Web Development with PHP
Practical - 2
PHP Variables & Loops

PHP (Hypertext Preprocessor) is a popular server-side scripting language that is widely used for web development. One of the most fundamental concepts in PHP is variables. Variables are used to store data or values that can be manipulated or used throughout a PHP script. In simple terms, a variable is like a container that holds a specific piece of information, such as a string of text, a number, or a boolean value. In PHP (Hypertext Preprocessor) a variable is a container that can store a value or a reference to a value. Variables can hold various data types such as strings, integers, floats, and booleans.

In PHP, a variable is declared using a $ sign, followed by the variable name. In PHP, you don't have to declare the variable first and then use it, but in PHP the variable is created at the moment you assign it a value. Also, in PHP we do not have to specify the type of data that we will be storing in a variable. You can create a variable and save any type of data in it. Hence PHP is quite a loosely typed language.

Syntax:

```
<?php
    $variableName = value;
?>
```

Rules for creating Variables in PHP

We have a few basic rules that you must keep in mind while creating variables in PHP.

- A variable name will always start with a $ sign, followed by the variable name.
- A variable name should not start with a numeric value. It can either start with an alphabet or an underscore sign _.
- A variable name can only contain alphabets, numbers and underscore symbols _.
- Variable names in PHP are case-sensitive, which means $var is not the same as $VAR.

Types of variables

In PHP there are various types of variables. This categorization is based on the scope of the variable.

- Global variables – These variables have global scope and visibility to access from anywhere. For accessing them in a separate file, class or function we have to use the global keywords. For example global $global_variable_name.
- Local variables – The variables that are defined and used within some specific function or any other program block are called local variables.
- Superglobals – These are predefined global variables. For example, $_GET, $_SERVER etc. It can be accessed from anywhere without the global keyword.

PHP *echo* and *print* functions

Echo and Print methods of PHP are not built-in functions of PHP language, but they are language constructs. A language construct is accepted/executed by the PHP parser as it is, in other words the PHP parser doesn't have to parse and modify a language construct to execute it, as it is ready for execution by default. Hence, language constructs are faster than any built-in functions.

echo() function is used to print or output one or more strings. It is specifically mentioned string here because, the syntax of the echo function is:

```
echo(string)
```

Although you can use echo() function to output anything, a PHP parser will automatically convert it into string type. echo doesn't need parenthesis, although you can use parenthesis if you want.

```
<?php
    echo "I am open";
    echo ("I am enclosed in parenthesis");

                                                      echo
'This','is','a','broken','sentence';

    echo "This is a
     multiline sentence
     example";
    $str = "I am a string variable";
    echo $str;

      echo  "Hello,  this  is  a  \"beautiful\"
picture";

    $weird = "Stupid";
    echo "I am $weird";
    echo 'I am $weird';
?>
```

When using double quotes the value of the string variable gets printed, while using single quotes, the variable is printed as it is. The PHP print is exactly the same as echo, with the same syntax and same usage. Replace echo with print in all the above examples.

S.No.	echo statement	print statement
1.	echo accepts a list of arguments (multiple arguments can be passed), separated by commas.	print accepts only one argument at a time.
2.	It returns no value or returns void.	It returns the value 1.
3.	It displays the outputs of one or more strings separated by commas.	The print outputs only the strings.
4.	It is comparatively faster than the print statement.	It is slower than an echo statement.

PHP Constants

Constants are variables whose value cannot be changed. In other words, once you set a value for a constant, you cannot change it. In PHP, there are two ways to define a constant:

- It defines a constant at runtime.

  ```
  define(name, value, case-insensitive)
  ```

  - ➔ name: Name of the constant
  - ➔ value: Value of the constant
  - ➔ case-insensitive: Specifies whether the constant name is case sensitive or not. Its default value is false, which means, by default, the constant name is case sensitive.

    ```
    <?php
        define(OMG, "Oh! my God.");
        echo OMG;
    ?>
    <?php
        define(OMG, "Oh! my God.", true);
        echo omg;
    ?>
    ```

- Using the const keyword.

  We can also define constants in PHP using the const keyword. But we can only use the const keyword to define scalar constants, i.e. only integers, floats, booleans and strings, while define() can be used to define array and resource constants as well. When we define a constant using the const keyword, the constant name is always case sensitive.

  The const keyword defines constants at compile time. It is a language construct, not a function.

  ```
  <?php

      const OMG = "Oh! my God.";
      echo OMG;
  ?>
  ```

Another very important point to remember is that while naming a constant, we don't have to use $ symbol with the constant's name.

PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

- ***Scalar Types***
    - It holds only a single value. There are 4 scalar data types in PHP.
        - ➔ **boolean** - A boolean data type can have two possible values, either True or False.
        - ➔ **integer** - An integer value can be negative or positive, but it cannot have a decimal.
        - ➔ **float** - Float data type is used to store any decimal numeric value. A float value can also be either negative or positive.
        - ➔ **string** - String data type in PHP and in general, is a sequence of characters(or anything, it can be numbers and special characters too) enclosed within quotes. You can use single or double quotes.
- ***Compound Types***
    - It can hold multiple values. There are 2 compound data types in PHP.
        - ➔ **array** - An array is used to store multiple values, generally of the same type, in a single variable.
        - ➔ **object** - An object is an instance of the class which holds the local variables with values assigned and using the object we can call the local methods defined in the class.
- ***Special Types***
    - There are 2 special data types in PHP.
        - ➔ **resource** - A resource is a special variable, holding a reference to an external resource. It typically holds special handlers to open files and database connections.
        - ➔ **NULL** - NULL data type is a special data type which means nothing. It can only have one value, and that is NULL.

PHP Strings

A string is a sequence of characters enclosed in quotes. In PHP, strings can be enclosed in single or double quotes.

```
          $str1 = 'Hello, World!';
```

Strings enclosed in double quotes are slower than single-quoted strings but can parse more escape sequences and variable values.

```
$str2 = "The value of x is $x."; // double-quoted
string with variable interpolation

$str3 = "He said, \"I'm going home.\""; //
double-quoted string with escape sequence
```

PHP - Scope of the variable

- Local
  - 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. Hence a variable declaration outside the function with the same name is completely different from the variable declared inside the function.

```
<?php
    function mytest()
    {
        $lang = "PHP";
        echo "Local Scope in  " .$lang;
    }
    mytest();
    //using $lang (local Var) outside the
function
    //will generate an error
    echo $lang;
?>
```

- Global
  - 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.

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

    - Another way to use the global variable inside the function is the predefined $GLOBALS array.

```
<?php
    $num1 = 55;      //global variable
    $num2 = 3;     //global variable
    function global_var()
    {
                    $sum  =  $GLOBALS['num1']  +
$GLOBALS['num2'];
            echo "Sum of global variables is: "
.$sum;
    }
    global_var();
?>
```

- Static
    - 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 variables. We use the static keyword before the variable to define a variable, and this variable is called a static variable. Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope.

```
<?php
    function static_var()
    {
        static $num1 = 0; //static variable
        $num2 = 100;      //Non-static variable
        //increment in non-static variable
        $num1++;
        //increment in static variable
        $num2++;
        echo "Static: " .$num1 ."</br>";
        echo "Non-static: " .$num2 ."</br>";
    }
    static_var();
    static_var();
    static_var();
    static_var();
    static_var();
?>
```

Operators

➔ Arithmetic operators

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

➔ Assignment operators

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of the right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

➔ Comparison operators

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

➔ Increment/Decrement operators

Operator	Name	Example	Explanation
++	Increme nt	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrem ent	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

➔ Logical operators

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

➔ String operators

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

➔ Array operators

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

➔ Bitwise Operators

Operator	**Name**	**Example**	**Explanation**
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

➔ PHP Conditional Assignment Operators

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescin g	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

➔ Execution Operators
PHP has an execution operator **backticks (``)**. PHP executes the content of backticks as a shell command. Execution operator and **shell_exec()** give the same result.

Operator	Name	Example	Explanation
``	backticks	echo `dir`;	Execute the shell command and return the result. Here, it will show the directories available in the current folder.

PHP Operators Precedence

Operators	Additional Information	Associativity
clone new	clone and new	non-associative
[	array()	left
**	arithmetic	right
++ -- ~ (int) (float) (string) (array) (object) (bool) @	increment/decrement and types	right
instanceof	types	non-associative
!	logical (negation)	right
* / %	arithmetic	left
+ - .	arithmetic and string concatenation	left
<< >>	bitwise (shift)	left
< <= > >=	comparison	non-associative
== != === !== <>	comparison	non-associative
&	bitwise AND	left

^	bitwise XOR	left
|	bitwise OR	left
&&	logical AND	left
||	logical OR	left
?:	ternary	left
= += -= *= **= /= .= %= &= |= ^= <<= >>= =>	assignment	right
and	logical	left
xor	logical	left
or	logical	left
,	many uses (comma)	left

PHP Conditional Statements

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

PHP Loops

Loops are used to execute the same block of code again and again, as long as a certain condition is true.

In PHP, we have the following loop types:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array

Whitespace & line breaks
In most cases, whitespace and line breaks don't have special meaning in PHP. Therefore, you can place a statement in one line or span it across multiple lines.

For example, the following code snippets are equivalent:

```
login( $username, $password );
```

And:

```
login(
   $username,
   $password
);
```

PHP constants
A constant is simply a name that holds a single value. As its name implies, the value of a constant cannot be changed during the execution of the PHP script.

To define a constant, you use the define() function. The define() function takes the constant's name as the first argument and the constant value as the second argument. For example:

```
define(name, value, case-insensitive);

<?php
    define('WIDTH','1140px');
    //define('WIDTH','1140px', true);
    Echo WIDTH;
?>
```

- By convention, constant names are uppercase. Unlike a variable, the constant name doesn't start with the dollar sign($).
- By default, constant names are case-sensitive. It means that WIDTH and width are different constants.
- It was possible to define case-insensitive constants. However, it's deprecated since PHP 7.3
- In PHP 5, a constant can hold a simple value like a number, a string, a boolean value. From PHP 7.0, a constant can hold an array. For example:

```
<?php
    define( 'ORIGIN', [0, 0] );
?>
```

- Like superglobal variables, you can access constants from anywhere in the script.

PHP provides you with another way to define a constant via the const keyword. Here's the syntax:

```
const CONSTANT_NAME = value;
```

In this syntax, you define the constant name after the const keyword. To assign a value to a constant, you use the assignment operator (=) and the constant value. The constant value can be scalar, e.g., a number, a string, or an array.

```
<?php
    const SALES_TAX = 0.085;
    $net_price = 100 * (1 + SALES_TAX);
    echo $net_price; // 108.5
?>
```

The following example uses the const keyword to define the COUNT constant that holds an array:

```
<?php
    const COUNT = ['ONE', 'TWO', 'THREE'];
?>
```

define vs const

- First, the define() is a function while the const is a language construct.
    - It means that the define() function defines a constant at run-time, whereas the const keyword defines a constant at compile time.
    - In other words, you can use the define() function to define a constant conditionally like this:
      ```
      <?php
          if(condition) {
              define('WIDTH', '1140px');
          }
      ?>
      ```
    - You cannot use the const keyword to define a constant this way. For example, the syntax of the following code is invalid:
      ```
      <?php
          if(condition) {
             const WIDTH = '1140px';
          }
      ?>
      ```
- Second, the define() function allows you to define a constant with the name that comes from an expression. For example, the following defines three constants OPTION_1, OPTION_2, and OPTION_3 with the values 1, 2, and 3.
  ```
  <?php
      define('PREFIX', 'OPTION');
      define(PREFIX . '_1', 1);
      define(PREFIX . '_2', 2);
      define(PREFIX . '_3', 3);
  ```

- However, you cannot use the const keyword to define a constant name derived from an expression.
- Unless you want to define a constant conditionally or use an expression, you can use the const keyword to define constants to make the code more clear.
- Note that you can use the const keyword to define constants inside classes.

var_dump function

- The var_dump() is a built-in function that allows you to dump the information about a variable. The var_dump() function accepts a variable and displays its type and value.

```
<?php
    $test = 100;
    echo '<pre>';
    var_dump($test);
    echo '</pre>';

    $message = Hello World';
    echo '<pre>';
    var_dump($message);
    echo '</pre>';
?>
```

die() function

- The die() function displays a message and terminates the execution of the script:

```
die($status);
```

- Sometimes, you want to terminate the script immediately. In this case, you can use the die() function.

```
<?php
    // Single-quoted string isn't processed.
    echo 'Hello\nWorld!'.'<br>';
    // Double-quoted string is processed.
    echo "Hello\nWorld!".'<br>';
    die(“Stop Script”);
    echo "Hello\nWorld!".'<br>';
?>
```

Examples:

```
<?php
    $greeting = 'Hello World!';
    echo $greeting[0];
    echo $greeting[2];
    echo $greeting[11];
?>

<?php
    echo gettype(10), "\n";
    echo gettype(10.0), "\n";
    echo gettype('Hello World!'), "\n";
    echo gettype(""), "\n";
    echo gettype(true), "\n";
    echo gettype(false), "\n";
    echo gettype([]), "\n";
    echo gettype(NULL), "\n";
?>

<?php
    $is_active = true;
    if ($is_active) {
      echo 'User is active';
    } else {
      echo 'User is inactive';
    }
    $num1 = 123;
    $num3 = 0123; // octal integer
    $z = 0x1a;
    $float1 = 1.23; // decimal float
    $float2 = 1.2e3; // exponential float (1.2 x
    10^3)
    $float3 = 0.123456789; // precision float
    $bool1 = true; // true boolean
    $bool2 = false; // false boolean
    $bool3 = 1; // true represented by integer 1
    $bool4 = 0; // false represented by integer 0

?>
```

Boolean

- A boolean value represents a truth value. In other words, a boolean value can be either true or false. PHP uses the bool type to represent boolean values.
- To represent boolean literals, you can use the true and false keywords. These keywords are case-insensitive.
- When you use non-boolean values in a boolean context, e.g., if statement. PHP evaluates that value to a boolean value. The following values evaluate to false:
    - The keyword false
    - The integer zero (0)
    - The floating-point number zero (0.0)
    - The empty string ('') and the string "0"
    - The NULL value
    - An empty array, i.e., an array with zero elements
    - PHP evaluates other values to true.
-

Integer

- Integers are whole numbers such as -3, -2, -1, 0, 1, 2, 3… PHP uses the int type to represent the integers.
- The range of integers depends on the platform where PHP runs. Typically, integers has a range from -2,147,438,648 to 2,147,483,647. It's equivalent to 32 bits signed.
- To get the size of the integer, you use the PHP_INT_SIZE constant. Also, you use the PHP_INT_MIN and PHP_INT_MAX constants to get the minimum and maximum integer values.
- PHP represents integer literals in decimal, octal, binary, and hexadecimal formats.
- PHP uses a sequence of digits without leading zeros to represent decimal values. The sequence may begin with a plus or minus sign. If it has no sign, then the integer is positive.
    - From PHP 7.4, you can use the underscores (_) to group digits in an integer (Decimal) to make it easier to read. For example, instead of using the following number:

      1000000

      you can use the underscores (_) to group digits like this:

      1_000_000

- Octal numbers consist of a leading zero and a sequence of digits from 0 to 7. Like decimal numbers, the octal numbers can have a plus (+) or minus (-)

sign.

- Hexadecimal numbers consist of a leading 0x and a sequence of digits (0-9) or letters (A-F). The letters can be lowercase or uppercase. By convention, letters are written in uppercase. Similar to decimal numbers, hexadecimal numbers can include a sign, either plus (+) or minus(-).
- Binary numbers beginning with 0b are followed by a sequence of digits 0 and 1. The binary numbers can include a sign.

Float

- Floating-point numbers represent numeric values with decimal digits.
- Floating-point numbers are often referred to as floats, doubles, or real numbers. Like integers, the range of the floats depends on the platform where PHP runs.
- PHP also supports the floating-point numbers in scientific notation
    - Since PHP 7.4, you can use the underscores in floats to make long numbers more readable. For example:

      1_234_457.89
- Since the computer cannot represent exact floating-point numbers, it can only use approximate representations.
- For example, the result of 0.1 + 0.1 + 0.1 is 0.299999999…, not 0.3. It means that you must be careful when comparing two floating-point numbers using the == operator.

```
$total = 0.1 + 0.1 + 0.1;
echo var_dump($total == 0.3);
```

String

- In PHP, a string is a sequence of characters. PHP provides you with four ways to define a literal string, including single-quoted, double-quoted, heredoc syntax, and nowdoc syntax.
- When evaluating a double-quoted string, PHP replaces the value of any variable that you place inside the string. This feature is called variable interpolation in PHP.

```
<?php
    $name = 'John';
    echo 'Hello ' . $name;  //single quote
    echo "Hello $name";     //double quote
    echo "Hello {$name}";   //double quote
    //echo 'Hello {$name}';   //Wrong
```

- Besides substituting the variables, the double-quoted strings also accept special characters, e.g., \n, \r, \t by escaping them.

- It’s a good practice to use single-quoted strings when you don’t use variable interpolation because PHP doesn’t have to parse and evaluate them for double-quoted strings.
- A string has a zero-based index. It means that the first character has an index of 0. The second character has an index of 1 and so on.

```
<?php
    $title = 'PHP string′;
    echo $title[0];
```

- When you place variables in a double-quoted string, PHP will expand the variable names. If a string contains the double quotes (“), you need to escape them using the backslash character(\).
- PHP heredoc strings behave like double-quoted strings, without the double-quotes. It means that they don’t need to escape quotes and expand variables.

```
//Double Quote
<?php
    $he = 'Bob';
    $she = 'Alice';
    $text = "$he said, \"PHP is awesome\".
    \"Of course.\" $she agreed.";
    echo $text;
?>
//Heredoc
<?php
    $he = 'Bob';
    $she = 'Alice';
    $text = <<<TEXT
    $he said "PHP is awesome".
    "Of course" $she agreed."
    TEXT;
    echo $text;
?>
```

- The following shows the syntax of a heredoc string:

```
<?php
    $str = <<<IDENTIFIER
    place a string here
    it can span multiple lines
    and include single quote ' and double
    quotes "
    IDENTIFIER;
?>
```

    - First, start with the <<< operator, an identifier, and a new line:
      <<<IDENTIFIER
    - Second, specify the string, which can span multiple lines and includes single quotes (‘) or double quotes (“).
    - Third, close the string with the same identifier.
        - The identifier must contain only alphanumeric characters and underscores and start with an underscore or a non-digit character.
        - The closing identifier must follow these rules:
            - Begins at the first column of the line
            - Contains no other characters except a semicolon (;).
            - The character before and after the closing identifier must be a newline character defined by the local operating system.
- A <u>nowdoc</u> string is similar to a heredoc string except that it doesn’t expand the variables.

```
<?php
    $str = <<<'IDENTIFIER'
    place a string here
    it can span multiple lines
    and include single quote ' and double
    quotes "
    IDENTIFIER;
?>
```

    - The nowdoc’s syntax is similar to the heredoc’s syntax except that the identifier which follows the <<< operator needs to be enclosed in single quotes. The nowdoc’s identifier also follows the rules for the heredoc identifier.

null

- The null is a special type in PHP. The null type has only one value which is also null. In fact, null indicates the absence of a value for a variable.
- Use the is_null() function or === operator to compare a variable with null.

Exercise :

1. Create two variables of integer type and display the value of the variables.
2. Create a php script which declares all the scalar data types of variables and displays the value .
3. Create a php script which changes the data types using appropriate values. And display the new data types (integer to string , string to integer, Boolean to integer, integer to float, float to integer)
4. Create php script which declare two variable and perform all arithmetic operations on the
variables.
5. Create a php script which shows all the comparison operators working/demonstration.
6. Create a php script which performs string operations.
7. Create a php script which demonstrates the scope of local variables. {display the appropriate error also in output by making changes into script)
8. Create a php script which declares two global variables and demonstrates the scope of the global variable . {display the appropriate error also in output by making changes into script)
9. Create a php script which demonstrates the working and scope of static variables. (hint create function having static variable)
10. Create a php script which prints the first ten odd numbers.
11. Create a php script which checks whether the number is palindrome or not. E.d 121 is a palindrome number.
12. Create a php script which prints the reverse of the number. E.g 123 => 321
13. Create a php script for given conditions.
    `If age < 20 or age > 60 print message` *`not valid age`*
    `If age is between 20 to 35 , print message` *`age is in range of 20 to 35`*
    `If age is between 36 to 55 print message` *`age is in range of 36 to 55`*
    `Else print message` *`age is more than 55`*

14. Create the php script which changes the background color of the div tag according to variable value.

    (hint `<div style="background-color:yellow;">`)