PHP Programming

Reference Notes for PHP Programming for B.Sc. 2nd year students

by

Prof. Anil Verma

Assistant Professor Computer Applications

Department of Computer Science

Maharana Pratap Govt. Degree College Amb

महाराणा प्रताप राजकीय महाविद्यालय

विद्यया अमृतमश्नुते

अम्ब, जिला ऊना (हि०प्र०)

affiliated with

Himachal Pradesh University, Shimla

COMP203TH: PHP Programming

Unit-I

Introduction to PHP

PHP (Hypertext Preprocessor) is a popular server-side scripting language used for web development. It was originally designed to create dynamic web pages, but it has since evolved to support a wide range of applications, including server-side scripting, command-line scripting, and desktop applications.

Some of the key features of PHP include:

Open Source: PHP is an open-source language, which means that it is free to use, modify, and distribute.

Cross-Platform: PHP runs on all major operating systems, including Windows, Linux, and macOS.

Easy to Learn: PHP has a simple and easy-to-learn syntax, making it a popular choice for beginners.

Wide Community: PHP has a large and active community of developers who contribute to the language and offer support to other developers.

Compatibility: PHP is compatible with a wide range of web servers, including Apache and Nginx.

Some of the popular use cases for PHP include:

Dynamic Web Pages: PHP is commonly used to create dynamic web pages, where the content of a web page changes based on user input or other factors.

Content Management Systems: PHP is used in many popular content management systems, such as WordPress and Drupal, to power the backend of websites.

E-Commerce: PHP is widely used in e-commerce websites to handle online transactions and manage customer data.

Command-Line Scripts: PHP can be used to create command-line scripts for automating tasks on a server or local computer.

Overall, PHP is a versatile language that has played a significant role in the development of the web. Its popularity can be attributed to its ease of use, cross-platform compatibility, and the large community of developers who have contributed to its development over the years.

History inventions and versions of PHP

PHP (Hypertext Preprocessor) was created by Rasmus Lerdorf in 1994. It was initially developed as a set of Common Gateway Interface (CGI) scripts written in the C programming language, which Lerdorf used to track visitors to his personal website.

Over time, Lerdorf added new features to PHP and released it as an open-source project. In 1997, two developers, Andi Gutmans and Zeev Suraski, rewrote the PHP engine from scratch and released it as PHP 3.

PHP 3 included many new features, such as support for databases and more robust object-oriented programming capabilities. It quickly gained popularity and was used to power many popular websites, including Yahoo!.

In 2000, PHP 4 was released with significant improvements to performance, stability, and security. It introduced new features, such as session handling and support for XML, and included an updated Zend Engine, which improved the language's performance.

PHP 5 was released in 2004 and was a major update to the language. It included significant improvements to object-oriented programming and introduced new features, such as exceptions and improved support for XML and web services.

In 2014, PHP 5.6 was released with improved performance, security, and language features, including support for variadic functions and constant scalar expressions.

In 2015, PHP 7 was released with significant improvements to performance, memory usage, and error handling. It introduced new language features, such as anonymous classes and return type declarations, and deprecated some older features, such as PHP's original MySQL extension.

The latest stable release of PHP as of my knowledge cutoff date (September 2021) is PHP 8.0. It includes many new language features, such as named arguments, union types, and attributes, as well as improvements to performance, error handling, and security.

Some notable inventions associated with PHP include:

PHP-FPM (FastCGI Process Manager): PHP-FPM is a PHP FastCGI implementation that provides high performance and scalability for PHP applications. It was introduced in PHP 5.3 and has become the preferred way to run PHP applications on high-traffic websites.

Composer: Composer is a dependency manager for PHP that simplifies the process of managing external libraries and packages in PHP projects. It was released in 2012 and has since become an essential tool for PHP developers.

Laravel: Laravel is a popular PHP web application framework that was released in 2011. It provides an elegant and expressive syntax for web development and includes a range of features such as routing, middleware, and authentication.

Overall, PHP has come a long way since its inception and continues to be a popular language for web development. Its popularity can be attributed to its ease of use, cross-platform compatibility, and the large community of developers who have contributed to its development over the years.

Important tools and software requirements for PHP (like Web Server, Database, Editors etc.)

To work with PHP, you'll need a few key tools and software requirements:

Web Server: PHP is a server-side scripting language, which means it needs a web server to run. Apache, Nginx, and Microsoft IIS are some popular web servers that support PHP.

PHP Interpreter: You'll need to install the PHP interpreter on your server or local computer to run PHP scripts. Most web servers come with a built-in PHP interpreter, but you can also download and install it separately.

Database: PHP is often used in conjunction with a database to create dynamic web applications. MySQL, PostgreSQL, and MongoDB are some popular databases that work well with PHP.

Code Editor: To write PHP code, you'll need a text editor or integrated development environment (IDE). Some popular options include Visual Studio Code, Sublime Text, and PHPStorm.

Debugging Tools: Debugging is an essential part of software development, and there are many tools available to help you find and fix errors in your PHP code. Xdebug and PHP Debug Bar are two popular debugging tools for PHP.

Version Control: To manage changes to your PHP code, you'll need a version control system such as Git. This will help you track changes to your code and collaborate with other developers on your team.

Overall, having a good understanding of these tools and software requirements is essential for working with PHP and developing robust, scalable web applications.

Use of PHP with other technologies

PHP can be used in conjunction with a variety of other technologies to create powerful, dynamic web applications. Here are some examples:

HTML/CSS/JavaScript: PHP is often used in conjunction with HTML, CSS, and JavaScript to create dynamic web pages. PHP scripts can generate HTML output, which can then be styled with CSS and enhanced with JavaScript.

Databases: PHP is frequently used in conjunction with databases to create dynamic, data-driven web applications. MySQL, PostgreSQL, and MongoDB are some popular databases that work well with PHP.

Web Frameworks: PHP web frameworks like Laravel, Symfony, and CodeIgniter provide a structured approach to web development, making it easier to build scalable, maintainable web applications.

Content Management Systems (CMS): Many popular CMS platforms, such as WordPress, Drupal, and Joomla, are built using PHP. These platforms provide an easy-to-use interface for creating and managing website content.

Web Services: PHP can be used to build web services and APIs that can be consumed by other applications. This makes it possible to build complex, distributed systems that can scale to handle large volumes of traffic.

Overall, PHP's versatility and ease of use make it a popular choice for building a wide range of web applications, from simple websites to complex, data-driven systems. Its ability to work with other technologies makes it a valuable tool for developers looking to build powerful, dynamic web applications.

Scope of PHP

PHP has a wide scope in the field of web development. It is a popular server-side scripting language used for building dynamic websites and web applications. PHP's popularity and versatility make it an excellent choice for building everything from simple, static web pages to complex, data-driven web applications.

Here are some of the areas where PHP is commonly used:

E-commerce websites: PHP is widely used for building e-commerce websites because it can easily integrate with payment gateways, shopping carts, and other e-commerce tools.

Content Management Systems (CMS): Many popular CMS platforms, such as WordPress, Drupal, and Joomla, are built using PHP.

Web applications: PHP is used for building web applications that can handle complex business logic and data processing.

Social networking websites: PHP is used in the development of social networking websites such as Facebook, LinkedIn, and Twitter.

Mobile application development: PHP can also be used to build web services and APIs that can be consumed by mobile applications.

Overall, PHP has a wide range of applications and is an essential tool for developers looking to build powerful, dynamic web applications. Its popularity, ease of use, and versatility make it a valuable tool for web developers around the world.

Basic Syntax of PHP Program

The basic syntax of a PHP program is as follows:

PHP code is typically embedded within HTML code, so a PHP script begins with the opening <?php tag and ends with the closing ?> tag.

PHP statements are terminated by a semicolon (;).

Comments can be added to the code using the // or /* */ syntax.

Here is an example of a simple PHP program:

```
<!DOCTYPE html>
<html>
<body>
<?php
// This is a comment
// Define a variable
$message = "Hello, world!";
// Output the message
echo $message;
?>
</body>
</html>
```

In this example, we define a variable $message and assign it the value "Hello, world!". We then use the echo statement to output the value of the variable to the web page.

Note that the PHP code is enclosed within the opening and closing PHP tags, and that the PHP statements end with semicolons. We also include a comment in the code to explain what it does.

This is just a simple example, but the basic syntax of a PHP program remains the same regardless of the complexity of the code.

PHP variables and constants with examples

PHP variables and constants are used to store values that can be used throughout a program. Here are some examples of how to define and use variables and constants in PHP:

Variables:

Variables in PHP are declared using the $ symbol followed by the variable name. Variable names can contain letters, numbers, and underscores, but they must begin with a letter or underscore.

Example:

```
$name = "John Doe";

$age = 30;
```

In this example, we declare two variables: $name, which stores a string value "John Doe", and $age, which stores an integer value of 30.

Constants:

Constants in PHP are similar to variables, but their values cannot be changed once they are defined. Constants are declared using the define() function.

Example:

```
define("PI", 3.14159);

define("GREETING", "Hello, world!");
```

In this example, we define two constants: PI, which stores the value of pi (3.14159), and GREETING, which stores a string value "Hello, world!".

Constants are useful for storing values that need to remain constant throughout the execution of a program.

Once defined, variables and constants can be used throughout the program. Here is an example of how to use a variable and a constant in PHP:

```
$name = "John Doe";

$age = 30;

define("GREETING", "Hello, world!");

echo "My name is " . $name . " and I am " . $age . " years old."; // Output: My name is John Doe and I am 30 years old.

echo GREETING; // Output: Hello, world!
```

In this example, we use the variables $name and $age to construct a string, which is then output to the web page using the echo statement. We also output the value of the constant GREETING using the echo statement.

Overall, variables and constants are essential for storing values that can be used throughout a PHP program. They provide a flexible and powerful way to manage data and simplify programming tasks.

Data Types in PHP with examples

PHP supports several data types that are used to store different types of data. Here are some examples of the data types supported in PHP:

Strings:

Strings are used to store textual data. They can be enclosed in single or double quotes.

Example:

$name = "John Doe";

$message = 'Hello, world!';

Integers:

Integers are used to store whole numbers.

Example:

$age = 30;

Floats:

Floats (also known as "doubles") are used to store decimal numbers.

Example:

$price = 12.99;

Booleans:

Booleans are used to store true/false values.

Example:

$is_active = true;

$is_logged_in = false;

Arrays:

Arrays are used to store multiple values in a single variable. They can store values of different data types.

Example:

$fruits = array("apple", "banana", "orange");

Objects:

Objects are used to represent complex data structures. They are instances of classes and can contain properties and methods.

Example:

```
class Person {
  public $name;
  public $age;
}
$person = new Person();
$person->name = "John Doe";
$person->age = 30;
```

Null:

Null is used to represent the absence of a value.

Example:

```
$value = null;
```

These are just a few examples of the data types supported in PHP. Understanding the different data types is essential for writing effective PHP code that can handle different types of data and perform various operations.

Expressions in PHP

Expressions in PHP are combinations of values, variables, operators, and functions that can be evaluated to produce a result. They are used to perform mathematical and logical operations and to manipulate data in various ways. Here are some examples of expressions in PHP:

Arithmetic expressions:

Arithmetic expressions are used to perform mathematical operations, such as addition, subtraction, multiplication, and division.

Example:

```
$x = 5;
$y = 10;
$sum = $x + $y; // sum is now 15
$product = $x * $y; // product is now 50
```

Comparison expressions:

Comparison expressions are used to compare values and return a boolean (true/false) result. The comparison operators include ==, !=, <, >, <=, and >=.

Example:

```
$x = 5;

$y = 10;

$is_greater = $y > $x; // is_greater is true

$is_equal = $x == $y; // is_equal is false
```

String expressions:

String expressions are used to concatenate strings together using the . operator.

Example:

```
$first_name = "John";

$last_name = "Doe";

$full_name = $first_name . " " . $last_name; // full_name is "John Doe"
```

Logical expressions:

Logical expressions are used to perform logical operations on boolean values. The logical operators include &&, ||, and !.

Example:

```
$is_logged_in = true;

$is_admin = false;

$is_allowed = $is_logged_in && $is_admin; // is_allowed is false
```

Overall, expressions are a fundamental aspect of PHP programming, as they enable the manipulation of data and the performance of complex operations. Understanding how to construct and evaluate expressions is essential for effective PHP programming.

Local and Global scopes of a variable in PHP

In PHP, the scope of a variable determines where it can be accessed and used within a script. PHP supports two types of variable scopes: local and global.

Local scope:

A variable with local scope can only be accessed and used within the function or block of code where it is defined. Once the function or block of code ends, the variable is destroyed.

Example:

```
function test() {

   $x = 10; // $x has local scope

   echo $x;

}

test(); // output: 10
```

```
echo $x; // throws an error, $x is not defined
```

In the example above, the variable $x has local scope within the test() function. It can only be accessed within the function, and trying to access it outside of the function will result in an error.

Global scope:

A variable with global scope can be accessed and used throughout the entire script, including inside functions and blocks of code.

Example:

```
$x = 10; // $x has global scope

function test() {

   global $x;

   echo $x;

}

test(); // output: 10

echo $x; // output: 10
```

In the example above, the variable $x has global scope, so it can be accessed both inside and outside the test() function. To access a global variable inside a function, you need to use the global keyword before the variable name.

It's important to use variable scopes correctly in PHP to avoid errors and conflicts in your code. Local variables should be used for temporary storage within a function, while global variables should be used for values that need to be accessed and modified across different functions and blocks of code.

Operators in PHP: Arithmetic, Assignment, Relational, Logical operators, Bitwise, ternary, and MOD operators with examples

In PHP, operators are used to perform various operations on values and variables. Here are some of the most commonly used operators:

Arithmetic operators:

Arithmetic operators are used to perform mathematical operations on values, such as addition, subtraction, multiplication, division, and modulus.

```
$x = 10;

$y = 5;

echo $x + $y; // addition

echo $x - $y; // subtraction

echo $x * $y; // multiplication

echo $x / $y; // division

echo $x % $y; // modulus
```

Assignment operators:

Assignment operators are used to assign values to variables, such as =, +=, -=, *=, /=, and %=.

```
$x = 10;

$x += 5; // equivalent to $x = $x + 5

$x -= 5; // equivalent to $x = $x - 5

$x *= 5; // equivalent to $x = $x * 5

$x /= 5; // equivalent to $x = $x / 5

$x %= 5; // equivalent to $x = $x % 5
```

Relational operators:

Relational operators are used to compare values and return a boolean (true/false) result, such as ==, !=, <, >, <=, and >=.

```
$x = 10;

$y = 5;

echo $x == $y; // equal to

echo $x != $y; // not equal to

echo $x < $y; // less than

echo $x > $y; // greater than

echo $x <= $y; // less than or equal to

echo $x >= $y; // greater than or equal to
```

Logical operators:

Logical operators are used to perform logical operations on boolean values, such as &&, ||, and !.

```
$x = true;

$y = false;

echo $x && $y; // logical AND

echo $x || $y; // logical OR

echo !$x; // logical NOT
```

Bitwise operators:

Bitwise operators are used to perform bitwise operations on integer values, such as &, |, ^, <<, and >>.

```
$x = 2; // binary 10

$y = 3; // binary 11

echo $x & $y; // bitwise AND (binary 10)
```

echo $x | $y; // bitwise OR (binary 11)

echo $x ^ $y; // bitwise XOR (binary 01)

echo $x << 1; // left shift (binary 100)

echo $x >> 1; // right shift (binary 01)

Ternary operator:

The ternary operator is a shorthand way of writing an if statement. It takes three operands: a condition, a value to return if the condition is true, and a value to return if the condition is false.

$x = 10;

$y = ($x > 5) ? "greater than 5" : "less than or equal to 5";

echo $y; // output: greater than 5

MOD operator:

The MOD operator returns the remainder of a division operation. It is represented by the % symbol.

$x = 10;

$y = 3;

echo $x % $y; // output: 1

Overall, operators are a fundamental aspect of PHP programming, as they enable the manipulation of constants and variables in any expression.

Precedence and associativity of operators in PHP

In PHP, the precedence of operators determines the order in which operations are performed in an expression. Operators with higher precedence are evaluated before operators with lower precedence. If operators have the same precedence, the associativity of the operators determines the order in which they are evaluated.

Here is a table showing the precedence and associativity of operators in PHP, in order from highest to lowest:

Precedence	Operator	Description	Associativity
1	**()**	Grouping	n/a
2	**++, --**	Increment/Decrement	n/a
3	**!, ~, +, -**	Logical/Bitwise/Unary plus and minus	right-to-left
4	***, /, %**	Multiplication/Division/Modulus	left-to-right
5	**+, -**	Addition/Subtraction	left-to-right
6	**<<, >>**	Bitwise shift	left-to-right
7	**<, <=, >, >=**	Comparison	left-to-right

8	**==, !=, ===, !==**	Equality	left-to-right
9	**&**	Bitwise AND	left-to-right
10	**^**	Bitwise XOR	left-to-right
11	**|**	Bitwise OR	left-to-right
12	**&&**	Logical AND	left-to-right
13	`		`
14	**?:**	Ternary	right-to-left
15	**=**	Assignment	right-to-left
16	**,**	Comma	left-to-right

As an example, consider the following expression:

$x = 10 + 5 * 2;

Based on the table above, the multiplication operator has higher precedence than the addition operation. Therefore, the expression is evaluated as follows:

$x = 10 + (5 * 2);

$x = 10 + 10;

$x = 20;

So the final value of $x is 20.

It's important to note that the order of operations can be changed using parentheses. For example:

$x = (10 + 5) * 2;

This expression evaluates the addition operation first, resulting in the value 15. Then, the multiplication operation is performed, resulting in the value 30. Therefore, the final value of $x is 30.

Unit-II

Handling HTML form with PHP

Capturing Form Data in PHP

In PHP, form data can be captured using the $_POST or $_GET superglobal arrays, depending on the method used to submit the form.

If the form is submitted using the POST method, the form data can be accessed using the $_POST array. For example, consider the following HTML form:

Html Program

```
<form method="post" action="process.php">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <input type="submit" value="Submit">
</form>
```

In the process.php script, the form data can be accessed as follows:

PHP Program

```
$name = $_POST['name'];
$email = $_POST['email'];
```

Similarly, if the form is submitted using the GET method, the form data can be accessed using the $_GET array. For example:

Html Program

```
<form method="get" action="process.php">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <input type="submit" value="Submit">
</form>
```

In the process.php script, the form data can be accessed as follows:

PHP Program

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

$email = $_GET['email'];
```

It's important to note that form data should always be sanitized before use, to prevent security vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks. This can be done using functions such as htmlspecialchars() and mysqli_real_escape_string().

GET and POST form methods in PHP with examples

In PHP, the two main methods used to submit form data are GET and POST.

GET method

The GET method is used to submit form data in the URL query string. This method is suitable for forms with a small amount of data, such as search forms.

Here's an example of a form that uses the GET method:

Html Program

```
<form method="get" action="search.php">
         <label for="query">Search:</label>
        <input type="text" id="query" name="query">
        <button type="submit">Search</button>
</form>
```

When the form is submitted, the data is sent in the URL query string, like this:

Arduino Code

```
http://example.com/search.php?query=search+term
```

The data can be accessed in PHP using the $_GET superglobal array:

PHP Program

```
$query = $_GET['query'];
```

POST method

The POST method is used to submit form data in the request body. This method is suitable for forms with a large amount of data, such as registration or login forms.

Here's an example of a form that uses the POST method:

Html Program

```
<form method="post" action="login.php">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <label for="password">Password:</label>
        <input type="password" id="password" name="password">

        <button type="submit">Login</button>

</form>
```

When the form is submitted, the data is sent in the request body, and can be accessed in PHP using the $_POST superglobal array:

Php Program

```
$username = $_POST['username'];

$password = $_POST['password'];
```

It's important to note that the POST method is more secure than the GET method, as the data is not visible in the URL. However, both methods should always be validated and sanitized to prevent security vulnerabilities.

Difference between GET and POST methods in PHP

In PHP, the two main methods used to submit form data are GET and POST. The key differences between these two methods are:

Data transmission: The GET method sends form data in the URL query string, while the POST method sends it in the request body.

Data size: The GET method is suitable for forms with a small amount of data, while the POST method is suitable for forms with a large amount of data.

Security: The POST method is more secure than the GET method, as the data is not visible in the URL.

Caching: The GET method allows caching of the form data, while the POST method does not.

Here's a summary of the differences:

Feature	GET	POST
Data transmission	Sends data in the URL query string	Sends data in the request body
Data size	Suitable for small amounts of data	Suitable for large amounts of data
Security	Less secure, as data is visible in the URL	More secure, as data is not visible in the URL
Caching	Allows caching of form data	Does not allow caching of form data

In general, the GET method should be used for forms that retrieve data, such as search forms, while the POST method should be used for forms that submit data, such as registration or login forms. It's important to validate and sanitize the form data, regardless of the method used.

How to deal with multi-value fields in PHP

In PHP, multi-value fields in forms can be handled using arrays.

For example, if a form has a field for selecting multiple options, such as checkboxes or a multiple select input, the name attribute for the field can be specified as an array, like this:

Html Program

```
<form method="post" action="process.php">
        <label for="colors">Select your favorite colors:</label>
        <input type="checkbox" name="colors[]" value="red"> Red
        <input type="checkbox" name="colors[]" value="green"> Green
        <input type="checkbox" name="colors[]" value="blue"> Blue
        <button type="submit">Submit</button>
</form>
```

When the form is submitted, the selected options will be sent as an array in the $_POST superglobal variable. Here's how to access the selected options in PHP:

PHP Program

```
$selected_colors = $_POST['colors'];
```

The $selected_colors variable will be an array containing the values of the selected options.

Similarly, if a form has a field that allows the user to enter multiple values, such as a text input or textarea, the name attribute for the field can be specified as an array, like this:

Html Program

```
<form method="post" action="process.php">
        <label for="ingredients">Enter your favorite ingredients (one per line):</label>
        <textarea name="ingredients[]" id="ingredients"></textarea>
        <button type="submit">Submit</button>
</form>
```

When the form is submitted, the entered values will be sent as an array in the $_POST superglobal variable. Here's how to access the entered values in PHP:

PHP Program

```
$ingredients = $_POST['ingredients'];
```

The $ingredients variable will be an array containing the entered values, with each value as a separate element in the array.

Redirecting a form after submission in PHP

In PHP, after a form is submitted, you can redirect the user to another page using the header() function. This function sends a raw HTTP header to the browser, which will cause the browser to redirect to the specified URL.

Here's an example of how to redirect a user to a "thank you" page after submitting a form:

First, process the form data and perform any necessary actions, such as saving the data to a database or sending an email.

Use the header() function to redirect the user to the "thank you" page. The Location header should be set to the URL of the page you want to redirect to.

PHP Program

```
// Process the form data here

// Redirect to the thank you page

header("Location: thank-you.php");

exit; // make sure to exit after redirecting
```

Note that the exit statement is used after the header() function to ensure that no further PHP code is executed and that the redirect happens immediately.

On the "thank you" page, you can display a message to the user thanking them for submitting the form.

Html Program

```
<h1>Thank you for submitting the form!</h1>

<p>We will get back to you as soon as possible.</p>
```

Make sure that the header() function is called before any output is sent to the browser, including HTML tags, whitespace, or other characters. If output has already been sent, the header() function will not work and you may receive an error message.

PHP conditional events and Loops

IF Else conditional statement

In PHP, conditional statements are used to execute code blocks based on certain conditions. The most commonly used conditional statements in PHP are if, else if, and else.

Here's an example of how to use conditional statements in PHP:

PHP Program

```
$age = 25;

if ($age >= 18) {

        echo "You are an adult.";

        }
```

```
else {
        echo "You are not yet an adult.";
        }
```

In this example, the $age variable is checked to see if it is greater than or equal to 18. If it is, the message "You are an adult." is printed to the screen. If it is not, the message "You are not yet an adult." is printed instead.

You can also use else if statements to check additional conditions, like this:

PHP Program

```
$grade = 75;
if ($grade >= 90) {
         echo "You got an A.";
} else if ($grade >= 80) {
        echo "You got a B.";
} else if ($grade >= 70) {
        echo "You got a C.";
} else if ($grade >= 60) {
        echo "You got a D.";
} else {
        echo "You failed the course.";
}
```

In this example, the $grade variable is checked to see which range it falls into. Depending on the value of $grade, a different message is printed to the screen.

You can also use logical operators like && (and), || (or), and ! (not) to combine multiple conditions in a single statement. For example:

PHP Program

```
$age = 25;
$country = "USA";
if ($age >= 18 && $country == "USA") {
         echo "You are eligible to vote.";
}
else {
        echo "You are not eligible to vote.";
}
```

In this example, both the age and country must meet the specified conditions in order for the message "You are eligible to vote." to be printed.

Nester IF Else conditional statement

In PHP, you can nest if statements inside other if statements to create more complex conditional logic. Here's an example:

Php Program

```
$age = 25;

$country = "USA";

if ($country == "USA") {

        if ($age >= 18) {

                echo "You are eligible to vote.";

        } else {

                echo "You are not yet an adult.";

        }

        } else {

                echo "You are not a citizen of the USA.";

}
```

In this example, the $country variable is checked first. If it equals "USA", then a nested if statement checks the value of $age. If $age is greater than or equal to 18, the message "You are eligible to vote." is printed. If not, the message "You are not yet an adult." is printed. If $country is not equal to "USA", the message "You are not a citizen of the USA." is printed.

It's important to note that when nesting if statements, you need to make sure to close each one with its own set of curly braces. In the example above, the inner if statement is closed with its own set of curly braces, even though it is contained within the curly braces of the outer if statement. This helps to avoid syntax errors and makes the code more readable.

Switch case statement

In PHP, the switch statement is used to perform different actions based on different conditions. It is an alternative to using multiple if statements.

Here's an example of how to use the switch statement in PHP:

Php Program

```
$day = "Wednesday";

switch ($day) {

        case "Monday":
            echo "Today is Monday.";
            break;
        case "Tuesday":
            echo "Today is Tuesday.";
            break;
        case "Wednesday":
            echo "Today is Wednesday.";
            break;
        case "Thursday":
            echo "Today is Thursday.";
            break;
        case "Friday":
            echo "Today is Friday.";
            break;
        default:
            echo "Today is not a weekday.";
  break;
}
```

In this example, the $day variable is checked to see which case it matches. If it matches "Monday", the message "Today is Monday." is printed. If it matches "Tuesday", the message "Today is Tuesday." is printed, and so on. If $day does not match any of the specified cases, the default case is executed and the message "Today is not a weekday." is printed.

Note that each case is followed by a break statement. This is important to prevent the code from executing the next case even if the condition is not met. If you omit the break statement, the code will continue to execute the next case until it reaches a break statement or the end of the switch statement.

While Loop in PHP

In PHP, a while loop is used to execute a block of code repeatedly as long as a specified condition is true. Here's an example of how to use a while loop in PHP:

PHP Program

```
$count = 1;
while ($count <= 10) {
        echo $count . "<br>";
        $count++;
}
```

In this example, the $count variable is initialized to 1. The while loop checks if $count is less than or equal to 10. If it is, the code inside the loop is executed, which in this case is to print the value of $count and then increment it by 1 using the $count++ statement. This process continues until $count is no longer less than or equal to 10.

The output of this code will be:

```
1
2
3
4
5
6
7
8
9
10
```

It's important to make sure that the condition specified in the while loop eventually becomes false, otherwise, the loop will continue to execute indefinitely, causing your program to hang or crash.

For Loop

In PHP, a for loop is used to execute a block of code a specified number of times. Here's an example of how to use a for loop in PHP:

PHP Program

```
for ($i = 1; $i <= 10; $i++) {
        echo $i . "<br>";
}
```

In this example, the for loop initializes a variable $i to 1, checks if $i is less than or equal to 10, executes the code inside the loop, and then increments $i by 1 using the $i++ statement. This process continues until $i is no longer less than or equal to 10.

The output of this code will be:

```
1
2
3
```

```
4

5

6

7

8

9

10
```

The for loop has three parts: initialization, condition, and increment. The initialization part initializes the loop variable, the condition part specifies the condition for the loop to continue running, and the increment part is executed after each iteration of the loop.

You can also use a for loop to iterate over arrays in PHP. Here's an example:

PHP Program

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

for ($i = 0; $i < count($colors); $i++) {

        echo $colors[$i] . "<br>";

}
```

In this example, the for loop iterates over an array $colors using its index, which starts from 0 and goes up to the length of the array minus 1. The count() function is used to get the length of the array.

The output of this code will be:

```
Red

Green

Blue
```

Do While Loop

In PHP, a do-while loop is used to execute a block of code at least once, and then repeatedly as long as a specified condition is true. Here's an example of how to use a do-while loop in PHP:

PHP Program

```
$count = 1;

do {

        echo $count . "<br>";

        $count++;

        } while ($count <= 10);
```

In this example, the $count variable is initialized to 1. The do block of code is executed first, which in this case is to print the value of $count and then increment it by 1 using the $count++ statement. Then, the while loop checks if $count is less than or equal to 10. If it is, the code inside the loop is executed again. This process continues until $count is no longer less than or equal to 10.

The output of this code will be:

```
1

2

3

4

5

6

7

8

9

10
```

Note that unlike the while loop, the do-while loop always executes the block of code at least once, regardless of whether the condition is true or not.

It's important to make sure that the condition specified in the do-while loop eventually becomes false, otherwise the loop will continue to execute indefinitely, causing your program to hang or crash.

Goto

In PHP, the goto statement is used to transfer control to a different part of the program. It allows you to jump to a specific label within your code, bypassing any code in between.

Here's an example of how to use goto in PHP:

PHP Program

```
echo "Start<br>";

goto middle;

echo "This won't be displayed<br>";

middle:

echo "Middle<br>";

goto end;

echo "This won't be displayed either<br>";

end:

echo "End<br>";
```

In this example, the goto statement is used to jump over the code that would normally be executed between the goto statement and the middle: label.

The output of this code will be:

```
Start

Middle

End
```

While the goto statement can be useful in certain situations, it can also make your code harder to read and debug. As a result, it's generally considered a bad practice to use goto in PHP, and you should try to find other ways to structure your code instead.

Break, Continue, and exit

In PHP, the break, continue, and exit statements are used to control the flow of execution within loops and conditional statements.

1. break: The break statement is used to exit out of a loop or switch statement. When encountered, the break statement immediately terminates the loop or switch statement, and execution resumes at the next statement after the loop or switch.

Here's an example of how to use break in a loop:

PHP Program

```
for ($i = 1; $i <= 10; $i++) {
        if ($i == 6) {
        break;
        }
        echo $i . "<br>";
}
```

In this example, the loop will execute 5 times, printing the numbers 1 through 5. When $i equals 6, the break statement is encountered, causing the loop to terminate immediately.

2. continue: The continue statement is used to skip over the rest of the current iteration of a loop and move on to the next iteration.

Here's an example of how to use continue in a loop:

PHP Program

```
for ($i = 1; $i <= 10; $i++) {
        if ($i % 2 == 0) {
        continue;
        }
        echo $i . "<br>";
}
```

In this example, the loop will execute 10 times, but only print the odd numbers (1, 3, 5, 7, 9) because the continue statement is used to skip over the even numbers.

3. exit: The exit statement is used to immediately terminate the execution of a PHP script. It's often used to handle fatal errors or to redirect the user to a different page.

Here's an example of how to use exit to redirect the user to a different page:

PHP Program

```
if (!isset($_SESSION['username'])) {
        header("Location: login.php");
        exit;
}
```

In this example, if the user is not logged in, they will be redirected to the login.php page using the header() function. The exit statement is used to immediately terminate the execution of the script, preventing any further code from being executed.

Unit-III

PHP Functions

Function and Need of Function in PHP

A function in PHP is a block of code that performs a specific task and can be reused throughout the program. Functions are useful for organizing code, reducing redundancy, and improving code readability and maintainability.

Functions in PHP are defined using the function keyword, followed by the function name, a set of parentheses, and a block of code enclosed in curly braces. Here's an example:

PHP Program

```
function greet($name) {
        echo "Hello, " . $name . "!";
        }
greet("John"); // Output: Hello, John!
```

In this example, we define a function called greet that takes one parameter ($name) and prints a personalized greeting using the parameter. We then call the function with the argument "John", resulting in the output "Hello, John!".

Functions can also have return values using the return keyword. Here's an example:

PHP Program

```
function add($num1, $num2) {
        $sum = $num1 + $num2;
        return $sum;
        }
$result = add(3, 5); // $result now contains 8
```

In this example, we define a function called add that takes two parameters ($num1 and $num2) and returns their sum using the return keyword. We then call the function with arguments 3 and 5, storing the returned value in the $result variable.

The need for functions in PHP (and programming in general) is to avoid duplicating code and to make the code easier to read and maintain. By encapsulating a block of code in a function, we can reuse it throughout the program without having to write the same code multiple times. Functions also make it easier to debug code, as errors can be isolated to a specific function rather than being spread throughout the program.

Declaration and calling of a function in PHP

To declare a function in PHP, you use the function keyword followed by the name of the function, a set of parentheses, and a block of code enclosed in curly braces. Here's an example of a simple function:

PHP Program

```
function sayHello() {
        echo "Hello!";
        }
```

To call this function, you simply write the function name followed by a set of parentheses:

PHP Program

```
sayHello();
```

This will execute the code inside the sayHello() function and output "Hello!" to the screen.

Functions can also take parameters, which are variables that are passed to the function when it is called. Here's an example of a function that takes a parameter:

PHP Program

```
function greet($name) {
         echo "Hello, $name!";
        }
```

To call this function with a specific parameter value, you simply include the value inside the parentheses when you call the function:

PHP Program

```
greet("John");
```

This will execute the code inside the greet() function, using the value "John" for the $name parameter, and output "Hello, John!" to the screen.

Functions can also return values, using the return keyword. Here's an example:

PHP Program

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

        $sum = $num1 + $num2;

        return $sum;

        }
```

To call this function and store the returned value in a variable, you simply assign the function call to a variable:

PHP Program

```
$result = add(3, 5);

        echo $result; // Output: 8
```

This will execute the code inside the add() function, using the values 3 and 5 for the $num1 and $num2 parameters, and return the sum of those values. The returned value is then assigned to the $result variable, which is output to the screen using the echo statement.

PHP Function with arguments

In PHP, you can define functions that take arguments, which are values that are passed to the function when it is called. Here's an example of a function that takes two arguments:

PHP Code

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

        $sum = $num1 + $num2;

        echo "The sum of $num1 and $num2 is $sum.";

        }
```

To call this function and pass it two arguments, you simply include the values inside the parentheses when you call the function:

PHP Code

```
add(3, 5);
```

This will execute the code inside the add() function, using the values 3 and 5 for the $num1 and $num2 parameters, and output "The sum of 3 and 5 is 8." to the screen.

You can also define functions with default values for their arguments, which are used if the function is called without a specific value for that argument. Here's an example:

PHP Code

```
function greet($name = "world") {
        echo "Hello, $name!";
        }
```

To call this function with a specific value for the $name argument, you include the value inside the parentheses when you call the function:

PHP Code

```
greet("John");
```

This will execute the code inside the greet() function, using the value "John" for the $name parameter, and output "Hello, John!" to the screen.

If you call the function without a specific value for the $name argument, the default value of "world" will be used:

PHP Code

```
greet();
```

This will execute the code inside the greet() function, using the default value of "world" for the $name parameter, and output "Hello, world!" to the screen.

PHP Function with Default Arguments

In PHP, you can define functions with default arguments, which are used if the function is called without a specific value for that argument. Here's an example of a function that takes two arguments, with a default value for the second argument:

PHP Code

```
function greet($name, $message = "Hello") {
        echo "$message, $name!";
        }
```

To call this function and pass it two arguments, you simply include the values inside the parentheses when you call the function:

PHP Code

```
greet("John", "Hi");
```

This will execute the code inside the greet() function, using the values "John" and "Hi" for the $name and $message parameters, and output "Hi, John!" to the screen.

If you call the function without a specific value for the $message argument, the default value of "Hello" will be used:

PHP Code

```
greet("Jane");
```

This will execute the code inside the greet() function, using the value "Jane" for the $name parameter, and the default value of "Hello" for the $message parameter, and output "Hello, Jane!" to the screen.

Function argument with call by value

In PHP, function arguments can be passed by value or by reference. When an argument is passed by value, a copy of the value is made and used inside the function. Any changes made to the argument inside the function will not affect the original value of the argument outside the function.

Here's an example of a function that takes an argument by value:

PHP Code

```
function increment($number) {
        $number++;
        echo "Inside function: $number<br>";
        }
$num = 5;
increment($num);
echo "Outside function: $num<br>";
```

In this example, the increment() function takes an argument $number by value. When the function is called with the variable $num as an argument, a copy of the value 5 is made and used inside the function. The function increments the value of $number by 1, so inside the function the value of $number is 6.

However, when the function is called outside the function, the original value of $num is still 5. This is because the value of $num was passed by value to the increment() function, and the changes made to the copy of the value inside the function did not affect the original value of $num outside the function.

The output of this code will be:

```
Inside function: 6
Outside function: 5
```

So, if you want to modify the value of a variable passed as an argument to a function, you need to pass it by reference instead of by value.

Function argument with call by reference in PHP

In PHP, you can pass function arguments by reference by using the & symbol before the argument name. When a function argument is passed by reference, changes made to the argument inside the function will affect the original value of the argument outside the function.

Here's an example of a function that takes an argument by reference:

PHP Code

```
function increment(&$number) {
        $number++;
        echo "Inside function: $number<br>";
        }
$num = 5;
increment($num);
echo "Outside function: $num<br>";
```

In this example, the increment() function takes an argument $number by reference using the & symbol. When the function is called with the variable $num as an argument, a reference to the original value of $num is passed to the function.

The function increments the value of $number by 1, so inside the function the value of $number is 6. However, when the function is called outside the function, the original value of $num is also 6. This is because the changes made to the value of $number inside the function affected the original value of $num outside the function.

The output of this code will be:

```
Inside function: 6
Outside function: 6
```

So, if you want to modify the value of a variable passed as an argument to a function, you need to pass it by reference using the & symbol.

Global and Local Scope of Function in PHP

In PHP, variables have different scopes depending on where they are defined. The same is true for functions - functions can have local or global scope.

A function with local scope means that any variables defined inside the function are only accessible within that function. Here's an example:

PHP Code

```
function myFunction() {
        $x = 10;
        echo "The value of x is: $x<br>";
        }
myFunction(); // Output: The value of x is: 10
echo "The value of x is: $x<br>"; // Output: Undefined variable: x
```

In this example, the variable $x is defined inside the myFunction() function, so it has local scope. This means that it can only be accessed from within the function. When we try to access $x outside of the function, we get an "Undefined variable" error because $x does not exist outside of the function.

A function with global scope means that any variables defined inside the function are accessible anywhere in the code, including outside the function. Here's an example:

PHP Code

```
$x = 5;

function myFunction() {

        global $x;

        $x = 10;

        }

myFunction();

echo "The value of x is: $x<br>"; // Output: The value of x is: 10
```

In this example, we define the variable $x outside the function myFunction(), so it has global scope. Inside the function, we use the global keyword to access the global variable $x. We then change the value of $x to 10 inside the function.

When we call myFunction() and then try to output the value of $x outside the function, we see that the value of $x has changed to 10. This is because we changed the value of the global variable $x inside the function.

So, when defining functions in PHP, you can choose whether to use local or global scope depending on your needs. If you want a variable to be accessible only within the function, you should define it with local scope. If you want a variable to be accessible from anywhere in the code, you should define it with global scope.

Array

Anatomy of an Array in PHP

In PHP, an array is a data structure that stores a collection of elements, which can be of any data type. Each element in the array is identified by a unique index or a key, depending on the type of array.

Here is the basic anatomy of an array in PHP:

1. Declaration: An array can be declared using the array() construct or the shorthand [] notation. For example:

PHP Code

```
// using array() construct

$array1 = array('apple', 'banana', 'orange');

// using shorthand [] notation
```

```
$array2 = ['apple', 'banana', 'orange'];
```

2. Indexing: The elements in the array can be accessed using their index/key. In PHP, arrays can have both numeric and string keys. For example:

PHP Code

```
// numeric indexing

echo $array1[0]; // outputs 'apple'

// string indexing

$array3 = ['name' => 'John', 'age' => 30];

echo $array3['name']; // outputs 'John'
```

3. Adding/Removing Elements: Elements can be added or removed from an array using various built-in functions in PHP. For example:

PHP Code

```
// adding elements

$array1[] = 'grape'; // adds 'grape' to the end of the array

$array3['email'] = 'john@example.com'; // adds a new key-value pair to the array

// removing elements

unset($array1[1]); // removes the element at index 1

unset($array3['age']); // removes the 'age' key-value pair
```

4. Iteration: You can iterate over the elements in an array using various loop constructs in PHP, such as the for, foreach, and while loops. For example:

PHP Code

```
// using a for loop

for ($i = 0; $i < count($array1); $i++) {

        echo $array1[$i] . ' ';

        }

// using a foreach loop

foreach ($array3 as $key => $value) {

  echo $key . ': ' . $value . ' ';

        }
```

5. Array Functions: PHP provides many built-in functions to manipulate arrays, such as array_push(), array_pop(), array_merge(), and array_slice(). For example:

PHP Code

```
// adding elements using array_push()

array_push($array1, 'kiwi');

// removing the last element using array_pop()

$last_element = array_pop($array1);

// merging two arrays using array_merge()

$array4 = ['pear', 'pineapple'];

$merged_array = array_merge($array1, $array4);

// extracting a slice of an array using array_slice()

$sliced_array = array_slice($merged_array, 1, 3);
```

Overall, arrays are a fundamental data structure in PHP, and understanding their anatomy and various operations is crucial for writing effective PHP code.

Creating index based and Associative array in PHP

In PHP, you can create two types of arrays: indexed arrays and associative arrays.

1. Indexed Array: An indexed array is an array in which each element is identified by its index, which starts from 0 and increases by 1 for each subsequent element. You can create an indexed array in PHP using the array() construct or the shorthand [] notation, as shown below:

PHP Code

```
// Using array() construct

$fruits = array('apple', 'banana', 'orange', 'mango');

// Using shorthand [] notation

$fruits = ['apple', 'banana', 'orange', 'mango'];
```

2. Associative Array: An associative array is an array in which each element is identified by a string key instead of a numeric index. You can create an associative array in PHP using the array() construct and by specifying key-value pairs within it, as shown below:

PHP Code

```
// Using array() construct

$user = array(

        'name' => 'John Doe',

        'email' => 'john.doe@example.com',

        'age' => 30,

        'city' => 'New York'

       );
```

In the above example, 'name', 'email', 'age', and 'city' are the keys of the array, and their corresponding values are 'John Doe', 'john.doe@example.com', 30, and 'New York', respectively.

You can also create an associative array using the shorthand [] notation by specifying key-value pairs within it, as shown below:

PHP Code

```
$user = [
        'name' => 'John Doe',
        'email' => 'john.doe@example.com',
        'age' => 30,
        'city' => 'New York'
        ];
```

In both cases, you can access the elements of the array using their keys or indexes, as shown below:

PHP Code

```
// Accessing elements of indexed array
echo $fruits[0]; // Outputs 'apple'


// Accessing elements of associative array
echo $user['name']; // Outputs 'John Doe'
```

In summary, indexed arrays use numeric indexes to identify elements, while associative arrays use string keys to identify elements.

Accessing array in PHP

In PHP, you can access array elements using their index or key. The index/key is used to retrieve a specific element or a range of elements from the array.

Here are some ways to access arrays in PHP:

1. Accessing a single element: To access a single element in an indexed array, you can use its index as shown below:

PHP Code

```
$fruits = array('apple', 'banana', 'orange');
echo $fruits[1]; // Outputs 'banana'
```

To access a single element in an associative array, you can use its key as shown below:

PHP Code

```
$user = array(
        'name' => 'John Doe',

        'email' => 'john.doe@example.com',

        'age' => 30,

        'city' => 'New York'

        );

echo $user['name']; // Outputs 'John Doe'
```

2. Accessing a range of elements: To access a range of elements in an indexed array, you can use the array_slice() function as shown below:

PHP Code

```
$fruits = array('apple', 'banana', 'orange', 'mango');

$sliced_fruits = array_slice($fruits, 1, 2); // Returns 'banana', 'orange'
```

In the above example, the array_slice() function is used to retrieve elements from index 1 to index 2 (inclusive) from the $fruits array.

To access a range of elements in an associative array, you can use a loop, such as a foreach loop, as shown below:

PHP Code

```
$user = array(

        'name' => 'John Doe',

        'email' => 'john.doe@example.com',

        'age' => 30,

        'city' => 'New York'

        );

foreach (array('name', 'email') as $key) {

        echo $user[$key] . '<br>';

        }
```

In the above example, the loop iterates through the array of keys ('name' and 'email') and retrieves the corresponding values from the $user array.

3. Checking if an element exists: To check if an element exists in an array, you can use the isset() function or the array_key_exists() function. The isset() function checks if the element exists and is not null, while the array_key_exists() function checks if the key exists in the array.

PHP Code

```
$fruits = array('apple', 'banana', 'orange');

if (isset($fruits[1])) {
        echo 'The element exists';
        } else {
        echo 'The element does not exist';
        }
if (array_key_exists('name', $user)) {
        echo 'The key exists';
        } else {
        echo 'The key does not exist';
        }
```

In the above example, the isset() function is used to check if the element at index 1 exists in the $fruits array, while the array_key_exists() function is used to check if the 'name' key exists in the $user array.

Overall, accessing arrays is a fundamental operation in PHP, and understanding the various ways to access arrays is important for writing effective PHP code.

Looping with Index based array in PHP

In PHP, you can loop through indexed arrays using different types of loops, such as for, foreach, and while loops. Here are some examples:

1. Using for loop:

PHP Code

```
$fruits = array('apple', 'banana', 'orange', 'mango');
for ($i = 0; $i < count($fruits); $i++) {
        echo $fruits[$i] . '<br>';
        }
```

In the above example, a for loop is used to iterate through the $fruits array using its index. The count() function is used to get the number of elements in the array.

2. Using foreach loop:

PHP Code

```
$fruits = array('apple', 'banana', 'orange', 'mango');
foreach ($fruits as $fruit) {
        echo $fruit . '<br>';
        }
```

In the above example, a foreach loop is used to iterate through the $fruits array. The $fruit variable holds the value of each element in the array.

3. Using while loop:

PHP Code

```
$fruits = array('apple', 'banana', 'orange', 'mango');

$i = 0;

while ($i < count($fruits)) {

        echo $fruits[$i] . '<br>';

        $i++;

        }
```

In the above example, a while loop is used to iterate through the $fruits array using its index. The $i variable holds the index of each element in the array.

In all of the above examples, the output will be:

```
apple

banana

orange

mango
```

Overall, looping through indexed arrays is a common operation in PHP, and understanding the different types of loops and how to use them can help you write more efficient and effective PHP code.

Associative array in PHP using each() and foreach() functions

In PHP, an associative array is an array that uses named keys instead of numeric indices. You can create an associative array in PHP using the array() function, or by using the short syntax [ ].

Here are some examples of creating associative arrays in PHP:

PHP Code

```
// Using array() function

$user = array(

        'name' => 'John Doe',

        'email' => 'john.doe@example.com',

        'age' => 30,

        'city' => 'New York'

        );

// Using short syntax

$user = [
        'name' => 'John Doe',

        'email' => 'john.doe@example.com',

        'age' => 30,

        'city' => 'New York'

        ];
```

Once you have an associative array, you can loop through its elements using the each() and foreach() functions.

1. Using the each() function:

PHP Code

```
$user = array(

             'name' => 'John Doe',

            'email' => 'john.doe@example.com',

            'age' => 30,

            'city' => 'New York'

);

while ($element = each($user)) {

            echo $element['key'] . ': ' . $element['value'] . '<br>';

            }
```

In the above example, the each() function is used to iterate through the $user associative array. The function returns each element of the array as an array with two keys: 'key' and 'value'. The while loop continues until all elements have been processed.

2. Using the foreach() function:

PHP Code

```
$user = array(

            'name' => 'John Doe',

            'email' => 'john.doe@example.com',

            'age' => 30,

            'city' => 'New York'

            );

foreach ($user as $key => $value) {

            echo $key . ': ' . $value . '<br>';

            }
```

In the above example, the foreach() function is used to iterate through the $user associative array. The $key variable holds the name of each key, while the $value variable holds the corresponding value.

Both the each() and foreach() functions produce the same output:

```
name: John Doe

email: john.doe@example.com

age: 30

city: New York
```

Overall, looping through associative arrays in PHP is a common operation, and understanding how to use the each() and foreach() functions can help you write more efficient and effective PHP code.

Some useful Library function for array handling in PHP

PHP provides many built-in library functions for handling arrays. Here are some useful ones:

1. array_push() - adds one or more elements to the end of an array

PHP Code

```php
$fruits = array('apple', 'banana', 'orange');

array_push($fruits, 'mango', 'kiwi');

print_r($fruits);
```

Output:

```
Array

(

        [0] => apple

        [1] => banana

        [2] => orange

        [3] => mango

        [4] => kiwi

)
```

2. array_pop() - removes the last element from an array and returns it

PHP Code

```php
$fruits = array('apple', 'banana', 'orange');

$last_fruit = array_pop($fruits);

echo $last_fruit; // outputs "orange"
```

3. array_shift() - removes the first element from an array and returns it

PHP Code

```
$fruits = array('apple', 'banana', 'orange');
$first_fruit = array_shift($fruits);
echo $first_fruit; // outputs "apple"
```

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

PHP Code

```
$fruits = array('apple', 'banana', 'orange');
array_unshift($fruits, 'mango', 'kiwi');
print_r($fruits);
```

Output:

```
Array
(
        [0] => mango
        [1] => kiwi
        [2] => apple
        [3] => banana
        [4] => orange
)
```

5. array_slice() - returns a slice of an array

PHP Code

```
$fruits = array('apple', 'banana', 'orange', 'mango', 'kiwi');
$slice = array_slice($fruits, 1, 3);
print_r($slice);
```

Output:

```
Array
(
        [0] => banana
        [1] => orange
        [2] => mango
)
```

6. array_merge() - merges two or more arrays into a single array

PHP Code

```
$fruits1 = array('apple', 'banana', 'orange');

$fruits2 = array('mango', 'kiwi');

$all_fruits = array_merge($fruits1, $fruits2);

print_r($all_fruits);
```

Output:

```
Array
(
        [0] => apple
        [1] => banana
        [2] => orange
        [3] => mango
        [4] => kiwi
        )
```

These are just a few of the many useful functions for handling arrays in PHP. Understanding these functions and their usage can make working with arrays in PHP easier and more efficient.

Unit-IV

String Manipulation and Regular Expression

Creating and accessing String in PHP

In PHP, a string is a sequence of characters enclosed in quotes. Here are some examples of creating and accessing strings in PHP:

1. Creating a string

PHP Code

```
$name = "John Doe";
```

2. Concatenating two strings using the dot (.) operator

PHP Code

```
$first_name = "John";

$last_name = "Doe";

$name = $first_name . " " . $last_name;

echo $name; // outputs "John Doe"
```

3. Accessing individual characters in a string using square brackets []

PHP Code

```
$name = "John Doe";

echo $name[0]; // outputs "J"

echo $name[5]; // outputs "D"
```

4. Finding the length of a string using the strlen() function

PHP Code

```
$name = "John Doe";

echo strlen($name); // outputs 8
```

5. Finding the position of a substring within a string using the strpos() function

PHP Code

```
$phrase = "The quick brown fox jumps over the lazy dog";

$position = strpos($phrase, "brown");

echo $position; // outputs 10
```

6. Replacing a substring within a string using the str_replace() function

PHP Code

```
$phrase = "The quick brown fox jumps over the lazy dog";

$new_phrase = str_replace("brown", "red", $phrase);

echo $new_phrase; // outputs "The quick red fox jumps over the lazy dog"
```

These are just a few examples of working with strings in PHP. There are many more built-in functions available for manipulating strings, such as strtolower(), strtoupper(), substr(), and more.

Searching and Replacing String, Formatting String, joining and splitting String in PHP

Here are some examples of how to search and replace strings, format strings, and join/split strings in PHP using PHP's built-in functions:

1. Searching and replacing strings using str_replace()

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

$new_text = str_replace("quick", "slow", $text);

echo $new_text; // outputs "The slow brown fox jumps over the lazy dog"
```

2. Formatting strings using sprintf()

PHP Code

```
$price = 9.99;
```

```
$formatted_price = sprintf("The price is $%.2f", $price);

echo $formatted_price; // outputs "The price is $9.99"
```

3. Joining strings using implode()

PHP Code

```
$fruits = array("apple", "banana", "orange");

$fruit_string = implode(", ", $fruits);

echo $fruit_string; // outputs "apple, banana, orange"
```

4. Splitting strings using explode()

PHP Code

```
$fruit_string = "apple, banana, orange";

$fruits = explode(", ", $fruit_string);

print_r($fruits); // outputs Array ( [0] => apple [1] => banana [2] => orange )
```

5. Searching for a string within another string using strpos()

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

$pos = strpos($text, "brown");

if ($pos !== false) {

   echo "The word 'brown' was found at position $pos.";

} else {

   echo "The word 'brown' was not found.";

}
```

Output: "The word 'brown' was found at position 10."

These are just a few examples of the many built-in string functions available in PHP. Other useful functions include strcasecmp(), substr(), ucwords(), and more.

String Related Library functions in PHP

Here are some commonly used library functions for string manipulation in PHP:

1. strtolower() - Converts a string to lowercase

PHP Code

```
$text = "HELLO WORLD";

$text_lower = strtolower($text);

echo $text_lower; // outputs "hello world"
```

2. strtoupper() - Converts a string to uppercase

PHP Code

```
$text = "hello world";

$text_upper = strtoupper($text);

echo $text_upper; // outputs "HELLO WORLD"
```

3. ucfirst() - Converts the first character of a string to uppercase

PHP Code

```
$text = "hello world";

$text_ucfirst = ucfirst($text);

echo $text_ucfirst; // outputs "Hello world"
```

4. ucwords() - Converts the first character of each word in a string to uppercase

PHP Code

```
$text = "hello world";

$text_ucwords = ucwords($text);

echo $text_ucwords; // outputs "Hello World"
```

5. str_replace() - Replaces all occurrences of a string in another string

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

$text_new = str_replace("brown", "red", $text);

echo $text_new; // outputs "The quick red fox jumps over the lazy dog"
```

6. strlen() - Returns the length of a string

PHP Code

```
$text = "Hello world";

$length = strlen($text);

echo $length; // outputs 11
```

7. substr() - Returns a substring from a string

PHP Code

```
$text = "Hello world";

$substring = substr($text, 0, 5);

echo $substring; // outputs "Hello"
```

8. trim() - Removes whitespace or other characters from the beginning and end of a string

PHP Code

```
$text = "   Hello world   ";

$text_trimmed = trim($text);

echo $text_trimmed; // outputs "Hello world"
```

These are just a few examples of the many built-in string functions available in PHP. Other useful functions include strcasecmp(), str_repeat(), str_split(), and more.

Use and advantage of regular expression over inbuilt function in PHP

Regular expressions offer several advantages over built-in string functions in PHP, including:

1. Flexibility: Regular expressions provide a flexible way to match and manipulate strings based on patterns. You can use regular expressions to match complex patterns that cannot be easily achieved with built-in functions.
2. Portability: Regular expressions are supported in many programming languages and tools, so once you learn how to use them in one language, you can use them in others.
3. Efficiency: Regular expressions can be more efficient than using a series of built-in functions, especially when dealing with large amounts of data or complex patterns.
4. Standardization: Regular expressions provide a standardized way of working with strings, making it easier for other developers to understand and maintain your code.
5. Expressiveness: Regular expressions allow you to express complex patterns using a concise syntax, which can make your code easier to read and understand.

However, regular expressions do have some disadvantages as well. They can be more difficult to learn and understand than built-in functions, and they can also be slower in some cases. Additionally, regular expressions can be prone to errors if not written carefully. It is important to weigh the advantages and disadvantages of regular expressions before deciding whether to use them over built-in functions in a given situation.

Use of preg_match(), preg_replace(), preg_split() functions in regular expression

In PHP, the preg_match(), preg_replace(), and preg_split() functions are commonly used with regular expressions.

1. preg_match(): This function searches a string for a pattern and returns true if it finds a match, or false if no match is found. It can also return an array of matches if capturing groups are used in the pattern.

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

if (preg_match('/brown/', $text)) {

  echo "Match found!";

} else {

  echo "No match found.";
}

// Output: Match found!
```

2. preg_replace(): This function searches a string for a pattern and replaces all occurrences of the pattern with a specified string. It can also use capturing groups to insert matched values into the replacement string.

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

$new_text = preg_replace('/brown/', 'red', $text);

echo $new_text;

// Output: The quick red fox jumps over the lazy dog
```

3. preg_split(): This function splits a string into an array of substrings based on a specified pattern.

PHP Code

```
$text = "The quick brown fox jumps over the lazy dog";

$words = preg_split('/\s+/', $text);

print_r($words);

// Output: Array ( [0] => The [1] => quick [2] => brown [3] => fox [4] => jumps [5] => over [6] => the [7]
=> lazy [8] => dog )
```

These functions provide a powerful way to search, replace, and split strings using regular expressions. However, regular expressions can be complex and difficult to read, so it is important to use them carefully and to test them thoroughly to ensure they are working as intended.