PROGRAMMING ON PHP

BY LECTURER SURAJ PANDEY CCT COLLEGE

PROGRAMMING ON PHP

3.1 Conditional Statements(If, If else, switch statement)

3.2 Iteration and Looping (do while, while, for loop, foreach loop)

3.3 Functions: Built-In and user-defined functions

3.4 String functions and pattern: String comparison, String Concatenation.

3.5 Array: Numeric Array, Associative Array

3.6 One dimensional and multi-dimensional array

CONDITIONAL STATEMENT

While writing programs/scripts, there will be scenarios where you would want to execute a particular statement only **if** some condition is satisfied. In such situations we use **Conditional statements**.

In PHP, there are 4 different types of Conditional Statements.

1. if statements
2. if...else statements
3. if...elseif...else statements
4. switch statement

The if statement

When we want to execute some code when a condition is **true**, then we use if statement.

Syntax:

```
if(condition)
{ // code to be executed if 'condition' is true }
```

Here is a simple example,

```
<?php
$age = 20;
if($age <= 25)
{
echo "You are not allowed to consume alchohol";
}
?>
```

The if...else statement

When we want to execute some code when a condition is **true**, and some other code when that condition is **false**, then we use the if...else pair.

Syntax:

```
if(condition)

{ // code to be executed if 'condition' is true }

else { // code to be executed if 'condition' is false }
```

Here is a simple example,

```
<?php

$age = 26;

if($age <= 25)

{

echo "You are not allowed to consume alchohol";

}

 else

{

 echo "Enjoy the drinks";

}

?>
```

The if...else...elseif statement

When we want to execute different code for different set of conditions, and we have more than 2 possible conditions, then we use if...elseif...else pair.

Syntax:

```
if(condition1)

 { // code to be executed if 'condition1' is true }

elseif(condition2)

{ // code to be executed if 'condition2' is true }

else { /* code to be executed if both 'condition1' and 'condition2' are false */ }
```

Here is a simple example,

```
<?php

// speed in kmph

$speed = 110;

if($speed < 60)

{

 echo "Safe driving speed";

}

elseif($speed > 60 && $speed < 100)

 {

 echo "You are burning extra fuel";

 }

 else

{

 // when speed is greater than 100

echo "Its dangerous";

}

 ?>
```

The switch statement

A switch statement is used to perform different actions, based on different conditions.

Using a switch statement, we can specify multiple conditions along with the code to be executed when that condition is **true**, thereby implementing a menu style program.

syntax:

```
switch(X)
{
case value1:
// execute this code when X=value1
 break;
case value2:
 // execute this code when X=value2
break;
case value3:
 // execute this code when X=value3
break;
...
default:
/* execute this when X matches none of of the specified options */
 }
```

In a switch statement, we provide the deciding factor which can be a variable or an expression to our switch statement, and then we specify the different **cases**, each with a **value**, a piece of code and a **break** statement.

break statement is specified to break the execution of the switch statement once the action related to a specified value has been performed.

If we do not specify a break statement, then all the switch cases, after the matched case, will get executed, until the
next break statement.

The **default** statement is executed if no matching case is there.

```
<?php
$car = "Jaguar";
switch($car)
 {
case "Audi":
echo "Audi is amazing";
 break;
case "Mercedes":
 echo "Mercedes is mindblowing";
 break;
case "Jaguar":
 echo "Jaguar is the best";
break;
 default:
 echo "$car is Ok";
}
?>
```

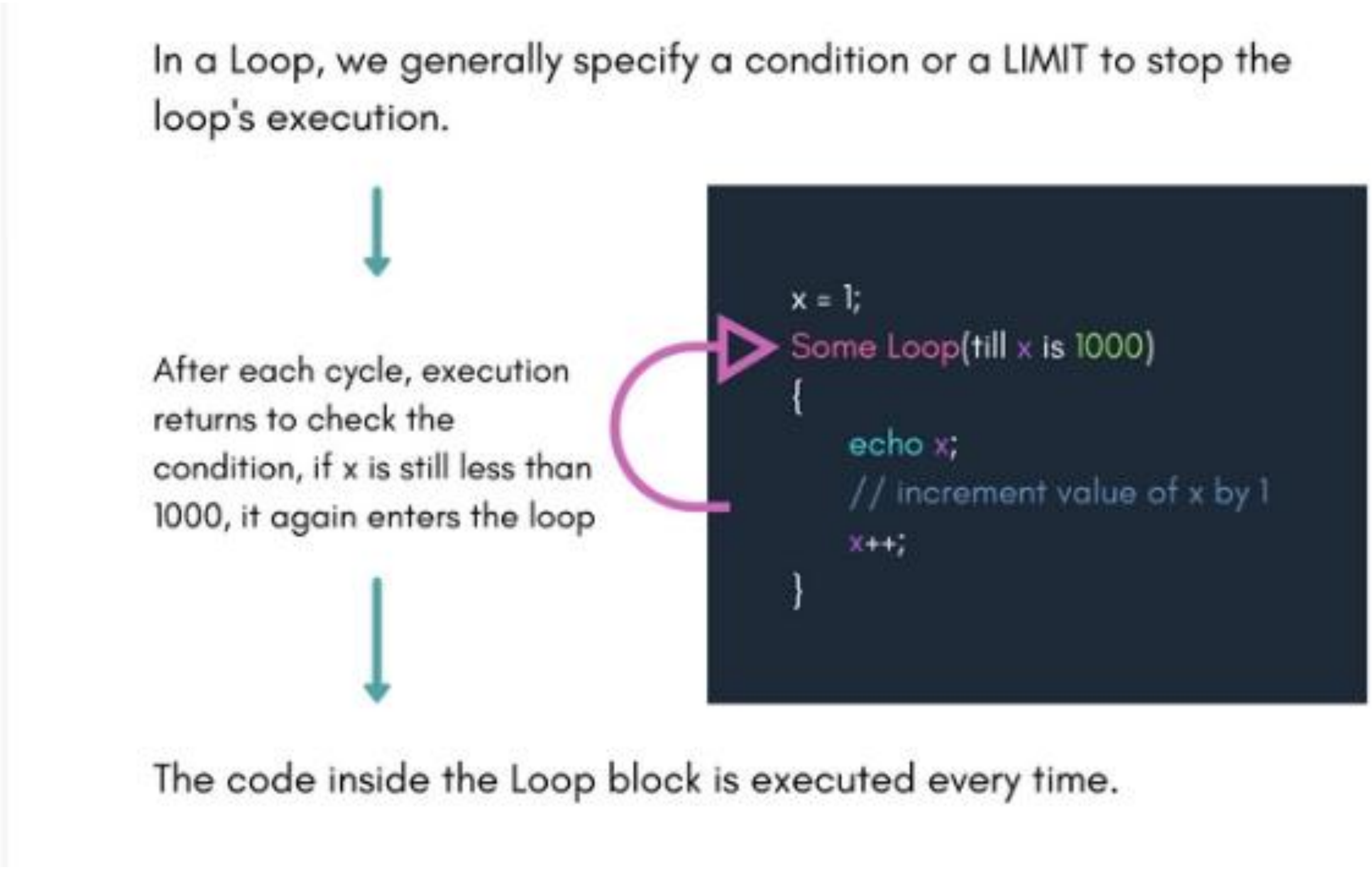
```
Jaguar is the best
```

ITERATION AND LOOPING

As the name suggests, a **Loop** is used to execute something over and over again.

For example, if you want to display all the numbers from 1 to 1000, rather than using echo statement 1000 times, or specifying all the numbers in a single echo statement with newline character \n, we can just use a loop, which will run for 1000 times and every time it will display a number, starting from 1, incrementing the number after each iteration or cycle.

In a Loop, we generally specify a condition or a LIMIT up till which the loop will execute, because if we don't specify such a condition, how will we specify when the loop should end and not go on for infinite time.



In a Loop, we generally specify a condition or a LIMIT to stop the loop's execution.

After each cycle, execution returns to check the condition, if x is still less than 1000, it again enters the loop

```
x = 1;
Some Loop(till x is 1000)
{
    echo x;
    // increment value of x by 1
    x++;
}
```

The code inside the Loop block is executed every time.

PHP while Loop

The while loop in PHP has two components, one is a condition and other is the code to be executed. It executes the given code until the specified condition is true.

Syntax:

```
<?php
while(condition)
{
 /* execute this code till the condition is true */
 }
?>
```

For example, Let's print numbers from 1 to 10.

```
<?php
$a = 1;
 while($a <= 10)
{
echo "$a | ";
$a++;
// incrementing value of a by 1
}
?>
```

OUTPUT

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

PHP do...while Loop

The do...while loop is a little different from all the loops in PHP because it will execute at least one time, even if the condition is false, can you guess how? Well because the condition is checked after the loop's execution, hence the first time when the condition is checked, the loop has already executed once.

Syntax:

```
<?php
Do
 {
 /* execute this code till the condition is true */
}
 while(condition)
?>
```

Let's implement the above example using do...while loop,

```
<?php
$a = 1;
do
{ echo "$a | ";
$a++;
// incrementing value of a by 1
 }
 while($a <= 10)
?>
```

OUTPUT

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

Let's take another example where even if the condition is **false**, still the loop will be executed once.

```
<?php
$a = 11;
 do
 {
 echo $a;
$a++;
 // incrementing value of a by 1
}
while($a <= 10)
?>
```

OUTPUT

```
11
```

As we can see clearly, that the condition in the
above do...while loop will return **false** because value of
variable $a is **11** and as per the condition the loop should be
executed only if the value of $a is less than or equal to **10**.

PHP for Loop

The for loop in PHP doesn't work like while or do...while loop, in case of for loop, we have to declare beforehand how many times we want the loop to run.

Syntax:

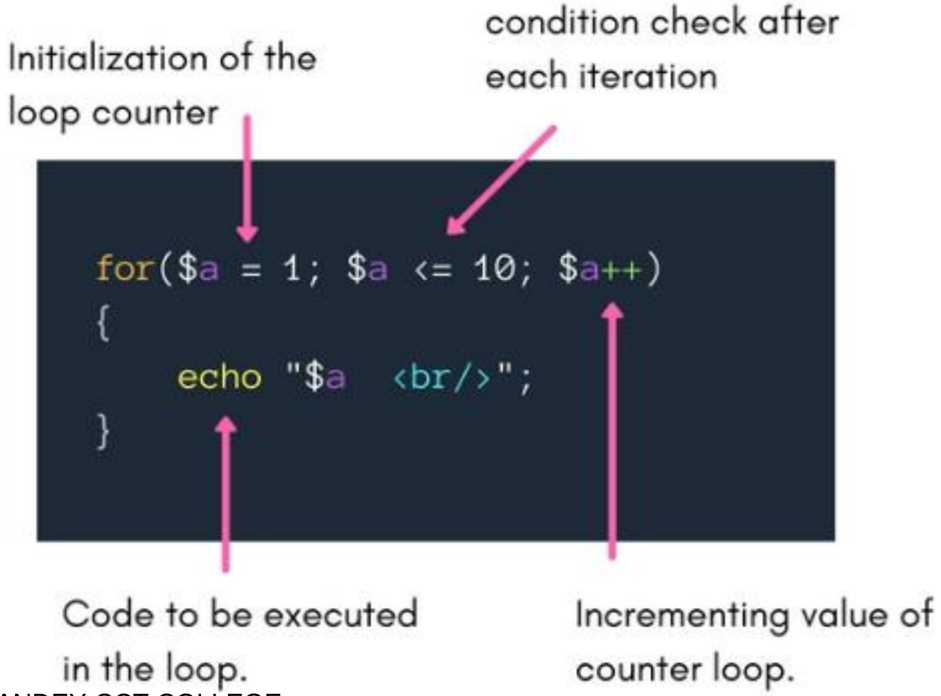
```
<?php
for(initialization; condition; increment/decrement)
 {
 /* execute this code till the condition is true */
}
 ?>
```

The parameters used have following meaning:

initialization: Here we initialize a variable with some value. This variable acts as the loop counter.

condition: Here we define the condition which is checked after each iteration/cycle of the loop. If the condition returns **true**, then only the loop is executed.

increment/decrement: Here we increment or decrement the loop counter as per the requirements.


Initialization of the loop counter
condition check after each iteration
Code to be executed in the loop.
Incrementing value of counter loop.

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

print numbers from 1 to 10, this time we will be using the for loop.

```php
<?php
for($a = 1; $a <= 10; $a++)
{
 echo "$a <br/>";
}
?>
```

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

Nested for Loops

We can also use a for loop inside another for loop. Here is a simple example of nested for loops.

```
<?php
for($a = 0; $a <= 2; $a++)
{
for($b = 0; $b <= 2; $b++)
{
echo "$b $a ";
}
}
?>
```

```
0
 0
 1
 0
 2
 0 0 1 1 1 2 1 0 2 1 2 2 2
```

foreach Loop

The foreach loop in PHP is used to access key-value pairs of an array. This loop only works with arrays and you do not have to initialise any loop counter or set any condition for exiting from the loop, everything is done implicitly(internally) by the loop.

Syntax:

```
<?php
foreach($array as $var)
{
/* execute this code for all the array elements $var will represent all the array elements starting from first element, one by one */
 }
 ?>
```

Here is a simple example.

```
<?php
 $array = array("Jaguar", "Audi", "Mercedes", "BMW"); foreach($array
as $var)
{
 echo "$var <br/>";
 }
 ?>
```

OUTPUT

```
Jaguar
Audi
 Mercedes
BMW
```

break statement

We have already seen and used break statements in the switch conditional statements.

To recall, in switch code blocks, we used break statement to break out of the switch block when a valid case block gets executed.

Let's see an example for simple switch code:

```
<?php
$a = 1;
switch($a)
{
case 1: echo "This is case 1";
 break;
case 2: echo "This is case 2";
break;
default: echo "This is default case";
}
?>
```

OUTPUT

```
This is case 1
```

FUNCTIONS: BUILT-IN AND USER-DEFINED FUNCTIONS

A **Function** is nothing but a 'block of statements' which generally performs a specific task and can be used repeatedly in our program. This 'block of statements' is also given a **name** so that whenever we want to use it in our program/script, we can call it by its **name**.

In PHP there are thousands of built-in functions which we can directly use in our program/script.

PHP also supports **user defined functions**, where we can define our own functions.

A function doesn't execute when its defined, it executed when it is called.

PHP User Defined Functions

Let's understand how we can define our own functions in our program and use those functions.

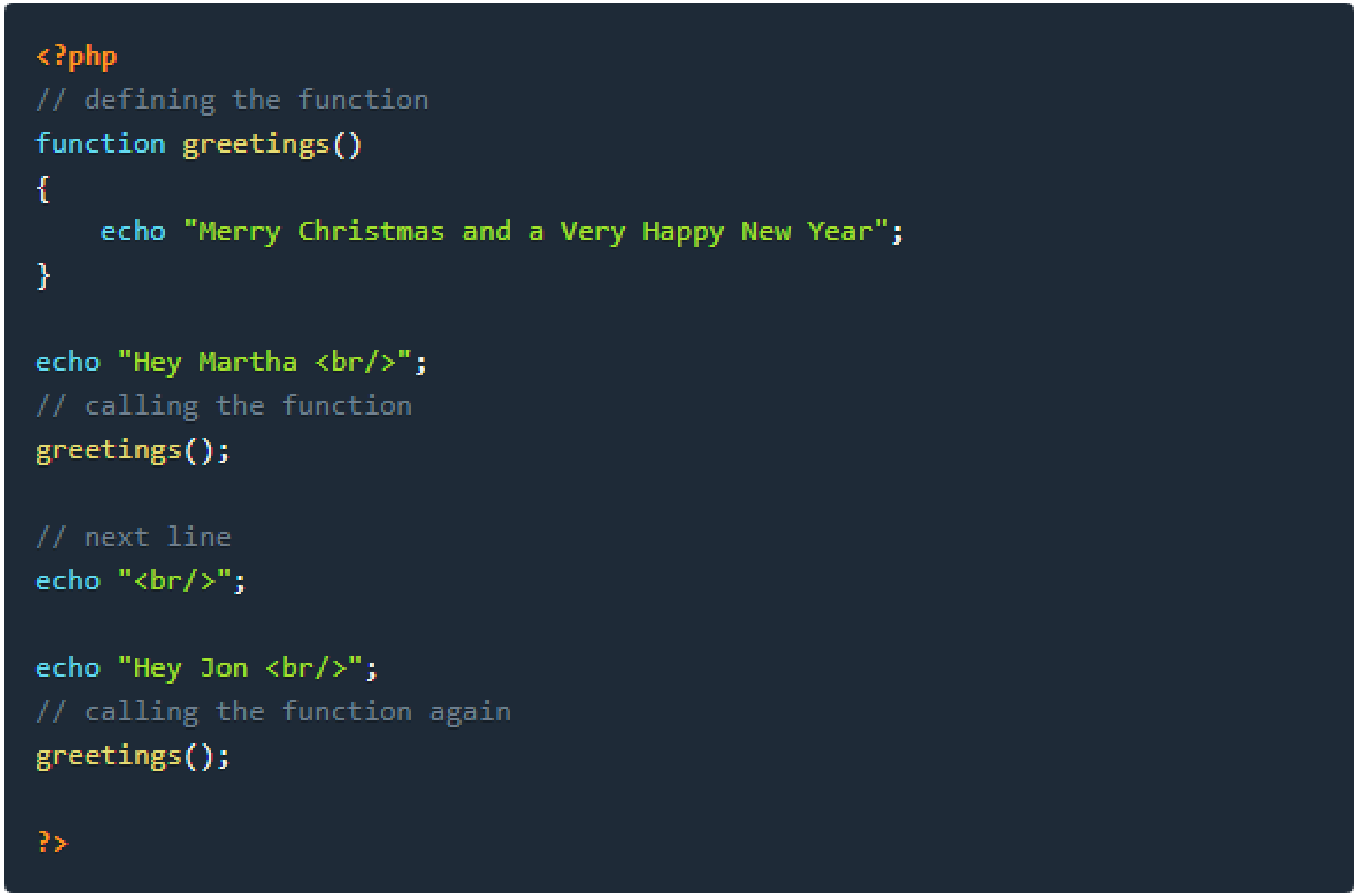
Syntax:

```
<?php
function function_name()
{
// function code statements
}
?>
```

Few Rules to name Functions

1. A **function name** can only contain alphabets, numbers and underscores. No other special character is allowed.
2. The name should start with either an alphabet or an underscore. It should not start with a number.
3. And last but not least, function names are not case-sensitive.
4. The opening curly brace { after the function name marks the start of the function code, and the closing curly brace } marks the end of function code.

write a very simple function which will display a simple "Merry Christmas and a Very Happy New Year" message. This script can actually be very useful when you have to send festive emails to every friend of yours and you have to write the same message in all of them.



```php
<?php
// defining the function
function greetings()
{
    echo "Merry Christmas and a Very Happy New Year";
}

echo "Hey Martha <br/>";
// calling the function
greetings();

// next line
echo "<br/>";

echo "Hey Jon <br/>";
// calling the function again
greetings();

?>
```

Advantages of User-defined Functions

As we have already seen a simple example of using a function above, you must have understood how time-saving it can be for large programs. Here are a few advantages of using functions for you:

Reuseable Code: As it's clear from the example above, you write a function once and can use it for a thousand times in your program.

Less Code Repetition: In the example above we just had one line of code in the function, but what if we have 10 lines of code. So rather than repeating all those lines of code over and over again, we can just create a function for them and simply call the function.

Easy to Understand: Using functions in your program, makes the code more readable and easy to understand.

PHP Function Arguments

We can even pass data to a function, which can be used inside the function block. This is done using arguments. An argument is nothing but a variable.

Arguments are specified after the function name, in parentheses, separated by comma. When we define a function, we must define the number of arguments it will accept and only that much arguments can be passed while calling the function.

Syntax:

```
<?php
/* we can have as many arguments as we want to have in a function */
function function_name(argument1, argument2)
 {
 // function code statements
}
?>
```

```
<?php
 // defining the function with argument function greetings($festival)
 {
echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
// calling the function
greetings("Diwali");
// next line
echo "<br/>";
echo "Hey Jon <br/>";
// calling the function again
greetings("New Year");
 ?>
```

PHP Default Function Arguments

Sometimes function arguments play an important role in the function code execution. In such cases, if a user forgets to provide the argument while calling the function, it might lead to some error.

To avoid such errors, we can provide a default value for the arguments which is used when no value is provided for the argument when the function is called.

Let's take an example.

```
<?php
// defining the function with default argument function greetings($festival = "Life")
 {
 echo "Wish you a very Happy $festival";
}
echo "Hey Jai <br/>";
 // calling the function with an argument
greetings("Diwali");
// next line
echo "<br/>";
echo "Hey Jon <br/>";
// and without an argument
 greetings();
 ?>
```

PHP Function Returning Values

Yes, functions can even return results. When we have functions which are defined to perform some mathematical operation etc, we would want to output the result of the operation, so we return the result.

return statement is used to return any variable or value from a function in PHP.

Let's see an example.

```
<?php

function add($a, $b)

{

$sum = $a + $b;

// returning the result

return $sum;

}

echo "5 + 10 = " . add(5, 10) . "";

?>
```

OUTPUT

```
5 + 10 = 15
```

PHP Function Overloading

Function overloading allows you to have multiple different variants of one function, differentiated by the number and type of arguments they take.

For example, we defined the function add() which takes two arguments, and return the sum of those two. What if we wish to provide support for adding 3 numbers.

To tackle such situations, what we can do is, we can define two different variants of the function add(), one which takes in 2 arguments and another which accepts 3 arguments. This is called **Function Overloading**.

Let's take an example,

```
<?php
// add function with 2 arguments
function add($a, $b)
{
$sum = $a + $b;
// returning the result return $sum;
}
// overloaded add function with 3 arguments
function add($a, $b, $c)
 {
 $sum = $a + $b + $c;
// returning the result
return $sum;
}
// calling add with 2 arguments
echo "5 + 10 = " . add(5, 10) . "<br/>";
 // calling add with 3 arguments
echo "5 + 10 + 15 = " .add(5, 10, 15) . "<br/>";
 ?>
```

STRING FUNCTIONS AND PATTERN: STRING COMPARISON, STRING CONCATENATION.

A string in PHP is a simple character concatenation like "Hello world!", "Lorem Ipsum..." etc. Strings in Php are also a collection of alphanumeric characters, enclosed in single quotes for simple string data and double quotes for complex string data.

```
<?php
 $a = 'Simple string value';
 $b = "Another string value using double quotes.";
echo $a;
echo "<br />";
echo $b;
?>
```

We can define a string using **double quotes** as well as **single quotes**.

But, what if the string has a single or double quote in it, then what?

```
<?php
 $srt1 = "This is "a" String";
 $str2 = 'This is also 'a' String';
 ?>
```

Both the above string definitions will result in **error**, as we cannot include a double quote in the string, if the string is defined inside double quotes.

But it will work, if the string contains double quotes, but it is defined using single quotes and vice versa, for example,

```
<?php
// string inside double quotes, with a single quote
$srt1 = "This is 'a' String"; // string inside single quotes, with a double quote
$str2 = 'This is also "a" String';
?>
```

There is one more simple technique to deal with the quotes problem, it is known as **Escaping Special Character**, which can be done using a **backslash**.

Let's take an example,

```
<?php
 // escaping double quote using backslash
$srt1 = "I\"ll handle this.";
 // escaping single quote using backslash
 $str2 = 'I\'ll handle this.';
 echo $str1;
echo "\n"; // new line
echo $str2;
?>
```

STRING FUNCTIONS

In general practice, using the right string function will save you a lot of time as they are pre-defined in PHP libraries and all you have to do is call them to use them.

Commonly used PHP 5 String Functions

Below we have a list of some commonly used string functions in PHP:

strlen($str)

This function returns the length of the string or the number of characters in the string including whitespaces

```
<?php
$str = "Welcome to Studytonight";
// using strlen in echo method
echo "Length of the string is: ". strlen($str);
?>
```

OUTPUT

```
Length of the string is: 23
```

str_word_count($str)

This function returns the number of words in the string. This function comes in handly in form field validation for some simple validations.

```
<?php

$str = "Welcome to Studytonight";

 // using str_word_count in echo method

echo "Number of words in the string are: ".
str_word_count($str);

?>
```

OUTPUT

```
Number of words in the string are: 3
```

strrev($str)

This function is used to reverse a string.

Let's take an example and see,

```
<?php
$str = "Welcome to Studytonight";
// using strrev in echo method
echo "Reverse: ". strrev($str);
?>
```

OUTPUT

```
Reverse: thginotydutS ot emocleW
```

strpos($str, $text)

This function is used to find the position of any text/word in a given string. Just like an array, string also assign index value to the characters stored in it, starting from zero.

```
<?php
$str = "Welcome to Studytonight";
// using strpos in echo method
echo "Position of 'Studytonight' in string: ". strpos($str, 'Studytonight');
?>
```

OUTPUT

```
Position of 'Studytonight' in string: 11
```

Concatenation of two strings in PHP

There are two string operators. The first is the concatenation operator (‘**.**‘), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator (‘**.=**‘), which appends the argument on the right side to the argument on the left side.

Examples :

Input : string1: Hello

string2 : World!

Output : HelloWorld!

Input : string1: geeksfor

string2: geeks

Output : geeksforgeeks

Output :

```
HelloWorld!
```

```
// First String
$a = 'Hello';

// Second String
$b = 'World!';

// Concatenation Of String
$c = $a.$b;

// print Concatenate String
echo " $c \n";
?>
```

Output :
```
John Carter!
```

```
// First String
$fname = 'John';

// Second String
$lname = 'Carter!';

// Concatenation Of String
$c = $fname." ".$lname;

// print Concatenate String
echo " $c \n";
?>
```

Output :
```
HelloWorld!
```

```
// First String
$a = 'Hello';

// now $a contains "HelloWorld!"
$a. = "World!";

// Print The String $a
echo " $a \n";
?>
```

COMPARISON OF STRING

== operator

The most common way you will see of comparing two strings is simply by using the == operator if the two strings are equal to each other then it returns true.

```
// Using the == operator, Strings match is printed
if('string1' == 'string1')
{
 echo 'Strings match.';
 }
 else
{
echo 'Strings do not match.';
 }
```

```
// Using the == operator, Strings do not match is printed
if('string1' == 'STRING1')
 {
 echo 'Strings match.';
 }
Else
 {
 echo 'Strings do not match.';
}
```

strcmp Function

Another way to compare strings is to use the PHP function **strcmp**, this is a binary safe string comparison function that will return a 0 if the strings match.

```
// strcmp function, Strings match is printed if(strcmp('string1', 'string1') == 0)
 {
 echo 'Strings match.';
}
else
{
echo 'Strings do not match.';
}
```

strcasecmp Function

The previous examples will not allow you to compare different case strings, the following function will allow you to compare case insensitive strings.

```
// Both strings will match
if(strcasecmp('string1', 'string1') == 0)
 {
 echo 'Strings match.';
}
else
{
echo 'Strings do not match.';
}
// Both strings will match even with different case if(strcasecmp('string1', 'String1') == 0)
 {
 echo 'Strings match.';
 }
else
{ echo 'Strings do not match.';
 }
// Both strings will match even with different case
if(strcasecmp('string1', 'STRING1') == 0)
{
echo 'Strings match.';
 }
else
{
echo 'Strings do not match.';
 }
```

ARRAY: NUMERIC ARRAY, ASSOCIATIVE ARRAY

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data.

An array is created using an **array()** function in PHP.

There are basically three types of arrays in PHP:

Indexed or Numeric Arrays: An array with a numeric index where values are stored linearly.

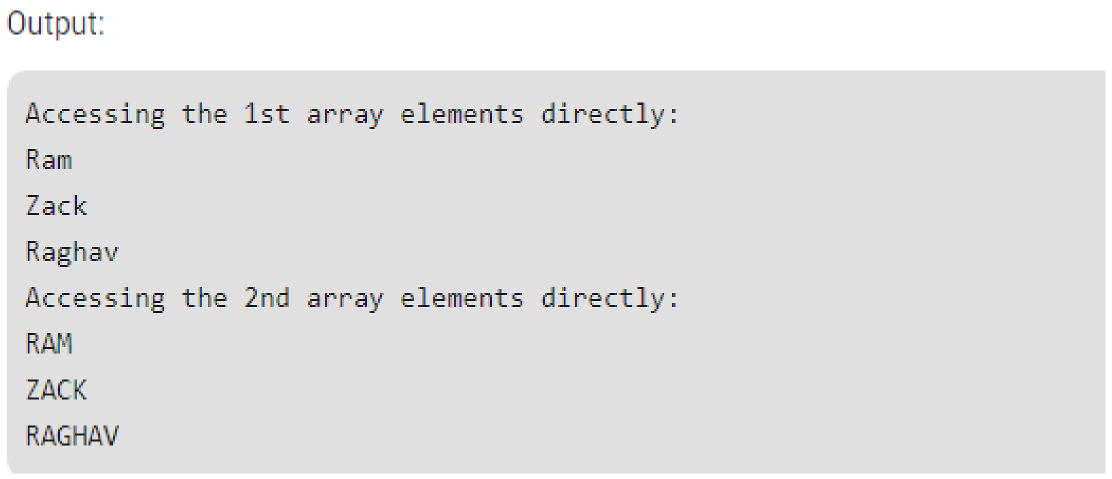
Associative Arrays: An array with a string index where instead of linear storage, each value can be assigned a specific key.

Multidimensional Arrays: An array which contains single or multiple array within it and can be accessed via multiple indices.

Indexed or Numeric Arrays

These type of arrays can be used to store any type of elements, but an index is always a number. By default, the index starts at zero. These arrays can be created in two different ways as shown in the following example:

```php
<?php

// One way to create an indexed array
$name_one = array("Zack", "Anthony", "Ram", "Salim", "Raghav");

// Accessing the elements directly
echo "Accessing the 1st array elements directly:\n";
echo $name_one[2], "\n";
echo $name_one[0], "\n";
echo $name_one[4], "\n";

// Second way to create an indexed array
$name_two[0] = "ZACK";
$name_two[1] = "ANTHONY";
$name_two[2] = "RAM";
$name_two[3] = "SALIM";
$name_two[4] = "RAGHAV";

// Accessing the elements directly
echo "Accessing the 2nd array elements directly:\n";
echo $name_two[2], "\n";
echo $name_two[0], "\n";
echo $name_two[4], "\n";

?>
```


Output:

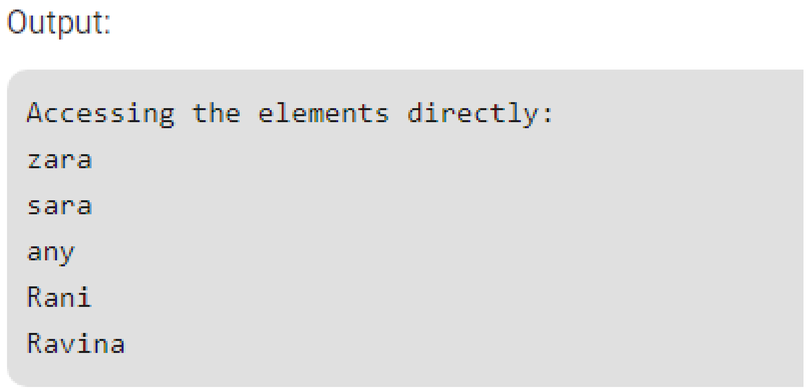
```
Accessing the 1st array elements directly:
Ram
Zack
Raghav
Accessing the 2nd array elements directly:
RAM
ZACK
RAGHAV
```

Associative Arrays

These type of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

Example:

```php
<?php

// One way to create an associative array
$name_one = array("Zack"=>"Zara", "Anthony"=>"Any",
                  "Ram"=>"Rani", "Salim"=>"Sara",
                  "Raghav"=>"Ravina");

// Second way to create an associative array
$name_two["zack"] = "zara";
$name_two["anthony"] = "any";
$name_two["ram"] = "rani";
$name_two["salim"] = "sara";
$name_two["raghav"] = "ravina";

// Accessing the elements directly
echo "Accessing the elements directly:\n";
echo $name_two["zack"], "\n";
echo $name_two["salim"], "\n";
echo $name_two["anthony"], "\n";
echo $name_one["Ram"], "\n";
echo $name_one["Raghav"], "\n";

?>
```



Output:

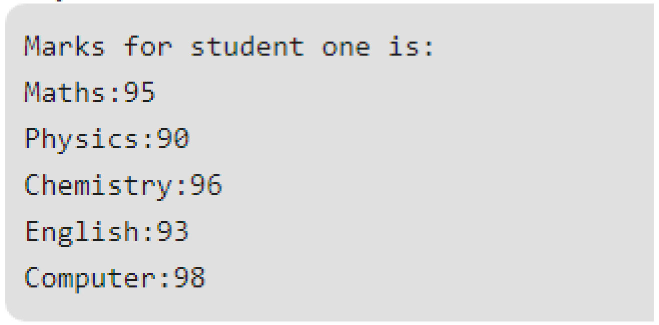
```
Accessing the elements directly:
zara
sara
any
Rani
Ravina
```

Associative arrays are used to store key value pairs. For example, to store the marks of different subject of a student in an array, a numerically indexed array would not be the best choice. Instead, we could use the respective subject’s names as the keys in our associative array, and the value would be their respective marks gained.

Example:
Here **array()** function is used to create associative array.

```
<?php
/* First method to create an associate array. */
$student_one = array("Maths"=>95, "Physics"=>90,
                    "Chemistry"=>96, "English"=>93,
                    "Computer"=>98);

/* Second method to create an associate array. */
$student_two["Maths"] = 95;
$student_two["Physics"] = 90;
$student_two["Chemistry"] = 96;
$student_two["English"] = 93;
$student_two["Computer"] = 98;

/* Accessing the elements directly */
echo "Marks for student one is:\n";
echo "Maths:" . $student_two["Maths"], "\n";
echo "Physics:" . $student_two["Physics"], "\n";
echo "Chemistry:" . $student_two["Chemistry"], "\n";
echo "English:" . $student_one["English"], "\n";
echo "Computer:" . $student_one["Computer"], "\n";
?>
```

Output:



```
Marks for student one is:
Maths:95
Physics:90
Chemistry:96
English:93
Computer:98
```

```php
<?php
// Defining a multidimensional array
$favorites = array(
    array(
        "name" => "Dave Punk",
        "mob" => "5689741523",
        "email" => "davepunk@gmail.com",
    ),
    array(
        "name" => "Monty Smith",
        "mob" => "2584369721",
        "email" => "montysmith@gmail.com",
    ),
    array(
        "name" => "John Flinch",
        "mob" => "9875147536",
        "email" => "johnflinch@gmail.com",
    )
);
// Accessing elements
echo "Dave Punk email-id is: " . $favorites[0]["email"], "\n";
echo "John Flinch mobile number is: " . $favorites[2]["mob"];
?>
```

Multidimensional Arrays

Multi-dimensional arrays are such arrays which stores an another array at each index instead of single element. In other words, we can define multi-dimensional arrays as array of arrays. As the name suggests, every element in this array can be an array and they can also hold other sub-arrays within. Arrays or sub-arrays in multidimensional arrays can be accessed using multiple dimensions.
Example:

```
<?php
// Defining a multidimensional array
$favorites = array(
    array(
        "name" => "Dave Punk",
        "mob" => "5689741523",
        "email" => "davepunk@gmail.com",
    ),
    array(
        "name" => "Monty Smith",
        "mob" => "2584369721",
        "email" => "montysmith@gmail.com",
    ),
    array(
        "name" => "John Flinch",
        "mob" => "9875147536",
        "email" => "johnflinch@gmail.com",
    )
);
// Accessing elements
echo "Dave Punk email-id is: " . $favorites[0]["email"], "\n";
echo "John Flinch mobile number is: " . $favorites[2]["mob"];
?>
```