise.

syntax :in_array($needle, $haystack, $strict = false);

$needle: The value you are searching for.

$haystack: The array to search in.

$strict (optional): If set to true, in_array() will also check the types of the values. By default, it checks only the values.

```
<?php
$fruits = ["apple", "orange"];
if (in_array(“orange”, $fruits)) {
   echo “element is in the array.";
} else {
   echo “element is not in the array.";
}
?>
```

Output

element is in the array

2.14 Sorting Arrays

- Sorting arrays is one of the most common operation in programming, and PHP provides a several functions to handle array sorting. Sorting arrays in PHP can be done by values or keys, in ascending or descending order. PHP also allows you to create custom sorting functions.

Sorting one array at a time

1.sort() - Sorts arrays in ascending order.

```
$numbers = array(4, 6, 2, 22, 11);
sort($numbers); print_r($numbers);
 // Output: Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 11 [4] => 22 )
```

2.rsort() - Sorts arrays in descending order.

```
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers); print_r($numbers);
// Output: Array ( [0] => 22 [1] => 11 [2] => 6 [3] => 4 [4] => 2 )
```

3.asort() - Sorts associative arrays in ascending order, according to the value.

```
$age = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
asort($age); print_r($age);
// Output: Array ( [Peter] => 35 [Ben] => 37 [Joe] => 43 )
```

4.ksort() - Sorts associative arrays in ascending order, according to the key.

```
$age = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
 ksort($age); print_r($age);
// Output: Array ( [Ben] => 37 [Joe] => 43 [Peter] => 35 )
```

5.arsort() - Sorts associative arrays in descending order, according to the value.

```
$age = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
 arsort($age); print_r($age);
// Output: Array ( [Joe] => 43 [Ben] => 37 [Peter] => 35 )
```

6.krsort() - Sorts associative arrays in descending order, according to the key.

```
$age = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
krsort($age); print_r($age);
 // Output: Array ( [Peter] => 35 [Joe] => 43 [Ben] => 37 )
```

7.usort() - Sorts an array by values using a user-defined comparison function.

```
$array = array(3, 2, 5, 6, 1); usort($array, function($a, $b) { return $a <=> $b; // Spaceship operator for comparison }); print_r($array); // Output: Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 5 [4] => 6 )
```

8.uasort() - Sorts an array with a user-defined comparison function and maintains index association.

```
$array = array("a" => 3, "b" => 2, "c" => 5, "d" => 6, "e" => 1);
uasort($array, function($a, $b)
{
return $a <=> $b;
});
 print_r($array);
// Output: Array ( [e] => 1 [b] => 2 [a] => 3 [c] => 5 [d] => 6 )
```

9.uksort() - Sorts an array by keys using a user-defined comparison function.

```
$array = array("d" => 3, "b" => 2, "e" => 5, "a" => 6, "c" => 1);
uksort($array, function($a, $b)
 {
 return strcmp($a, $b);
// String comparison
});
print_r($array);
 // Output: Array ( [a] => 6 [b] => 2 [c] => 1 [d] => 3 [e] => 5 )
```

2.14.1 Natural Order sorting

- The natsort() function sorts an array by using a "natural order" algorithm. The values keep their original keys.
- In a natural algorithm, the number 2 is less than the number 10. In computer sorting, 10 is less than 2, because the first number in "10" is less than 2.

Ex:<?php $array = ["file10.txt", "file2.txt", "file1.txt", "file20.txt", "file11.txt"];

natsort($array);

foreach ($array as $key => $value)

{ echo "$value\n"; } ?>

Output

file1.txt

file2.txt

file10.txt

file11.txt

file20.txt

2.14.2 Sorting multiple array at once

- The array_multisort() is an inbuilt function in PHP which is used to sort multiple arrays at once or a multi-dimensional array with each individual dimension.
  With this function, one should remember that string keys will be maintained, but numeric keys will be re-indexed, starting at 0 and increases by 1.
- **Syntax:**
- *bool* array_multisort($array1, sorting_order, sorting_type, $array2..)
- **Parameters:** The array generally takes one parameter that is the array which needs to be sorted. But in addition, the function can take two more optional parameters sorting_order and sorting_type.
- **$array1**: This parameter specifies the array which we want to sort.
- **sorting_order**: This parameter specifies the order to use i.e. in ascending or descending order. The default value of this parameter is SORT_ASC. That is, sorting in ascending order. In order to sort in descending order we will have to set this parameter to SORT_DESC.

- **sorting_type**: This parameter specifies the sort options for the arrays and they are as follows:
  - SORT_REGULAR: Compare elements regularly(Standard ASCII).
  - SORT_NUMERIC: Compare elements as numeric-values.
  - SORT_STRING: Compare elements as string values.
  - SORT_LOCALE_STRING: Compare elements as string, based on the current locale.
  - SORT_NATURAL: Compare elements as strings using “natural ordering”.
  - SORT_FLAG_CASE: Can be combined (bitwise OR) with SORT_STRING or SORT_NATURAL to sort strings case-insensitively.

```
<?php
  // Input arrays
$array1=array("Dog", "BBB", "Cat");
$array2=array("Pluto", "Fido", "Missy");
  // sorting multiple arrays
array_multisort($array1, SORT_ASC, $array2);
  // Printing sorted arrays
print_r($array1);
print_r($array2);
  ?>
Array ( [0] => BBB [1] => Cat [2] => Dog )
Array ( [0] => Fido [1] => Missy [2] => Pluto )
```

2.14.3 Reversing an array

The most straightforward way to reverse an array in PHP is by using the array_reverse() function. This built-in function takes an array as input and returns a new array with the elements in the opposite order. Here's a simple example:

```
<?php
function Reverse($array)
{return(array_reverse($array));
}
$array = array("ram", "aakash", "saran", "mohan");
echo "Before:\n";
print_r($array);
echo "\nAfter:\n";
print_r(Reverse($array));
?>
```

Output

Before: Array ([0] => ram [1] => aakash [2] => saran [3] => mohan) After: Array ([0] => mohan [1] => saran [2] => aakash [3] => ram)

2.14.4 Randomizing Order

- You can use the PHP shuffle() function to randomly shuffle the order of the elements or values in an array.
- Ex

```
<?php
$numbers = range(1, 10);
shuffle($numbers);
 foreach ($numbers as $value)
{ echo "$value" . "<br>"; }
?>
```

1
6
3
5
7
8
9
4
10
2

2.14.5 Recursive function

- A recursive function in PHP is a function that calls itself to solve a problem. This technique is particularly useful for tasks that can be broken down into smaller, similar tasks. Here's a simple example of a recursive function to calculate the factorial of a number:
- <?php

```
function factorial($n)
{
 if ($n == 1) { return 1; } else { return $n * factorial($n - 1); } }
echo "Factorial of 5 is " . factorial(5);
 // Output: Factorial of 5 is 120 ?>
```

Unit 3

Object Oriented Programming

3.1 Introduction

advantages of Object-Oriented Programming (OOP) in PHP

1. **Modularity**: Organizes code into separate, reusable parts, reducing repetition.
2. **Encapsulation**: Protects data and hides the details, making the code safer and easier to manage.
3. **Abstraction**: Simplifies complex systems by showing only what's needed, making the code easier to use.
4. **Reusability**: Lets you reuse code, saving time and effort.
5. **Flexibility and Scalability**: Makes it easier to add new features and grow the application.
6. **Better Code Organization**: Keeps the code neat and organized by grouping related functions together.
7. **Easier Collaboration**: Helps teams work together by clearly defining different parts of the code.
8. **Error Reduction**: Reduces mistakes by reusing tested code.

3.2 Classes

- **1. Classes**
- A **class** in PHP is like a blueprint or template for creating objects. It defines properties (variables) and methods (functions) that an object created from the class will have.
- A class is defined by using the class keyword.

Ex:<?php

```
class Fruit {
  // code goes here...
}
?>
```

3.2.1 Declaring properties

properties are like variables inside a class. You can declare them with different levels of access (visibility)

```
<?php
class Car
 {
public $make;
protected $model;
private $year;}
```

public: Properties can be accessed from anywhere (inside or outside the class).

protected: Properties can only be accessed within the class and by subclasses (child classes).

private: Properties can only be accessed within the class where they are declared.

3.2.2 Declaring Methods

methods are functions that are declared inside a class. They define actions that an object can perform. Just like properties, methods can have different visibility (public, private, protected).

```php
<?php
class Car {
public $make;
   // A public method to display car's make
   public function displayMake() {
      echo "The car make is: " . $this->make;
   }
   // A private method (only accessible inside the class)
   private function secretMethod() {
      echo "This is a secret method!";
   }
}
?>
```

3.2.3 Static property and Static Method

1. Static Properties in PHP

A **static property** belongs to the class itself, not to any specific object created from the class. All objects of that class share the same static property.

Key Points:

1. **Shared across all objects**: Every instance (object) of the class shares the same static property.
2. **No need to create an object to access**: You can access static properties directly using the class name.
3. **Accessed using self::**: Inside the class, you access static properties using self::$property.

```php
<?php
class Car {
    public static $count = 0; // Static property
    public function __construct() {
        self::$count++; // Increase count when a new object is created
    }
}
// Creating objects
$car1 = new Car();
$car2 = new Car();
echo Car::$count; // Outputs: 2 (Because 2 objects were created)
?>
```

2. Static Methods in PHP

- A **static method** belongs to the class itself and can be called without creating an object. Static methods can only work with static properties and other static methods.
- **Key Points:**
- **Called directly on the class**: You don't need to create an object to call static methods.
- **Can only access static properties**: Static methods can only work with static properties, not instance-specific properties.
- **Accessed using self::**: Inside the class, you access static methods using self::methodName().

```php
<?php
class Car {
    public static $count = 0; // Static property

    // Static method to get the car count
    public static function getCount() {
        return self::$count; // Access static property inside static method
    }

    public function __construct() {
        self::$count++; // Increase count when a new object is created
    }
}
// Creating objects
$car1 = new Car();
$car2 = new Car();

// Calling static method without an object
echo Car::getCount(); // Outputs: 2 (Because 2 objects were created)
?>
```

Key Differences Between Static and Instance (Non-static) Properties/Methods:

Feature	Static Properties/Methods	Non-Static (Instance) Properties/Methods
Belongs to	The class itself	The specific object (instance)
Accessed by	`ClassName::$property` or `ClassName::method()`	`$object->property` or `$object->method()`
Can access instance properties	No, only static properties/methods	Yes, can access instance properties/methods
Used for	Shared data or behavior across all objects	Data or behavior specific to each object

3.3 Object

- An **Object** is an individual instance of the data structure defined by a class. We define a class once and then make many objects that belong to it. Objects are also known as instances.
- **Creating an Object:**
  Following is an example of how to create object using **new** operator.

```
class Books {
 // Members of class Books
} // Creating three objects of
 Books $physics = new Books;
$maths = new Books;
 $chemistry = new Books;
```

- **Member Functions:**
  After creating our objects, we can call member functions related to that object. A member function typically accesses members of current object only.
- Example:

```
$physics->setTitle( "Physics for High School" );
 $chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );
$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Constructors:

A constructor is a key concept in object oriented programming in PHP. Constructor in PHP is special type of function of a class which is automatically executed as any object of that class is created or instantiated.

Constructor is also called magic function because in PHP, magic methods usually start with two underscore characters.

Below is the sample code for the implementation of constructors:

Program for Constructors:

```
<?php
class GeeksforGeeks
{ public $Geek_name = "GeeksforGeeks";
   // Constructor is being implemented.
   public function __construct($Geek_name)
   {
      $this->Geek_name = $Geek_name;
   } }
$Geek = new GeeksforGeeks("GeeksforGeeks");
echo $Geek->Geek_name;
?>
```

destructor

- In PHP, a destructor is a special method that is automatically called when an object is destroyed or when the script execution ends. The destructor method is defined using the __destruct() function. It is particularly useful for cleaning up resources, such as closing database connections or releasing file handles.

Here's a simple example of a destructor in PHP:

```
<?php class Fruit
{ public $name;
public $color;
function __construct($name, $color)
{ $this->name = $name; $this->color = $color; }
 function __destruct()
{
echo "The fruit is {$this->name} and the color is
{$this->color}.";
 } }
$apple = new Fruit("Apple", "red"); ?>
```

3.4 Introspection

- Introspection in PHP offers the useful ability to examine an object's characteristics, such as its name, parent class (if any) properties, classes, interfaces, and methods.

  PHP offers a large number of functions that you can use to accomplish the task.

  The following are the functions to extract basic information about classes such as their name, the name of their parent class and so on.

  In-built functions in PHP Introspection :

Function	Description
class_exists()	Checks whether a class has been defined.
et_class()	Returns the class name of an object.
get parent_class()	Returns the class name of a Return object's parent class.
is_subclass_of()	Checks whether an object has a given parent class.
get_declared_classes()	Returns a list of all declared classes.
get_class_methods()	Returns the names of the class methods.
get_class_vars()	Returns the default properties of a class
interface_exists()	Checks whether the interface is defined.
method_exists()	Checks whether an object defines a method.

3.5 Serialization in PHP:

Serialization is a technique used by programmers to preserve their working data in a format that can later be restored to its previous form.

- Serializing an object means converting it to a byte stream representation that can be stored in a file. Serialization in PHP is mostly automatic, it requires little extra work from you, beyond calling the serialize() and unserialize() functions.

Serialize():
The serialize() converts a storable representation of a value.
The serialize() function accepts a single parameter which is the data we want to serialize and returns a serialized string.

- A serialize data means a sequence of bits so that it can be stored in a file, a memory buffer, or transmitted across a network connection link.
- It is useful for storing or passing PHP values around without losing their type and structure.

Syntax:

serialize(value);

unserialize():

unserialize() can use string to recreate the original variable values i.e. converts actual data from serialized data.

Syntax:

unserialize(string);

Example:

3.6 Inheritance

Inheritance is a fundamental object-oriented programming concept in PHP where a class (subclass or child class) can inherit properties and methods from another class (superclass or parent class). It enables code reusability and promotes hierarchical relationships between classes.

- We can extends the features of a class by using 'extends' keyword.
- It supports the concept of **hierarchical classification**.
- Inheritance has three types, **single, multiple and multilevel Inheritance**.
- **PHP** supports only **single inheritance**, where only one class can be **derived from single parent class**.
- We can simulate multiple inheritance by using **interfaces**.

1.Single inheritance

```
<?php
   class demo
   {
      public function display()
      {
         echo "example of inheritance  ";
      }
   }
   class demo1 extends demo
   {
      public function view()
      {
         echo "in php";
      }
   }
   $obj= new demo1();
   $obj->display();
   $obj->view();
?>
```

Output example of inheritance in php

- **2. Multiple Inheritance (Simulated via Interfaces or Traits)**
- PHP doesn't directly support multiple inheritance (where a class can inherit from more than one class). However, PHP allows you to use interfaces or traits to simulate multiple inheritance.
- Multiple Inheritance with Interfaces:
- A class can implement multiple interfaces.

```
<?php
interface Animal {
public function speak();
}
interface Pet {
public function play();
}
class Dog implements Animal, Pet
 {   public function speak()
{       echo "Dog barks\n";
 }   public function play()
 {       echo "Dog plays\n";
  }}
$dog = new Dog();
$dog->speak();  $dog->play(); ?>
```

3. multilevel inheritance

in multilevel inheritance, a class can inherit from a parent class, and then a child class can inherit from that subclass.

```php
<?php
class Animal {
    public function eat() {
        echo "Animal eats\n";
    }
}

class Dog extends Animal {
    public function bark() {
        echo "Dog barks\n";
    }
}

class Puppy extends Dog {
    public function play() {
        echo "Puppy plays\n";
    }
}

$puppy = new Puppy();
$puppy->eat();   // From Animal class
$puppy->bark();  // From Dog class
$puppy->play();  // From Puppy class
?>
```

4.Final Method

- **final method** is a method that cannot be overridden by any child class. Once a method is marked as final in a parent class, it becomes fixed, and any attempt to override that method in a subclass will result in a **fatal error**.
- **Key Points:**
- **Final Method**: A method declared as final cannot be overridden by child classes.
- **Purpose**: This is useful when you want to ensure a specific method's behavior is not changed in subclasses.
- **Final Class**: Similarly, a final class cannot be extended by any other class.

Example of a Final Method in PHP:

```
<?php
Class A
{
Private $msg=“sdfgsjkjfgh”;
Final function display()
{
Echo $this->msg
}}
Class B extends A
{
Function dis(){
Echo”hrkethrekjth”;
}}
$abc=new B;
$abc->dis();
?>
```

5.Abstract Classes and Methods with Inheritance:

- **Abstract Class**: An **abstract class** cannot be instantiated directly. It can contain both abstract methods (which have no implementation) and regular methods (which do have implementation). It is meant to be inherited by other classes.
- **Abstract Method**: An **abstract method** is a method declared in an abstract class but without any implementation. Any class that inherits from the abstract class **must implement** these abstract methods.
- **How Abstract Classes and Methods Work in Inheritance:**
- **Parent Abstract Class**: Defines the structure (methods) that must be inherited by its subclasses.
- **Child Class**: Inherits from the abstract class and provides the specific implementation of abstract methods.

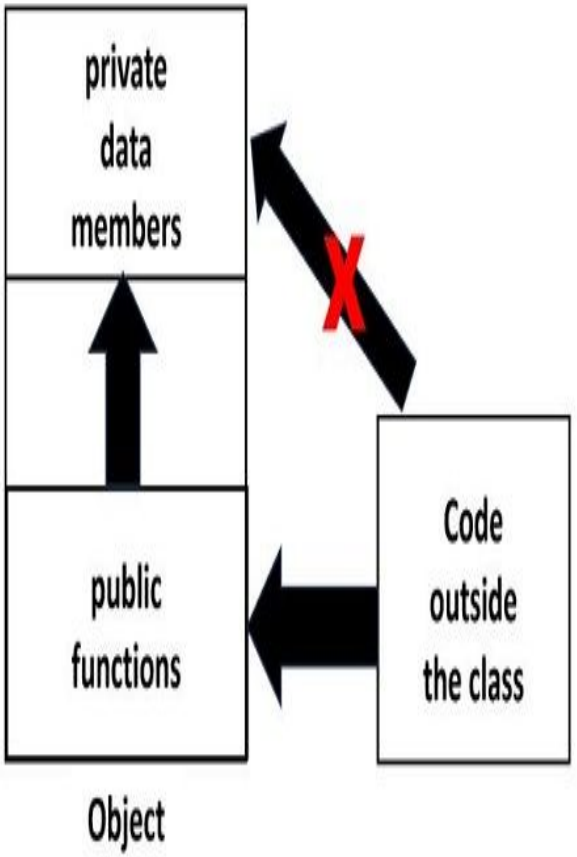
```
<?php
abstract class Base {
    // This is abstract function
    abstract function printdata();
}
class Derived extends base {
    function printdata() {
      echo "Derived class";
    } }
$b1 = new Derived;
$b1->printdata();
?>
```

3.7 Interfaces

- You may use interfaces in PHP to define which methods a class should implement. Interfaces make it easy to use several classes at the same time. The term "polymorphism" refers to when two or more groups share the same interface.
- A PHP interface defines a contract which a class must fulfill. If a PHP class is a blueprint for objects, an interface is a blueprint for classes. Any class implementing a given interface can be expected to have the same behavior in terms of what can be called, how it can be called, and what will be returned.
- characteristics of an Interface are:
- An interface consists of methods that have no implementations, which means the interface methods are abstract methods.
- All the methods in interfaces must have public visibility scope.
- Interfaces are different from classes as the class can inherit from one class only whereas the class can implement one or more interfaces.
- Example:

3.8 Encapsulation

- **Data Hiding and Abstraction:** Unnecessary details, internal representation and implementation are hidden from the end-users for protection of data structure and the Class. Data access is prohibited from members of other classes by creating private methods. It protects the internal state of any object by keeping member variables private and preventing any inconsistent state. It is the enclosing of data and related operations into that object.
- **Data security:** Encapsulation helps in making data very robust and secured, as the data and member functions are wrapped together to form an object. All the tasks are done inside without any external worry and it also makes life very easy.
- **Reduces complexity:** Encapsulation helps in reducing development complexity of the software by hiding the details of implementation and exposing the methods or operations.
- **Reusability:** There are instances, you don't have to re-write same functionality that you inherited from the parent class.
- **Reliability:** You can make the class read-only or write-only by writing SET or GET methods.
- **Easier testing of code:** Encapsulated PHP code is easy to test as the functions used for testing child class ensures the testing of parent class functions also.
- **Increased flexibility:** Class variables can be accessed by GET or SET methods increasing flexibility. It is easy to maintain as the internal implementation can be changed without changing the code.



- A Class is kind of a container or capsule which contains properties and a set of methods to manipulate these properties. This is called as encapsulation. It binds together code and data it manipulates.

Function Overloading

In function overloading, more than one function can have same method signature but different number of arguments.

Function overloading contains same function name and that function performs different task according to number of arguments.

1. __call($name, $arguments): This method is invoked when you call a method that is not defined in the class.
2. __callStatic($name, $arguments): This method is invoked when you call a static method that is not defined in the class.
3. __get($name): Allows access to inaccessible or undefined properties.
4. __set($name, $value): Allows setting of inaccessible or undefined properties.
5. __isset($name): Checks if a property is set.
6. __unset($name): Unsets a property.

Function Overriding

Function overriding is same as other OOPs programming languages. In function overriding, both parent and child classes should have same function name with and number of arguments.

It is used to replace parent method in child class.

The purpose of overriding is to change the behavior of parent class method. The two methods with the same name and same parameter is called overriding.

	Abstract Class	Interface
Type of methods	Abstract class can have abstract and non-abstract methods	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
Type of variables	Abstract class can have final, non-final, static and non-static variables	Interface has only static and final variables.
Implementation	Abstract class can provide the implementation of the interface.	Interface can’t provide the implementation of an abstract class.
Inheritance vs Abstraction	An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
Multiple implementations	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
Multiple Inheritance	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Accessibility of Data Members	A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

Feature	Abstract Class	Final Class
Definition	Cannot be instantiated, meant to be subclassed	Cannot be subclassed once declared
Keyword	`abstract`	`final`
Usage	Defines common properties/methods to be inherited, may have abstract methods	Used to prevent class inheritance, ensures class immutability
Abstract Methods	Can have abstract methods that must be implemented in derived classes	Cannot have abstract methods
Instantiation	Cannot be instantiated directly	Can be instantiated directly
Inheritance	Can be extended by other classes	Cannot be extended by any other classes
Example		
	`php abstract class Vehicle {  abstract public function move();  }`	`php final class Car {  public function drive() {  echo "Driving";  } }`

STATIC VERSUS FINAL

STATIC	FINAL
A keyword that denotes a member variable or a method that can be accessed without requiring an instantiation of the class to which it belongs	A keyword that denotes an entity that can only be assigned once
Variables can be initialized again	Variables cannot be initialized again
Methods can be called by other static methods and only access the static members of the class	Methods cannot be overridden
An object cannot be created from a static class; it only consists of static members	Final classes cannot be inherited by other classes

Unit 4

Web Techniques

4.1 Variables

Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

1.$_COOKIE

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax :-setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

2. $_GET

$_GET contains an array of variables received via the HTTP GET method.There are two main ways to send variables via the HTTP GET method:

1. Query strings in the URL

```
<html><body><?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?></body></html>
```

1. HTML Forms

HTML Form

```
<html>
<body><form action="welcome_get.php" method="GET">
 </form></body> </html>
```

3.$_POST

A HTML form submits information via the HTTP POST method if the form's method attribute is set to "POST".

To demonstrate this, we start by creating a simple HTML form:

```
<html><body>

<form method="POST" action="demo_request.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form></body></html>
```

4.$_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

5.$_FILES

$_FILES is one of the 'superglobal', or automatic global, variables in PHP. It is available in all scopes throughout a script. The variable $_FILES is an associative array containing items uploaded via HTTP POST method.

A file is uploaded when a HTML form contains an input element with a file type, its enctype attribute set to multipart/form-data, and the method attribute set to HTTP POST method.

$HTTP_POST_FILES also contains the same information, but it is not a superglobal, and it has now been deprecated.

The $_FILES array contains the following properties −

- **$_FILES['file']['name']** − The original name of the file that the user has chosen to be uploaded.
- **$_FILES['file']['type']** − The mime type of the file. An example would be "image/gif". This mime type is however not checked on the PHP side.
- **$_FILES['file']['size']** − The size, in bytes, of the uploaded file.
- **$_FILES['file']['tmp_name']** − The temporary filename of the file in which the uploaded file was stored on the server.
- **$_FILES['file']['full_path']** − The full path as submitted by the browser. Available as of PHP 8.1.0.
- **$_FILES['file']['error']** − The error code associated with this file upload.

```
<input type="file" name="file">
<?php
   echo "Filename: " . $_FILES['file']['name']."<br>";
   echo "Type : " . $_FILES['file']['type'] ."<br>";
   echo "Size : " . $_FILES['file']['size'] ."<br>";
   echo "Temp name: " . $_FILES['file']['tmp_name'] ."<br>";
   echo "Error : " . $_FILES['file']['error'] . "<br>";
?>
```

6**$_ENV** : Itis a superglobal variable in PHP. It is an associative array that stores all the environment variables available in the current script. **$HTTP_ENV_VARS** also contains the same information, but it is not a superglobal, and it has now been deprecated.

The environment variables are imported into the global namespace. Most of these variables are provided by the shell under which the PHP parser is running. Hence, the list of environment variables may be different on different platforms.

4.3 SERVER Information

Element/Code	Description
`$_SERVER['PHP_SELF']`	Returns the filename of the currently executing script
`$_SERVER['GATEWAY_INTERFACE']`	Returns the version of the Common Gateway Interface (CGI) the server is using
`$_SERVER['SERVER_ADDR']`	Returns the IP address of the host server
`$_SERVER['SERVER_NAME']`	Returns the name of the host server (such as www.w3schools.com)
`$_SERVER['SERVER_SOFTWARE']`	Returns the server identification string (such as Apache/2.2.24)
`$_SERVER['SERVER_PROTOCOL']`	Returns the name and revision of the information protocol (such as HTTP/1.1)
`$_SERVER['REQUEST_METHOD']`	Returns the request method used to access the page (such as POST)
`$_SERVER['REQUEST_TIME']`	Returns the timestamp of the start of the request (such as 1377687496)
`$_SERVER['QUERY_STRING']`	Returns the query string if the page is accessed via a query string
`$_SERVER['HTTP_ACCEPT']`	Returns the Accept header from the current request
`$_SERVER['HTTP_ACCEPT_CHARSET']`	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)

`$_SERVER['HTTP_HOST']`	Returns the Host header from the current request
`$_SERVER['HTTP_REFERER']`	Returns the complete URL of the current page (not reliable because not all user-agents support it)
`$_SERVER['HTTPS']`	Is the script queried through a secure HTTP protocol
`$_SERVER['REMOTE_ADDR']`	Returns the IP address from where the user is viewing the current page
`$_SERVER['REMOTE_HOST']`	Returns the Host name from where the user is viewing the current page
`$_SERVER['REMOTE_PORT']`	Returns the port being used on the user's machine to communicate with the web server
`$_SERVER['SCRIPT_FILENAME']`	Returns the absolute pathname of the currently executing script
`$_SERVER['SERVER_ADMIN']`	Returns the value given to the SERVER_ADMIN directive in the web server configuration file
`$_SERVER['SERVER_PORT']`	Returns the port on the server machine being used by the web server for communication (such as 80)
`$_SERVER['SERVER_SIGNATURE']`	Returns the server version and virtual host name which are added to server-generated pages

`$_SERVER['PATH_TRANSLATED']`	Returns the file system based path to the current script
`$_SERVER['SCRIPT_NAME']`	Returns the path of the current script
`$_SERVER['SCRIPT_URI']`	Returns the URI of the current page

$_SERVER in PHP is a superglobal variable. This variable is an associative array that stores the information about the server, path, and script. PHP provides various parameters that can be passed in $_SERVER like $_SERVER['PHP_SELF'], $_SERVER['SERVER_PROTOCOL'], etc. These parameters decide the behavior of the $_SERVER variable.
$_SERVER array contains values for the meta variable listed with the specification of CGI(Common gateway interface).

4.4 Processing Forms

Controls used in forms: Form processing contains a set of controls through which the client and server can communicate and share information. The controls used in forms are:

- **Textbox:** Textbox allows the user to provide single-line input, which can be used for getting values such as names, search menu and etc.
- **Textarea:** Textarea allows the user to provide multi-line input, which can be used for getting values such as an address, message etc.
- **DropDown:** Dropdown or combobox allows the user to provide select a value from a list of values.
- **Radio Buttons:** Radio buttons allow the user to select only one option from the given set of options.
- **CheckBox:** Checkbox allows the user to select multiple options from the set of given options.
- **Buttons:** Buttons are the clickable controls that can be used to submit the form.

Form Validation: Form validation is done to ensure that the user has provided the relevant information. Basic validation can be done using HTML elements. For example, in the above script, the email address text box is having a type value as “email”, which prevents the user from entering the incorrect value for an email. Every form field in the above script is followed by a required attribute, which will intimate the user not to leave any field empty before submitting the form. PHP methods and arrays used in form processing are:

- **isset():** This function is used to determine whether the variable or a form control is having a value or not.
- **$_GET[]:** It is used the retrieve the information from the form control through the parameters sent in the URL. It takes the attribute given in the url as the parameter.
- **$_POST[]:** It is used the retrieve the information from the form control through the HTTP POST method. IT takes name attribute of corresponding form control as the parameter.
- **$_REQUEST[]:** It is used to retrieve an information while using a database.

$_GET

The $_GET variable is used to get data from a form that is written in HTML. Also in the url $_GET variable will display the data that was taken by the $_GET variable.For example using the second example on this page it will display in the url as ?name='it will equal to what text was entered in the text box'. $_GET has limits on the amount of information that can be sent.

How to use it

Before you can use the the $_GET variable you have to have a form in html that has the method equal to GET. Then in the php, you can use the $_GET variable to get the data that you wanted. The $_GET syntax is ($_GET['name of the form field goes here']).

$_POST

The $_POST variable is also used to collect data from forms, but the $_POST is slightly different because in $_GET it displayed the data in the url and $_POST does not.
The data that $_POST gets, is invisible to others and the amount that can be sent is not limited besides the 8MB max size.

How to use it?

Before you can use the the $_POST variable you have to have a form in html that has the method equal to POST. Then in the php, you can use the $_POST variable to get the data that you wanted. The $_POST syntax is ($_POST['name of the form field goes here']).

$_REQUEST

The $_REQUEST variable is a variable with the contents of $_GET and $_POST and $_COOKIE variables.

How to use it?

Before you can use the the $_REQUEST variable you have to have a form in html that has the method equal to GET and POST. Then in the php, you can use the $_REQUEST variable to get the data that you wanted. Depending on what you wrote for the method in the form and using $_REQUEST in the php, $_REQUEST will use $_Get if GET is written for the method and $_REQUEST will use $POST if POST is written in the method. The $_REQUEST syntax is ($_REQUEST['name of the form field goes here']).

isset()

The isset function in PHP is used to determine whether a variable is set or not. A variable is considered as a set variable if it has a value other than NULL. In other words, you can also say that the isset function is used to determine whether you have used a variable in your code or not before. It is a boolean-type function. It means that if a variable, array, or array key is not set or it has a NULL value, the isset in PHP will return false, otherwise, you will get true as the return value. You can pass multiple arguments in the isset function and it will check for all the passed parameters whether or not they are set. In this case, the isset function will return true only if all the passed arguments are set. If any of the arguments have a NULL value, the isset function will return false.

4.4.1 Self processing Pages

"Self-processing forms" in PHP refer to forms that submit data to the same PHP script that generates the form. In other words, the form and the PHP processing logic are contained in the same file. When the user submits the form, the same PHP script is responsible for handling the form data, performing any necessary processing, and displaying the results.

- **$_SERVER[‘PHP_SELF’]:** The **$_SERVER[“PHP_SELF”]** is a super global variable that returns the filename of the currently executing script. It sends the submitted form data to the same page, instead of jumping on a different page.

PHP_SELF Exploits can be avoided by following function

1.htmlspecialcharacters(): The **htmlspecialchars()** function converts special characters to HTML entities. It will replace HTML characters like with < and >. This prevents scripting attacks by attackers who exploit the code by inserting HTML or Javascript code in the form fields.

2.htmlentities():htmlentities() Function

The htmlentities() function is an inbuilt function in PHP that is used to transform all characters that apply to HTML entities. It is used when additional character encoding is required.

4.4.2 Sticky forms

A sticky form is simply a standard HTML form that remembers how you filled it out. This is a particularly nice feature for end users, especially if you are requiring them to resubmit a form. A sticky form in PHP is a form that remembers the information you entered, even if there's an error and the page reloads. This makes it easier for users to correct mistakes because they don't have to fill out the entire form again.

Steps to make a sticky form:

- Create the form: You have a regular HTML form where users can enter data (like name, email).
- Check if the form is submitted: Use PHP to check if the user has submitted the form.
- Validate the data: Check if the fields are empty or incorrect. If so, show an error message.
- Show the entered values again: If there was an error, PHP will put the data the user entered back into the form, so they don't need to retype everything.

4.4.3 Sticky multi values parameters

So now you're wondering, can I make multiple selection form elements sticky? You can, but it isn't easy. You'll need to check to see whether each possible value in the form was one of the submitted values.

4.5 Setting Response Headers

The HTTP response that a server sends back to a client contains headers that identify the type of content in the body of the response, the server that sent the response, how many bytes are in the body, when the response was sent, etc. PHP and Apache normally take care of the headers for you, identifying the document as HTML, calculating the length of the HTML page, and so on. Most web applications never need to set headers themselves. However, if you want to send back something that's not HTML, set the expiration time for a page, redirect the client's browser, or generate a specific HTTP error, you'll need to use the header( ) function.

Content-Type

The Content-Type header identifies the type of document being returned. It is normally "text/html" but you can specify "text/plain" also.

```
<?
header(“content-type:text/plain”);
?>
```

Redirections

Redirections Will send the browser to a new URL, known as a redirection , you set the Location header.

<?php

‘location:”https://www.google.com”’

?>

Expiration

If you provide a partial URL (e.g., "/elsewhere.html"), the redirection is handled internally by the web server. This is only rarely useful, as the browser generally won't learn that it isn't getting the page it requested.

A server can explicitly inform the browser, and any proxy caches that might be between the server and browser, of a specific date and time for the document to expire. Proxy and browser caches can hold the document until that time or expire it earlier. Repeated reloads of a cached document do not contact the server. However, an attempt to fetch an expired document does contact the server.

To set the expiration time of a document, use the Expires header

```
header('Expires: Fri, 18 Jan 2002 05:30:00 GMT');
```

To expire the web page before 3 hours of page generation time use time() method and gmstrftime() to generate expiration time

```
$now = time( );
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now +
60*60*3);
header("Expires: $then");
```

Http Authentication

It is possible to use the header() function to send an "Authentication Required" message to the client browser causing it to pop up a Username/Password input window. Once the user has filled in a username and a password, the URL containing the PHP script will be called again with the predefined variables *PHP_AUTH_USER*, *PHP_AUTH_PW*, and *AUTH_TYPE* set to the user name, password and authentication type respectively. These predefined variables are found in the *$_SERVER* array. *Only* the "Basic" authentication method is supported. See the header() function for more information.

4.6 Maintaining state

HTTP is a stateless protocol, which means that once a web server completes a client's request for a web page, the connection between the two goes away. In other words, there is no way for a server to recognize that a sequence of requests all originate from the same client.
To keep track of state information between requests and to maintain state cookies are used.
A cookie is a bit of information that the server can give to a client. On every subsequent request the client will give that information back to the server, thus identifying itself. Cookies are useful for retaining information through repeated visits by a browser, but they're not without their own problems. The main problem is that some browsers don't support cookies, and even with browsers that do, the user can disable cookies. So any application that uses cookies for state maintenance needs to use another technique as a fallback mechanism. We'll discuss cookies in more detail shortly.

Cookie

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Key Cookie Attributes:

- **Expires**: This attribute sets the expiration date and time for a cookie. After this point, the cookie becomes invalid and is automatically deleted by the browser. Cookies can be set to expire at a specific date or when the browser session ends.
- **Path**: It defines the URL path for which the cookie is accessible. This attribute allows you to specify a directory or a part of the website where the cookie should be available. It helps in segmenting cookies for different purposes.
- **Domain**: The “domain” attribute specifies the domain (e.g., example.com) for which the cookie is valid. This attribute is useful when you want a cookie to be accessible across subdomains, enabling seamless user experiences.
- **Secure**: When this attribute is set to true, ensures that the cookie is only transmitted over secure (HTTPS) connections. It enhances the security of sensitive information stored in cookies.

Create a Cookie in PHP

To set a cookie in PHP, The setcookie() function takes parameters to define the cookie properties such as name, value, expiration time, path, domain, and security.

```
setcookie("username", "John", time() + 3600, "/");

setcookie("user_id", "12345", time() + (86400 * 30), "/");//86400 sec=1 day
```

This example sets a cookie named “username” with the value “John” that expires in an hour, is accessible from the root directory, is associated with the domain “example.com”, and is marked as secure.

Delete a Cookie

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

```
<?php// set the expiration date to one hour ago

setcookie("user", "", time() - 3600);

?><html><body>

<?php

echo "Cookie 'user' is deleted.";?></body></html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Ex:

```
<?php

setcookie("test_cookie", "test", time() + 3600, '/');?>

<html><body><?php

  if(count($_COOKIE) > 0) {echo "Cookies are enabled.";

  } else {echo "Cookies are disabled."; }?></body></html>
```

Session

What is a Session?

- A session is a global variable stored on the server.
- Each session is assigned a unique id which is used to retrieve stored values.
- Whenever a session is created, a cookie containing the unique session id is stored on the user's computer and returned with every request to the server. If the client browser does not support cookies, the unique php session id is displayed in the URL
- Sessions have the capacity to store relatively large data compared to cookies.
- The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database.
- Just like the $_COOKIE array variable, session variables are stored in the $_SESSION array variable. Just like cookies, the session must be started before any HTML tags.

Why and when to use Sessions?

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

Creating a Session

In order to create a session, you must first call the PHP session_start function and then store your values in the $_SESSION array variable.

Let’s suppose we want to know the number of times that a page has been loaded, we can use a session to do that.

Destroying Session Variables

The session_destroy() function is used to destroy the whole PHP session variables.

If you want to destroy only a session single item, you use the unset() function.

Session in PHP	Cookie in PHP
A session is a server-side storage mechanism that keeps track of user interactions and data during their visit to a website, allowing the server to maintain state across multiple requests from the same user.	Cookies are small pieces of data stored on the client side within the user's browser, used to remember information about the user, such as login status or site preferences, across multiple visits to a website.
Stored on the server.	Stored on the client's browser.
Accessible only through PHP on server.	Accessible by client-side scripts and PHP.
Lasts until the browser is closed or session times out.	Expires based on the specified duration, can persist beyond browser sessions.
Generally, no practical limit. Depends on server's memory.	Limited to about 4KB per cookie.
More secure, as it's not exposed to the client-side.	Less secure, vulnerable to client-side access and manipulation.
Ideal for sensitive information, like user authentication details.	Suitable for less sensitive data like user preferences or settings.
Can handle large amounts of data.	Limited capacity, suitable for small pieces of data.
Less dependent. Sessions work even if cookies are disabled in the browse (using URL rewriting).	Highly dependent. If cookies are isabled in the browser, cookies won't work.

Additional Session Functions

- session_abort — Discard session array changes and finish session
- session_cache_expire — Get and/or set current cache expire
- session_cache_limiter — Get and/or set the current cache limiter
- session_commit — Alias of session_write_close
- session_create_id — Create new session id
- session_decode — Decodes session data from a session encoded string
- session_destroy — Destroys all data registered to a session
- session_encode — Encodes the current session data as a session encoded string
- session_gc — Perform session data garbage collection
- session_get_cookie_params — Get the session cookie parameters
- session_id — Get and/or set the current session id
- session_module_name — Get and/or set the current session module
- session_name — Get and/or set the current session name
- session_regenerate_id — Update the current session id with a newly generated one
- session_register_shutdown — Session shutdown function
- session_reset — Re-initialize session array with original values
- session_save_path — Get and/or set the current session save path
- session_set_cookie_params — Set the session cookie parameters
- session_set_save_handler — Sets user-level session storage functions
- session_start — Start new or resume existing session
- session_status — Returns the current session status
- session_unset — Free all session variables
- session_write_close — Write session data and end session

SSL

The Secure Sockets Layer (SSL) provides a secure channel over which regular HTTP requests and responses can flow. PHP doesn't specifically concern itself with SSL, so you cannot control the encryption in any way from PHP. An https:// URL indicates a secure connection for that document, unlike an http:// URL.

The HTTPS entry in the $_SERVER array is set to 'on' if the PHP page was generated in response to a request over an SSL connection. To prevent a page from being generated over a non encrypted connection, simply use.

```
if ($_SERVER{'HTTPS'] !== 'on')
{ die("Must be a secure connection."); }
```

A common mistake is to send a form over a secure connection (e.g., https://www.example.com/form.html), but have the action of the form submit to an http:// URL. Any form parameters entered by the user are sent over an insecure connection—a trivial packet sniffer can reveal them.
You can add SSL Certificate to site with following steps
1.set up SSL Certificate signing request.
2.Create SSL certificate
3.Install SSL on PHP on Cloudways

Unit 5

Databases

Introduction

- A database acts as a backend. Database stores a large amount of related data. A collection of related tables forms a database.
- Each row in a table is called a record and each column in a record is called a field. Using PHP, we can add records, retrieve records, update records of the table in the database.
- PHP supports databases like Ingres, Oracle, dBase, InterBase, PostgreSQL, MySQL, Sybase, IBM DB2 etc.

Using PHP To Access A Database:

There are two methods to access databases from PHP. These are:

(i) Database Specific Method

(i) Database Independent Method (PEAR DB)

Steps to open and interact with a database through PHP:

1. Open a connection to the database by passing a connection string (two methods have different syntax).

In case of database specific method, we use pg_connect() function for connecting to PostgreSQL database. In case of database independent method (PEAR (PHP extension and application repository) DB), we use DB::connect().

For example,

(i) Database Specific Method:

```
$db=pg_connect("host=localhost dbname=sybca user=postgres password=postgres") or die("Cannot Connect");
```

(ii) Database Independent Method:

```
$db=DB::connect("pgsql://postgres: root@localhost/sybca");
```

2. Execute the query on the connection created in first step. For database specific method, pg_query() is used. For database independent method, query() is used.

2. Process the result set obtained after executing the query. For retrieval queries, we need to process the result. This is because the query may return more than one record in the result set. For both the methods, there are various functions.

For example, pg_fetch_row() is used for database specific method, fetchRow() is used for database independent method.

4. Release the resources. This means free the result variable and free the connection resource. For database specific method, pg_free_result() and pg_close() functions are used. For database independent method, free() and disconnect() functions are used.

Database Specific Method

As there is a database specific prefix before the function, it is easy to identify the database specific method.

Opening a Database Connection:

There are database specific prefixes to connect to a specific database. For example if we want to connect to PostgreSQL database we use pg_connect() function. If we want to connect to MySQL database we use mysql_connect() function.

Syntax: resource pg_connect(string $conn_string)

The $conn_string can have one or more parameter setting separated by white space. The list of parameters is host, port, dbname, user, password. On success, the pg_connect() function returns PostgreSQL connection resource. On failure, it returns false.

For example,

```
$db=pg_connect("host=localhost port=5432 dbname=sybca user=postgres password=root") or die("Cannot connect");
```

In the above example, we connect to a database named "sybca".

```
$db=pg_connect("dbname=sybca user=postgres password=root") or die("Cannot connect");
```

In the above example, we connect to a database named "sybca". Here, the default values are taken for the remaining parameters.

Closing a Database Connection

pg_close() function is used to close a PostgreSQL connection.

Syntax:
bool pg_close([resource $conn_resource]);

The $conn_resource is a PostgreSQL database connection resource. Generally pg_close() is not required as non-persistent connections are closed at the end of the script automatically.

The pg close() function returns true on success or false on failure.

PHP program to check whether a connection to a database is successful or not.

```
<?php

$db=pg_connect("host=localhost port=5432 dbname=sybca user=postgres password=root") or die("Cannot connect");

echo "Connection is successfully done";
pg_close($db);

?>
```

Executing a Query on a Database Connection

pg_query() function executes a query on the specified database connection.

Syntax:

resource pg_query([resource $conn_resource], string $query_string);

If the first parameter is omitted, the default connection is used. But it is not recommended to keep the first parameter optional, as it can lead to the problems which can be hard to debug. The function returns query result resource on success or false on failure.

For example,

$result=pg_query($db, "select * from student") or die("Cannot execute");

Processing the result set

1. pg_fetch_row:

- The pg_fetch_row() function returns the values of all the columns in a record.

Syntax: array pg_fetch_row(resource $result [, integer $row])

- The fields are indexed with number starting with zero(0).

The second parameter is optional.

If pg_fetch_row() is called with only first parameter, it returns the next row or false when no more rows remain in the result set.

The second parameter is the row number in the result to fetch. The rows are numbered in the increasing order starting from zero(0).

PHP program to display students studying in ‘sybca' class using pg_fetch row() function.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres") or die("Cannot
connect");
$result=pg_query($db, "select * from student") or die("Cannot execute");

while($row=pg_fetch_row($result))

{
echo "Roll No.: $row[0] Name: $row[1]";
}
pg_free_result($result);
pg_close($db);
?>
```

Output:

Roll No. : 1 Name : Amit Roll No. : 2
Name : Akash Roll No. : 3 Name : Yash
Roll No. : 4 Name : Sagar Roll No. : 5
Name : Pranav

pg_fetch_array

- The pg_fetch_array() function returns the row as an array. It returns an array containing each field value for the given row.

Syntax:

- array pg_fetch_array(resource $row, $result[,integer integer $result_type])

- Every call to pg_fetch_array() function returns the next row, or false when there are no more rows are remaining. The optional parameter $result_type gives you option to specify the type of returned array.

- $result_type takes the following values: PGSQL_NUM, PGSQL_ASSOC, PGSQL_BOTH. If we use PGSQL_NUM, it returns an indexed array with numerical indices.

- If we use PGSQL_ASSOC, it returns an array with associative array with indices having field names. If we use PGSQL_BOTH, it returns both numerical and associative indices.

- pg_fetch_array() is an extended version of pg_fetch_row(). In addition to storing the data in the numeric indices (field number) to the result array, it can also store the data using associative indices (field name). It stores both indices by default.

PHP program to display the students studying in 'sybca' class using pg_fetch_array() function.

```
<?php
$db=pg_connect("host=localhost     dbname=sybca  user=postgres    ")
      or
die("Cannot connect");
$result=pg_query($db,  "select  *     from     student  where
cls_name='sybca'") or die("Cannot execute");
while($row=pg_fetch_array($result))
echo "Roll No.: $row[0] Name: $row[1]";
}
pg_free_result($result);
pg_close($db);
?>
```

Output:

Roll No. : 4 Name : Sagar

Roll No. : 5 Name : Pranay

pg_fetch_assoc

- The pg_fetch_assoc() function returns the row as an associative array. It is equivalent to pg_fetch_array() with PGSQL_ASSOC as the optional third parameter.

Syntax:

- array pg_fetch_assoc(resource $result[, int $row])

PHP program to display the students studying in 'sybca' class using pg_fetch_assoc() function.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres") or die("Cannot connect");
$result=pg_query($db, "select rno,sname from student where cls_name='sybca'") or die("Cannot execute");
while($row=pg_fetch_assoc($result))
{
echo "Roll No.: $row['rno'] Name: $row[‘sname']";
}
pg_free_result($result);
pg_close($db);
?>
```

Output:

Roll No. : 4 Name : Sagar

Roll No. : 5 Name : Pranav

pg_fetch_object

- The pg_fetch_object() function fetches the row as an object.

Syntax:

object pg_fetch_object(resource $result[, integer $row])

- pg_fetch_object() returns an object with properties that are nothing but fetched row's field names. The $row is the row number in the result to fetch.

- The rows are numbered in increasing order starting from zero(0). If called with single parameter each call to pg_fetch_object() function returns the next row or false when no more rows remain in the result set.

PHP program to display student information of students studying in 'sybca' class using pg_fetch_object() function.

```
<?php
$db=pg_connect("host=localhost                    user=postgres") or
dbname=sybca
$result=pg_query($db, "select rno,sname from student where
die("Cannot connect");
cls_name='sybca'") or die("Cannot execute");
while($row=pg_fetch_object($result))
{
echo "Roll No.: $row->rno Name: $row->sname";
}
pg_free_result($result);
pg_close($db);
?>
```

Output:

Roll No. : 4 Name : Sagar

Roll No. : 5 Name : Pranav

pg_fetch_result

- The pg_fetch_result() function returns the value of a user specified row and column in a result set.

Syntax:

- string pg_fetch_result(resource $result, integer $row, mixed $column)
- string pg_fetch_result(resource $result, mixed $column)

- $row is the row number in the result to fetch. The rows are numbered from zero(0) onwards.

- If $row parameter is not mentioned, next row is fetched.

- $column is either a string representing the name of the column to fetch or an integer describing which column number to fetch. Columns are numbered in increasing order starting from zero(0).

PHP program to display the class name of first student and value in the second column of the first student studying in 'sybca'.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres") or die("Cannot
connect");

$result=pg_query($db, "select rno, sname, cls_name='sybca'") or die("Cannot
execute");"
//Display the "class" of "first"
student echo pg_fetch_result($result,
0, 2)); echo "<br>";
//Display the value in the "second" column (name) of the first
student
echo pg_fetch_result($result, 1);
pg_free_result($result);
pg_close($db);
?>
```

cls_name from student where

Output:

sybca
Sagar

pg_fetch_all

- The pg_fetch_all() function returns all rows from a result in the form of an array.

- PHP assigns each row with an integer starting with zero(0). Each element is itself an array in which the column names are used as indices.

Syntax: array pg_fetch_all(resource $result)

PHP program to display the student details who are studying in 'sybca' using pg_fetch_all() function.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres ") or die("Cannot connect");
$result=pg_query($db, "select rno, name, cls_name from student where cls_name='sybca'") or die("Cannot execute");
print_r(pg_fetch_all($result));
pg_free_result($result);
pg_close($db);
?>
```

Output:

Array([0]=>Array([rno]=>4 [sname]=>Sagar [cls_name]=>sybca
[1]=> Array([rno]=>5 [sname]=>Pranav [cls_name]=>sybca))

pg_num_fields

This function returns total number of columns in a result set. On error it returns -1.

Syntax: integer pg_num_fields(resource $result)

PHP program to display the total number of columns returned by a query containing the students studying in 'sybca' class.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres ") or die("Cannot
    connect");
$result=pg_query($db, "select rno, sname, cls_name from student
where cls_name='sybca'") or die("Cannot execute");
echo "Total number of columns:
".pg_num_fields($result); pg_free_result($result);
pg_close($db);
?>
```

Output:

Total number of columns: 3

pg_num_rows

This function returns total number of rows in a result set. On error, it returns -1
Syntax: integer pg_num_rows(resource $result);

PHP program to display total number of rows returned by a query containing students studying in 'sybca' class.

```
<?php
    $db=pg_connect("host=localhost dbname=sybca user=postgres") or die("Cannot connect");
    $result=pg_query($db, "select rno, sname, cls_name from student where cls_name='sybca'") or die("Cannot execute");
    echo "Total number of rows: ".pg_num_rows($result);
    pg_free_result($result);
    pg_close($db);
?>
```

Output:
Total number of rows: 2

pg_field_name:

This function is used to returns the field(column) name.

Syntax:

string pg_field_name(resource $result, integer $column_number)

Field(column) numbering starts from zero(0). It returns field(column) name on success and false on error.

pg_field_num:

This function returns the field(column) number of the named field(column).

Syntax:

integer pg_field_num(resource $result, string $column_name)

It returns the field(column) number or -1 on error. The field(column) numbering start from zero.

PHP program to display the first column name and the column number of field 'class' in the result of a query containing students of 'sybca' class.

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres") or
die("Cannot connect");
$result=pg_query($db, "select rno,sname, cls_name from
student
where cls_name='sybca'") or die("Cannot execute");
echo "First Column is : ".pg_field_name($result,0));
echo "<br>". "The column number of class field is :
".pg_field_num($result,"cls_name"));
pg_free_result($result);
pg_close($db);
?>
```

Output:

First Column is : rno

The column number of class field is : 2

pg_result_error

This function displays the error message related with the result.
Syntax: string pg_result_error(resource $result)
PHP program to demonstrate the use of pg result_error()
function and display the error message(if any).

```
<?php
$db=pg_connect("host=localhost dbname=sybca user=postgres");
if(!pg_connection_busy($db))
{    pg_send_query($db, "select * from stud;");
}
$result=pg_get_result($db);
echo pg_result_error($result);
?>
```

output
ERROR: relation "stud" does not exist LINE 1: select * from stud;

- The pg_send_query() function sends asynchronous query to the database.
- The pg_get_result() gets asynchronous query result.
- The pg_connection_busy() function checks whether the connection is busy or not. It waits if the previous query is still running, in that case pg_get_result() remains blocked.

Releasing the Result Memory:

The pg_free_result() function is used to free the memory occupied by the PostgreSQL query result.

Syntax: boolean pg_free_result(resource $result)

PEAR DB BASICS (DATABASE INDEPENDENT METHOD)

- The PHP Extension and Application Repository (PEAR) is a framework.
- PEAR DB is an advanced, object-oriented database library providing us the full database abstraction.
- It is also known as the database independent method. It means that the same code is applicable to all databases which are not possible with the database specific method.

Opening a Database Connection Data

Source Names (DSN):

In order to open a database connection using PEAR DB, first we need to build the DSN. A data source name (DSN) is a string having different components like type of database, username, password, protocol, host specification, database name.

The DSN has the following Syntax:

dbsyntax://username:password@protocol+hostspec/database

Following table specifies the currently supported database types.

dbsyntax	Database
mysql	MySQL
pgsql	PostgreSQL
msql	Mini SQL
ibase	InterBase
sybase	Sybase
fbsql	FontBase
ifx	Informix
mssql	Microsoft SQL Server
oci8	Oracle 7/8/9
odbc	ODBC(Open Database Connectivity)

- The protocol is the communication protocol to use. The common values are "tcp" and "unix" which are related with Internet and Unix domain sockets. Every database does not necessarily support every communication protocol.

Following are some valid data source names:

- pgsql://postgres:root@tcp+localhost/sybca
- pgsql://postgres:root@localhost/sybca
- pgsql://postgres@localhost/sybca //if there is no password

Opening a Connection

- After building a DSN, we can open a connection using the connect() method. We first need to include 'DB.php' header file in our code by using function require_once('DB.php');

The syntax for connecting to a database:

$db=DB::connect(DSN [, options]);

- The options part is not mandatory. The options can be 'persistent', 'optimize' or 'debug'. The connection is not persistent and no debugging information is display by default. The permissible values for 'optimize' option are 'portability' and 'performance' (default).

For example,

- $dsn="pgsql://postgres@localhost/sybca";
- $db=DB::connect($dsn, array('debug' =>1, 'optimize'=>'performance'));

Checking for the Errors

By using DB::iserror() method, we can check whether there is an error or not.

Syntax:

```
$db=DB::connect($dsn);
if(DB::iserror($db))
{    die($db->getMessage());
}
```

For example,

```
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
```

In the above example, $db is the database connection resource. The above code checks whether any error occurs while the connection is established with the database.

The DB::iserror() returns true if there is an error occurred. In that case, the program execution is stopped using die() method and an error message is displayed by the getMessage() method.

Executing a Query

The query()methodexecutes a query on the database.This method

dispatches SQL query to the database.

Syntax: $result=$db->query($sql);

Where $sql is the SQL statementpassed as a parameter and $db is the database connection resource.

In case of DML queries having INSERT, DELETE, UPDATE commands, the method returns the DB_OK constant to indicate success. In case of SELECT command, the method returns an object that can be used to access the result.

To check whether the query is successfully completed or not DB::iserror() is used as follows:

Syntax:

```
$result=$db->query($sql);
if(DB::iserror($result))
{   die($result->getMessage());
}
```

Fetching Results from a Query

PEAR DB has **two methods for retrieving data from a query** result set. The **fetchRow()** method **returns an array** while **fetchInto()** method **stores it into an array.**

Returning the Row

The fetchRow() method returns an array of the next row of results.

Syntax: $row=$result->fetchRow([mode]);

This method returns either an array of data (if there is some data), NULL (if there is no more data), or DB_ERROR (if an error occurred). The mode parameter is optional. If it is mentioned, specifies the format of the array returned. Generally, the fetchRow() method is used in a while loop to process a result, one row at a time.

Syntax:

```
while($row=$result->fetchRow())
{     if(DB::iserror($row))
        {    die($row->getMessage());
        }
        //statements to process the row
}
```

PEAR DB Program

Consider a table book(book_id, isbn_no, title, author, publ, price) Write a PHP script to display the top 3 costliest book written by user specified author name (Use PEAR DB method)

```
<html>
<head>
<title> Author Name Input Screen </title>
</head>
<body>
<form action="pedb.php" method="POST">
<pre>
Enter Author Name <input type="text" name="aname">
<input type="submit" value="Submit">
<input type="reset" value="Clear">
</pre>
</form>
</body>
</html>
```

```
<html>
<head>    <title>Top three costliest books</title> </head>
<body>
<?php
require_once('DB.php');
$auth=$_POST['aname'];
$db=DB::connect("pgsql://postgres@localhost/Vidya");
if(DB::iserror($db))
{      die($db->getMessage());
}
$sql="select title,price from book where author='amit' order by price desc limit 3";
$result=$db->query($sql);
if(DB::iserror($result))
{      die($result->getMessage());
}
echo "<h4>Top 3 costliest books of <u>$auth</u> are
</h4>"; while($row=$result->fetchRow())
{      echo "Title: $row[0] Price: $row[1]<br>";
}
$result->free();
$db->disconnect();
?>
</body>
</html>
```

Storing the Row

- The fetchInto() method **instead of returning the next row, stores it into an array variable** passed as a parameter.

Syntax: $answer=$result->fetchInto(array, [,mode]);

- Like fetchRow(), fetchInto() **returns NULL (if there is no more data),** or DB_ERROR (if an error occurred). **To process the result one row at a time fetchInto() is used as follows:**

```
while($answer=$result->fetchInto($row))
{
   if(DB::iserror($answer))
   {    die($answer->getMessage());
        //statements to process the row
   }
}
```

- The optional mode parameter to fetchRow() or fetchInto() controls the format of the row array.

- The possible values are

  DB_FETCHMODE_ORDERED(default)
  DB_FETCHMODE_ASSOC
  DB_FETCHMODE_OBJECT

- The DB_FETCHMODE_ASSOC **creates an array whose keys are column names and whose values are the values from those columns.**

- The DB_FETCHMODE_ORDERED **creates an indexed array.** To access the data in the **object format**, use DB_FETCHMODE_OBJECT mode.

For Example:

1. **Using default mode, i.e.** DB_FETCHMODE_ORDERED
   (no need to mention)

```
$result=$db->query("select * from student");
if(DB::iserror($result))
{
   die($result->getMessage());
}
while($row=$result->fetchRow())
{
   echo $row[0]." ".$row[1]." ".$row[2]."<br>";
}
```

2. Using DB_FETCHMODE_ASSOC.

```
$result=$db->query("select * from student");
if(DB::iserror($result))
{
    die($result->getMessage());
}
while($row=$result->fetchRow(DB_FETCHMODE_ASSOC))
{
    echo $row['rno']." ".$row['sname']." ".$row['cls_name']."<br>";
}
```

3. Using DB_FETCHMODE_OBJECT.

```
$result=$db->query("select * from student");
if(DB::iserror($result))
{    die($result->getMessage());
}
while($row=$result->fetchRow(DB_FETCHMODE_OBJECT)
)
{
echo $row->rno." ".$row->sname." ".$row->cls_name."<br>";
}
```

Finishing the Result

- A query result object holds all the rows fetched by the query. This may require a lot of memory for storing the result object. **When the processing on the result object is over, the memory occupied by it is released using the free() method.**

Syntax: $result->free();

This is not mandatory as free() is **called automatically when the PHP script ends.**

-

Disconnecting

- To release the database connection, use the disconnect() method on the database object.

Syntax: $db->disconnect();

This is not mandatory, as all **the database connections are terminated when the PHP script ends**

Advanced Database Techniques

- In addition to the basic database methods, PEAR DB provides several other methods for fetching rows, likewise a unique row ID system and a pair of prepare/execute statements that can improve the performance of repeated queries.

Placeholders

- While writing the query like INSERT or UPDATE, we can mention a placeholder '?' in place of specific value. In the query() method, we can add a second parameter consisting of an array of values to insert in place of '?' into the SQL.

Syntax: $result=$db->query(SQL, values);

- For example, the following code inserts two entries into the student table.

```
$studinfo=array(array(10, 'Mandar', 'sybca'), array(11, 'Chinmay', 'fybca'));
foreach($studinfo as $s)
{
$db->query('insert into student(rno, sname,cls_name) values(?,?,?)', $s);
}
```

- **Three characters that can be used as placeholder** values in an SQL query are as follows:
- **?: A string or number**, which will be **quoted if necessary** (recommended).
- **|: A string or number**, which will **never be quoted**.

Prepare/Execute and Prepare/ExecuteMultiple

- If we want to execute the same query repeatedly, then it is always better to compile the query once and then execute the compiled query many times, this can be done by using the prepare(), execute() and executeMultiple() methods.
- The prepare(), execute() and executeMultiple() methods are used in a pair, which means either you have to use prepare() and execute() or you have to use prepare() and executeMultiple().
- The use of prepare() and execute() is explained below.

Step 1: Call prepare() method on the query.

```
$compiled_query=$db->prepare($sql);
```

$sql is the SQL statement. The prepare() method returns the compiled version of the query object.

Step 2: The **execute() method fills in any placeholders** in the query by using second parameter and sends it back to RDBMS.

```
$response=$db->execute($compiled_query, $values);
```

- In case of an error, DB_ERROR is returned, otherwise query response object is returned. The $values array contains the values for the fields(placeholders) mentioned in the SQL query. The placeholders are shown with '?' in the SQL query.

- For example, in the following example two student records are added to the student table after the successful execution of prepare() and execute() methods.

```php
<?php
//Using prepare() and execute() methods
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{   die($db->getMessage());
}
$st=array(array(7, 'Anil', 'tybca'), array(9, 'Sanjiv', 'sybca'));
$sql="insert into student(rno, sname, cls_name) values(?,?,?)";
$compiled_query=$db->prepare($sql);
foreach($st as $val)
{   $db->execute($compiled_query, $val);
}
$db->disconnect();
?>
```

- The use of **prepare() and executeMultiple()** methods is explained below.

Step 1: Call prepare() method on the query.

$compiled_query=$db->prepare($sql);

$sql is the SQL statement. The prepare() method returns the compiled version of the query object.

Step 2: The executeMultiple() method avoids the iterations of the foreach loop mentioned for the execute() method.

Syntax of executeMultiple:

$response=$db->executeMultiple($compiled_query, $values);

The $values array is two dimensional indexed array.

For example, in the following example two student records are inserted in student table.

```
<?php
//Using prepare() and executeMultiple() methods
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$st=array(array(7, 'Anil', 'tybca'), array(9, 'Sanjiv', 'sybca'));
$sql="insert into student(rno, sname, cls_name) values(?,?,?)";
$compiled_query=$db->prepare($sql);
//Iterations of foreach loop is not required.
$db->executeMultiple($compiled_query, $st);
$db->disconnect();
?>
```

Other Methods

- PEAR DB provides other methods that execute a query and fetch the results in single step. These methods are getOne(), getCol(), getRow(), getAssoc(), and getAll(). All of these methods support placeholders.

1. **getOne():**

- This method executes a query and returns the first column of the first row.

Syntax:

- mixed getOne(string $sql [, mixed $param])
- Here $sql is a SQL query and $param can numeric, array or string.

PHP program to display the first column of the first row of the result containing students of 'sybca' class.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{
    die($db->getMessage());
}
$sql="select sname, cls_name from student where cls_name='sybca'";
$row=$db->getOne($sql);
print_r($row);
$db->disconnect();
?>
```

Output: Mahesh

2. getCol():

The getCol() methodreturns a single column from the result returned by the SQL query.

Syntax:

array getCol(string $sql [, mixed $col, mixed $values])

Here, $sql is the SQL query, $col can be integer or string specifying which column to return,$values can be number, string or array which specify parameters for SELECT statement (if any). Last two parameters are optional.

PHP program to display the value in the specified column 'class' of
a student having roll number 105.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{
die($db->getMessage());
}
$sql="select sname, cls_name from student where rno=105";
$row=$db->getCol($sql,"cls_name");
print_r($row);
$db->disconnect();
?>
```

Output: Array([0]=>fybca)

3. getRow():

- The getRow() method returns the first row of result data returned by the SQL query.

Syntax:

array getRow(string $sql [, array $param])

- Here $sql is the SQL query, $param is array of parameters.

4. getAssoc():

- The getAssoc() method returns the whole result set in the form of an associative array.

Syntax:

array getAssoc(string $sql [, boolean $opt])

- Here $sql is the SQL query and $opt is used when query returns exact two fields.

5. getAll()

- The getAll() method returns an array containing all the rows returned by the SQL query. If we specify the placeholder in the SQL statement, it retrieves only specific rows.

Syntax:

- array getAll(string $sql [, array $param [, mixed $mode]])
- Here, $sql is the SQL query, $param is an array of parameters and $mode is either numeric or in string format.

- In the following example, rno is passed as a parameter to the SQL query. As a result of this, we found that the name and class of the specified roll number is displayed

PHP program to display the name and class of a student whose roll number is 9 using getRow() method (pass roll number as a parameter).

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{     die($db->getMessage());
}
 $sql="select sname, cls_name from student where rno=?";
 $a=array(9);
 $row=$db->getRow($sql,$a);
 print_r($row);
 $db->disconnect();
 ?>
```

Output: Array ([0]=>Sanjiv [1]=>sybca)

In the above example, the name and class of student whose roll number is 9 is returned as an array.

PHP program to display name and class of a student whose roll

number is 9 using getAssoc method.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{   die($db->getMessage());
}
$sql="select sname, cls_name from student where rno=9";
$row=$db->getAssoc($sql,true);
print_r($row);
$db->disconnect();
?>
```

Output: Array ([Sanjiv] => Array ([0] => sybca))

PHP program to display the name and class of a student whose roll number is 9 using getAll. (pass roll number as a parameter to getAll).

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$sql="select sname, cls_name from student where rno=?";
$a=array(9);
$row=$db->getAll($sql, $a,2);   //2 is equivalent with
DB_FETCHMODE_ASSOC
print_r($row);
$db->disconnect();
?>
```

Output: Array ([0] => Array ([sname]=>Sanjiv [cls_name]=>sybca))

Getting information about Query Response

- PHP provides four PEAR DB methods to get information about query result object. These are numRows(), numCols(), affectedRows() and tableInfo()

1. **numRows():**

- The numRows() method returns the total number of rows returned by the SELECT query.

Syntax: integer numRows()

2. **numCols():**

- The numCols() methodreturns the total number of columns returned by the SELECT query.

Syntax: integer numCols()

PHP program to display the total number of rows and columns returned by a query result.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$sql="select sname, cls_name from student";
$result-$db->query($sql);
echo "Total number of rows returned is : ".$result->numRows();
echo "<br>";
echo "Total number of columns returned is : ".$result->numCols();
$result->free();
$db->disconnect();
?>
```

Output:

Total number of rows returned is : 5
Total number of columns returned is : 2

3. affectedRows():

- The INSERT, UPDATE, DELETE statements affect the number of rows in a table.

- If we use INSERT statement, then the number of rows will increase. In the same way, if we use either UPDATE or DELETE statements with specific condition, then some specific number of records gets affected.

- The affectedRows() method gives us the information about the number of rows affected by UPDATE, INSERT or DELETE statements.

Syntax: integer affectedRows()

PHP program to update the class to 'sybca' of students whose roll number is 4 and 5. Also display the total number of affected rows by this update command.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$sql="UPDATE student SET cls_name='sybca' WHERE rno
IN(4,5)";
$db->query($sql);
echo "Affected rows : ".$db->affectedRows();
$db->disconnect();
?>
```

Output: Affected rows : 2

tableInfo()

- The tableInfo() method returns the detail information on the data type and other points like flags of columns returned from a SELECT operation.

Syntax: array tableInfo(mixed $result)

- The following code finds the information about a table 'student'.

PHP program to display detail information about 'student' table.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{     die($db->getMessage());
}
$info=$db->tableInfo('student');
echo "<table border='2'>";
foreach($info as $key=>$val)
{
     echo "<tr>";
     foreach($val as $k=>$v)
     {       echo "<tr><td>$k</td><td>$v</td></tr>";
     }
     echo "<tr>";
}
echo "</table>";
$db->disconnect();
?>
```

Metadata

- The getListof() method query the database and gives information on available databases, users, views, and functions.

Syntax: array getListOf(string $param)

- Here $param can be tables, databases, users, functions, views etc.

- The following script displays all the views created in "sybca" database.

PHP program to display all the views defined in 'sybca' database.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$data=$db->getListOf('views');
print_r($data);
$db->disconnect();
?>
```

Output: Array ([0] =>emp_view)

Transactions

- A transaction is an atomic unit of execution which is performed as a whole unit or none of the instruction from the transaction is executed.

- A series of changes made to the database can be committed (all changes applied) or rolled back(changes not applied).

- For example, in a transaction of transferring an amount from account of person 'A' to account of person 'B', the withdrawal from A's account and deposit into B's account must happen as a whole unit(together).

- These two actions cannot be performed independently as a separate instruction. If it happens, it leaves the database in inconsistent state, which is not acceptable. PEAR DB offers commit() and rollback() methods to help with transactions.

- $result = $db->commit();
- $result = $db->rollback();

- Calling either commit() or rollback() on a non supported database returns DB_ERROR.

Sequences

- PEAR DB sequences are somewhat similar to the database specific ID assignment. The nextId() method returns the next ID for that given sequence.

Syntax: integer nextId(string $sequence_name)

- A sequence is a table in the database that maintains the last-assigned ID. We can create and delete sequences with createSequence() and dropSequence() methods.

Syntax:

integer createSequence(string $sequence_name)
integer dropSequence(string $sequence_name)

The following example creates a sequence named "newseq"

PHP program to create a new sequence named 'newseq' and display the next ID for that sequence.

```
<?php
require_once('DB.php');
$db=DB::connect("pgsql://postgres@localhost/sybca");
if(DB::iserror($db))
{    die($db->getMessage());
}
$ret=$db->createSequence('newseq');
var_dump($ret);
$id=$db->nextId('newseq');
echo "<br>The next id is : ".$id;
$db->disconnect();
?>
```

Output:

```
int(1)
The next id is : 1
```