INTRODUCTION TO PHP

Ceena Mathews
Associate Professor
Dept of Computer Science
Prajyoti Niketan College
Pudukad

- PHP is an acronym for "PHP: Hypertext Preprocessor"

- PHP is a **widely-used, open source server scripting** language

- used for making **dynamic and interactive** Web pages.

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code

- PHP files have extension “.php"

- Latest release is PHP version 7

Features of PHP

- It **runs on various platforms** (Windows, Linux, Unix, Mac OS X, etc.)
- It is **compatible with almost all servers** used today (Apache, IIS, etc.)
- It supports a **wide range of databases**
- It is **free.**
- It is **easy to learn and runs efficiently** on the server side
- It can **generate dynamic page content**
- It can **create, open, read, write, delete, and close files** on the server
- It can collect form data
- It can **send and receive cookies**
- It can **add, delete, modify data in your database**
- It can be used to **control user-access**
- It can **encrypt data**

New Features of PHP 7

- It is much faster than the previous popular stable release (PHP 5.6)
- It has improved Error Handling
- It supports stricter Type Declarations for function arguments
- It supports new operators (like the spaceship operator: <=>)

Basic PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>:

```
<?php
// PHP code goes here
?>
```

- A PHP file normally contains HTML tags, and some PHP scripting code.

```
<?php
echo “Hello”;
?>
```

Case Sensitivity

- keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.
- However, all variable names are case-sensitive

Comments

- Single line comment
  - //
  - #
- Multiline comment

  /*

  ......

  */

Variables

- PHP has **no command for declaring** a variable. It is **created the moment you first assign a value to it.**

Rules for PHP variables:

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

- There are two basic ways to get output

    1. **echo**
    2. **print**

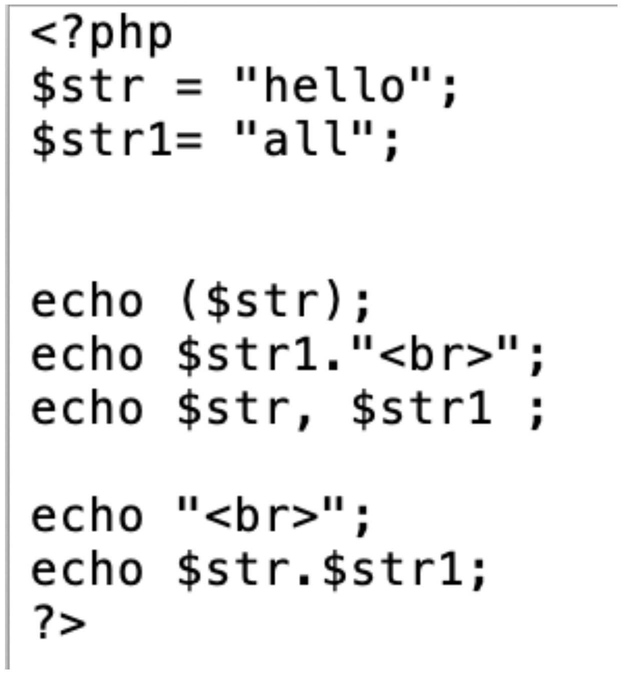
    echo and print are more or less the same.

    They are both used to output data to the screen.

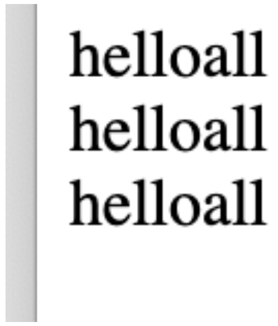
    Differences

    1. **echo has no return value** while **print has a return value of 1** so it can be used in expressions.
    2. **echo can take multiple parameters while print can take one argument.**
    3. **echo is marginally faster than print.**

- echo statement can be used with or without parentheses:
- echo or echo()

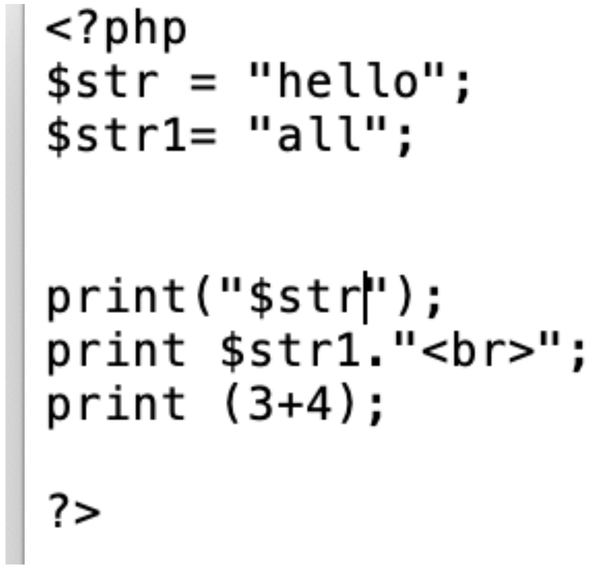


```php
<?php
$str = "hello";
$str1= "all";


echo ($str);
echo $str1."<br>";
echo $str, $str1 ;

echo "<br>";
echo $str.$str1;
?>
```

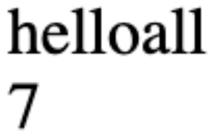


```
helloall
helloall
helloall
```

- print statement can be used with or without parentheses:
- print or print().



```php
<?php
$str = "hello";
$str1= "all";


print("$str");
print $str1."<br>";
print (3+4);

?>
```



```
helloall
7
```

Data types & Operators

Ceena Mathews, PNC

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

1. String

* A string is a collection of characters enclosed within single or double quotes

eg: "Hello", 'Hello'

2. Integer

* An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

eg :$a=124 (decimal number); $b=0x1A(hexadecimal number receded by 0x)

$c=0123(Octal number preceded by 0)

$d=0b1111(Binary number preceded by 0b)

Float

* A floating point number is a number with a decimal point or a number in exponential form.

eg : $w=123.45, $a=7.64E+5, $b=5.56E-5

Boolean

* It represents two values: TRUE or FALSE.

* used in conditional testing

eg: $t=True

$t=False

PHP Array

* An array stores multiple values in one single variable.

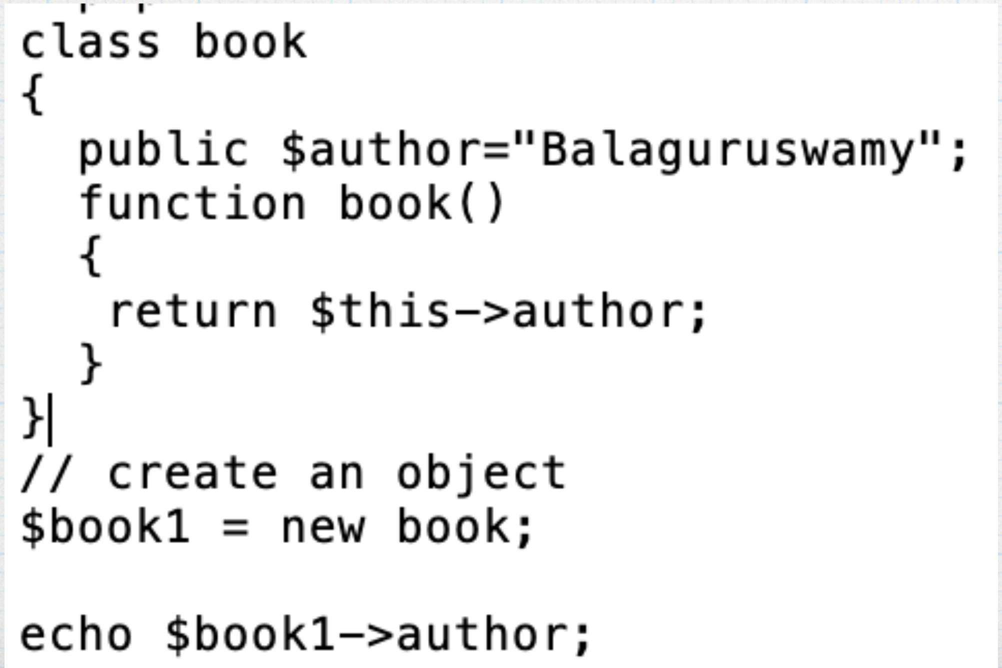
eg :$fruits = array("apple","mango","banana");

Object

* An object is a data type which stores data and information on how to process that data.

* an object must be explicitly declared.

eg:



```
class book
{
  public $author="Balaguruswamy";
  function book()
  {
   return $this->author;
  }
}
// create an object
$book1 = new book;

echo $book1->author;
```

NULL

- special data type which can have only one value: NULL.
- special NULL value is used to represent empty variables in PHP.
- A variable of type NULL is a variable without any data.
- eg. `$a=NULL`

Resource

* A resource is a special variable, holding a reference to an external resource.

* Resource variables typically hold special handlers to opened files and database connections.

eg:

```
// Open a file for reading

$f= fopen("note.txt", "r");
```

Operators

* PHP divides the operators in the following groups:
    1. Arithmetic operators
    2. Assignment operators
    3. Comparison operators
    4. Increment/Decrement operators
    5. Logical operators
    6. String operators
    7. Array operators
    8. Conditional assignment operato

Arithmetic operators

Operator	Name	Example
+	Addition	$a + $b
-	Subtraction	$a - $b
*	Multiplication	$a* $b
/	Division	$a/ $b
%	Modulus	$a % $b(Remainder of $a divided by $b)
**	Exponentiation	$a ** $b ($a raise to the $bth power)
		eg. 3 ** 2 Ans:9 (3²)

Ceena Mathews, PNC

Assignment Operator

- They are used with numeric values to write a value to a variable.
- The basic assignment operator is "=".
- It means that the left operand gets set to the value of the assignment expression on the right.

Assignment/shorthand operators

- =, +=, -=, *=, /=, %=

```
$x = $y
$x += $y    same as $x = $x + $y
$x -= $y    same as $x = $x - $y
$x *= $y    same as $x = $x * $y
$x /= $y    same as $x = $x / $y
$x %= $y    same as $x = $x % $y
```

Comparison Operators

* The PHP comparison operators are used to compare two values (number or string):

* Operator

==	Equal	$x ==$y	Returns true if $x is equal to $y(they need not be of same datatype)
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>			
<			
>=			
<=			
<=>	Spaceship	$x <=> $y	Returns -1 if $x<$y , returns 0 if they are equal, returns 1 if $x>$y

Increment / Decrement Operators

* Operator

++$x	Pre-increment	:Increments $x by one, then returns $x
$x++	Post-increment	:Returns $x, then increments $x by one
--$x	Pre-decrement	:Decrements $x by one, then returns $x
$x--	Post-decrement	:Returns $x, then decrements $x by one

Logical Operators

- They are used to combine conditional statements.
- Operators

and eg:$x and $y (True if both $x and $y are true)

or eg:$x or $y (True if either $x or $y is true)

xor eg:$x xor $y (True if either $x or $y is true, but not both)

&& (And)eg:$x && $y (True if both $x and $y are true)

|| (Or) eg:$x || $y (True if either $x or $y is true)

! (Not)eg: !$x (True if $x is false)

String Operators

* PHP has two operators that are specially designed for strings.
* Operator
  - **.** **Concatenation eg:$txt1 . $txt2](Concatenation of $txt1 and $txt2)**
  - **.=** **Concatenation assignment**

```
eg. $txt1="Good";
        $txt2=" Morning";
         echo $txt1.$txt2; //displays Good Morning
```

Array Operators

* They are used to compare arrays.
* Operator

  + (Union)

  eg:$x + $y (Union of $x and $y)

  == (Equality)

  eg:$x == $y (Returns true if $x and $y have the same key/ value pairs)

  ===(Identity)

  eg:$x === $y (Returns true if $x and $y have the same key/value pairs

  in the same order and of the same types)

  != (Inequality)

  eg:$x != $y(Returns true if $x is not equal to $y)

  <> (Inequality)

  eg:$x <> $y(Returns true if $x is not equal to $y)

  !==(Non-identity) eg:$x !== $y (Returns true if $x is not equal to $yand not in the same order and of the same type)

Conditional Assignment Operators

- They are used to set a value depending on conditions

- Operator

  ?: –Ternary

  eg:$x = expr1 ? expr2 : expr3   Returns the value of $x.

  The value of $x is expr2 if expr1 = TRUE.

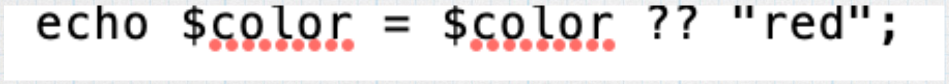
  The value of $x is expr3 if expr1 = FALSE

  ?? –Null coalescing

  eg:$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.

eg 

```
echo $color = $color ?? "red";
```

SUPER GLOBAL VARIABLES

Ceena Mathews @PNC

PHP Global Variables - Superglobals

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

1. **$_SERVER**
2. **$_REQUEST**
3. **$_POST**
4. **$_GET**
5. **$_COOKIE**
6. **$_SESSION**

$_SERVER

- PHP super global variable which holds information about headers, paths, and script locations.
  - $_SERVER['PHP_SELF'] :

    Returns the filename of the currently executing script
  - $_SERVER['GATEWAY_INTERFACE']:

    Returns the version of the Common Gateway Interface (CGI) the server is using
  - $_SERVER['SERVER_ADDR']:

    Returns the IP address of the host server
  - $_SERVER['SERVER_NAME'] :

    Returns the name of the host server
  - $_SERVER['HTTP_HOST']

    Returns the Host header from the current request
  - $_SERVER['SCRIPT_NAME']

    Returns the path of the current script

Example

```
<html>
<body>

<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";

echo $_SERVER['GATEWAY_INTERFACE'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>

</body>
</html>
```

Output

```
/ceenaphp/super.php
localhost
localhost:8080
CGI/1.1
/ceenaphp/super.php
```

$_REQUEST

- a PHP super global variable which is used to collect data after submitting an HTML form.
- used with both get and post method
- Example :

```
Name: <input type="text" name=t1>

$name = $_REQUEST['t1'];
```

Conditional Statements in PHP

Ceena Mathews, PNC

Conditional Statements

They are used to perform actions based on conditions.

- `if` **statement**
- `if...else` **statement**
- `if...elseif...else` **statement**
- **Nested if**
- `switch` **statement**

if Statement

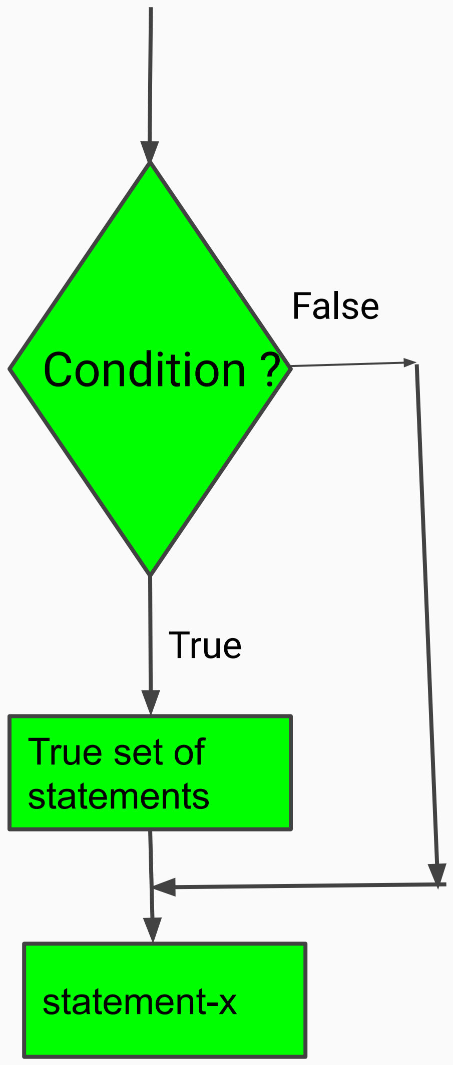
if Statement

executes code if condition is true.

Syntax

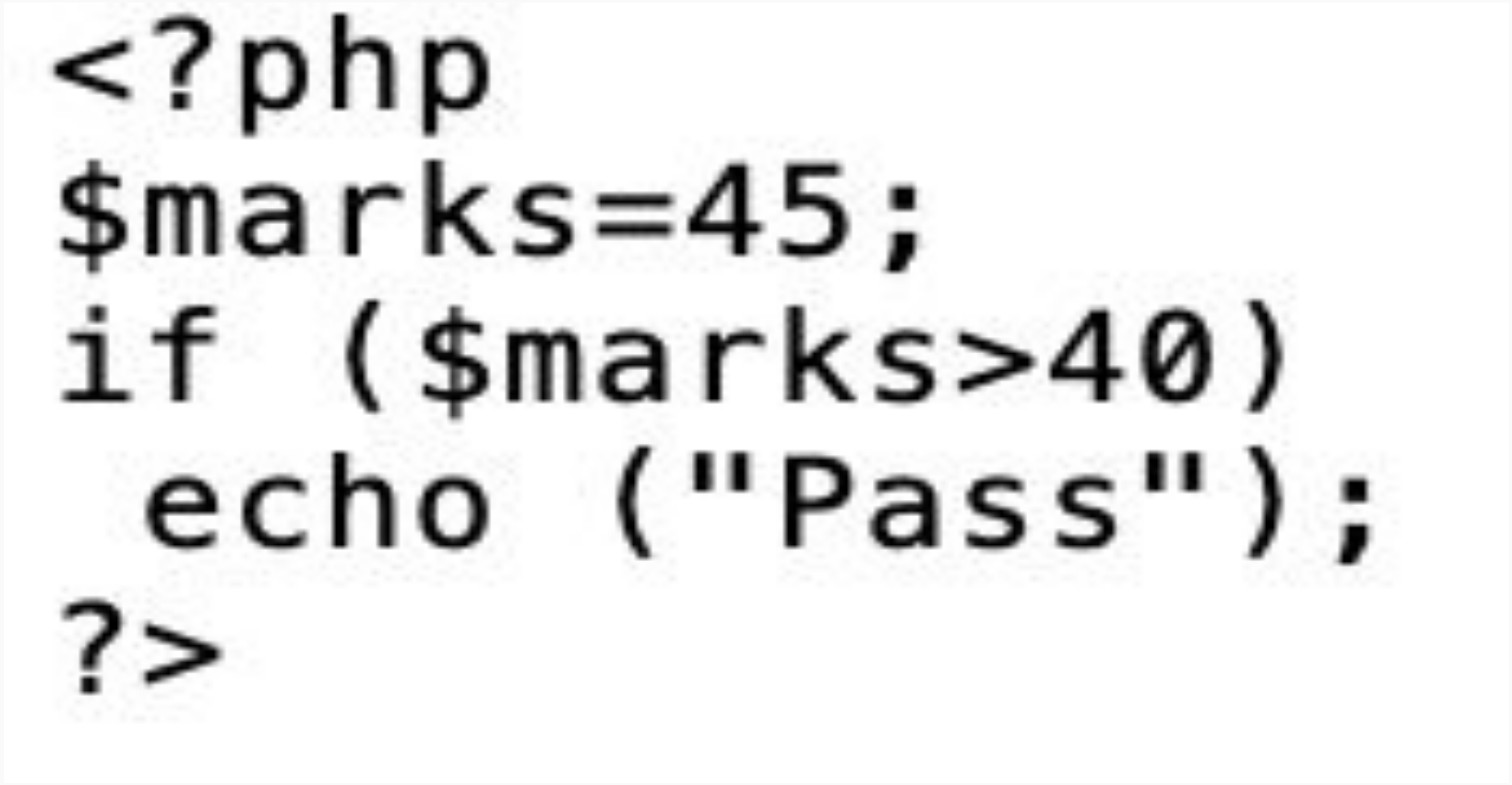
```
if (condition)
{
   //code to be executed if
condition is true;
}
```

statement -x



Ceena Mathews, PNC

if Statement - Example



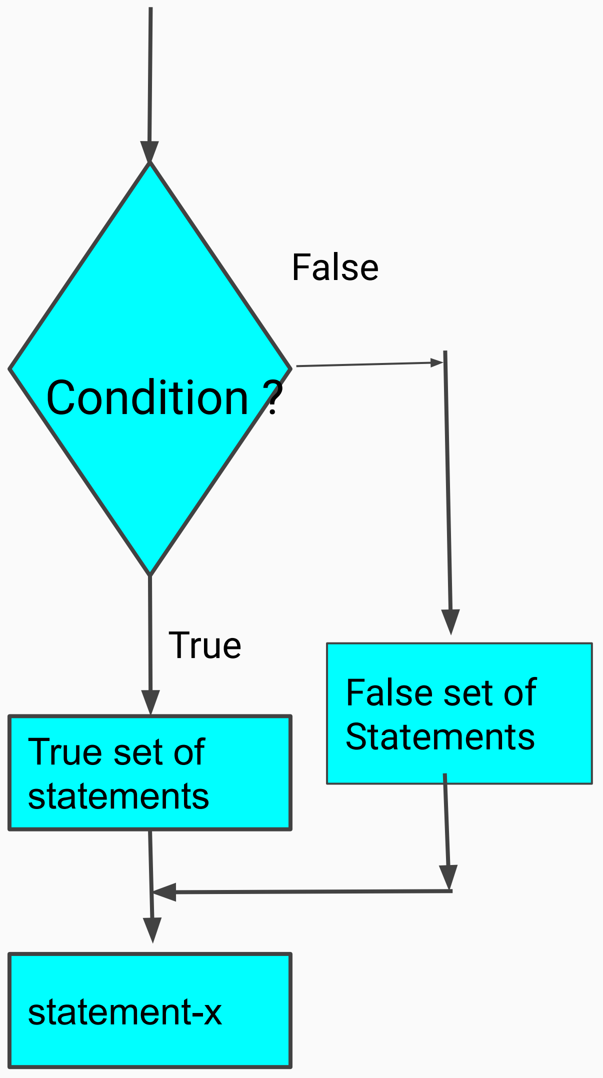
```php
<?php
$marks=45;
if ($marks>40)
 echo ("Pass");
?>
```

if...else Statement

executes code if a condition is true and another code if that condition is false.

Syntax

```
if (condition)
{
   //code to be executed if
condition is true;
}
else
{
   //code to be executed if
//condition is false;
}
statement-x;
```



Ceena Mathews, PNC

if...else Statement- Example

```php
<?php
$marks=45;
if ($marks>40)
 echo ("Pass")
else
 echo("Fail");
?>
```

if...elseif...else Statement

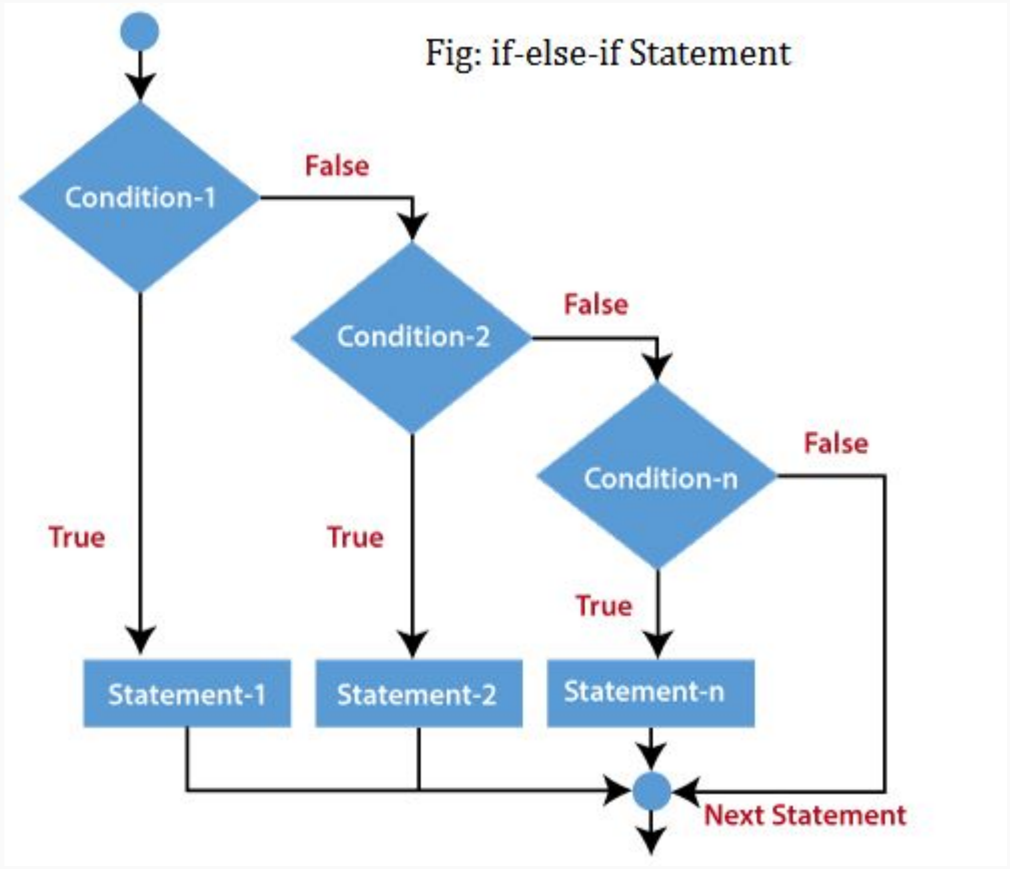
executes different codes for more than two conditions.

Syntax

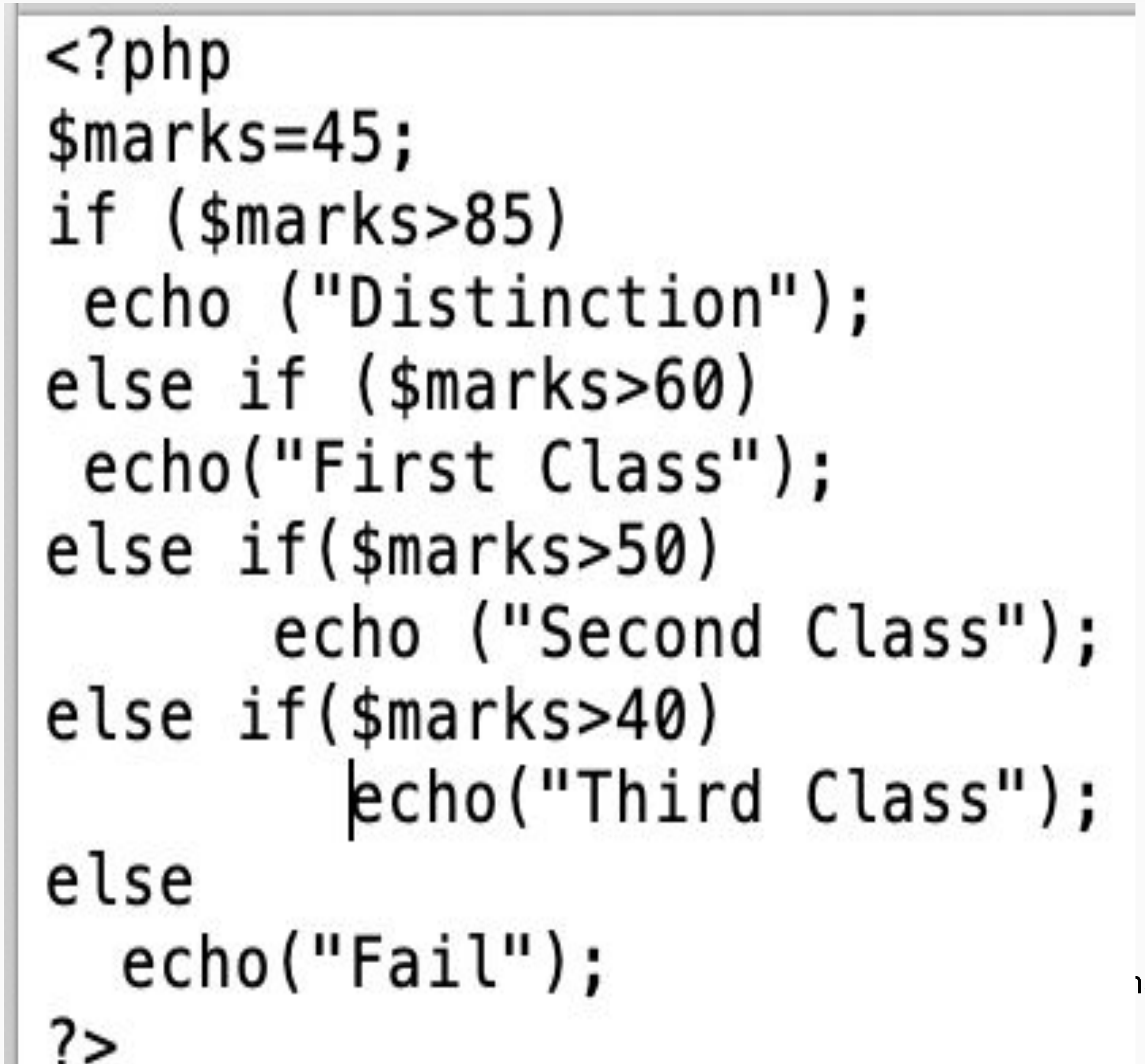
```
 if (condition-1)
 {
   //code to be executed if condition-1 is true;
 }
 elseif (condition-2)
  {
   //code to be executed if condition-1 is false and
   //condition-2 is true;
}
else
{
  //code to be executed if all conditions are false;
}
```

Ceena Mathews, PNC

if...elseif...else Statement



Example - if...elseif...else Statement



```
<?php
$marks=45;
if ($marks>85)
 echo ("Distinction");
else if ($marks>60)
 echo("First Class");
else if($marks>50)
      echo ("Second Class");
else if($marks>40)
        echo("Third Class");
else
  echo("Fail");
?>
```

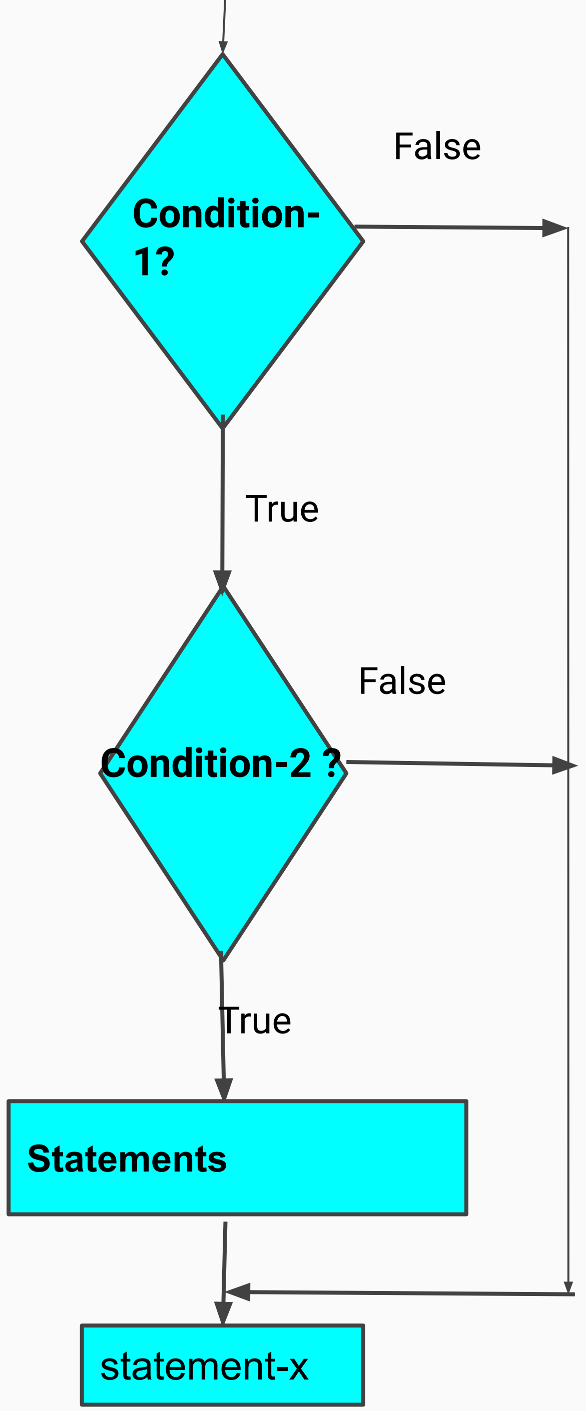
Nested if Statement

- contains the if block inside another if block.
- The inner if statement executes only when specified condition in outer if statement is **true**.

Syntax

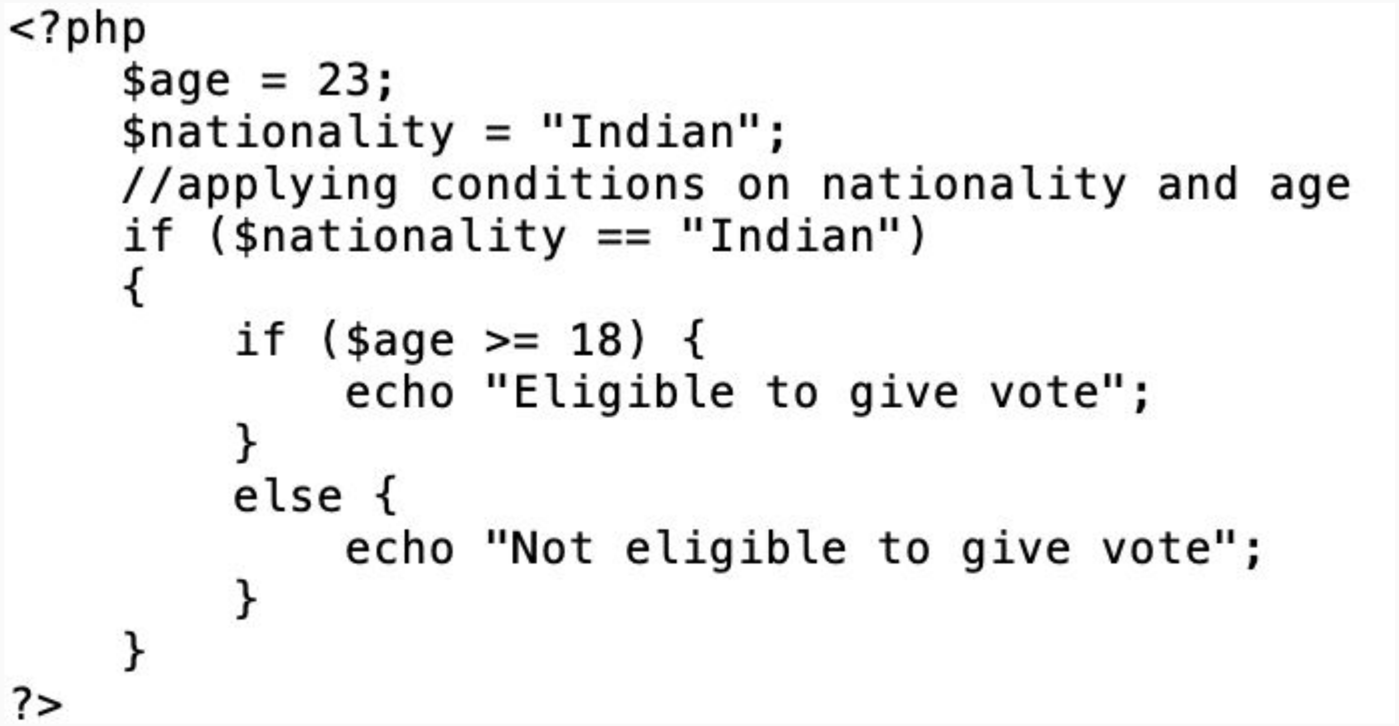
```
if (condition-1)
{
    if (condition-2)
{
    //code to be executed if both conditions is true
}
}
```

Nested if Statement



Ceena Mathews, PNC

Example -Nested if Statement



```php
<?php
    $age = 23;
    $nationality = "Indian";
    //applying conditions on nationality and age
    if ($nationality == "Indian")
    {
        if ($age >= 18) {
            echo "Eligible to give vote";
        }
        else {
            echo "Not eligible to give vote";
        }
    }
?>
```

Switch Statement

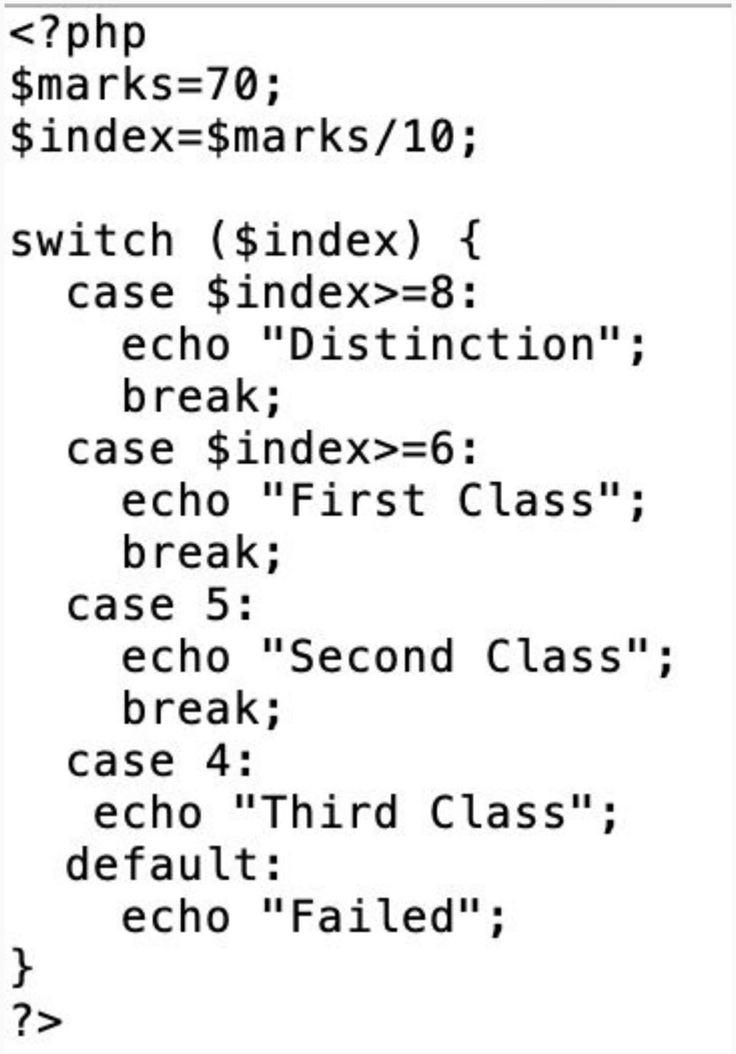
select one of many blocks of code to be executed.

Syntax

```
switch (n) {
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
   break;
    ...
  default:
    code to be executed if n is different from all
labels;
}
```

Ceena Mathews, PNC

Switch Statement



```php
<?php
$marks=70;
$index=$marks/10;

switch ($index) {
  case $index>=8:
    echo "Distinction";
    break;
  case $index>=6:
    echo "First Class";
    break;
  case 5:
    echo "Second Class";
    break;
  case 4:
   echo "Third Class";
  default:
    echo "Failed";
}
?>
```

Loops in PHP

Ceena Mathews, PNC

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`
- `do...while`
- `for`
- `foreach`

While loop

executes a block of code as long as the specified condition is true.

Syntax

```
while (condition)
  {
   //code to be executed;
 }
```

Example

```php
<?php
$x = 1;
$sum = 0;
while($x <= 10)
{
  $sum = $sum + $x;
  $x++;
}
 echo "The number is: $sum <br>";


?>
```

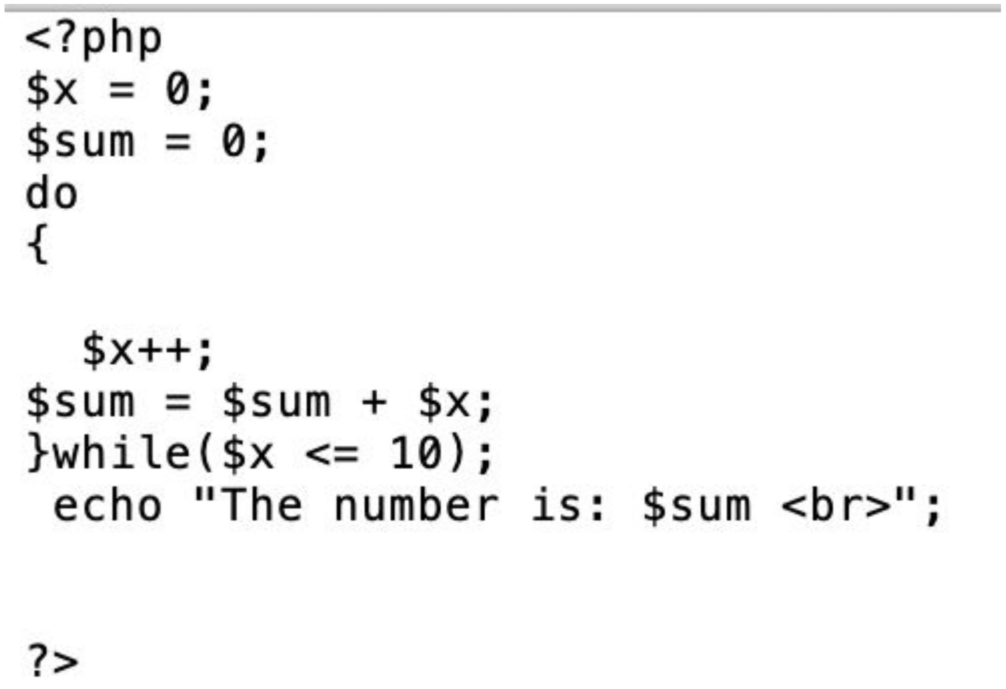
do...while Loop

The `do...while` loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do
{
  //code to be executed;
}
while (condition);
```

Example



```
<?php
$x = 0;
$sum = 0;
do
{

  $x++;
$sum = $sum + $x;
}while($x <= 10);
 echo "The number is: $sum <br>";


?>
```

for Loop

used when you know in advance how many times the script should run.

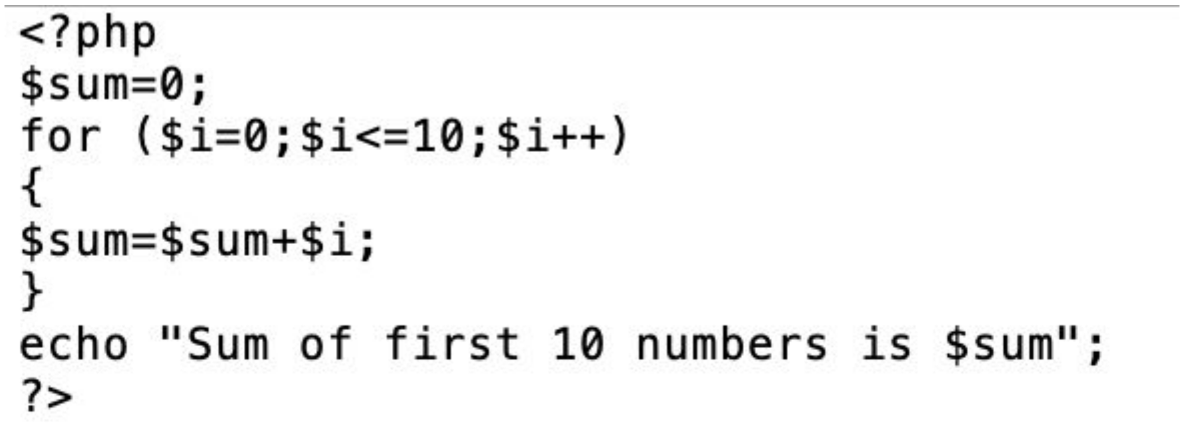
Syntax

```
for (initialization; test condition; increment/decrement
counter)
{
  //code to be executed for each iteration
}
```

Parameters:

- *initialization*: Initialize the loop counter value
- *test condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment/ decrement counter*: Increases/decreases the loop counter value

Example



```php
<?php
$sum=0;
for ($i=0;$i<=10;$i++)
{
$sum=$sum+$i;
}
echo "Sum of first 10 numbers is $sum";
?>
```

`foreach` loop

- works **only on arrays**
- is used to loop **through each key/value pair** in an array.

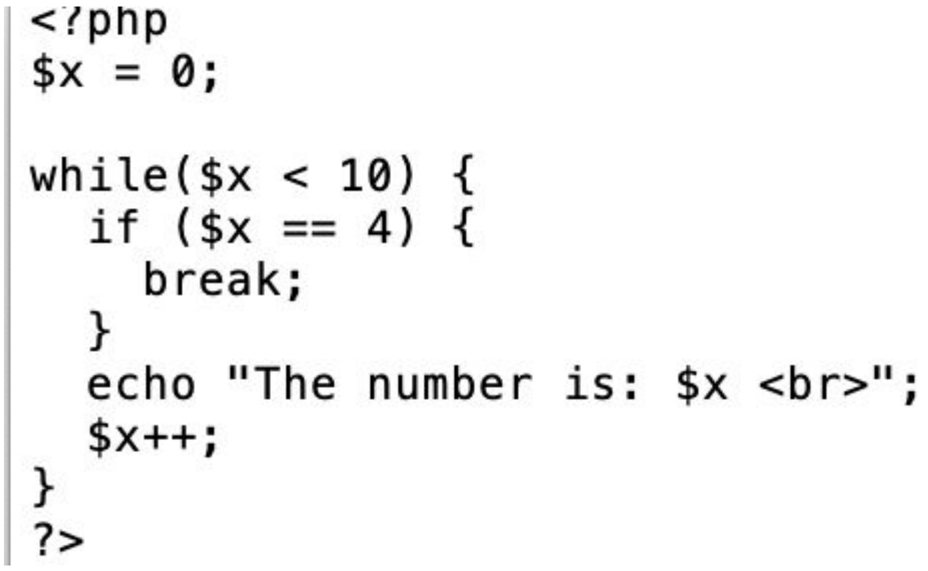
Syntax

```
foreach ($array as $value)
{
  //code to be executed;
}
```

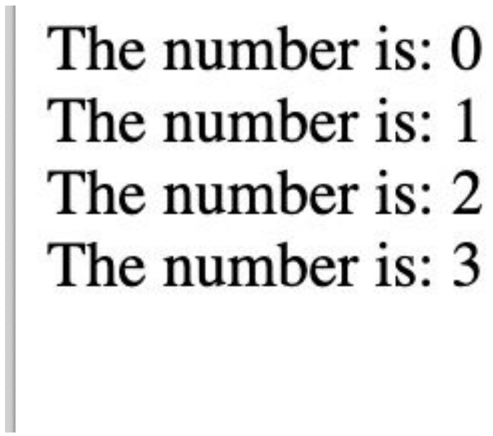
For every loop iteration, the **value of the current array element is assigned to $value** and the **array pointer is moved by one**, until it reaches the last array element.

Break statement

- Used to prematurely exit from a loop
- used to jump out of a loop.
- Used with switch statement also



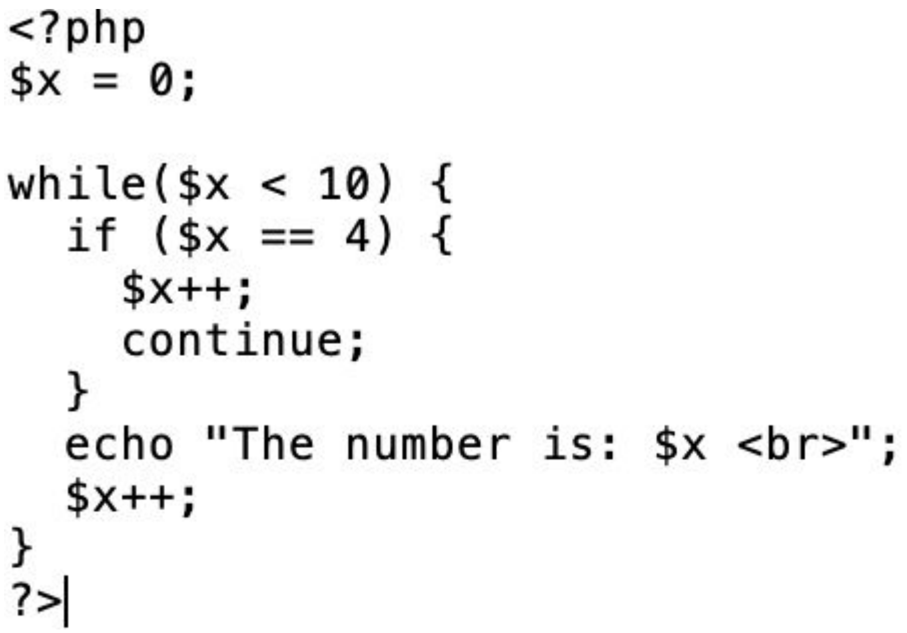
```php
<?php
$x = 0;

while($x < 10) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
  $x++;
}
?>
```



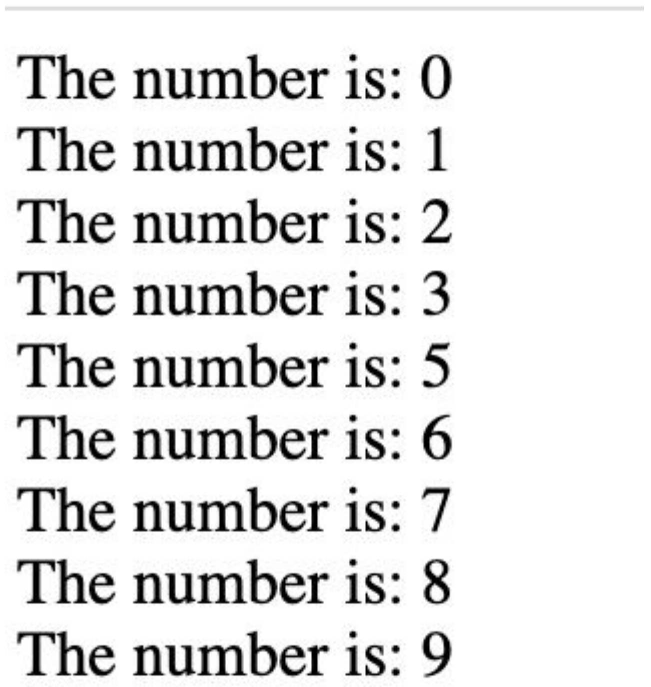
```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
```

Continue statement

Skips the current iteration and continues with the next iteration



```php
<?php
$x = 0;

while($x < 10) {
  if ($x == 4) {
    $x++;
    continue;
  }
  echo "The number is: $x <br>";
  $x++;
}
?>
```



```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
```

String Operations

Strings

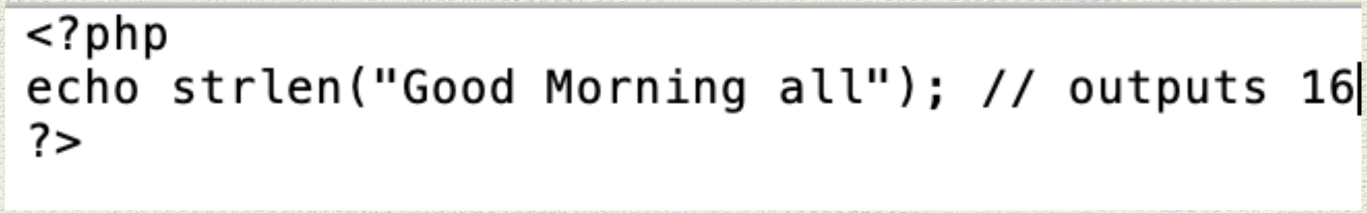
- A string is a sequence of characters enclosed in single or double quotes.

String Functions

1. strlen()
2. strrev()
3. strpos()
4. stripos()
5. strrpos()
6. strcmp()
7. strncmp()
8. strstr()
9. stristr()
10. strtolower()
11. strtoupper()
12. lcfirst()
13. ucfirst()
14. ucwords()
15. substr()
16. substr_replace()

1. strlen(string)

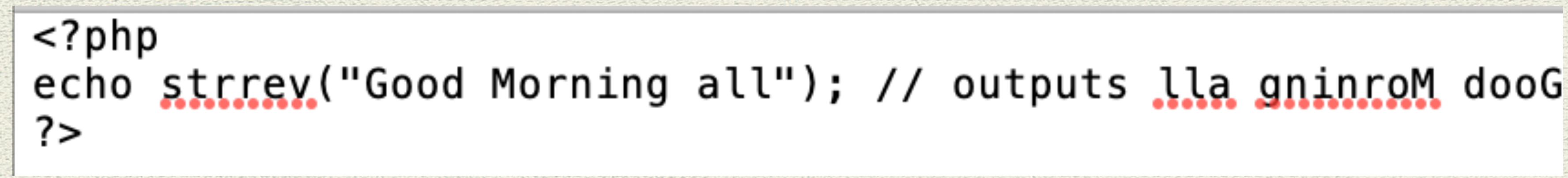
- function returns the length of a string.



```php
<?php
echo strlen("Good Morning all"); // outputs 16
?>
```

2. strrev(string)

- function reverses a string.



```php
<?php
echo strrev("Good Morning all"); // outputs lla gninroM dooG
?>
```

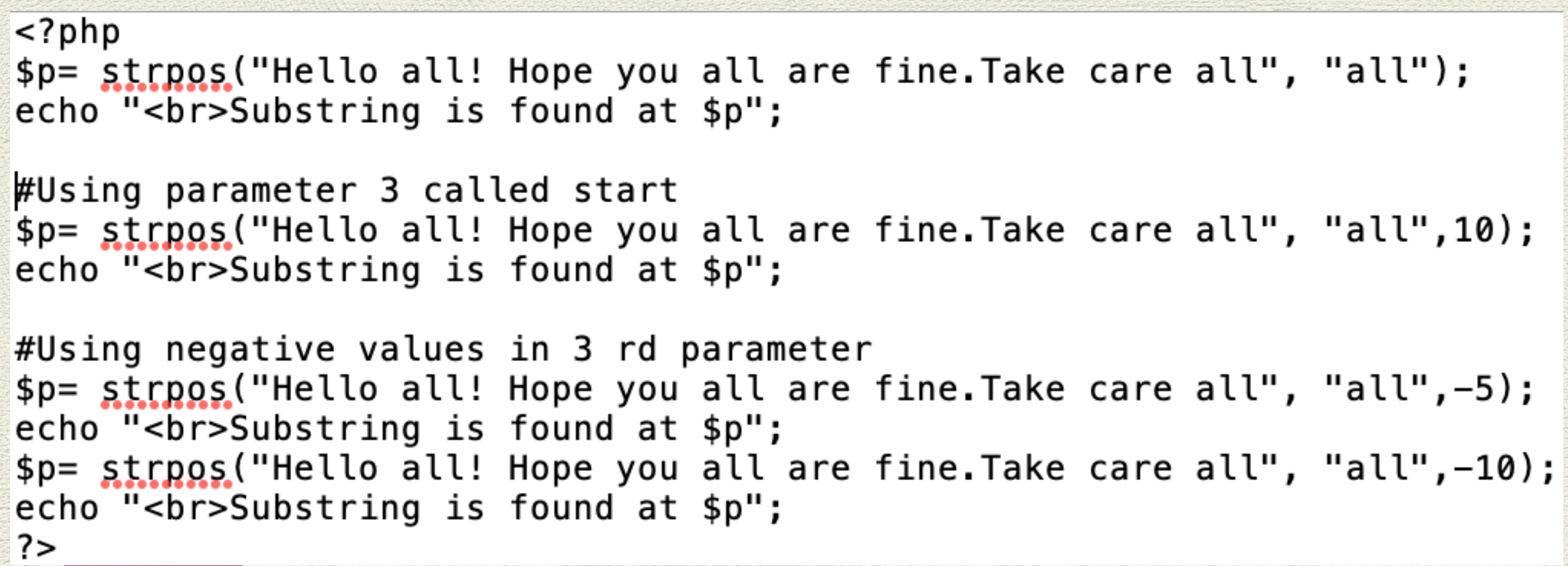
3. strpos(string,find,start)

- function searches for a specific text within a string.
- If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.
- case sensitive search

Syntax : strpos(string,find[,start])

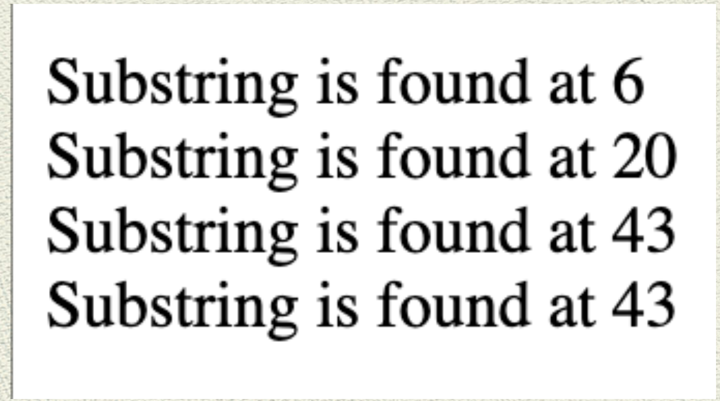
- **string: Specifies the string to search**
- **find : Specifies the string to find**
- **start: Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string**

Example



```
<?php
$p= strpos("Hello all! Hope you all are fine.Take care all", "all");
echo "<br>Substring is found at $p";

#Using parameter 3 called start
$p= strpos("Hello all! Hope you all are fine.Take care all", "all",10);
echo "<br>Substring is found at $p";

#Using negative values in 3 rd parameter
$p= strpos("Hello all! Hope you all are fine.Take care all", "all",-5);
echo "<br>Substring is found at $p";
$p= strpos("Hello all! Hope you all are fine.Take care all", "all",-10);
echo "<br>Substring is found at $p";
?>
```

Output



```
Substring is found at 6
Substring is found at 20
Substring is found at 43
Substring is found at 43
```

4.stripos()

- function finds the position of the first occurrence of a string inside another string.
- if not found, it returns false
- case-insensitive.
- This function is binary-safe.

Syntax

stripos(string,find,start)

  - string   :Specifies the string to search
  - find: Specifies the string to find
  - start: Specifies where to begin the search

5.strrpos()

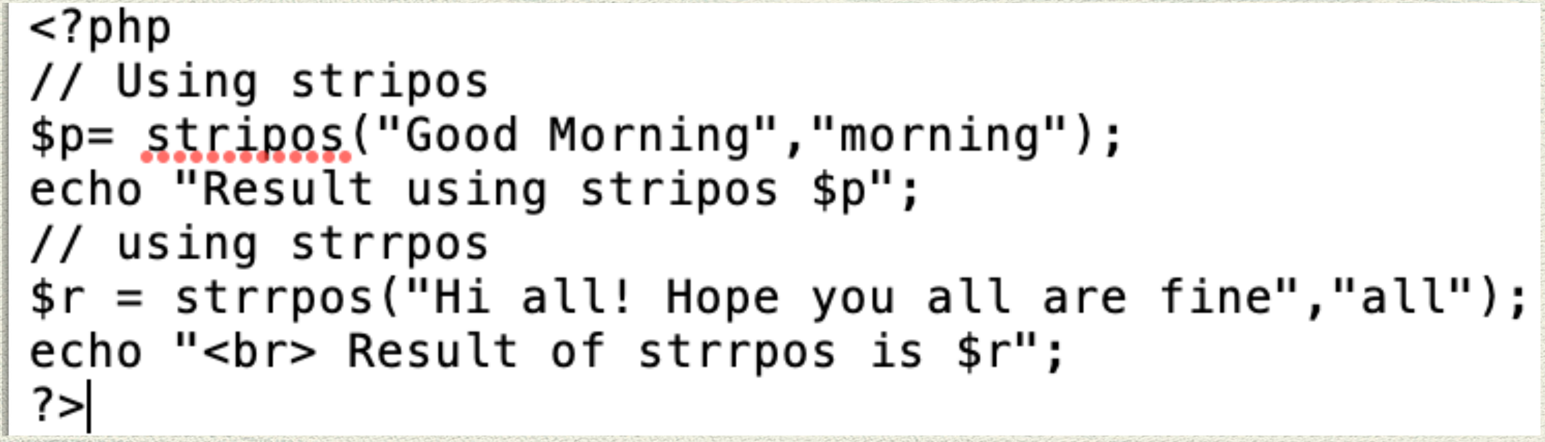
- function finds the position of the last occurrence of a string inside another string.
- if not found, it returns false
- case-insensitive.
- This function is binary-safe.

Syntax

strrpos(string,find,start)

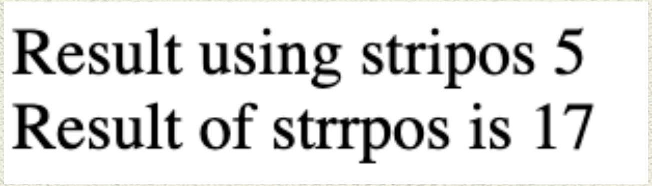
- string :Specifies the string to search
- find: Specifies the string to find
- start: Specifies where to begin the search

Example



```php
<?php
// Using stripos
$p= stripos("Good Morning","morning");
echo "Result using stripos $p";
// using strrpos
$r = strrpos("Hi all! Hope you all are fine","all");
echo "<br> Result of strrpos is $r";
?>
```

Output



```
Result using stripos 5
Result of strrpos is 17
```

6. strcmp() function

- compares two strings.
- It is binary-safe and case-sensitive.
- This function is similar to the strncmp() function, with the difference that you can specify the number of characters from each string to be used in the comparison with strncmp().

This function returns:
0 - if the two strings are equal
<0 - if string1 is less than string2
>0 - if string1 is greater than string2

Syntax

strcmp(string1,string2)

- **string1 :Specifies the first string to compare**
- **string2 :Specifies the second string to compare**

7.strncmp() function

- compares two strings.
- It is binary-safe and case-sensitive.

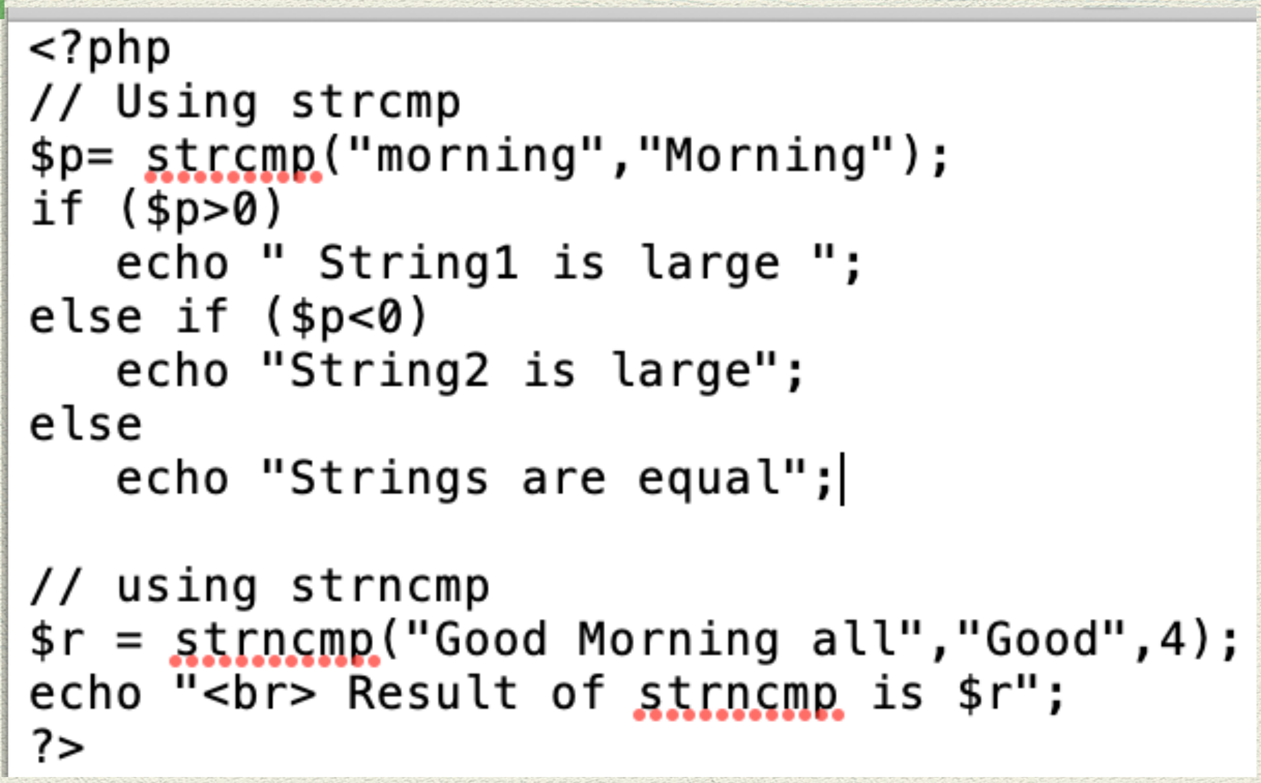
Syntax

strncmp(string1,string2,length)

This function returns:
0 - if the two strings are equal
<0 - if string1 is less than string2
>0 - if string1 is greater than string2

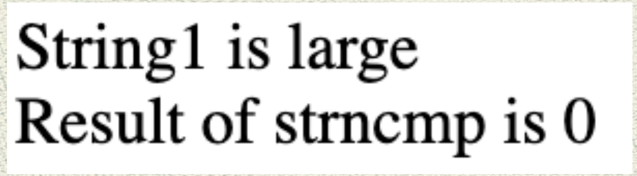
- string1:Specifies the first string to compare
- string2: Specifies the second string to compare
- length: Specify the number of characters from each string to be used in the comparison

Example



```
<?php
// Using strcmp
$p= strcmp("morning","Morning");
if ($p>0)
   echo " String1 is large ";
else if ($p<0)
   echo "String2 is large";
else
   echo "Strings are equal";

// using strncmp
$r = strncmp("Good Morning all","Good",4);
echo "<br> Result of strncmp is $r";
?>
```

Output



```
String1 is large
Result of strncmp is 0
```

8. strstr() function

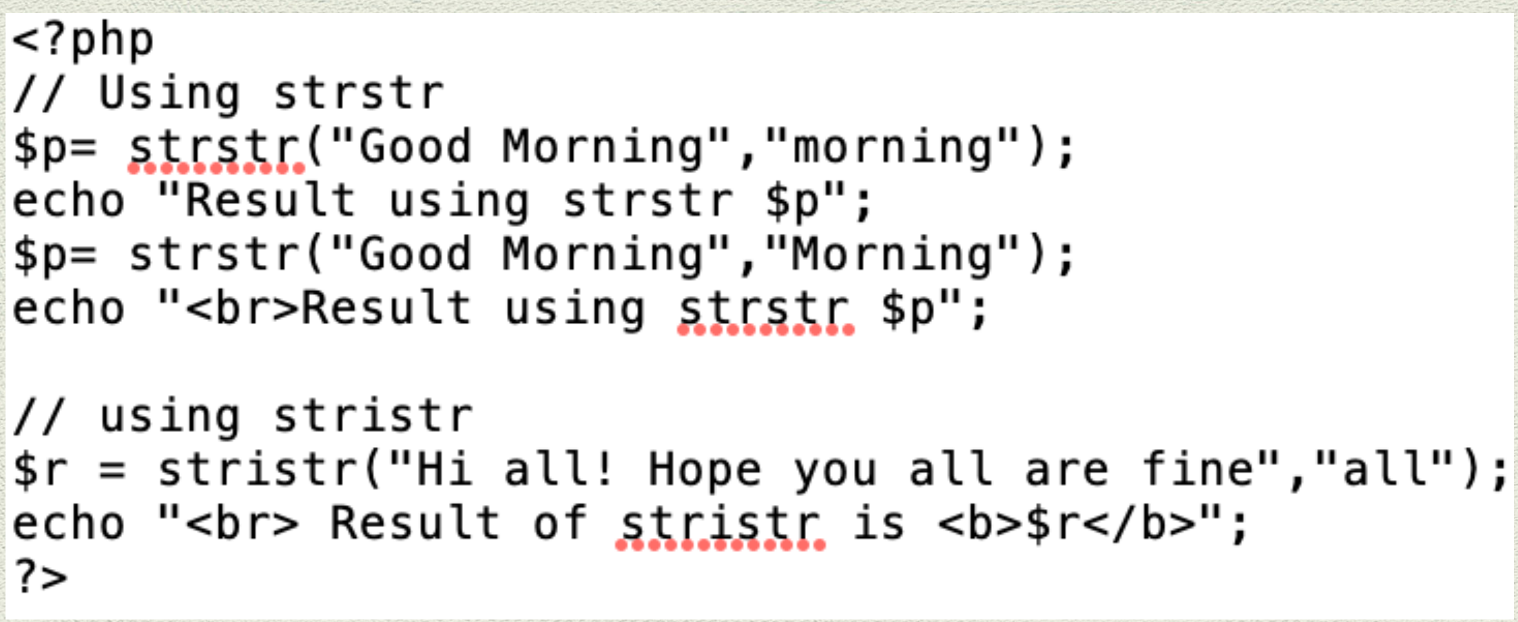
- searches for the first occurrence of a string inside another string.
- This function is binary-safe.
- This function is case-sensitive. For a case-insensitive search, use stristr() function.

Syntax

strstr(string,search[,before_search])

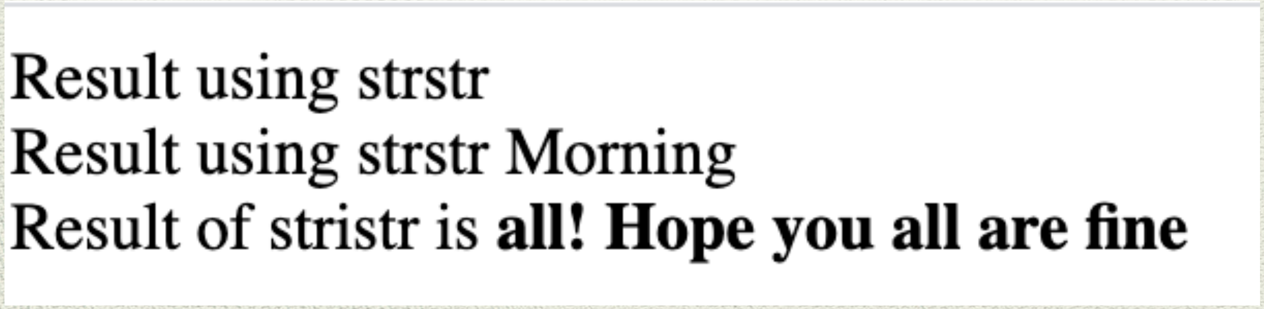
- string: Specifies the string to search
- search: Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number
- before_search:Optional. Possible values are true or False(default)
    - If "true", it returns the part of the string before the first occurrence of the search parameter.

Example



```php
<?php
// Using strstr
$p= strstr("Good Morning","morning");
echo "Result using strstr $p";
$p= strstr("Good Morning","Morning");
echo "<br>Result using strstr $p";

// using stristr
$r = stristr("Hi all! Hope you all are fine","all");
echo "<br> Result of stristr is <b>$r</b>";
?>
```

Output



```
Result using strstr
Result using strstr Morning
Result of stristr is all! Hope you all are fine
```

9. strtolower(string) - converts a string to lowercase

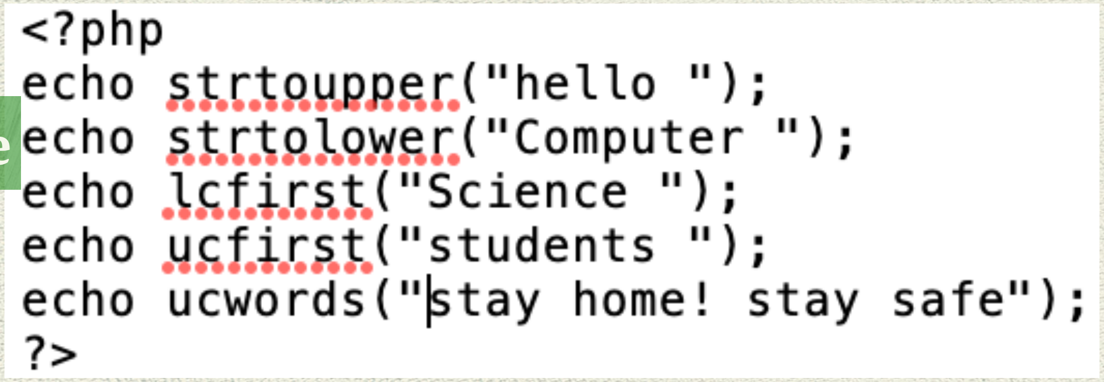
10. strtoupper(string) - converts a string to uppercase

11.lcfirst(string) - converts the first character of a string to lowercase

12.ucfirst(string) - converts the first character of a string to uppercase

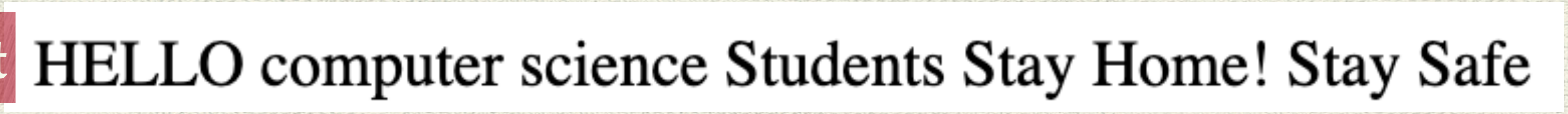
13. ucwords(string) - converts the first character of each word in a string to uppercase

Example



```
<?php
echo strtoupper("hello ");
echo strtolower("Computer ");
echo lcfirst("Science ");
echo ucfirst("students ");
echo ucwords("stay home! stay safe");
?>
```

Output



```
HELLO computer science Students Stay Home! Stay Safe
```

14. substr() function- returns a part of a string.

Syntax

substr(string,start[,length])

- string: Specifies the string to return a part of
- start: Specifies where to start in the string

A positive number - Start at a specified position in the string
A negative number - Start at a specified position from the end of the string
0 - Start at the first character in string

- length: Optional. Specifies the length of the returned string.
    - Default is to the end of the string.

A positive number - The length to be returned from the start parameter
Negative number - The length to be returned from the end of the string
If the length parameter is 0, NULL, or FALSE - it return an empty string

Example

```php
<?php

//without length parameter
echo substr("stay home! stay safe",11);
echo "<br>";
//with length parameter
echo substr("stay home! stay safe",11,4);

?>
```

Output

```
stay safe
stay
```

15. substr_replace()

- replaces a part of a string with another string.

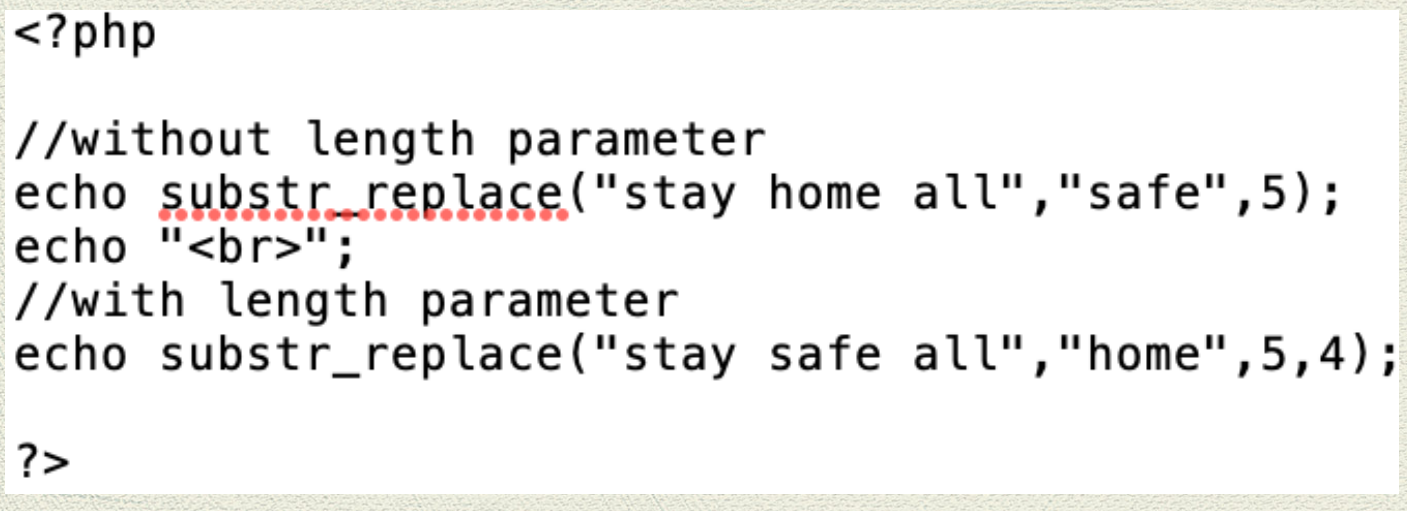
Syntax

substr_replace(string,replacement,start[,length])

- string: Specifies the string to check
- replacement: Specifies the string to insert
- start : Specifies where to start replacing in the string
  - **A positive number - Start replacing at the specified position in the string**
  - **Negative number - Start replacing at the specified position from the end of the string**
  - **0 - Start replacing at the first character in the string**
- length:Optional. Specifies how many characters should be replaced. Default is the same length as the string.

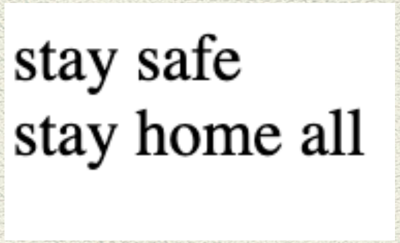
A positive number - The length of string to be replaced
A negative number - How many characters should be left at end of string after replacing
0 - Insert instead of replace

Example



```
<?php

//without length parameter
echo substr_replace("stay home all","safe",5);
echo "<br>";
//with length parameter
echo substr_replace("stay safe all","home",5,4);

?>
```

Output



```
stay safe
stay home all
```

User Defined Function PHP

Ceena Mathews
Associate Professor, PNC

- Function is a block of code to perform a particular task.

- User defined functions are those defined by user/programmer

Advantages

- **Reuseable Code**: A defined function can be reused any number of times in other programs as well.
- **Less Code Repetition**: can be used repeatedly in our program
- **Easy to Understand**: makes the code more readable and easy to understand.
- **Debugging is made easy**

Syntax

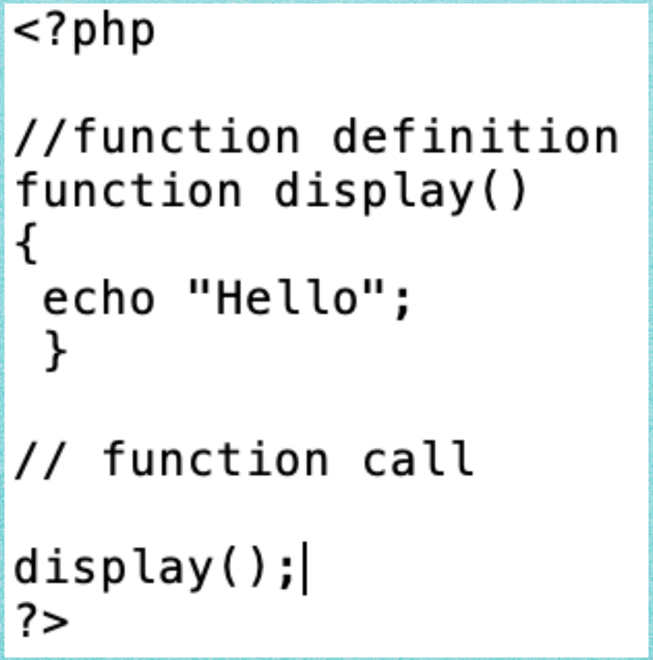
```
function functionName([[datatype] argument1, [datatype]
argument2,....])
 {
  //code to be executed;
}
```

- function name **must start with a letter or an underscore.**
- Function names **are NOT case-sensitive**.
- **Arguments are** variables which pass information to functions.
- Arguments are specified **after the function name, inside the parentheses**
- Multiple arguments can be given, just **separate them with a comma**

- In order to access the user defined function, call the function as given below
- **Syntax**

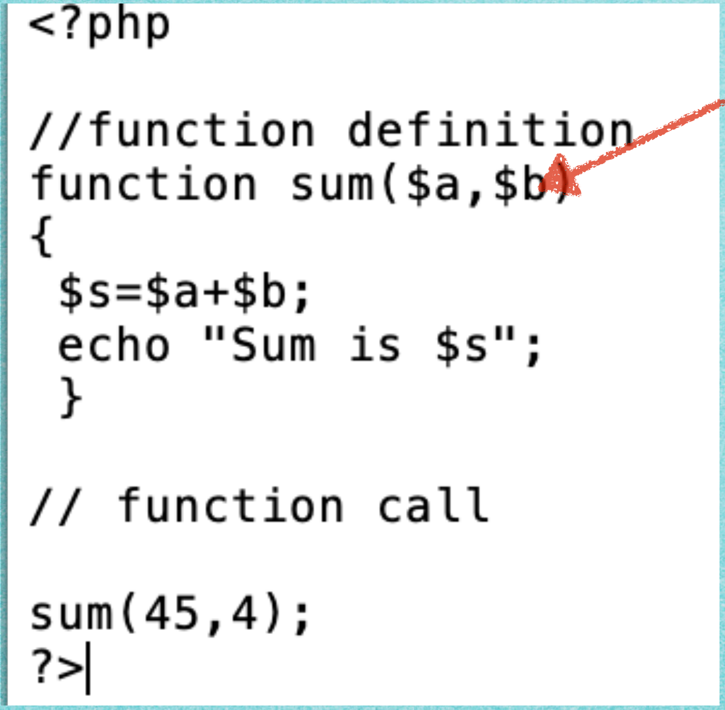
  functionname([argumentlist]);

  Example for function with no arguments



```
<?php

//function definition
function display()
{
 echo "Hello";
 }

// function call

display();
?>
```

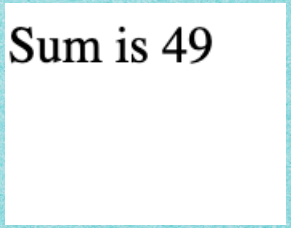
Example for function with arguments



```
<?php

//function definition
function sum($a,$b)
{
 $s=$a+$b;
 echo "Sum is $s";
 }

// function call

sum(45,4);
?>
```

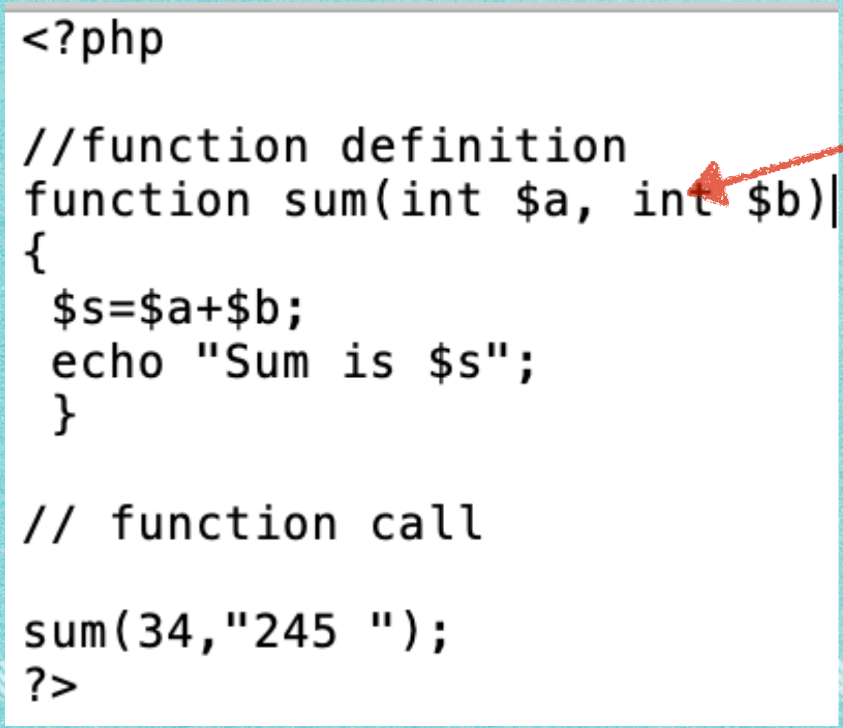
No datatype

Output



```
Sum is 49
```

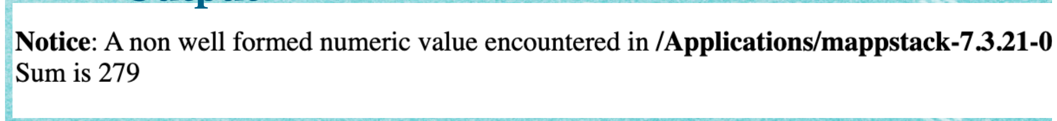
- Type declarations
  - **In PHP 7** type declarations were added.
  - This gives an option to **specify the expected data type when declaring a function**
  - by adding the strict declaration, it will **throw a "Fatal Error" if the data type mismatches.**



```
<?php

//function definition
function sum(int $a, int $b)
{
 $s=$a+$b;
 echo "Sum is $s";
 }

// function call

sum(34,"245 ");
?>
```

Not strict

Output

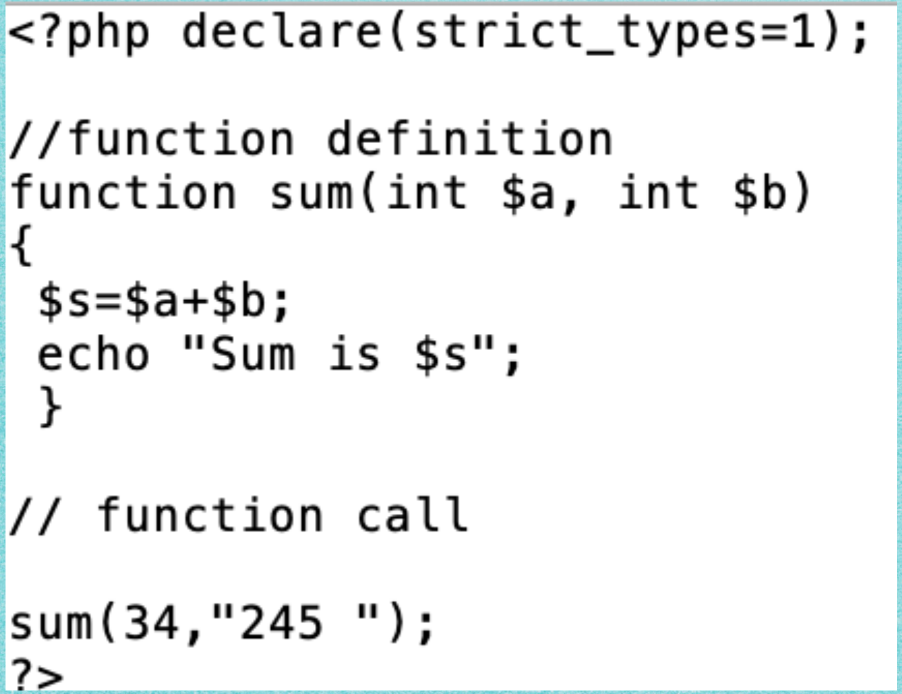


```
Notice: A non well formed numeric value encountered in /Applications/mappstack-7.3.21-0
Sum is 279
```

- To specify strict declaration, use

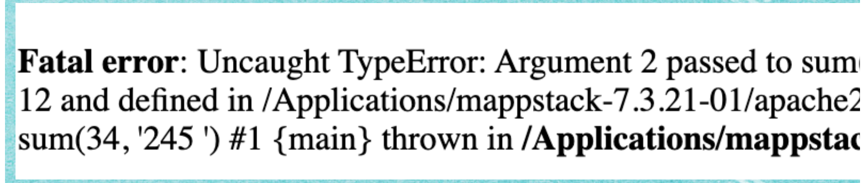
  declare(strict_types=1);

- This must be on the **very first line** of the PHP file.



```php
<?php declare(strict_types=1);

//function definition
function sum(int $a, int $b)
{
 $s=$a+$b;
 echo "Sum is $s";
 }

// function call

sum(34,"245 ");
?>
```

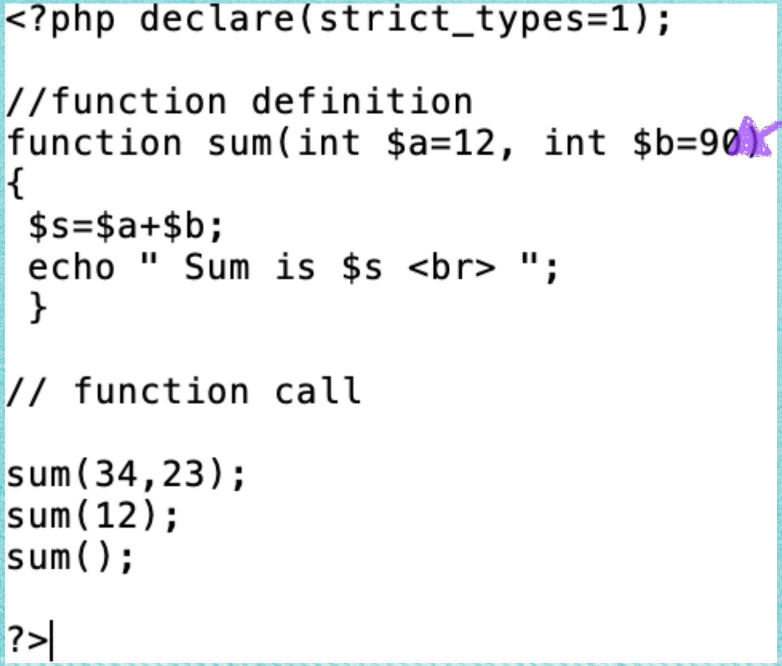
Generates type error



```
Fatal error: Uncaught TypeError: Argument 2 passed to sum
12 and defined in /Applications/mappstack-7.3.21-01/apache2
sum(34, '245 ') #1 {main} thrown in /Applications/mappstac
```

Default Argument

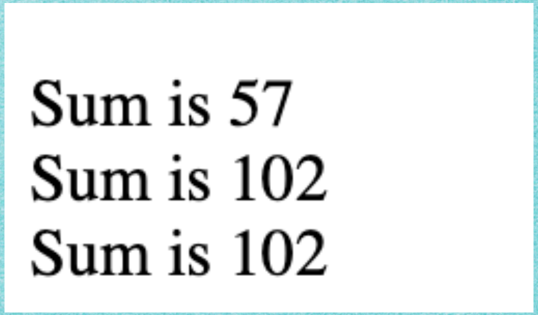
- PHP allows us to set default argument values for function parameters.
- If we do not pass any argument for a parameter with default value then it will use the default set value for this parameter in the function call.



```
<?php declare(strict_types=1);

//function definition
function sum(int $a=12, int $b=90)
{
 $s=$a+$b;
 echo " Sum is $s <br> ";
 }

// function call

sum(34,23);
sum(12);
sum();

?>
```

default value given for argument

Output



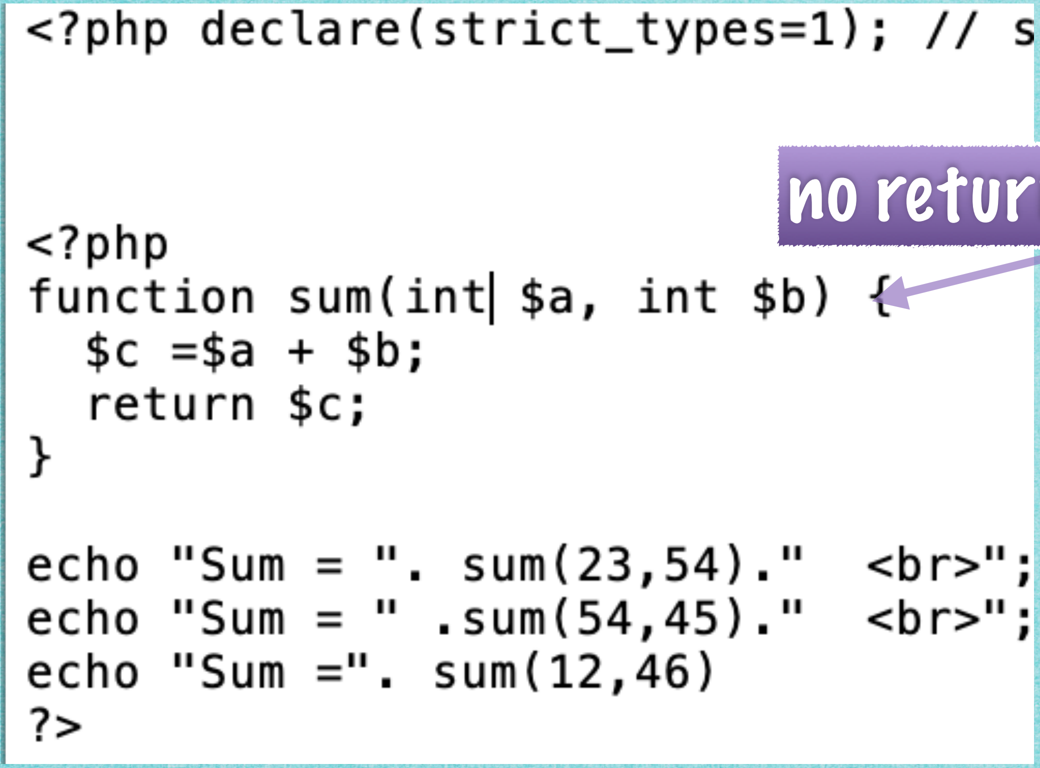
```
Sum is 57
Sum is 102
Sum is 102
```

Ceena Mathews, PNC

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
- **specifying a return datatype is optional**

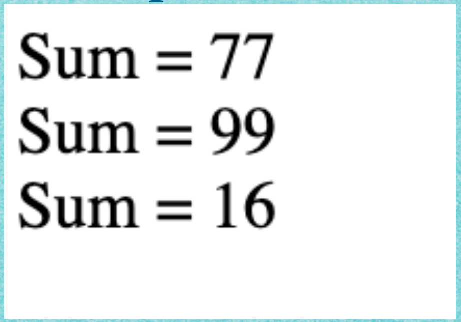
Return Example with no return datatype



```
<?php declare(strict_types=1); // s



<?php
function sum(int $a, int $b) {
  $c =$a + $b;
  return $c;
}

echo "Sum = ". sum(23,54)."  <br>";
echo "Sum = " .sum(54,45)."  <br>";
echo "Sum =". sum(12,46)
?>
```

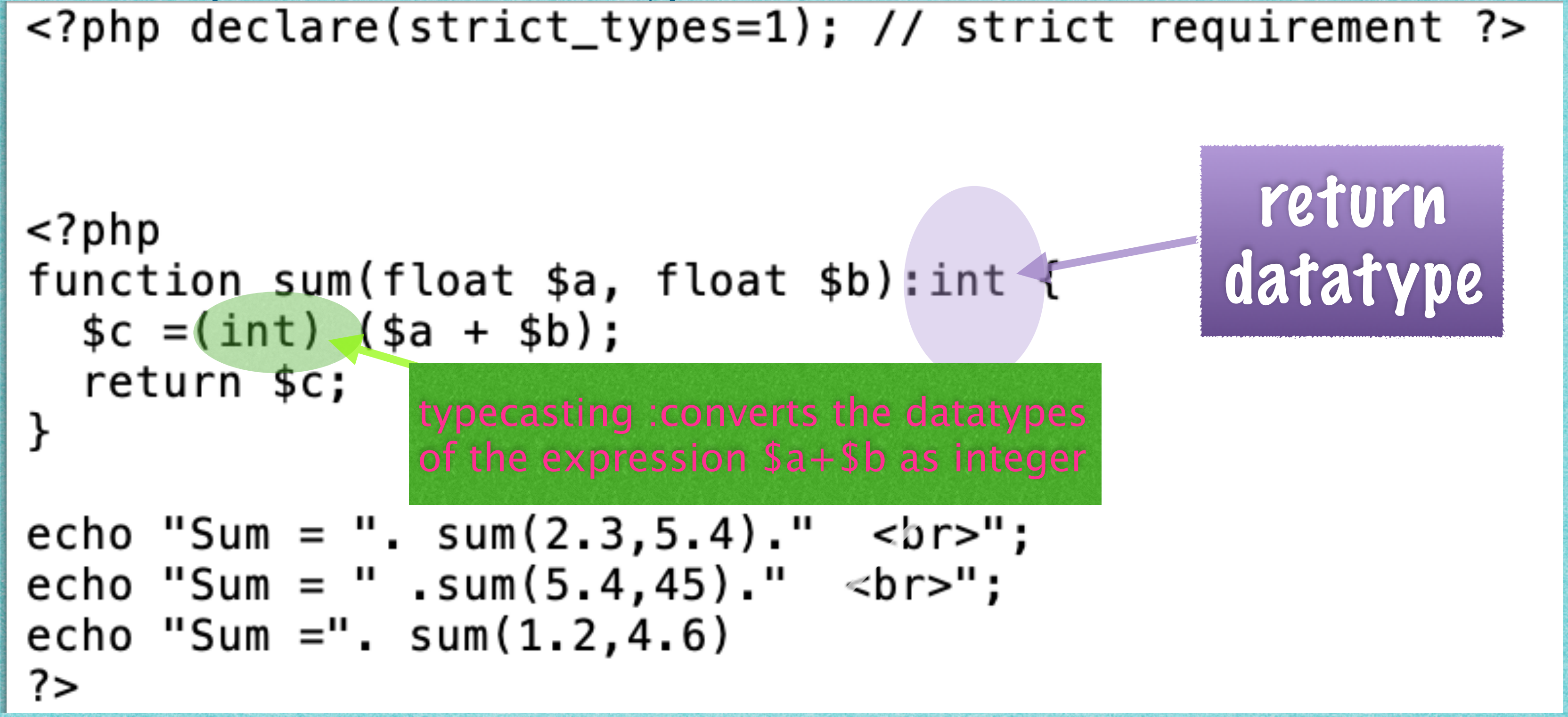
no return data type is specified

Output



```
Sum = 77
Sum = 99
Sum = 16
```

Example for strict datatype



```
<?php declare(strict_types=1); // strict requirement ?>



<?php
function sum(float $a, float $b):int {
  $c =(int) ($a + $b);
  return $c;
}

echo "Sum = ". sum(2.3,5.4)."  <br>";
echo "Sum = " .sum(5.4,45)."  <br>";
echo "Sum =". sum(1.2,4.6)
?>
```

return datatype

typecasting :converts the datatypes of the expression $a+$b as integer

Output

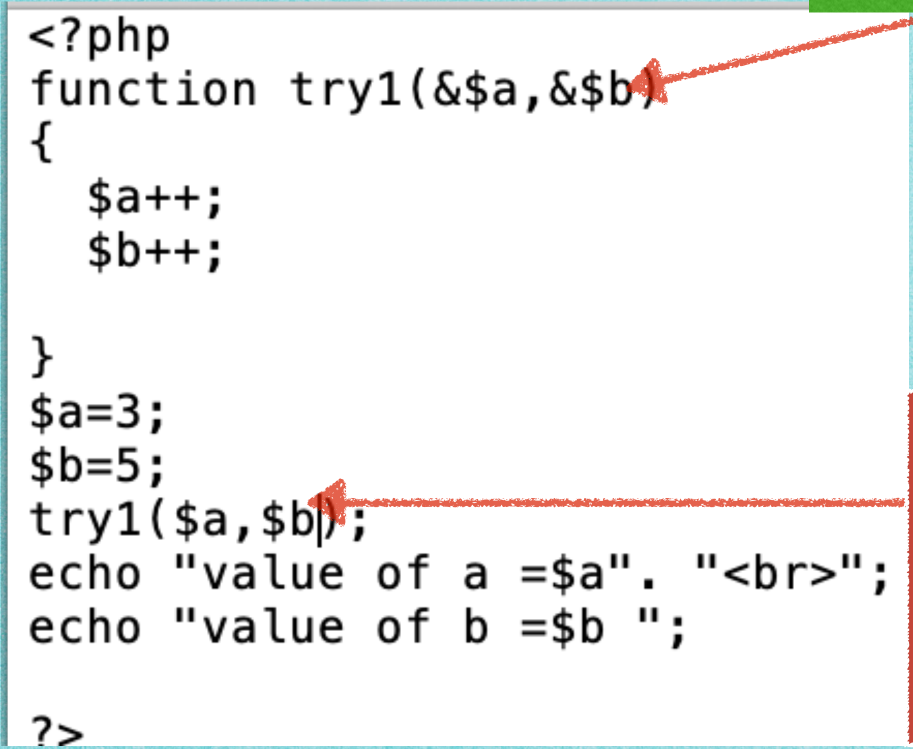
```
Sum = 7
Sum = 50
Sum =5
```

truncates the result to integer

Parameter passing to Functions

- PHP allows us two ways in which an argument can be passed into a function:
- Pass by Value:
  - On passing arguments using pass by value, the value of the argument gets changed within a function, but the original value outside the function remains unchanged.
  - That means a duplicate of the original value is passed as an argument.
- Pass by Reference:
  - On passing arguments as pass by reference, the original value is passed.
  - Therefore, the original value gets altered.
  - In pass by reference we actually pass the address of the value, where it is stored using ampersand sign(&).

Example for pass by reference

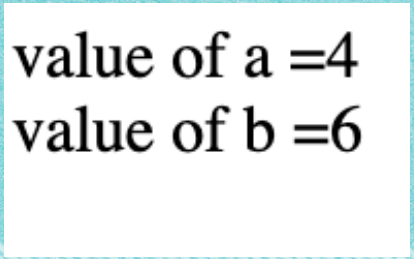


```
<?php
function try1(&$a,&$b)
{
   $a++;
   $b++;

}
$a=3;
$b=5;
try1($a,$b);
echo "value of a =$a". "<br>";
echo "value of b =$b ";

?>
```

& - indicates pass by reference

no & during function call
only variables can be passed
and no constants are allowed
eg: try1(3, 5) generates error

Output



```
value of a =4
value of b =6
```

Passing Information between Pages

Ceena Mathews
Associate Professor, PNC

Information can be passed between web pages

1. URL

2.Cookies

3. Session

assing information between pages using URL

- Forms are used mainly to collect data and then transfer them to a processing page.

- Here the processing page uses one of the above methods to pass values to different pages as per the requirements.

- two ways the client can send information to the web server.

    1. The GET Method
    2. The POST Method

- Before the browser sends the information, it encodes it using a scheme called URL encoding.
- In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

- Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values.
- After the information is encoded, it is sent to the server.

GET method

* GET method sends the encoded user information appended to the page request.

* The page and the encoded information are separated by the ? character.

  http://www.test.com/index.htm?name1=value1&name2=value2

* GET also has limits on the amount of information to send. [1024 characters only].

* cannot be used to send binary data, like images or word documents, to the server.

* However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

* GET may be used for sending non-sensitive data.

* $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

Post method

* Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request[headers])
* can send large amount of information
* Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
* However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
* can be used to send ASCII as well as binary data.
* $_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post".
* $_POST is also widely used to pass variables
* can be used to send sensitive information

_GET super global variable

- It is used to get the value passed using Get method

  Syntax

  variablename= $_GET[‘name’];

- where name is the name of control whose value is being accessed

  eg

```
<input type=text name=a>
<?php
$n=$_GET[‘a’]
?>
```

_POST super global variable

- It is used to get the value passed using Post method

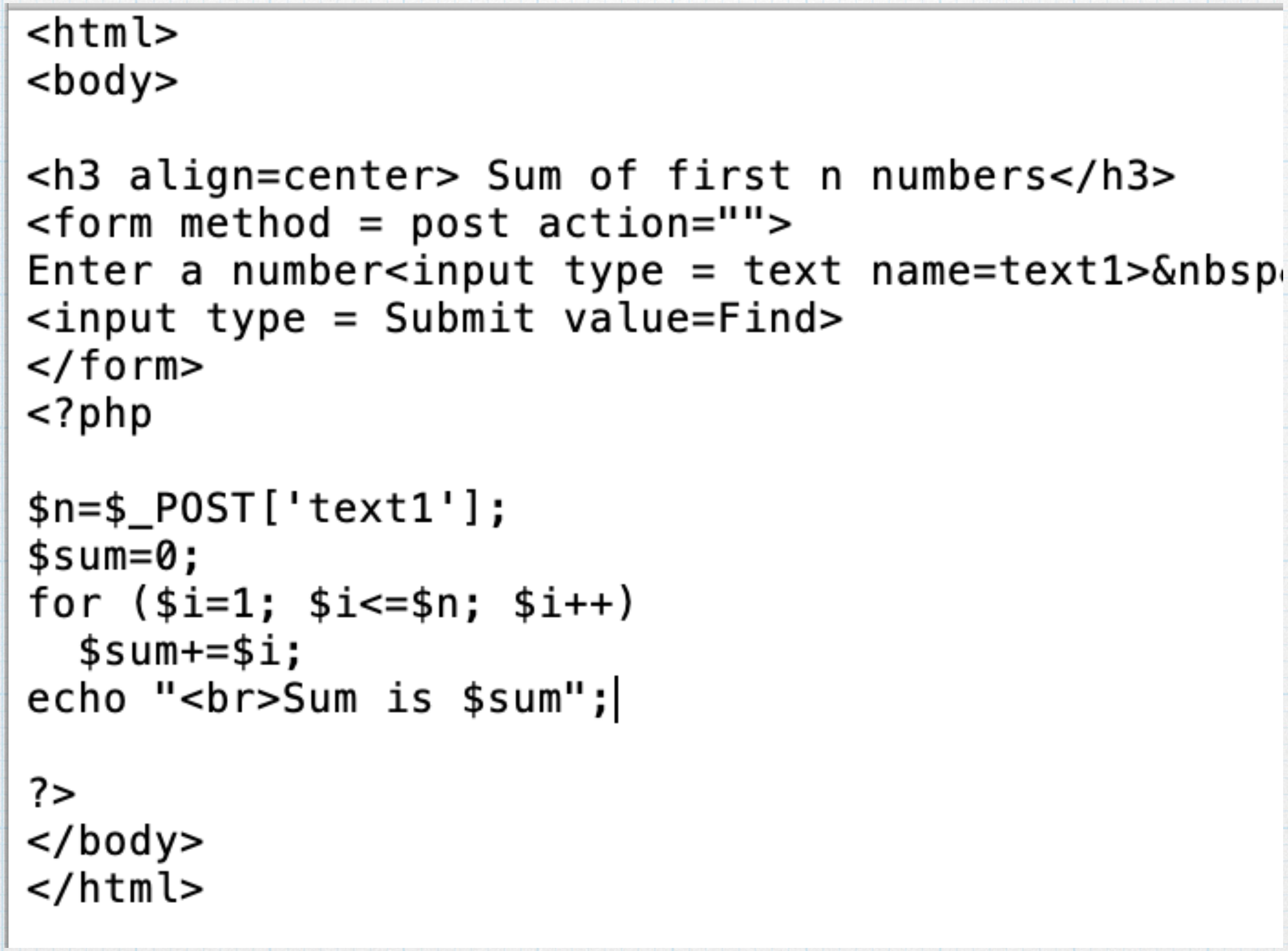
Syntax

variablename= $_POST['name'];

- where name is the name of control whose value is being accessed

```
eg
<input type=text name=a>
<?php
$n=$_POST['a']
?>
```

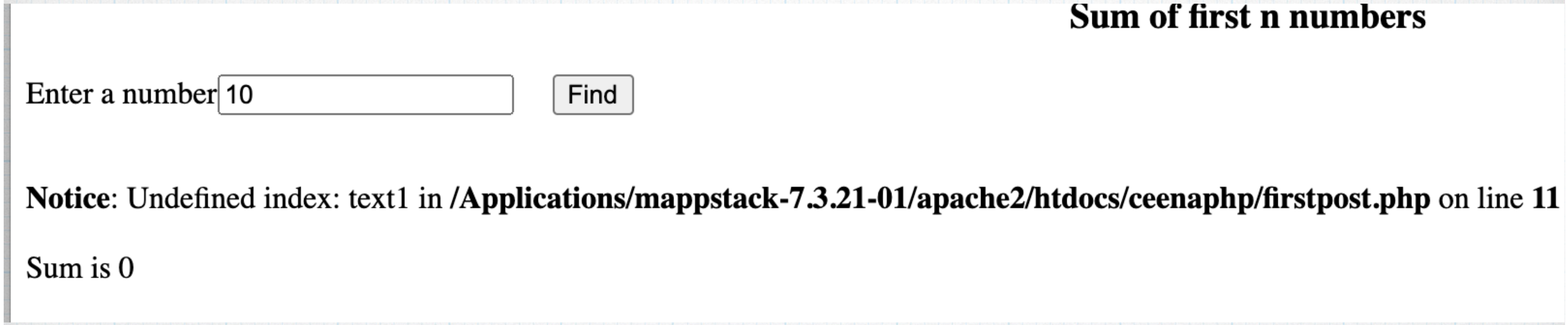
Example 1



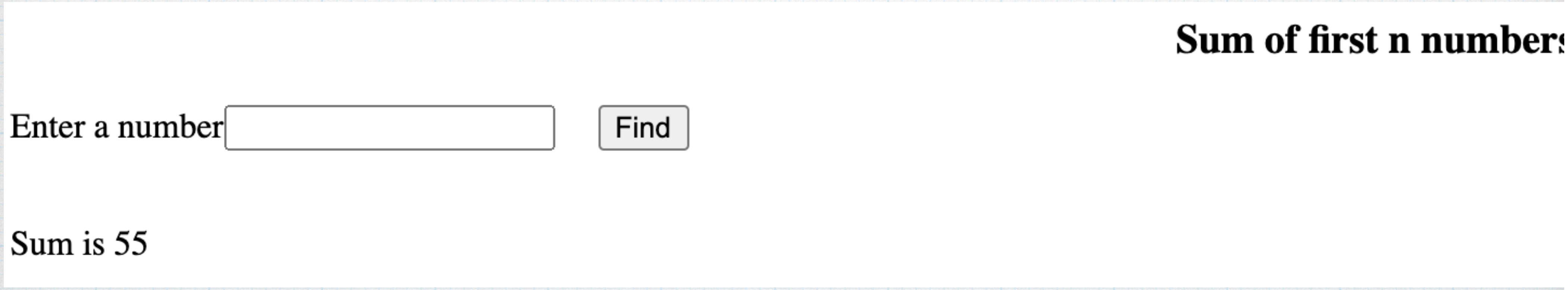
```
<html>
<body>

<h3 align=center> Sum of first n numbers</h3>
<form method = post action="">
Enter a number<input type = text name=text1>&nbsp
<input type = Submit value=Find>
</form>
<?php

$n=$_POST['text1'];
$sum=0;
for ($i=1; $i<=$n; $i++)
  $sum+=$i;
echo "<br>Sum is $sum";

?>
</body>
</html>
```

Ceena Mathews, PNC

Output



```
Sum of first n numbers

Enter a number 10          Find

Notice: Undefined index: text1 in /Applications/mappstack-7.3.21-01/apache2/htdocs/ceenaphp/firstpost.php on line 11

Sum is 0
```



```
Sum of first n numbers

Enter a number             Find

Sum is 55
```

Ceena Mathews, PNC

isset()

- isset() function checks whether a variable is set, which means that it has to be declared and is not NULL.

- This function returns true if the variable exists and is not NULL, otherwise it returns false.

- Note: If multiple variables are supplied, then this function will return true only if all of the variables are set.

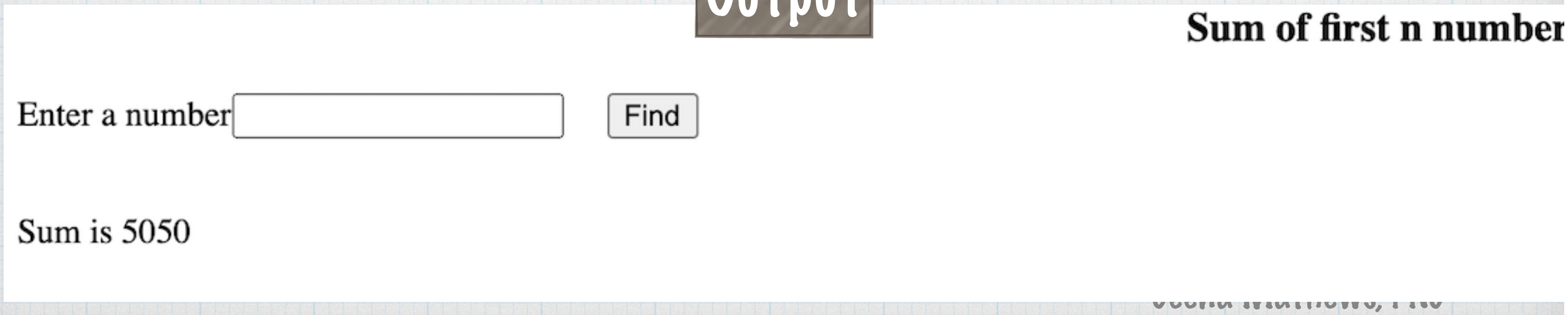
Syntax

isset(variable, ...);

Example 2

```
<html>
<body>

<h3 align=center> Sum of first n numbers</h3>
<form method = post action="">
Enter a number<input type = text name=text1>&nbsp&nbsp&nbsp&nbsp
<input type = Submit value=Find>
</form>
<?php
if (isset($_POST['text1']))
{
$n=$_POST['text1'];
$sum=0;
for ($i=1; $i<=$n; $i++)
  $sum+=$i;
echo "<br>Sum is $sum";
}
?>
</body>
</html>
```

Output



Sum of first n number

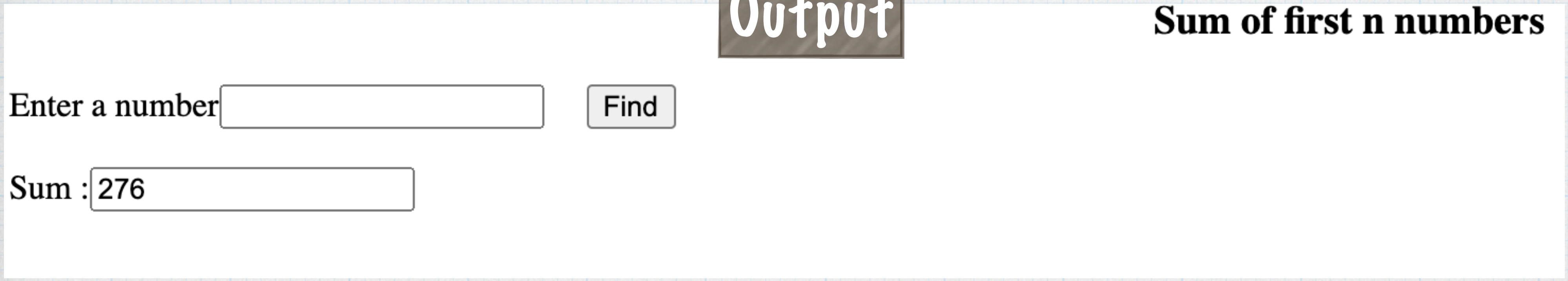
Enter a number

Sum is 5050

Example 3

```
<html>
<body>
<?php
if (isset($_POST['text1']))
{
$n=$_POST['text1'];
$sum=0;
for ($i=1; $i<=$n; $i++)
  $sum+=$i;

}
?>
<h3 align=center> Sum of first n numbers</h3>
<form method = post action="">
Enter a number<input type = text name=text1>&nbsp&nbsp&nbsp&nbsp
<input type = Submit value=Find>
<br><br>
Sum :<input type =text readonly value="<?php if (isset($_POST['text1'])) { echo $sum;}?>">
</form>

</body>
</html>
```

Output



Sum of first n numbers

Enter a number [   ]   [Find]

Sum : [276]

Cookies

- cookie is a small piece of information which is stored at client browser. [A cookie is a small file that the server embeds on the user's computer.]
- It is used to recognize the user.
- Cookie is created at server side and saved to client browser.
- Each time the same computer requests a page with a browser, it will send the cookie too.
- With PHP, you can both create and retrieve cookie values
- PHP Cookie must be used before <html> tag

setcookie() function

- PHP setcookie() function is used to set cookie with HTTP response.

  Syntax:setcookie(name[, value[, expire[, path[, domain[, secure[, httponly]]]]]]);

- Name – This sets the name of the cookie . This variable is used while accessing cookies.
- Value – This sets the value of the named variable and is the content that you actually want to store.
- Expiry – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- Path – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- Domain – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- Security – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP

```
Ex 1: setcookie(“name", "John Watkin", time()+3600, "/","", 0);

Ex 2 :setcookie(“age", "36", time()+3600, "/", "",  0);
```

Accessing Cookies

* Once cookie is set, access it by $_COOKIE superglobal variable.

```
Ex :echo $_COOKIE[“name"]
```

Deleting Cookie

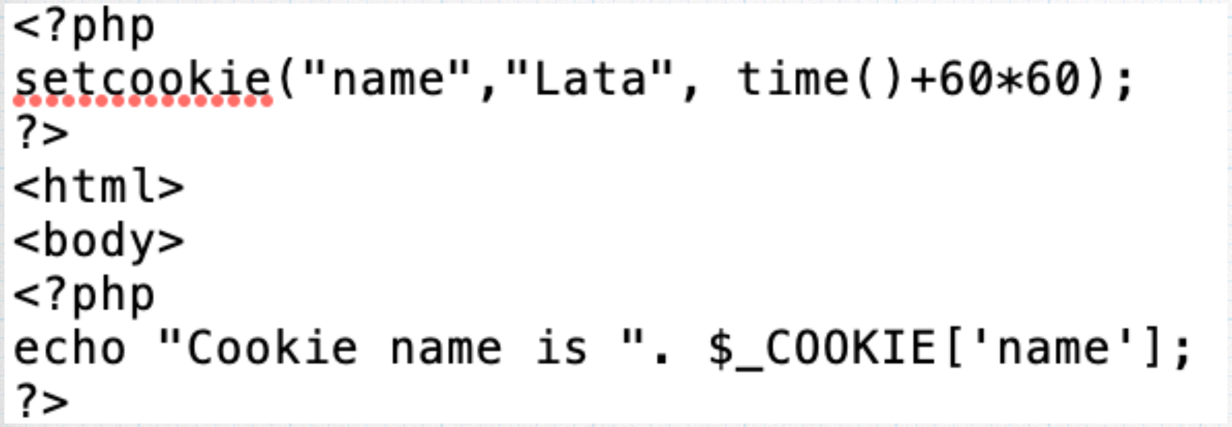
use the setcookie() function with an expiration date in the past

Ex: setcookie("name", "", time()- 60, "/","", 0);

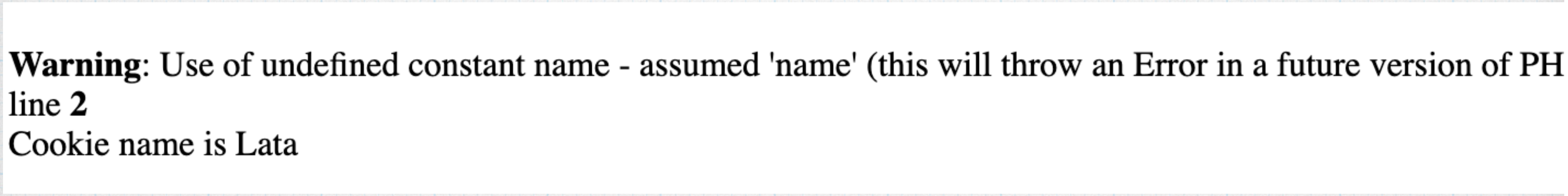
Ex : setcookie(“user", "", time() - 3600);

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function



```
<?php
setcookie("name","Lata", time()+60*60);
?>
<html>
<body>
<?php
echo "Cookie name is ". $_COOKIE['name'];
?>
```



```
Warning: Use of undefined constant name - assumed 'name' (this will throw an Error in a future version of PH
line 2
Cookie name is Lata
```

Session

- Session is a way to store information (in variables) to be used across multiple pages.

- PHP session stores data on the server rather than user's computer.

- In a session based environment, every user is identified through a unique number called session identifier or SID.

- session IDs are randomly generated by the PHP engine

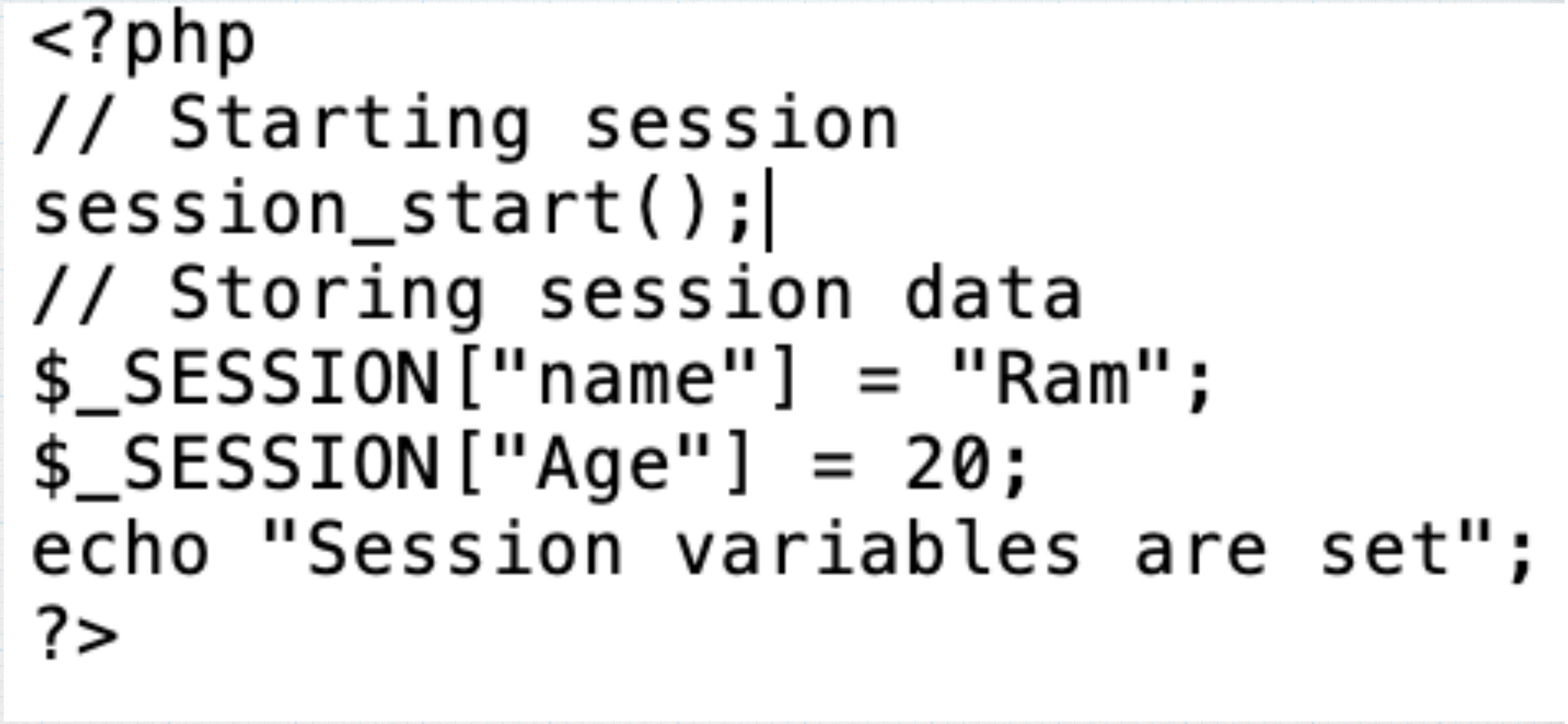
- This unique session ID is used to link each user with their own information on the server like emails, posts, etc

* PHP session is easily started by making a call to the session_start() function.
* function first checks to see if a session already exists by looking for the presence of a session ID.
    - If it finds one, i.e. if the session is already started, it sets up the session variables and if doesn't, it starts a new session by creating a new session ID
* It is recommended to put the call to session_start() at the beginning of the page.

```
<?php

// Starting session

session_start();

?>
```

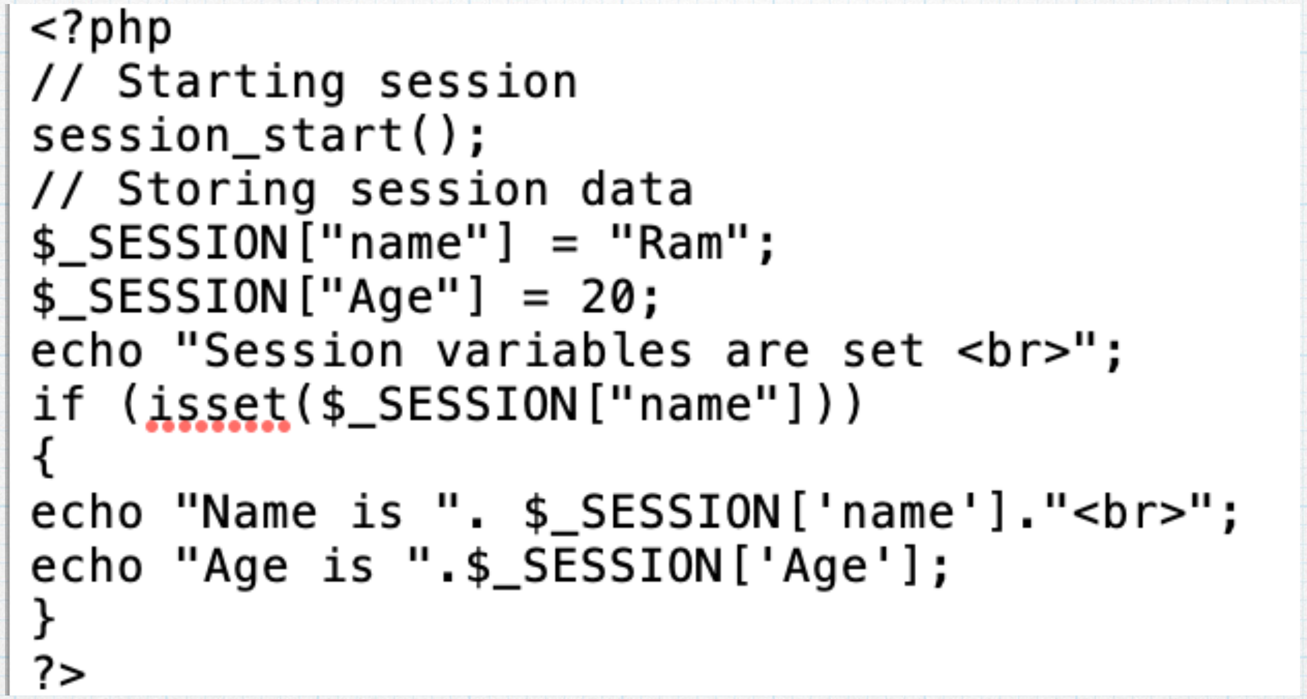
* Session variables are set with the PHP global variable: $_SESSION
* These variables can be accessed during lifetime of a session.

Example



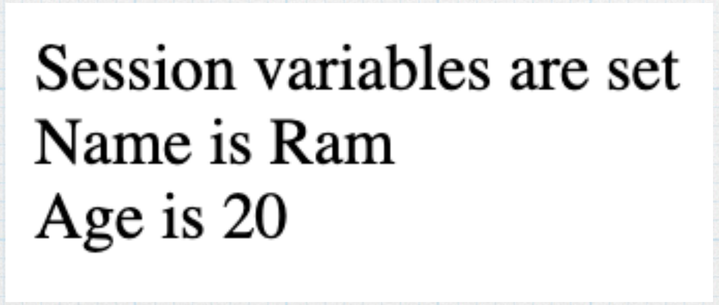
```
<?php
// Starting session
session_start();
// Storing session data
$_SESSION["name"] = "Ram";
$_SESSION["Age"] = 20;
echo "Session variables are set";
?>
```

Get PHP Session Variable Values



```
<?php
// Starting session
session_start();
// Storing session data
$_SESSION["name"] = "Ram";
$_SESSION["Age"] = 20;
echo "Session variables are set <br>";
if (isset($_SESSION["name"]))
{
echo "Name is ". $_SESSION['name']."<br>";
echo "Age is ".$_SESSION['Age'];
}
?>
```

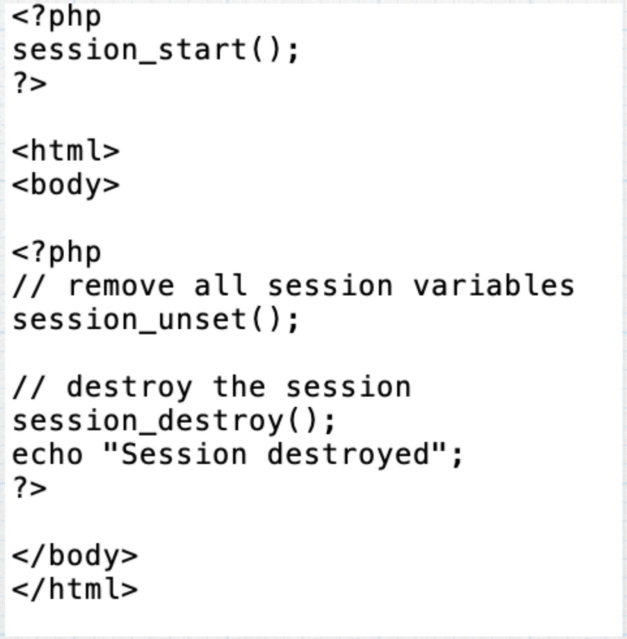
Output



```
Session variables are set
Name is Ram
Age is 20
```

Destroying a PHP Session

* session_destroy()
    - A PHP session can be destroyed by session_destroy() function.
    - This function does not need any argument
* session_unset()
    - no arguments
    - is used to destroy a single session variable



```
<?php
session_start();
?>

<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
echo "Session destroyed";
?>

</body>
</html>
```

HEADERS IN PHP

- HTTP header fields provide **required information about the request or response, or about the object sent** in the message body.
-
- There are four types of HTTP message headers

1. **General Header**
2. **Client request header**
3. **Server response header**
4. **Entity header**

- General-header: These header fields used with **both request and response messages.**

- Client Request-header: These header fields contains **information about the fetched request by the client.**

- Server Response-header: These header fields contains the **location of the source that has been requested by the client.**

- Entity-header: These header fields define **meta information about the entity-body or, if no body is present**, about the resource identified by the request.

General header

- Connectionn: allows the sender to specify options that are desired for that particular connection
    - Example

      connection : “Connection”

      Connection: close

Client Request Headers

- Accept: field used to specify certain media types which are acceptable for the response. The general syntax is as follows:
    - Accept: type/subtype [q=qvalue]
    - Multiple media types can be listed separated by commas and the optional qvalue represents an acceptable quality level for accept types on a scale of 0 to 1.
    - example:

      Accept: text/plain; q=0.5, text/html;
- Cookie: name=value

Server response header

- Location: field is used to redirect the recipient to a location other than the Request-URI for completion.
    - The general syntax is:

      Location : absoluteURI

    - example:

      Location: http://www.example.org/http/index.htm

- Set-Cookie: NAME=VALUE; OPTIONS

Entity Headers

➤ Allow: lists the set of methods supported by the resource identified by the Request-URI.

- Syntax

  Allow : Method

- You can specify multiple methods separated by commas.
- example:

  Allow: GET, HEAD, PUT

➤ Content-Encoding: used as a modifier to the media-type.

- The general syntax is:

  Content-Encoding : content-coding

- example:

  Content-Encoding: gzip

header() function

- sends a raw HTTP header to a client.
- It must be called before any actual output is sent!

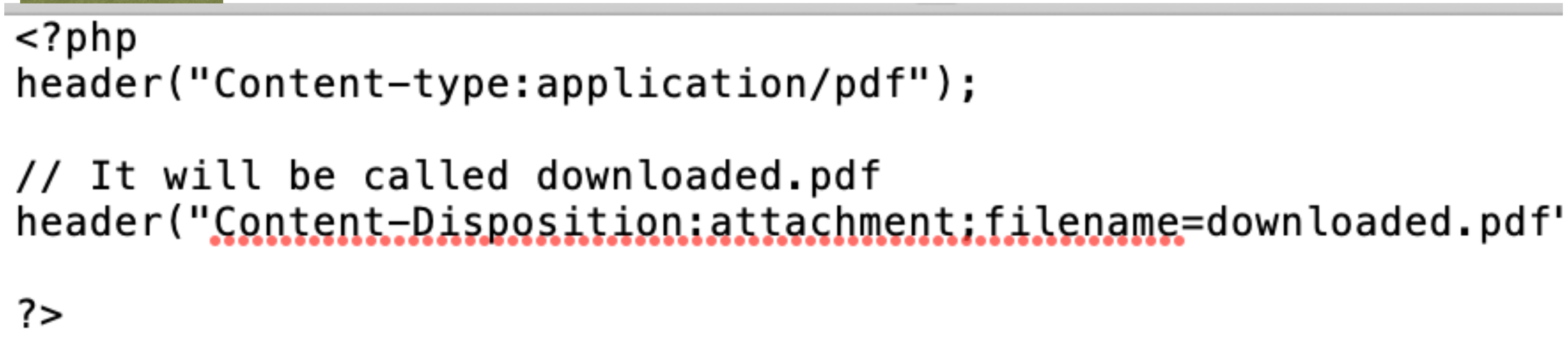
Syntax

header(header, replace, http_response_code)

  - **header : Specifies the header string to send**
  - **replace: Optional. Indicates whether the header should replace a previous similar header or add a new header of the same type.**
    - Default is TRUE (will replace).
    - FALSE allows multiple headers of the same type
  - **http_response_code:Optional. Forces the HTTP**

- Content-Disposition header is used to supply a recommended filename and force the browser to display the save dialog box

Example1



```
<?php
header("Content-type:application/pdf");

// It will be called downloaded.pdf
header("Content-Disposition:attachment;filename=downloaded.pdf'

?>
```

Example2

header("Location: https://www.tutorialspoint.com/");

ARRAYS IN PHP

- array can hold many values under a single name

- access the values by referring to an **index number**

- there are **three types of arrays:**

  1. **Indexed arrays(numeric arrays) -** Arrays with a numeric index
  2. **Associative arrays -** Arrays that uses named keys
  3. **Multidimensional arrays -** Arrays containing one or more arrays

➤ **Indexed arrays/Numeric arrays**

➤ Create arrays : 2 ways

1. array() function is used to create an array where index is automatically assigned(index starts at zero)

Syntax

```
varname=array([list of elements]);
```

index 0

```
eg:$fruits = array("apple", "grapes", "orange");
```

2. Index can be assigned manually

- Eg:

  $fruits[1]=“apple”;

  $fruits[2]=“grapes”;

- Loop Through an Indexed Array

  To loop through and print all the values of an indexed array, c use a for loop

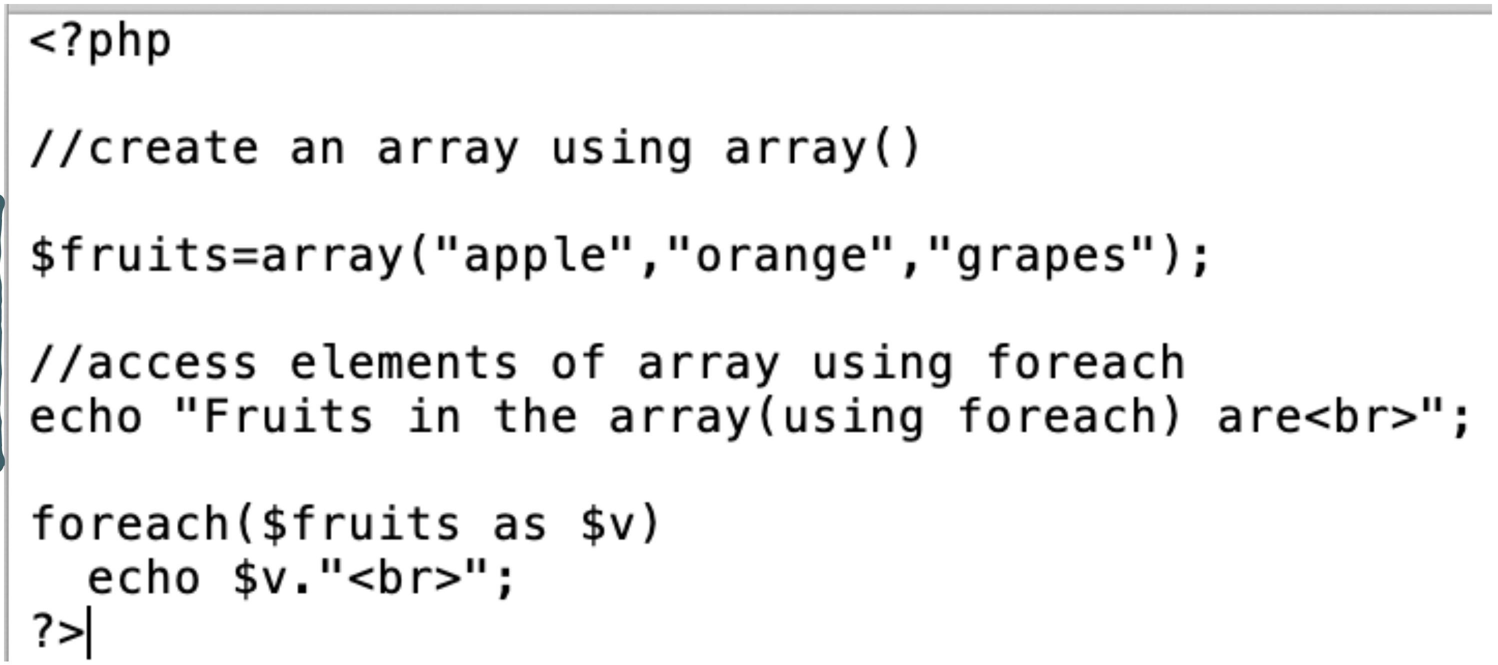
Example

```
<?php

//create an array using array()

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

//access elements of array using array()
echo "Fruits in the array are<br>";

for($i=0;$i<3;$i++)
  echo $fruits[$i]."<br>";
?>
```

Output

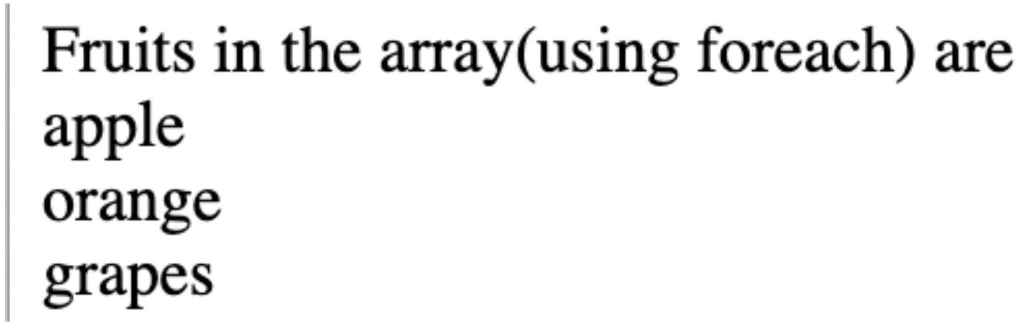
```
Fruits in the array are
apple
orange
grapes
```

Example
r indexed array
using
foreach()



```
<?php

//create an array using array()

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

//access elements of array using foreach
echo "Fruits in the array(using foreach) are<br>";

foreach($fruits as $v)
  echo $v."<br>";
?>
```

Output



```
Fruits in the array(using foreach) are
apple
orange
grapes
```

Associative Arrays

- They are arrays that use named keys.

- There are two ways to create an associative array

1. Using array()

Syntax

key is optional. if not given, it will use numeric indexes

```
varname=array([key1=>]value1,[key2=>]value2,…);
```

Eg:

Key *Value*

```
$fruits = array("red"=>"apple", “orange"=>"orange");
```

2. Syntax : arrayname[key]=value;

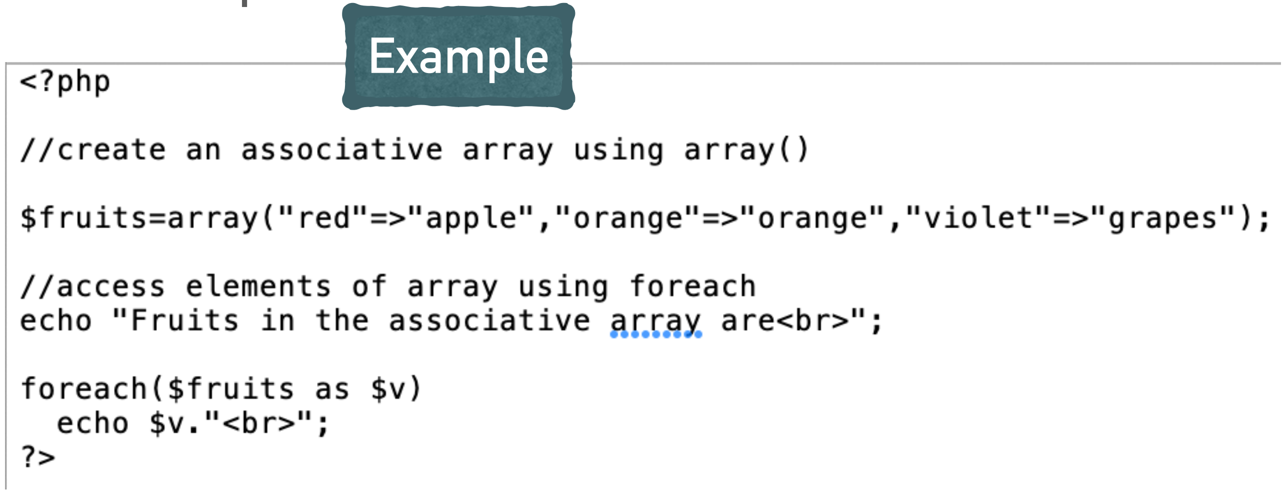
➤ Eg:

$fruits['red']="apple";

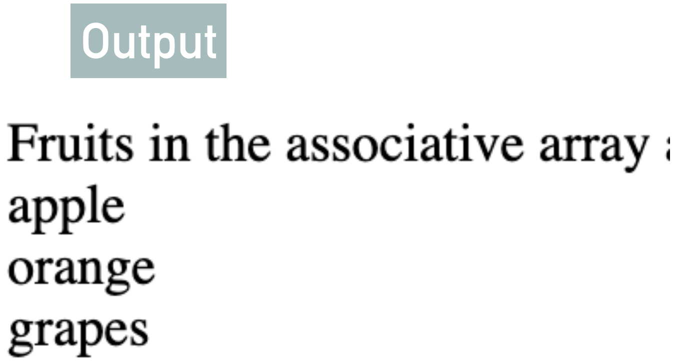
$fruits['orange']="orange";

Loop Through an Associative Array

To loop through and print all the values of an associative array, use a foreach loop


Example

```
<?php

//create an associative array using array()

$fruits=array("red"=>"apple","orange"=>"orange","violet"=>"grapes");

//access elements of array using foreach
echo "Fruits in the associative array are<br>";

foreach($fruits as $v)
  echo $v."<br>";
?>
```


Output

```
Fruits in the associative array
apple
orange
grapes
```

Multidimensional Arrays

- an array containing one or more arrays.

- It can be two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

- each element in the sub-array can be an array, and so on.

- Values in the multi-dimensional array are accessed using multiple index.

Example for Multidimensional array

```
<?php
         $marks = array(
            "Santo" => array (
               "physics" => 35,
               "maths" => 30,
               "chemistry" => 39
            ),

            "Domz" => array (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
            ),

            "Francis" => array (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
            )
         );

         /* Accessing multi-dimensional array values */
         echo "Marks for Santo in physics : " ;
         echo $marks['Santo']['physics'] . "<br />";

         echo "Marks for Francis in maths : ";
         echo $marks['Francis']['maths'] . "<br />";

         echo "Marks for Domz in chemistry : " ;
         echo $marks['Domz']['chemistry'] . "<br />";
      ?>
```

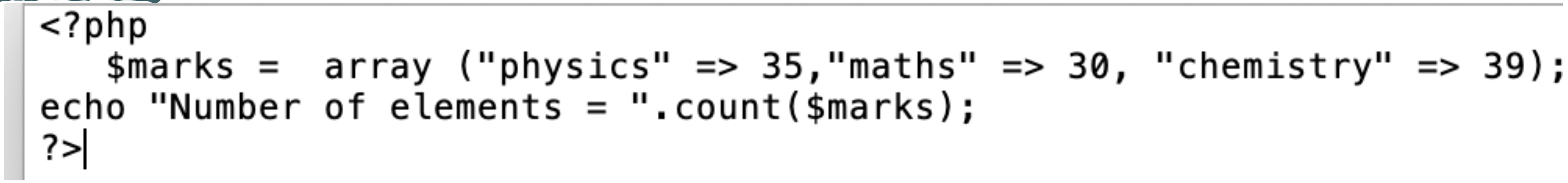
Output

```
Marks for Santo in physics : 35
Marks for Francis in maths : 22
Marks for Domz in chemistry : 29
```

The Length of an Array

➤ The count() function is used to return the length (the number of elements) of an array

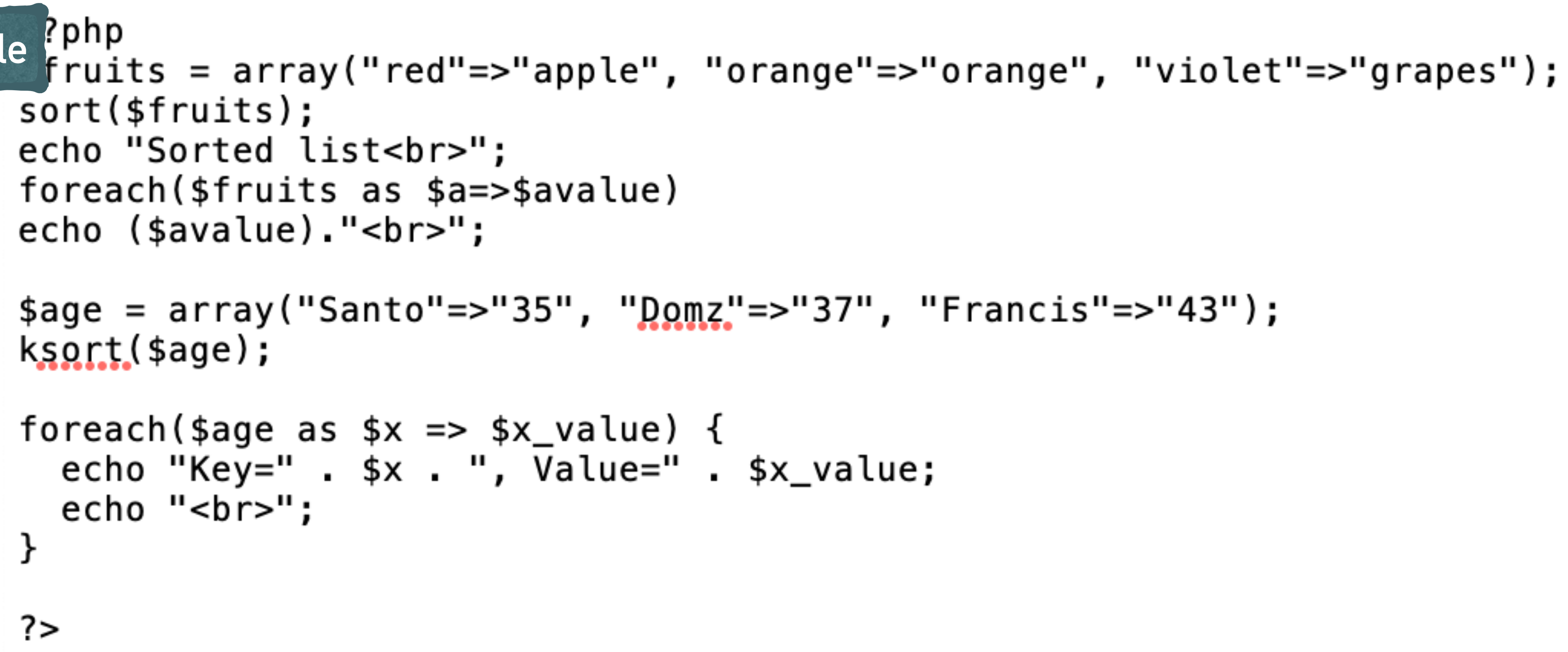
Example



```
<?php
   $marks =  array ("physics" => 35,"maths" => 30, "chemistry" => 39);
echo "Number of elements = ".count($marks);
?>
```

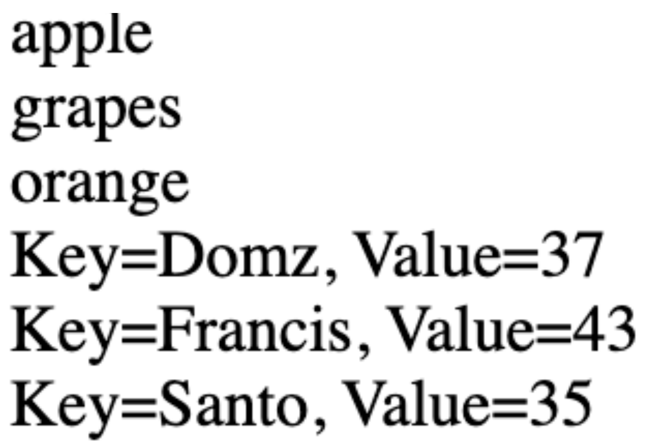
- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Example



```
?php
fruits = array("red"=>"apple", "orange"=>"orange", "violet"=>"grapes");
sort($fruits);
echo "Sorted list<br>";
foreach($fruits as $a=>$avalue)
echo ($avalue)."<br>";

$age = array("Santo"=>"35", "Domz"=>"37", "Francis"=>"43");
ksort($age);

foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}

?>
```

Output



```
apple
grapes
orange
Key=Domz, Value=37
Key=Francis, Value=43
Key=Santo, Value=35
```

array_unique() function

- removes duplicate values from an array.
- If two or more array values are the same, the first appearance will be kept and the other will be removed.

- The returned array will keep the first array item's key type.

Syntax

array_unique(array, sorttype)

sorttype Optional. Specifies how to compare the array elements/ items.
Possible values:
SORT_STRING - Default. Compare items as strings
SORT_REGULAR - Compare items normally (don't change types)
SORT_NUMERIC - Compare items numerically
SORT_LOCALE_STRING - Compare items as strings, based on current locale

Example

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"red");
$s=array_unique($a);
foreach($s as $v)
 echo "<br>".$v;
?>
```

Output

```
red
green
```

list() function

➤ used to assign values to a list of variables in one operation.

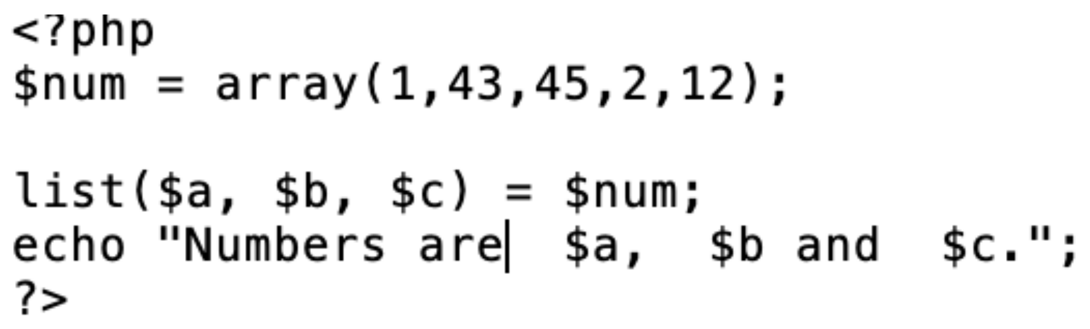
Syntax

list(var1, var2, ...)

Parameter Values

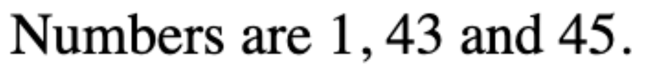
var1	Required. The first variable to assign a value to
var2,...	Optional. More variables to assign values to

Example



```
<?php
$num = array(1,43,45,2,12);

list($a, $b, $c) = $num;
echo "Numbers are  $a,  $b and  $c.";
?>
```

Output



```
Numbers are 1, 43 and 45.
```

POSTGRESQL USING PHP FUNCTIONS

Ceena Mathews,PNC

I. pg_connect

➤ **opens a connection to a PostgreSQL database** specified by the connection_string.

Syntax

    $res pg_connect (string $connection_string [, int $connect_type])

Parameters

1. connection_string

   ➤ connection_string can be empty to use all default parameters, or it can contain one or more parameter settings separated by whitespace.

   ➤ Each parameter setting is in the form keyword = value.

   ➤ Single quotes and backslashes within the value must be escaped with a backslash, i.e., \' and \\.

   ➤ The currently recognized parameter keywords are: host, port, dbname (defaults to value of user), user, password, connect_timeout, options and service.

   ➤ The options parameter can be used to set command line parameters to be invoked by the server.

2. connect_type

It has 2 values

1. PGSQL_CONNECT_FORCE_NEW is passed then a new connection is created, even if the connection_string is identical to an existing connection.
2. PGSQL_CONNECT_ASYNC is given, then the connection is established asynchronously.

Return Values
PostgreSQL connection resource on success, FALSE on failure.

- If a second call is made to pg_connect() with the same connection_string as an existing connection, the existing connection will be returned unless you pass PGSQL_CONNECT_FORCE_NEW as connect_type.

Example

```
$connStr = "host=localhost port=5432 dbname=postgres user=postgres password=qwerty";
$conn = pg_connect($connStr);
```

II. pg_dbname

Syntax

$db=pg_dbname ([resource $connection])

➢ It returns the name of the database of the given PostgreSQL connection resource.

Parameters

1.connection

- ➢ PostgreSQL database connection resource.
- ➢ When connection is not present, the default connection is used.
- ➢ The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values ¶
A string containing the name of the database specifying the connection or FALSE on error.

Example

```
pg_connect("host=localhost port=5432 dbname=postgres user=postgres
password=qwerty");
  echo pg_dbname(); //postgres
```

III. pg_connection_status

Syntax

pg_connection_status (resource $connection)

➤ It returns the status of the specified connection.

Parameters

connection : PostgreSQL database connection resource.

Return Values

PGSQL_CONNECTION_OK or PGSQL_CONNECTION_BAD

Example

```
 $stat = pg_connection_status($conn);
 if ($stat === PGSQL_CONNECTION_OK) {
     echo 'Connection status ok';
 } else {
     echo 'Connection status bad';
 }
```

IV. pg_close

Syntax

pg_close ([resource $connection])

- It closes the non-persistent connection to a PostgreSQL database associated with the given connection resource.

Parameters

1.connection

- ➤ PostgreSQL database connection resource.
- ➤ When connection is not present, the default connection is used.
- ➤ The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

Returns TRUE on success or FALSE on failure.

Example

```
pg_close($dbconn);
```

Executing Query

V.pg_query

Syntax

pg_query ([resource $connection], string $query)

- It executes the query on the specified database connection.
- If an error occurs, and FALSE is returned, details of the error can be retrieved using the pg_last_error() function if the connection is valid.

Parameters

1. connection

   PostgreSQL database connection resource. When connection is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

2. query

   The SQL statement or statements to be executed

Example

```
<?php

$connStr = "host=localhost port=5432 dbname=postgres user=postgres password=qwerty";
$conn = pg_connect($connStr);

if (!$conn)
{
  echo "An error occurred.\n";
  exit;
}
else
{ echo “Success”;}

$result = pg_query($conn, "SELECT author, email FROM authors");
if (!$result) {
  echo "An error occurred.\n";
  exit;
}
?>
```

VI. pg_execute

Syntax

pg_execute ([resource $connection], string $stmtname , array $params)

- ➤ Sends a request to execute a prepared statement with given parameters, and waits for the result.
- ➤ the command to be executed is specified by naming a previously-prepared statement, instead of giving a query string.
- ➤ This feature allows commands that will be used repeatedly to be parsed and planned just once, rather than each time they are executed.
- ➤ The statement must have been prepared previously in the current session.
- ➤ pg_execute() is supported only against PostgreSQL 7.4 or higher connections; it will fail when using earlier versions.

Parameters

1.connection

- PostgreSQL database connection resource. When connection is not present, the default connection is used. The default connection is the last connection made by pg_connect()

2.stmtname

- The name of the prepared statement to execute.
- if "" is specified, then the unnamed statement is executed.
- The name must have been previously prepared using pg_prepare().

3.params

- An array of parameter values to substitute for the $1, $2, etc. placeholders in the original prepared query string.
- The number of elements in the array must match the number of placeholders.

Example

```
// Prepare a query for execution

$result = pg_prepare($dbconn, "my_query", 'SELECT * FROM shops WHERE name
=$1');


$result = pg_execute($dbconn, "my_query", array("Joe's Widgets”));
```

VII. pg_last_error

Syntax
$str=pg_last_error ([resource $connection])

- pg_last_error() returns the last error message for a given connection.

- Error messages may be overwritten by internal PostgreSQL (libpq) function calls.
- It may not return an appropriate error message if multiple errors occur inside a PostgreSQL module function.

Parameters

1.connection

- ➤ PostgreSQL database connection resource. When connection is not present, the default connection is used. The default connection is the last connection made by pg_connect() or pg_pconnect().

Return Values

A string containing the last error message on the given connection, or FALSE on error.

Retrieving Data

VIII. pg_fetch_row

Syntax

- pg_fetch_row (resource $result [, int $row])

- pg_fetch_row() fetches one row of data from the result associated with the specified result resource.

Parameters

1. result

PostgreSQL query result resource, returned by pg_query(),

2. row

Row number in result to fetch. Rows are numbered from 0 upwards. If omitted or NULL, the next row is fetched.

Return Values

➤ An array, indexed from 0 upwards, with each value represented as a string.
➤ Database NULL values are returned as NULL.

➤ FALSE is returned if row exceeds the number of rows in the set, there are no more rows, or on any other error.

Example

```
$result = pg_query($conn, "SELECT author, email FROM authors");

while ($row = pg_fetch_row($result))
{
  echo "Author: $row[0]  E-mail: $row[1]";
  echo "<br >";
}
```

IX pg_fetch_array

Syntax

pg_fetch_array (resource $result [, int $row [, int $result_type = PGSQL_BOTH]])

- It returns an array that corresponds to the fetched row (record).
- In addition to storing the data in the numeric indices (field number) to the result array, it can also store the data using associative indices (field name). It stores both indicies by default.
- pg_fetch_array() is NOT significantly slower than using pg_fetch_row(), and is significantly easier to use.

Parameters

1. result

PostgreSQL query result resource, returned by pg_query(),

2. row

Row number in result to fetch. Rows are numbered from 0 upwards. If omitted or NULL, the next row is fetched.

3.result_type

- An optional parameter that controls how the returned array is indexed.
- result_type is a constant and can take the following values:
    1. PGSQL_ASSOC : will return only associative indices
    2. PGSQL_NUM :will return an array with numerical indices
    3. PGSQL_BOTH: the default, will return both numerical and associative indices.

Return Values

An array indexed numerically (beginning with 0) or associatively (indexed by field name), or both.

Example

```
$arr = pg_fetch_array($result, 0, PGSQL_NUM);
echo $arr[0] . "  Author\n";
echo $arr[1] . "  E-mail\n";

$arr = pg_fetch_array($result, NULL, PGSQL_ASSOC);
echo $arr["author"] . " Author\n";
echo $arr["email"] . " E-mail\n";

$arr = pg_fetch_array($result);
echo $arr["author"] . " Author\n";
echo $arr[1] . " E-mail\n";
```

X.pg_fetch_assoc

Syntax

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

- ➤ *returns an associative array that corresponds to the fetched row (records).*

- ➤ *pg_fetch_assoc() is NOT significantly slower than using pg_fetch_row(), and is significantly easier to use.*

Parameters

1. result

PostgreSQL query result resource, returned by pg_query(),

2. row

Row number in result to fetch. Rows are numbered from 0 upwards. If omitted or NULL, the next row is fetched.

Return Values

- ➤ *An array indexed associatively (by field name). Each value in the array is represented as a string.*
- ➤ *Database NULL values are returned as NULL.*

- ➤ *FALSE is returned if row exceeds the number of rows in the set, there are no more rows, or on any other error.*

Example

```
$result = pg_query($conn, "SELECT id, author, email FROM authors");

while ($row = pg_fetch_assoc($result))
{
  echo $row['id'];
  echo $row['author'];
  echo $row['email'];
}
```

XI. pg_fetch_object

Syntax
pg_fetch_object (resource $result [, int $row])

- *It returns an object with properties that correspond to the fetched row's field names.*

Parameters ¶

1. result
PostgreSQL query result resource, returned by pg_query(),

2. row
Row number in result to fetch. Rows are numbered from 0 upwards. If omitted or NULL, the next row is fetched.

Return Values ¶

- *An object with one attribute for each field name in the result. Database NULL values are returned as NULL.*
- *FALSE is returned if row exceeds the number of rows in the set, there are no more rows, or on any other error.*

Example

```
$q = pg_query($db_conn, "SELECT * FROM books ORDER BY author");


while ($data = pg_fetch_object($q))
 {
  echo $data->author;
  echo $data->year ;
  echo $data->title;
}
```

XII. pg_fetch_all

Syntax

pg_fetch_all (resource $result [, int $result_type = PGSQL_ASSOC]) : array

➤ *It returns an array that contains all rows (records) in the result resource.*

Parameters

1.result

PostgreSQL query result resource, returned by pg_query(),

Return Values

➤ *An array with all rows in the result.*

➤ *Each row is an array of field values indexed by field name.*

➤ *FALSE is returned if there are no rows in the result, or on any other error.*

Example

```
$result = pg_query($conn, "SELECT * FROM authors");

$arr = pg_fetch_all($result);

print_r($arr);
```

output

```
Array
(
    [0] => Array
        (
            [id] => 1
            [name] => Fred
        )

    [1] => Array
        (
            [id] => 2
            [name] => Bob
        )
)
```

XIII.pg_free_result

Syntax

pg_free_result (resource $result)

- *It frees the memory and data, associated with the specified PostgreSQL query result resource.*
- *This function need only be called if memory consumption during script execution is a problem. Otherwise, all result memory will be automatically freed when the script ends.*

Parameters

1. result

PostgreSQL query result resource, returned by pg_query()

Return Values

Returns TRUE on success or FALSE on failure.

Example

```
$res = pg_query($db, "SELECT * from student”);

pg_free_result($res);
```

XIV. pg_num_fields

Syntax
pg_num_fields (resource $result)

➤ It returns the number of fields (columns) in a PostgreSQL result resource.

Parameters

1. result

    PostgreSQL query result resource, returned by pg_query()

Return Values

The number of fields (columns) in the result. On error, -1 is returned.

XV. pg_num_rows

Syntax
pg_num_rows (resource $result)

➤ It will return the number of rows in a PostgreSQL result resource.

Parameters
result
    PostgreSQL query result resource, returned by pg_query(),

Return Values
The number of rows in the result. On error, -1 is returned.

XVI. pg_affected_rows

Syntax

pg_affected_rows (resource $result)

➤ It returns the number of tuples (instances/records/rows) affected by INSERT, UPDATE, and DELETE queries.

Parameters

1. result

PostgreSQL query result resource, returned by pg_query(), pg_query_params() or pg_execute() (among others).

Return Values

The number of rows affected by the query. If no tuple is affected, it will return 0.

Example

```
$result = pg_query($conn, "INSERT INTO authors VALUES ('Arundati', 1996, ‘God of Small Things')");

$c = pg_affected_rows($result);

echo $c  . " rows are affected.";
```