Arrays in PHP: Definition, Creating, Accessing Array**, Types of Arrays:** Indexed, Associative arrays, Multidimensional arrays, Accessing Array, Manipulating Arrays, displaying array, Array Functions.
Using Functions in PHP: Definition, Creating, invoking, user-defined functions, Formal parameters, actual parameters, Function and variable scope, Recursion, Library functions, Date and Time Functions.
Strings in PHP: Definition, Creating, Declaring, formatting strings, String Functions.

PHP Functions
The real power of PHP comes from its functions.In PHP, there are more than 700 built-in functions.
A function will be executed by a call to the function.
You may call a function from anywhere within a page.
A function is a block of code written in a program to perform some specific task. Functions take information's as parameter, execute a block of statements or perform operations on these parameters and return the result.

Function definition:
In PHP, you can define functions using the function keyword. Here's the basic syntax for defining a function:

```
function functionName($parameter1, $parameter2, /* ... */)
{
// Function body
// Code to be executed when the function is called
// You can use $parameter1, $parameter2, and other parameters here
// Optionally, you can return a value using the return statement
eturn $someValue;
}
```

Let's break down the components:

- function: The keyword used to declare a function.
- Function Name: The name of your function. Choose a meaningful and descriptive name.
- $parameter1, $parameter2, etc.: Parameters are variables that you can pass to the function. They act as placeholders for values that will be provided when the function is called.
- { /* ... */ }: The function body contains the code that will be executed when the function is called. It can include any valid PHP code.
- return $someValue;: The return statement is used to send a value back to the calling code. If your function doesn't need to return a value, you can omit this part.

Here's an example of a simple function:

```
function addNumbers($num1, $num2) {
$sum = $num1 + $num2; return $sum;
}
$result = addNumbers(3, 4);
echo $result;
```

// Output: 7
In this example, the addNumbers function takes two parameters, adds them together, and returns the result. The function is then called with the arguments 3 and 4, and the result is echoed to the screen.

Remember that function names in PHP are not case-sensitive, but it's a good practice to follow a consistent naming convention for functions.

PHP provides us with two major types of functions:

1. **Built-in functions:** PHP provides us with huge collection of built-in library functions. These functions are already coded and stored in form of functions. To use those we just need to call them as per our requirement like, var_dump, fopen(), print_r(), gettype() and so on.
2. **User Defined Functions:** Apart from the built-in functions, PHP allows us to create our own customized functions called the user- defined functions.

Why should we use functions?

1. **Reusability:** If we have a common code that we would like to use at various parts of a program, we can simply contain it within a function and call it whenever required. This reduces the time and effort of repetition of a single code. This can be done both within a program and also by importing the PHP file, containing the function, in some other program

2. **Easier error detection:** Since, our code is divided into functions, we can easily detect in which function, and the error could lie and fix them fast and easily.
3. **Easily maintained:** If anything or any line of code needs to be changed, we can easily change it inside the function and the change will be reflected everywhere, where the function is called. Hence, easy to maintain.

CREATING A FUNCTION

- While creating a user defined function we need to keep few things in mind:
  1. Any name ending with an open and closed parenthesis is a function.
  2. A function name always begins with the keyword function.
  3. To call a function we just need to write its name followed by parenthesis.
  4. A function name cannot start with a number. It can start with an alphabet or underscore.
  5. A function name is not case-sensitive.

Syntax:

function function_name()

```
{
        Executable code;
}
```

Example:

```
<?php
    function func()
    {
        echo "This is PHP program using Functions";
    }
    func();
?>
```

Output:

This is PHP program using Functions

Calling a Function in PHP

Once a function is declared, it can be invoked (called) by simply using the function name followed by parentheses.

```
<?php
  function sum($a, $b) {
   return $a + $b;
}
echo sum(5, 3);  // Outputs: 8
?>
```

Output

8

In this example:

- The **function sum()** is defined to take two parameters, $a and $b, and return their sum.
- The function is called with 5 and 3 as arguments, and it returns the result, which is 8.

Types of Functions in PHP

PHP has two types of functions:

1. User-Defined Functions

A user-defined function is created to perform a specific task as per the developer's need. These functions can accept parameters, perform computations, and return results.

```
<?php
 function addNumbers($a, $b) {
   return $a + $b;
}
echo addNumbers(5, 3); // Output: 8
?>
```

Output
8
In this example:

- The function addNumbers is defined by the user to take two parameters, $a and $b, and returns their sum ($a + $b).
- The function is called with 5 and 3 as arguments, so it adds these numbers together.

2. Built-in Functions
PHP comes with many built-in functions that can be directly used in your code. For example, strlen(), substr(), array_merge(), date() and etc are built-in PHP functions. These functions provide useful functionalities, such as string manipulation, date handling, and array operations, without the need to write complex logic from scratch.

```
<?php
 $str = "Hello, World!";
echo strlen($str); // Output: 13
?>
```

Output
13
In this example:

- A string variable $str is defined with the value "Hello, World!".
- The strlen() function is the built-in function in PHP that is used to calculate the length of the string $str.

FUNCTION PARAMETERS OR ARGUMENTS

- The information or variable, within the function's parenthesis, are called parameters.
- These are used to hold the values executable during runtime.
- A user is free to take in as many parameters as he wants, separated with a comma(,) operator. These parameters are used to accept inputs during runtime.
- While passing the values like during a function call, they are called arguments.
- An argument is a value passed to a function and a parameter is used to hold those arguments. In common term, both parameter and argument mean the same.

Syntax:

```
function function_name($first_parameter, $second_parameter)
{
        executable code;
}
```

Example:

```
<?php
    function pro($num1, $num2, $num3)
    {
        $product = $num1 * $num2 *
        $num3;  echo "The product is
        $product";
    }
    pro(2, 3, 5);
 ?>
```

Output: The product is 30

SETTING DEFAULT VALUES FOR FUNCTION PARAMETER

- PHP allows us to set default argument values for function parameters.
- If we do not pass any argument for a parameter then PHP will use the default set value for this parameter in the function call.

Example:

```
<?php
    function defk($str, $num=12)
    {
        echo "$str is $num years old \n";
    }
    defk("Ram", 15);
    defk("Adam");
 ?>
```

Output: Ram is 15 years old

Adam is 12 years old

In the above example, the parameter $num has a default value 12, if we do not pass any value for this parameter in a function call then this default value 12 will be considered. Also the parameter $str has no default value, so it is compulsory.

RETURNING VALUES FROM FUNCTIONS

- Functions can also return values to the part of program from where it is called.
- The return keyword is used to return value back to the part of program, from where it was called.
- The returning value may be of any type including the arrays and objects.
- The return statement also marks the end of the function and stops the execution after that and returns the value.

Example:

```
<?php
        function pro($num1, $num2, $num3)
        {
                $product = $num1 * $num2 *
                $num3; return $product;
        }
        $retValue = pro(2, 3, 5);
        echo "The product is $retValue";
 ?>
```

Output: The product is 30

PHP Function Parameters and Arguments

Functions can accept input values, known as parameters or arguments.

- Parameters are variables defined in the function declaration that accept values.
- Arguments are the actual values passed to the function when it is called.

These values can be passed to the function in two ways:

1. Passing by Value

When you pass a value by value, the function works with a copy of the argument, so the original value remains unchanged.

```
<?php
 function add($x, $y) {
   $x = $x + $y;
   return $x;
}

echo add(2, 3);  // Outputs: 5
?>
```

Output

5

In this example:

- The function add() is defined with parameters $x and $y.
- The function adds $x and $y and returns the sum.
- The function is called with values 2 and 3, and the result 5 is printed.

2. Passing by Reference

By passing an argument by reference, any changes made inside the function will affect the original variable outside the function.

```
<?php
function addByRef(&$x, $y) {
   $x = $x + $y;
}
$a = 5;
addByRef($a, 3);
echo $a;  // Outputs: 8
?>
```

Output

8

In this example:

- The function addByRef() is defined with $x passed by reference.
- The function modifies $x directly, adding $y to it.
- After calling the function with $a = 5 and 3, the value of $a becomes 8.

Anonymous Functions (Closures)

PHP supports anonymous functions, also known as closures. These functions do not have a name and are often used for passing functions as arguments to other functions.

```
<?php
 $greet = function($name) {
   echo "Hello, " . $name . "!";
};
$greet("GFG");
?>
```

Output

Hello, GFG!

In this example:

- An anonymous function is defined and assigned to the variable $greet.
- The function takes $name as a parameter and prints a greeting message.
- The function is called with the argument "GFG", and the greeting "Hello, GFG" is printed.

VARIABLE FUNCTIONS

- PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.
- Among other things, this can be used to implement callbacks, function

tables, and so forth.

Example:

```
function  foo()
{
    echo "hi<br />";
}

function  bar($arg  =  '')
{
    echo "  argument was '$arg'.<br />n";
}
function  echoit($string)
{
    echo $string;
}

$func  =  'foo';
$func();          // This calls foo()
$func  =  'bar';
$func('test');  // This calls bar()
$func  =  'echoit';
$func('test');  // This calls echoit()
?>
```

Output:

```
hi
argument
was 'test'.
test
```

Formal parameters versus Actual parameters:

In PHP (as well as in many other programming languages), there is a distinction between formal parameters and actual parameters (also known as formal arguments and actual arguments) when discussing functions or methods.

Formal Parameters:
Formal parameters are the variables declared in the function or method signature. They act as placeholders for the values that will be passed into the function when it is called.
These parameters are part of the function definition and are used within the function body as variables representing the values passed during the function invocation.
Formal parameters are what you see in the function declaration.
Example:

```
function addNumbers($num1, $num2) {
$sum = $num1 + $num2;
return $sum;
}
```

In the function addNumbers, $num1 and $num2 are formal parameters.

Actual Parameters:
Actual parameters, on the other hand, are the values or expressions that are passed to a function or method during its invocation.
These are the concrete values that are used in place of the formal parameters when calling the function.
Actual parameters are provided when calling the function.
Example:

```
$result = addNumbers(3, 4);
```

In this example, 3 and 4 are actual parameters. They are the values passed to the addNumbers function when it is called.

So, in summary, formal parameters are the variables declared in the function definition, and actual parameters are the values passed to the function when it is called. The actual parameters are used to fill the roles of the formal parameters within the function body.

Formal parameters	**Actual parameters**

Formal parameter is the parameter defined in the function header.
Syntax:
// function header
type func name (type paraml, type param2,...) { // function body
}
Example:
int sum(int a, int b)
{
return a+b;
}
Here, param1 and param2 are the formal parameters of sum function.
	Actual parameter is the argument passed to the function during the function call.
	Syntax:
	// function call statement
	func func_na name (paraml, param2, ...);
	
	Example:
	sum(x, y);
	Here, x and y are the actual parameters passed to the sum function and gets mapped to the formal parameters paraml and param2 respectively.

The formal parameters are locally available within the particular function only.	The actual parameters are part of the calling function. Example: If main method calls a function sum, then main is the calling function and sum is the called method.

Function and variable scope:
In PHP, the scope of a variable or function refers to the context in which it can be accessed or modified. PHP supports different scopes, including global scope, local scope, and static scope.
Variable Scope:

Global Scope:
Variables declared outside of any function or class have global scope.
They can be accessed from anywhere in the script, including within functions.
Example:$globalVar = 10;
function exampleFunction() {
echo $GLOBALS['globalVar']; // Accessing global variable
}
exampleFunction(); // Output: 10

Local Scope:
Variables declared inside a function have local scope.
They are only accessible within that function.
Example:
function exampleFunction() {
$localVar = 5;
echo $localVar;
}
exampleFunction(); // Output: 5
// This will result in an error, as $localVar is not defined outside the function
// echo $localVar;

Static Scope:
Variables declared as static inside a function retain their value between function calls.
They have function scope but persist across multiple invocations.
Example:
function counter() {
static $count = 0;
$count++;
echo $count;
}
counter(); // Output: 1
counter(); // Output: 2
Function Scope:

Global Scope:

Functions declared outside of any other function or class are in the global scope.
They can be called from anywhere in the script.
Example:
function globalFunction() {
echo "This is a global function.";

```
}
globalFunction(); // Output: This is a global function.
```

Local Scope:

Functions can also be declared within other functions or methods.
The inner function has access to variables in its own scope and the scope of the outer function.
Example:

```
function outerFunction() {
$outerVar = "Outer";
function innerFunction() {
global $outerVar;
echo $outerVar; // Accessing variable from outer function
}
innerFunction();
}
outerFunction(); // Output: Outer
```

Understanding variable and function scope is crucial for writing maintainable and bug-free code in PHP. Properly managing scope helps prevent unintended variable clashes and ensures that functions can access the data they need.

Recursion:
PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.
It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

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

Example 1: Printing number

```
<?php
function display($number) {
if($number<=5){
echo "$number <br/>";
display($number+1);
}
}
display(1);
?>
```

Library functions:
Library functions in PHP refer to built-in functions that are provided by the PHP language and can be used for a variety of tasks. These functions cover a wide range of functionalities, from string

manipulation to array operations, file handling, and more. Here are some categories of library functions commonly used in PHP:

String Functions:

strlen: Returns the length of a string.
strpos: Finds the position of the first occurrence of a substring in a string.
str_replace: Replaces all occurrences of a search string with a replacement string.

Array Functions:

count: Counts the number of elements in an array.
array_push: Pushes one or more elements onto the end of an array.
array_pop: Pops the element off the end of an array.

Math Functions:

abs: Returns the absolute value of a number.
round: Rounds a floating-point number to the nearest integer.
rand: Generates a random number.

File Functions:
file_get_contents: Reads entire file into a string.
file_put_contents: Write a string to a file.
fopen, fclose, fwrite: Functions for opening, closing, and writing to files.

Database Functions (for database interactions):

mysqli_connect, mysqli_query, mysqli_fetch_assoc: Functions for MySQL database interactions.
PDO: PHP Data Objects provide a uniform method of access to multiple databases.

Other Common Functions:

isset: Determines if a variable is set and is not NULL

empty: Checks if a variable is empty.
explode: Splits a string by a specified delimiter into an array.
These are just a few examples, and PHP provides a rich set of functions for various tasks. You can find the complete list of PHP functions in the official PHP documentation

Date and time function:
PHP provides a variety of functions for working with date and time. Here are some commonly used date and time functions in PHP:

- date function:

Returns the current date and time based on the specified format.
$currentDate = date("Y-m-d H:i:s");

```
echo $currentDate;
```

- time function:

Returns the current Unix timestamp (number of seconds since the Unix Epoch).

```
$timestamp = time();
echo $timestamp;
```

- strtotime function:

Parses an English textual date or time description and returns a Unix timestamp.

```
$nextWeek = strtotime("+1 week");
echo date("Y-m-d", $nextWeek);
```

- mktime function:

Returns the Unix timestamp for a date.

```
$timestamp = mktime(12, 0, 0, 2, 28, 2022);
echo date("Y-m-d H:i:s", $timestamp);
```

- date_create and date_format functions (DateTime object):

date_create creates a new DateTime object.
date_format formats the DateTime object.

```
$date = date_create('2022-02-28');
echo date_format($date, 'Y-m-d');
```

- strftime function:

Formats a local time/date according to locale settings.

```
setlocale(LC_TIME, 'en_US'); // Set locale to English (United States)
echo strftime("%A, %B %d, %Y %H:%M:%S");
```

- strtotime with date function for relative dates:

Using strtotime along with date allows you to work with relative dates.

```
$nextMonth = strtotime("+1 month");
echo date("Y-m-d", $nextMonth);
```

These are just a few examples of the many date and time functions available in PHP. The choice of which function to use depends on your specific requirements and the format in which you need to work with dates and times.