



Internet Programming Server Side Scripting

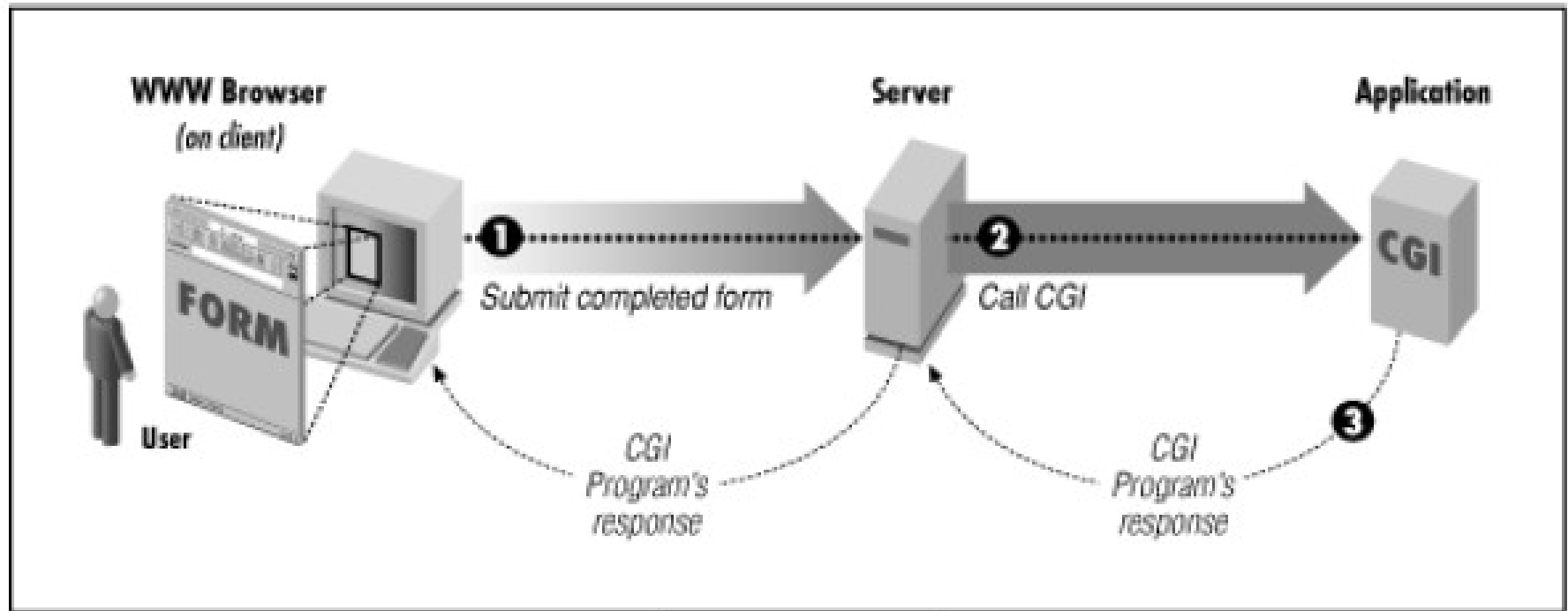
Server Side Scripting

- ▶ Scripts (programs) are stored on the server and are executed on request (from a client).
- ▶ Result of execution is sent to the client in the form of HTML.
- ▶ The server side script/program itself is never returned to the client!
- ▶ May be written in a wide variety of programming/scripting languages.
 - ▶ php, asp, c/c++, perl, java, ...
- ▶ Can perform several powerful operations.

CGI

- ▶ CGI (Common Gateway Interface) – a standard for interfacing external applications with information servers, such as HTTP (Web servers)
 - ▶ Not a programming language! It's a protocol that communicates web forms & server side programs.
- ▶ A CGI program is executed **on request in real-time.**
 - ▶ Can output dynamic information (as opposed to a static html page)
- ▶ It's a way of allowing people in the world to run (execute) a program on our machine.

CGI (cont'd)



Simple Diagram of CGI

CGI (cont'd)

- ▶ CGI programs are usually kept in a separate directory so that the server knows that they should be executed, not returned as files.
- ▶ A CGI program may be written in a variety of programming /scripting languages:
c/c++, perl, fortran, unix shell, VB, vb script, ...
 - ▶ Depends on what's available in the system
- ▶ The CGI program can be
 - ▶ a precompiled code in a compiled programming language (c, c++, fortran, ...)
 - ▶ a script which is interpreted directly without compilation (perl, shell, ...)

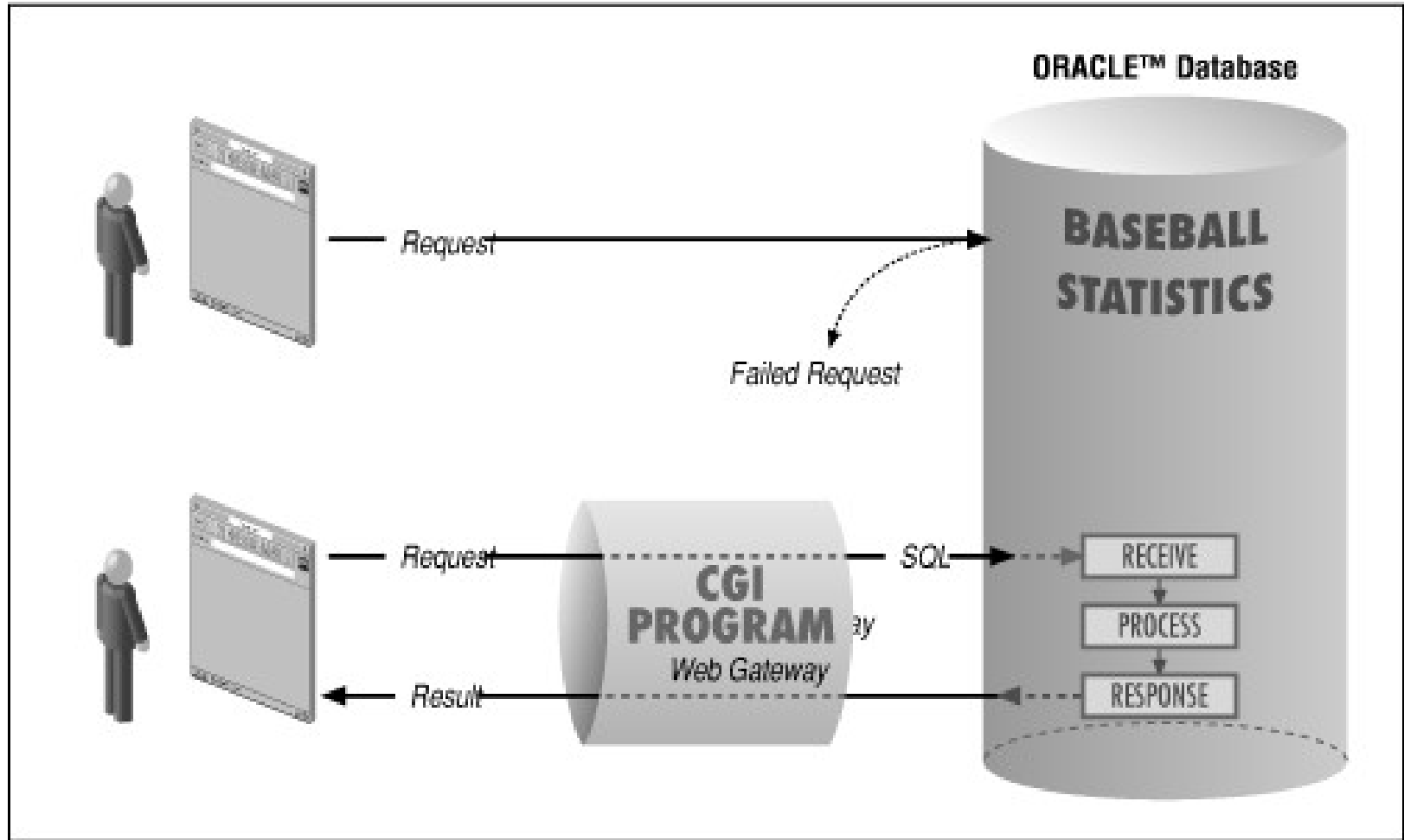
CGI (cont'd)

- ▶ A simple CGI example:

//first.cgi

```
#include <stdio.h>
void main(){
    cout<<"<html>\n";
    cout<<"<head><title>CGI Example</title></head>\n";
    cout<<"<body>\n";
    cout<<"<h1>Hello World!</h1>\n";
    cout<<"</body>\n";
    cout<<"</html>\n";
    exit(0);
}
```

CGI (cont'd)



Gateway to a database

CGI (cont'd)

- ▶ CGI programming can be made more easier through the use of libraries.
- ▶ Used less and less due to its complexity and popularity of other (relatively simpler) server side scripting languages such as PHP.
- ▶ Read more if interested.

PHP (Hypertext Preprocessor)

- ▶ A server side scripting language
- ▶ Open source
- ▶ Fairly easy but quite powerful!
- ▶ Can be embedded within HTML code
- ▶ Object Orientation supported
- ▶ Supports (almost) all major databases including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- ▶ when a page request arrives, the server recognizes PHP content via the file extension (.php)
- ▶ the server executes the PHP code, substitutes output into the HTML page
- ▶ the resulting page is then downloaded to the client
- ▶ user never sees the PHP code, only the output in the page

Applications of PHP

PHP is one of the most widely used language over the web. The following is list few of the **Applications of PHP**

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- You add, delete, modify elements within your database through PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data

▶ **Most servers support PHP**

- ▶ Download PHP for free here:
<http://www.php.net/downloads.php>
- ▶ Download MySQL for free here:
<http://www.mysql.com/downloads/index.html>
- ▶ Download Apache for free here:
<http://httpd.apache.org/download.cgi>

PHP – Language Basics

► Syntax

- PHP code should be enclosed within:

`<?php` and `?>`

So that it is distinguished from HTML.

- Hence, the PHP parser only parses code which is in between `<?php` and `?>`
 - PHP code can be embedded in HTML

Example:

```
<p>This is going to be ignored.</p>
```

```
<?php echo 'While this is going to be parsed.'; ?>
```

```
<p>This will also be ignored.</p>
```

Combining HTML and PHP

Example: A PHP Script Including HTML

```
<html>  
<head> <title>A PHP Script Including HTML</title> </head>  
<body>  
<div><b>  
<?php print "hello world"; ?>  
</b></div>  
</body>  
</html>
```

The PHP engine ignores everything outside PHP open and close tags.

Comments in PHP

- ▶ Standard C, C++, and shell comment symbols

// C++ and Java style comment

Shell type comment

/* C-style comments

These can span multiple lines

*/



Variables

- Variables are used to store data, like string of text, numbers, etc. Variable values can change over the course of a script. Here're some important things to know about variables:
- In PHP, a variable does not need to be declared before adding a value to it.
- PHP automatically converts the variable to the correct data type, depending on its value.
- After declaring a variable it can be reused throughout the code.
- The assignment operator (=) used to assign value to a variable.

Naming Conventions for PHP Variables

These are the following rules for naming a PHP variable:

- All variables in PHP start with a \$ 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 in PHP can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- A variable name cannot contain spaces.

Note: Variable names in PHP are case sensitive, it means \$x and \$X are two different variables. So be careful while defining variable names.

```
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

PHP: Loosely typed language

PHP is a loosely typed language, it means PHP automatically converts the variable to its correct data type.

PHP Type Juggling

- ▶ PHP is not strongly typed
- ▶ PHP is known as a dynamically typed language.
- ▶ Explicit type declaration of a variable is neither needed nor supported in PHP
- ▶ No variable declaration (implicit)
- ▶ To create a variable, just assign some value to it!

Example:

```
$var="Hello"; // variable is string type  
$var=100; //same variable now becomes int
```

- ▶ a variable when assigned value of different type, its type too changes. This approach of PHP to deal with dynamically changing value of variable is called type juggling.

- ▶ Type juggling also takes place during calculation of expression. In this example, a string variable containing digits is automatically converted to integer for evaluation of addition expression

```
<?php  
$var1=100;  
$var2="100";  
$var3=$var1+$var2;  
var_dump($var3);  
?>
```

PHP is case sensitive

Try out following example

```
<html>
  <body>

    <?php
      $capital = 67;
      print("Variable capital is $capital<br>");
      print("Variable CaPiTaL is $CaPiTaL<br>");
    ?>

  </body>
</html>
```

PHP Data Types

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

- Scalar Types (predefined)
- Compound Types (user-defined)
- Special Types

PHP Data Types: Scalar Types

It holds only single value. There are 4 scalar data types in PHP.

boolean

integer

float

string

PHP Data Types: Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

array
object

PHP Data Types: Special Types

There are 2 special data types in PHP.

resource
NULL

PHP var_dump() function

- The **var_dump()** function is a built-in function of PHP that dumps the information about the variables.
- This information includes the data type and value of the variable.
- In case of string, it also includes the **size of the string** passed inside the function.
- The array and object are explored recursively with values to show their structure.
- In simple words, this function provides structured information about one or more variables.

Syntax

Parameter

var_dump(var1, var2, ...);

Expression (var1, var2, ...): variable or the value of the variable, which you want to dump.

PHP Boolean

- Boolean (bool or boolean)
- Simplest of all
- Can be either TRUE or FALSE
- Booleans are the simplest data type works like switch. It holds only two values: TRUE (1) or FALSE (0). It is often used with conditional statements. If the condition is correct, it returns TRUE otherwise FALSE.

```
<?php
    if (TRUE)
        echo "This condition is TRUE.";
    if (FALSE)
        echo "This condition is FALSE.";
?>
```

► Integer

- Hold integer values (signed or unsigned)
- Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
- PHP allows an integer to be expressed in decimal, hexadecimal, octal or binary number system by prefixing appropriate symbol.
- By default Integer is assumed in decimal notation. For Hexadecimal, octal and binary number system, respectively 0x, 0 and 0b symbols are prefixed.

<?php

//Literal assignment of integer value to variable

\$var=232; // Decimal

\$var1=045; // Octal

\$var2=0xB2; //hexadecimal

\$var3=0b1001; //binary

?>

```
<?php
$x = 5985;
var_dump($x);
?>
```

► **PHP Float**

- A floating-point number is a number with a decimal point. Unlike integer, it can hold numbers with a fractional or decimal point, including a negative or positive sign.

```
<?php
    $n1 = 19.34;
    $n2 = 54.472;
    $sum = $n1 + $n2;
    echo "Addition of floating numbers: " . $sum;
?>
```

```
<?php
    $x = 10.365;
    var_dump($x);
?>
```

PHP String

- ▶ A string is a non-numeric data type.
- ▶ It holds letters or any alphabets, numbers, and even special characters.
- ▶ In PHP, literal representation of string can be done with single quotes, double quotes, with heredoc syntax and nowdoc syntax.

```
<?php
```

```
    $company = "XYZ Comapny";
```

```
    //both single and double quote statements will treat different
```

```
    echo "Hello $company";
```

```
    echo "</br>";
```

```
    echo 'Hello $company';
```

```
?>
```

Output:

Hello XYZ Company

Hello \$company

-
- Strings enclosed in single-quotes do not replace variables with the string representations of their values, they do not interpret certain escape sequences.

Double quoted

```
$str2 = "some other string";
```

Strings delimited by the double quotes replaces variables with the string representations of their values as well as specially interpreting certain escape sequences.

► Character escaping

- If the string is enclosed in double-quotes ("), PHP understands more escape sequences for special characters:
The escape-sequence replacements are:

\n is replaced by the newline character

\r is replaced by the carriage-return character

\t is replaced by the tab character

\\$ is replaced by the dollar sign itself (\$)

\\" is replaced by a single double-quote (")

\\ is replaced by a single backslash (\)

► **Variable parsing**

variables within double quoted strings are parsed.

Example

```
<?php
$beer = 'Heineken';
echo "$beer's taste is great"; // works
echo "He drank some $beers";  // won't work
echo "He drank some ${beer}s"; // works
echo "He drank some {$beer}s"; // works
?>
```

- ▶ Variables within double quoted strings ("") are parsed
-

Example

```
$name = "Abebe";  
$message = "Hello $name";  
echo $message; //Hello Abebe
```

- ▶ Variables within single quoted strings (') are not parsed!

Example

```
$name = "Abebe";  
$message = 'Hello $name';  
echo $message; //Hello $name
```

```
<?php
$my_str = 'World';
echo "Hello, $my_str!<br>";    // Displays: Hello World!
echo 'Hello, $my_str!<br>';    // Displays: Hello, $my_str!

echo '<pre>Hello\tWorld!</pre>'; // Displays: Hello\tWorld!
echo "<pre>Hello\tWorld!</pre>"; // Displays: Hello  World!
echo 'I\'ll be back';          // Displays: I'll be back
?>
```

The Heredoc string

- starts with <<< symbol followed by any identifier of user's choice. From next line, any multiline sequence of characters that may be having any of the above escape sequences. Last line should have same heredoc identifier ending with semicolon.

//Heredoc assignment of string value to variable

```
public $var = <<< XYZ
```

```
Hello World
```

```
Welcome to PHP
```

```
XYZ;
```

PHP *Nowdoc*

- ▶ The Nowdoc string creation method is similar to the heredoc method but works like the way single quotes work.
- ▶ No parsing takes place inside the Nowdoc.
- ▶ Nowdoc is ideal when working with raw data that do not need to be parsed.
- ▶ The code below shows the Nowdoc implementation

```
<?php
$age = 24;
$my_variable = <<<'EOT'
    Abebe is $age years old.
    He is a Computer Science student.
EOT;
echo $my_variable;
?>
```

String concatenation

- ▶ Use the . (dot) operator

Example

```
<?php
```

```
    $var1 = "Today is ";
```

```
    $message = $var1 . date("d-m-Y"); //Today is 28-  
    12-2007
```

```
?>
```

▶ Indexing strings

- ▶ \$str[index]
- ▶ String index starts with 0 (similar to arrays)

PHP String Functions

- ▶ PHP provides many built-in functions for manipulating strings like calculating the length of a string, find substrings or characters, replacing part of a string with different characters, take a string apart, and many others. Here are the examples of some of these functions.

Calculating the Length of a String

The **strlen()** function is used to calculate the number of characters inside a string. It also includes the blank spaces inside the string.

```
<?php  
$my_str = 'Welcome to Tutorial Republic';
```

```
// Outputs: 28  
echo strlen($my_str);
```

```
?>
```

The **str_word_count()** function counts the number of words in a string.

The **str_replace()** replaces all occurrences of the search text within the target string.

```
<?php
$my_str = 'If the facts do not fit the theory, change the facts.';

// Display replaced string
echo str_replace("facts", "truth", $my_str);
?>
```

The **strrev()** function reverses a string.

```
<?php
$my_str = 'You can do anything, but not
everything.';

// Display reversed string
echo strrev($my_str);
?>
```

Using the strpos() function

- The strpos() function is used to search for a string or character within a string.
- If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

```
<?php  
    echo strpos("Hello world!","world");  
?>
```

For a list of string related functions, refer to the Official PHP Reference Manual

NULL

- NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this –

```
$my_var = NULL;
```

- The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed –

```
$my_var = null;
```

- A variable that has been assigned NULL has the following properties –
 - It evaluates to FALSE in a Boolean context.
 - It returns FALSE when tested with IsSet() function.

PHP Array

- An array is a compound data type. It can store multiple values of same data type in a single variable.

```
<?php
    $bikes = array ("Royal Enfield", "Yamaha", "KTM");
    var_dump($bikes);
//the var_dump() function returns the datatype and values
    echo "</br>";
    echo "Array Element1: $bikes[0] </br>";
    echo "Array Element2: $bikes[1] </br>";
    echo "Array Element3: $bikes[2] </br>";
?>
```

PHP object

- Objects are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.

```

<?php
class Car {
    public $color;
    public $model;
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }
    public function message() {
        return "My car is a " . $this->color . " " . $this->model . "!";
    }
}

$myCar = new Car("black", "Volvo");
echo $myCar -> message();
echo "<br>";
$myCar = new Car("red", "Toyota");
echo $myCar -> message();
?>

echo "<pre>";
var_dump($mycar );
echo "</pre>";

```

Output:

```

object(Car)#1 (2) {
    ["color"]=>
    string(5) "black"
    ["model"]=>
    string(5) "Volvo"
}

```

Resource

- ▶ In PHP, Resource is a special data type that refers to any external resource.
- ▶ A resource variable acts as a reference to external source of data such as stream, file, database etc. PHP uses relevant functions to create these resources.
- ▶ For example, fopen() function opens a disk file and its reference is stored in a resource variable.

```
<?php
```

```
$fp=fopen("test.txt","w");
```

Output

```
var_dump($fp);
```

This will produce following result –

```
?>
```

resource(5) of type (stream)

NULL

- Represents that a variable has no value
- A variable is considered to be **NULL** if
 - it has been assigned the constant **NULL**.
 - it has not been set to any value yet.
 - it has been [unset\(\)](#)

```
<?php
```

```
$x = "Hello world!";
```

```
$x = null;
```

```
var_dump($x);
```

```
?>
```

- ▶ **You can use PHP's built-in function `gettype()` to acquire the type of any variable.**

```
<?php
    $x=10;
    $y=20;
    $sum=$x + $y;
    print "The sum is : $sum";
    echo gettype($sum);           //integer
    $sum =NULL;
    echo gettype($sum); //NULL
```

?>

Changing Type with settype()

```
<?php
$undecided = 3.14;
print gettype( $undecided ); // double
print " -- $undecided<br />"; // 3.14
settype( $undecided, string );
print gettype( $undecided ); // string
print " -- $undecided<br />"; // 3.14
settype( $undecided, int );
?>
```

Type Casting

- ▶ Type casting forces a variable to be used as a certain type. Following script shows example of different type cast operators
 - ▶ The casts allowed are:
 - ▶ (int), (integer) - cast to integer
 - ▶ (bool), (boolean) - cast to boolean
 - ▶ (float), (double), (real) - cast to float
 - ▶ (string) - cast to string
 - ▶ (array) - cast to array
 - ▶ (object) - cast to object
- ```
<?php
$var1=100;
$var2=(boolean)$var1;
$var3=(string)$var1;
$var4=(array)$var1;
$var5=(object)$var1;
var_dump($var2, $var3, $var4, $var5);
?>
```

# PHP Variables Scope

- ▶ In PHP, variables can be declared anywhere in the script.
- ▶ The scope of a variable is the part of the script where the variable can be referenced/used.
- ▶ PHP has three different variable scopes:
  - ▶ local
  - ▶ global
  - ▶ Static



## Local variable

- The variables that are declared within a function are called local variables for that function.
  - These local variables have their scope only in that particular function in which they are declared.
  - This means that these variables cannot be accessed outside the function, as they have local scope.
- 
- A variable declaration outside the function with the same name is completely different from the variable declared inside the function.
  - Let's understand the local variables with the help of an example:

```
<?php
function local_var()
{
 $num = 45; //local variable
 echo "Local variable declared inside the function is: ". $num;
}
local_var();
?>
```

### Output:

Local variable declared inside the function is: 45

```
<?php
function mytest()
{
 $lang = "PHP";
 echo "Web development language: " . $lang;
}
mytest();
//using $lang (local variable) outside the function will generate an error
echo $lang;
?>
```

### Output:

Web development language: PHP

Notice: Undefined variable: lang in D:\xampp\htdocs\program\p3.php on line

Tip: You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

## Global variable

---

- The global variables are the variables that are declared outside the function.
- These variables can be accessed anywhere in the program.
- To access the global variable within a function, use the **GLOBAL** keyword before the variable.
- However, these variables can be directly accessed or used outside the function without any keyword.
- Therefore there is **no need to use any keyword** to access a global variable outside the function.

Let's understand the global variables with the help of an example:

Note: Without using the global keyword, if you try to access a global variable inside the function, it will generate an error that the variable is undefined

```
<?php
 $name = "Hagos Abebe"; //global variable
 function global_var()
 {
 echo "Variable inside the function: ". $name;
 echo "</br>";
 }
 global_var();
?>
```

Output:

**Notice: Undefined variable: name in D:\xampp\htdocs\program\p3.php on line 6**  
**Variable inside the function:**

## Using \$GLOBALS instead of global

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

```
<?php
 $num1 = 5; //global variable
 $num2 = 13; //global variable
 function global_var()
 {
 $sum = $GLOBALS['num1'] + $GLOBALS['num2'];
 echo "Sum of global variables is: " . $sum;
 }
 global_var();
?>
```

**Output:**

**Sum of global variables is: 18**

- If two variables, local and global, have the same name, then the local variable has higher priority than the global variable inside the function.

```
<?php
 $x = 5;
 function mytest()
 {
 $x = 7;
 echo "value of x: " . $x;
 }
 mytest();
?>
```

Output:

Value of x: 7

## PHP The global Keyword

- ▶ The global keyword is used to access a global variable from within a function.
- ▶ To do this, use the global keyword before the variables (inside the function):

```
<?php
$x = 5;
$y = 10;

function myTest() {
 global $x, $y;
 $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

- PHP also stores all global variables in an array called **\$GLOBALS[index]**.
- The index holds the name of the variable.
- This array is also accessible from within functions and can be used to update global variables directly.

```
<?php
$x = 5;
$y = 10;

function myTest() {
 $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
 echo "<p>Variable x inside function is:
 $GLOBALS[x]</p>";
}

myTest();
echo $y; // outputs 15
?>
```

```
<?php
```

```
$name = "Hagos Abebe"; //Global Variable
```

```
function global_var()
```

```
{
```

```
 global $name;
```

```
 echo "Variable inside the function: ". $name;
```

```
 echo "</br>";
```

```
}
```

```
global_var();
```

```
echo "Variable outside the function: ". $name;
```

```
?>
```

### Output:

Variable inside the function: Hagos Abebe

Variable outside the function: Hagos Abebe

## Static variable

---

- It is a feature of PHP to delete the variable, once it completes its execution and memory is freed.
- Sometimes we need to store a variable even after completion of function execution.
- Therefore, another important feature of variable scoping is static variable.
- We use the static keyword before the variable to define a variable, and this variable is called as static variable.
- Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope.
- Understand it with the help of an example

```

<?php
function static_var()
{
 static $num1 = 3; //static variable
 $num2 = 6; //Non-static variable
 //increment in non-static variable
 $num1++;
 //increment in static variable
 $num2++;
 echo "Static: " . $num1 . "</br>";
 echo "Non-static: " . $num2 . "</br>";
}

//first function call
static_var();

//second function call
static_var();
?>

```

**Output:**

**Static: 4**

**Non-static: 7**

**Static: 5**

**Non-static: 7**

---

```
<?php
function myTest() {
 static $x = 0;
 echo $x;
 $x++;
}
```

```
myTest();
myTest();
myTest();
?>
```

- Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

# PHP Global Variables - Superglobals

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

► **The PHP superglobal variables are:**

`$GLOBALS`

`$_SERVER`

`$_REQUEST`

`$_POST`

`$_GET`

`$_FILES`

`$_ENV`

`$_COOKIE`

`$_SESSION`

## ▶ **PHP \$GLOBALS**

- ▶ **\$GLOBALS** is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
  - ▶ PHP stores all global variables in an array called **\$GLOBALS[index]**. The *index* holds the name of the variable.
- 
- ## ▶ **PHP Superglobal - \$\_SERVER**

## PHP \$\_SERVER

► **\$\_SERVER** is a PHP super global variable which holds information about headers, paths, and script locations

| Element/Code                                | Description                                                                                             |
|---------------------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>\$_SERVER['PHP_SELF']</code>          | Returns the filename of the currently executing script                                                  |
| <code>\$_SERVER['GATEWAY_INTERFACE']</code> | Returns the version of the Common Gateway Interface (CGI) the server is using                           |
| <code>\$_SERVER['SERVER_ADDR']</code>       | Returns the IP address of the host server                                                               |
| <code>\$_SERVER['SERVER_NAME']</code>       | Returns the name of the host server (such as <a href="http://www.w3schools.com">www.w3schools.com</a> ) |
| <code>\$_SERVER['SERVER_SOFTWARE']</code>   | Returns the server identification string (such as Apache/2.2.24)                                        |
| <code>\$_SERVER['SERVER_PROTOCOL']</code>   | Returns the name and revision of the information protocol (such as HTTP/1.1)                            |
| <code>\$_SERVER['REQUEST_METHOD']</code>    | Returns the request method used to access the page (such as POST)                                       |
| <code>\$_SERVER['REQUEST_TIME']</code>      | Returns the timestamp of the start of the request (such as 1255685406)                                  |

|                                               |                                                                                                                                                                                                                        |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$_SERVER['QUERY_STRING']</code>        | Returns the query string if the page is accessed via a query string                                                                                                                                                    |
| <code>\$_SERVER['HTTP_ACCEPT']</code>         | Returns the Accept header from the current request                                                                                                                                                                     |
| <code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code> | Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)                                                                                                                                  |
| <code>\$_SERVER['HTTP_HOST']</code>           | Returns the Host header from the current request                                                                                                                                                                       |
| <code>\$_SERVER['HTTP_REFERER']</code>        | Returns the complete URL of the current page (not reliable because not all user-agents support it)                                                                                                                     |
| <code>\$_SERVER['HTTPS']</code>               | Is the script queried through a secure HTTP protocol                                                                                                                                                                   |
| <code>\$_SERVER['REMOTE_ADDR']</code>         | Returns the IP address from where the user is viewing the current page                                                                                                                                                 |
| <code>\$_SERVER['REMOTE_HOST']</code>         | Returns the Host name from where the user is viewing the current page                                                                                                                                                  |
| <code>\$_SERVER['REMOTE_PORT']</code>         | Returns the port being used on the user's machine to communicate with the web server                                                                                                                                   |
| <code>\$_SERVER['SCRIPT_FILENAME']</code>     | Returns the absolute pathname of the currently executing script                                                                                                                                                        |
| <code>\$_SERVER['SERVER_ADMIN']</code>        | Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com) |
| <code>\$_SERVER['SERVER_PORT']</code>         | Returns the port on the server machine being used by the web server for communication (such as 80)                                                                                                                     |

---

► Example:

```
<?php
echo $_SERVER['PHP_SELF'];
echo "
";
echo $_SERVER['SERVER_NAME'];
echo "
";
echo $_SERVER['HTTP_HOST'];
echo "
";
echo $_SERVER['HTTP_REFERER'];
echo "
";
echo $_SERVER['HTTP_USER_AGENT'];
echo "
";
echo $_SERVER['SCRIPT_NAME'];
?>
```

## ► **PHP Superglobal - \$\_REQUEST**

- PHP \$\_REQUEST is a PHP super global variable which is used to collect data after submitting an HTML form.

```
► <html>
 <body>
 <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
 Name: <input type="text" name="fname">
 <input type="submit">
 </form>
 <?php
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
 // collect value of input field
 $name = $_REQUEST['fname'];
 if (empty($name)) {
 echo "Name is empty";
 } else {
 echo $name;
 }
 }
 } ?></body></html>
```

►

## ► PHP \$\_POST

- PHP \$\_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.

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?
>">
 Name: <input type="text" name="fname">
 <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
 // collect value of input field
 $name = $_POST['fname'];
 if (empty($name)) {
 echo "Name is empty";
 } else {
 echo $name;
 }
}
} 69
?> </body> </html>
```

# Case Sensitivity in PHP

---

- Variable names in PHP are case-sensitive. As a result the variables \$color, \$Color and \$COLOR are treated as three different variables.
- However the keywords, function and classes names are case-insensitive. As a result calling the gettype() or GETTYPE() produce the same result.

```
<?php
// Assign value to variable
$color = "blue";

// Get the type of a variable
echo gettype($color) . "
";
echo GETTYPE($color) . "
";
?>
```

---

## ► Example

```
<?php
 $var = 'Bob';
 $Var = 'Joe';
 echo "$var, $Var"; // outputs "Bob, Joe"
 $4site = 'not yet'; // invalid; starts with a number
 $_4site = 'not yet'; //valid; starts with an underscore
 $täyte = 'mansikka'; // valid; 'ä' is (Extended) ASCII 228.
?>
```

# Referencing

---

```
$var1 = "some string";
$var2 = &$var1;
$var2 = "another string"
```

```
echo $var1; //another string
echo $var2; //another string
```

## PHP Constants

- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no \$ sign before the constant name).
- Note: Unlike variables, constants are automatically global across the entire script.

## Create a PHP Constant

---

- To create a constant, use the `define()` function.

### Syntax

**`define(name, value)`**

Parameters:

name: Specifies the name of the constant

value: Specifies the value of the constant

```
<?php
```

---

```
define("SITE_URL", "https://www.mysite.com/");
// Using constant
echo 'Thank you for visiting - ' . SITE_URL;
```

```
define("PI",3.14);
echo "The value of PI is ",PI;
?>
```

- ▶ **PHP Magic constants**
- ▶ PHP provides a large number of predefined constants to any script which it runs.

`__LINE__`

The current line number of the file.

`__FILE__`

The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, `__FILE__` always contains an absolute path whereas in older versions it contained relative path under some circumstances.

`__FUNCTION__`

The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.

## \_\_CLASS\_\_

---

The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.

## \_\_METHOD\_\_

The class method name. (Added in PHP 5.0.0)  
The method name is returned as it was declared (case-sensitive)

# Array variables

## ► Types of Arrays in PHP

---

- There are three types of arrays that you can create. These are:
  - **Indexed array** — An array with a numeric key.
  - **Associative array** — An array where each key has its own specific value.
  - **Multidimensional array** — An array containing one or more arrays within itself.

# Indexed Arrays

- ▶ These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.
- ▶ There are two ways to define indexed array:

## Method 1

- ▶ **Using array()** function

```
<?php
```

```
/* First method to create array. */
```

```
$numbers = array(1, 2, 3, 4, 5);
```

```
foreach($numbers as $value) {
 echo "Value is $value
";
}
```

## Method 2: Using a short array syntax

---

```
<?php
$array = array(bar", "foo",);

// Using the short array syntax
$array = ["bar", "foo"];
?>
```

# Dumping an Array

---

The function `print_r()` dumps out PHP data - it is used mostly for debugging

Syntax: `print_r($arr);`

```
<?php
$b=array (10, 20, 30) ;
echo "
";
print_r ($b) ;
?>
```

```
Array ([0] => 10 [1] => 20 [2] => 30)
```



---

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " .
$cars[1] . " and " . $cars[2] . ".";
?>
```

## Method 3:

### ► Using square-bracket syntax

```
$size=array("Big","Medium","Short");
```

```
/* Third method to create array. */
```

```
$size[0]="Big";
```

```
$size[1]="Medium";
```

```
$size[2]="Short";
```

```
/* Third method to create array. */
```

```
$numbers[0] = "one";
```

```
$numbers[1] = "two";
```

```
$numbers[2] = "three";
```

```
$numbers[3] = "four";
```

```
$numbers[4] = "five";
```

<?php

---

```
$ages=[20,30,40];
```

```
echo $ages[0];
```

```
ages[]=50;
```

```
Ages[1]=60;
```

```
Print_r($ages);
```

?>

- 
- ▶ Adding a new element to an indexed array using **array\_push()** function:

```
<?php
$ages=[10,20,30];
array_push($ages,70);
$ages[]=80;
$ages[5]=90;
print_r($ages);
?>
```

- ▶ Counting the number of array elements using **count()** function.
- 

```
<?php
```

```
$ages=[10,20,30];
```

```
array_push($ages,70);
```

```
$ages[]=80;
```

```
$ages[5]=90;
```

```
echo "The no of array elements is :".
```

```
count($ages);
```

```
?>
```

- ▶ **Merging elements of two arrays to another array using array\_merge() function.**

```
<?php
```

```
$array_one=[1,2,3];
```

```
$array_two=[4,5];
```

```
$array_three=array_merge($array_one,
$array_two);
```

```
print_r($array_three);
```

```
?>
```

# Associative Arrays

- ▶ The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index.
- ▶ Associative array will have their index as string so that you can establish a strong association between key and values.
- ▶ There are two ways to create an associative array:

Method 1:

**Syntax: `array( [ key => ] value, ... )`**

**key : can be an integer or a string**

**value : can be any value**

```
$age = array("Peter"=>"35", "Ben"=>"37",
"Joe"=>"43");
```

## Method 2:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

```
<?php
$age
= array("Peter"=>"35", "Ben"=>"37", "Joe"=>
"43");
echo "Peter is " . $age['Peter'] . " years
old."
?>
```

## Creating associative array using [ ]

```
<?php
$age =["Peter"=>"35",
"Ben"=>"37", "Joe"=>"43"];
//accessing array element using key
echo $age["Peter"];
//adding a new element
age["Jone"]="55";
Print_r($age);

?>
```

## Loop Through an Associative Array

---

•To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```
<?php
$page
= array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($page as $x => $x_value) {
 echo "Key=" . $x . ", Value=" . $x_value;
 echo "
";
}
?>
```

## Example

```
<?php
 $arr = array("foo" => "bar", 12 => true);

 echo $arr["foo"]; // bar
 echo $arr[12]; // 1
?>
```

- ▶ Floats in key are truncated to integer.
- ▶ There are no different indexed and associative array types in PHP; there is only one array type, which can both contain integer and string indices.

- ▶ The value in an array can be anything (even an array)
- 

## Example

```
<?php
 $arr = array("somearray" => array(6 => 5, 13 =>
9, "a" => 42));

 echo $arr["somearray"][6]; // 5
echo $arr["somearray"][13]; // 9
echo $arr["somearray"]["a"]; // 42
?>
```

- ▶ If you do not specify a key for a given value, then the maximum of the integer indices is taken, and the new key will be that maximum value + 1.
  - ▶ If no integer indices exist yet, the key will be 0 (zero).
- ▶ If you specify a key that already has a value assigned to it, that value will be overwritten.

Example

```
<?php
// This array is the same as ...
array(5 => 43, 32, 56, "b" => 12);

// ...this array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

## ► **Creating/modifying with square-bracket syntax**

---

- You can also modify an existing array by explicitly setting values in it

### Syntax

```
$arr[key] = value;
```

```
$arr[] = value;
```

```
// key may be an integer or string
```

```
// value may be any value
```

- If `$arr` doesn't exist yet, it will be created
- To change a certain value, just assign a new value to an element specified with its key.
- If you want to remove a key/value pair, you need to `unset()` it

---

## Example

```
<?php
```

```
$arr = array(5 => 1, 12 => 2);
```

```
$arr[] = 56; // same as $arr[13] = 56;
```

```
$arr["x"] = 42; // adds a new element to the array
with key "x"
```

```
unset($arr[5]); // removes the element from the
array
```

```
unset($arr); // This deletes the whole array
?>
```

- ▶ `count()` function can also return the number of array elements in associative array.
- ▶ `array_merge()` can also be used to merge two associative arrays.

## Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

Example:

<b>Name</b>	<b>Stock</b>	<b>Sold</b>
<b>Volvo</b>	<b>22</b>	<b>18</b>
<b>BMW</b>	<b>15</b>	<b>13</b>
<b>Saab</b>	<b>5</b>	<b>2</b>
<b>Land Rover</b>	<b>17</b>	<b>15</b>

```
<?php
```

```
$cars = array (
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
);
```

```
print_r($cars);
//Using square bracket
```

```
$cars = [
 ["Volvo",22,18],
 ["BMW",15,1],
 ["Saab",5,2],
 ["Land Rover",17,15]
];
print_r($cars);
echo $cars[1][0];
```

```
<?php
$marks=[

"Abebe"=>["Physics"=>89,"Chemistry"=>90,"Biology"=>98],

"Aster"=>["Physics"=>88,"Chemistry"=>93,"Biology"=>95]
];

echo $marks["Abebe"]["Physics"];

//add a new element
$marks[]=["Alamz"=>["Physics"=>88,"Chemistry"=>92,"Biology"=>88]];

print_r($marks);
?>
```

# Array Functions

- `count($ar)` - How many elements in an array
- `is_array($ar)` - Returns TRUE if a variable is an array
- `isset($ar['key'])` - Returns TRUE if key is set in the array
- `sort($ar)` - Sorts the array values (loses key)
- `ksort($ar)` - Sorts the array by key
- `asort($ar)` - Sorts array by value, keeping key association
- `array_pop()`: Removes
- `shuffle($ar)`
- PHP `array()` function creates and returns an array. It allows you to create indexed, associative and multidimensional arrays.

## array\_pop() Function

- This inbuilt function of PHP is used to delete or pop out and return the last element from an **array** passed to it as parameter.
- It reduces the size of the **array** by one since the last element is removed

### Syntax:

`array_pop($arr);`

**Return Value:** This function returns the last element of the array.

If the array is empty or the input parameter is not an array, then NULL is returned.

//Example demonstrating count and isset functions

```
$arr = array();
$arr["name"] = "Kebede";
$arr["course"] = "PHP Intro";
print "Count: " . count($arr) . "\n";
if (is_array($arr)) {
 echo '$arr Is an array' . "\n";
} else {
 echo '$arr Is not an array' . "\n";
}
```

```
echo isset($arr['name']) ? "name is set\n" : "name is not set\n";
echo isset($arr['addr']) ? "addr is set\n" : "addr is not set\n";
```

## Sorting functions:

sort() and rsort() — For sorting indexed arrays

asort() and arsort() — For sorting associative arrays by value

ksort() and krsort() — For sorting associative arrays by key

```
<?php
// Define array
$age = array("Peter"=>20, "Harry"=>14, "John"=>45,
"Clark"=>35);
// Sorting array by key in ascending order and print
// for Sorting array by key in descending order , use
the function arsort() or krsort()
```

```
ksort($age);
print_r($age); ?>
```

# explode() function:

**explode()** is a built in function in PHP used to split a string in different strings.  
The **explode()** function splits a string based on a string delimiter, i.e. it splits the string wherever the delimiter character occurs.  
This function returns an array containing the strings formed by splitting the original string

\$set  
\$var  
pri **Syntax :**

***array explode(separator, OriginalString,  
NoOfElements)***

```
array(
 [0] => This
 [1] => is
 [2] => a
 [3] => senten
 [4] => with
 [5] => seven
 [6] => words
)
```

# PHP echo and print Statements

---

- echo and print are more or less the same.
- They are both used to output data to the screen.
- The differences are small:
- echo has **no return value** while
- print has a **return value of 1** so it can be used in expressions.
- echo can take **multiple parameters** (although such usage is rare) while print can **take one argument**.
- echo is marginally faster than print.

## The PHP echo Statement

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

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!
";
echo "I'm about to learn PHP!
";
echo "This ", "string ", "was ", "made ",
"with multiple parameters.";
```

## PHP Math

- ▶ PHP has a set of math functions that allows you to perform mathematical tasks on numbers.

### PHP pi() Function

The pi() function returns the value of PI:

```
<?php
echo(pi()); // returns 3.1415926535898
?>
```

**pow()** It is used to return x raised to the power of y.

**sqrt()** It is used to return the square root of a number.

**is\_nan()** It is used to check whether a value is 'not-a-number'

## PHP min() and max() Functions

The min() and max() functions can be used to find the lowest or highest value in a list of arguments:

```
<?php
echo(min(0, 150, 30, 20, -8, -200)); // returns -200
echo(max(0, 150, 30, 20, -8, -200)); // returns 150
?>
```

## Other maths functions:

---

`abs(-5);` // computes the absolute value of -5

`echo ceil(9.99);` // Outputs: 10

`echo ceil(-5.18);` // Outputs: -5

`echo floor(9.99);` // Outputs: 9

`echo floor(-5.18);` // Outputs: -6

`echo sqrt(10);` // Outputs: 3.1622776601684

`echo sqrt(-16);` // Outputs: NAN

## ► Generate a Random Number

---

- The rand() function can be used to generate a random number.
- You can optionally specify a range by passing the min, max arguments, as shown in the following example:

```
<?php // Generate some random numbers
echo rand() . "
";
echo rand() . "
"; // Generate some random numbers
between 1 and 10 (inclusive)
echo rand(1, 10) . "
";
echo rand(1, 10) . "
";
?>
```

---

For a complete list of php maths function, you can visit the site [www.php.net](http://www.php.net).

# PHP Operators

- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators
  - Conditional assignment operators

## PHP Arithmetic Operators

---

Operator	Name	Example	Result
+	Addition	$\$x + \$y$	Sum of $\$x$ and $\$y$
-	Subtraction	$\$x - \$y$	Difference of $\$x$ and $\$y$
*	Multiplication	$\$x * \$y$	Product of $\$x$ and $\$y$
/	Division	$\$x / \$y$	Quotient of $\$x$ and $\$y$
%	Modulus	$\$x \% \$y$	Remainder of $\$x$ divided by $\$y$
**	Exponentiation	$\$x ** \$y$	Result of raising $\$x$ to the $\$y$ 'th power

# PHP Assignment Operators

---

## Assignment

`x = y`  
right

## Same as...

`x = y`

The left operand gets set to the value of the expression on the

## Description

`x += y`

`x = x + y`

Addition

`x -= y`

`x = x - y`

Subtraction

`x *= y`

`x = x * y`

Multiplication

`x /= y`

`x = x / y`

Division

`x %= y`

`x = x % y`

Modulus

# PHP Comparison Operators

---

Operator	Name	Example	Result
==	Equal	<code>\$x == \$y</code>	Returns true if \$x is equal to \$y
===	Identical	<code>\$x === \$y</code>	Returns true if \$x is equal to \$y, and they are of the same type
!=	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
<>	Not equal	<code>\$x &lt;&gt; \$y</code>	Returns true if \$x is not equal to \$y
!==	Not identical	<code>\$x !== \$y</code>	Returns true if \$x is not equal to \$y, or they are not of the same type
>	Greater than	<code>\$x &gt; \$y</code>	Returns true if \$x is greater than \$y
<	Less than	<code>\$x &lt; \$y</code>	Returns true if \$x is less than \$y
>=	Greater than or equal to	<code>\$x &gt;= \$y</code>	Returns true if \$x is greater than or equal to \$y
<=	Less than or equal to	<code>\$x &lt;= \$y</code>	Returns true if \$x is less than or equal to \$y

## Spaceship Operator

- ▶ In PHP 7, a new feature, spaceship operator has been introduced. It is used to compare two expressions.
- ▶ It returns -1, 0 or 1 when first expression is respectively less than, equal to, or greater than second expression.

```
<?php
//integer comparison
print(1 <=> 1);print("
");
print(1 <=> 2);print("
");
print(2 <=> 1);print("
");
print("
");
//float comparison
print(1.5 <=> 1.5);print("
");
print(1.5 <=> 2.5);print("
");
print(2.5 <=> 1.5);print("
");
print("
");
//string comparison
print("a" <=> "a");print("
");
print("a" <=> "b");print("
");
print("b" <=> "a");print("
");
```

It produces the following browser output –

```
0
-1
1

0
-1
1

0
-1
1
```

# PHP Increment / Decrement Operators

---

Operator	Name	Description
++\$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

```
<?php
```

```
$x=8;
```

```
$y=9;
```

```
$z=++$x + ++$y;
```

```
echo "
". $z . "</br>"; $z=18
```

```
echo "
". $x . "</br>"; $x=9
```

```
echo "
". $y . "</br>"; $y=10
```

```
?>
```

## PHP Logical Operators

---

Operator	Name	Example	Result
and	And	\$x and \$y	True if both \$x and \$y are true
or	Or	\$x or \$y	True if either \$x or \$y is true
xor	Xor	\$x xor \$y	True if either \$x or \$y is true, but not both
&&	And	\$x && \$y	True if both \$x and \$y are true
	Or	\$x    \$y	True if either \$x or \$y is true
!	Not	!\$x	True if \$x is not true

# PHP String Operators

---

Operator	Name	Example	Result
.	Concatenation	\$txt1 . \$txt2	Concatenation of \$txt1 and \$txt2
.=	Concatenation assignment	\$txt1 .= \$txt2	Appends \$txt2 to \$txt1

# PHP Conditional Statements

---

There are several statements in PHP that you can use to make decisions:

- The if statement

- The if...else statement

- The if...elseif....else statement

- The switch...case statement

# The if Statement

---

```
if(condition){
 // Code to be executed
}
```

```
<?php
$d = date("D");
if($d == "Fri"){
 echo "Have a nice weekend!";
}
?>
```

# The if...else Statement

---

```
if(condition){
 // Code to be executed if condition is true
} else{
 // Code to be executed if condition is
false
}
```

```
<?php
$d = date("D");
if($d == "Fri"){
 echo "Have a nice weekend!";
} else{
 echo "Have a nice day!";
}
?>
```

# The if...elseif...else Statement

---

The if...elseif...else is a special statement that is used to combine multiple if...else statements.

```
if(condition1){
 // Code to be executed if condition1 is true
} elseif(condition2){
 // Code to be executed if the condition1 is false
 and condition2 is true
} else{
 // Code to be executed if both condition1 and
 condition2 are false
}
```

```
<?php
$d = date("D");
if($d == "Fri"){
 echo "Have a nice weekend!";
} elseif($d == "Sun"){
 echo "Have a nice Sunday!";
} else{
 echo "Have a nice day!";
}
?>
```

```
<html><head></head>
<body>
```

---

```
<?php
$d=date("D");
echo $d, "
";
if ($d=="Fri")
 echo "Have a nice weekend!
";
else
 echo "Have a nice day!
";
```

```
$x=10;
if ($x==10)
{
 echo "Hello
";
 echo "Good morning
";
}
```

```
?>
```

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

# The Ternary Operator

---

`( condition ) ? True_value : False_value`

Example:

```
...
<?php
 $year = (int)date("Y"); //var_dump($year)
 echo ($year % 4 == 0) ? "Leap Year" : "Not Leap
 Year";
?>
...
```

```
<?php
if($age < 18){
 echo 'Child'; // Display Child if age is less than 18
} else{
 echo 'Adult'; // Display Adult if age is greater than or equal to 18
}
?>
```

```
<?php echo ($age < 18) ? 'Child' : 'Adult'; ?
>
```

# Control structures (cont'd)

---

## ► **Switch**

```
switch (expression){
 case value 1:
 statements
 break;
 ...
 case value n:
 statements
 break;
 default:
 statements
}
```

```
<?php
```

```
$favcolor = "red";
```

---

```
switch ($favcolor) {
```

```
 case "red":
```

```
 echo "Your favorite color is red!";
```

```
 break;
```

```
 case "blue":
```

```
 echo "Your favorite color is blue!";
```

```
 break;
```

```
 case "green":
```

```
 echo "Your favorite color is green!";
```

```
 break;
```

```
 default:
```

```
 echo "Your favorite color is neither red,
blue, nor green!";
```

```
}
```

```
?>
```

```
<html><head></head>
<body>
<?php
$x = rand(1,5); // random integer
echo "x = $x

";
switch ($x)
{
case 1:
 echo "Number 1";
 break;
case 2:
 echo "Number 2";
 break;
case 3:
 echo "Number 3";
 break;
default:
 echo "No number between 1 and 3";
 break;
?>
</body>
</html>
```

## ► Looping constructs

---

In PHP, we have the following loop types:

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

## For loop

```
for(initialization; condition; increment){
 loop body
}
```

---

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

```
<?php
for ($x = 0; $x <= 100; $x+=10) {
 echo "The number is: $x
";
}
?>
```

# Foreach loop

---

- The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to \$value and the array pointer is moved by one and in the next pass next element will be processed.

## Syntax

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

```
<html>
 <body>

 <?php
 $array = array(1, 2, 3, 4, 5);

 foreach($array as $value) {
 echo "Value is $value
";
 }
 ?>

 </body>
</html>
```

---

► Example

...

```
$arr = array("name"=>"Abebe", "dept"=>"CS",
 "year"=>3, "cgpa"=>3.5);
foreach($arr as $key=>$value){
 echo $key . " = " . $value . "
";
}
```

//output

name = Abebe

dept = CS

year = 3

cgpa = 3.5

More examples:

```
<?php
for ($i=1; $i<=5; $i++)
{
echo "Hello World!
";
}
?>
```

```
<?php
$a_array = array(1, 2, 3, 4);
foreach ($a_array as $value)
{
 $value = $value * 2;
 echo "$value
 \n";
}
?>
```

```
<?php
```

---

```
$a_array=array("a","b","c");
foreach ($a_array as $key => $value)
{
 echo $key." = ".$value."\n";
}
?>
```

## While loop

---

```
while(condition){
loop body
}
```

```
<?php
$x = 1;
```

```
while($x <= 5) {
 echo "The number is: $x
";
 $x++;
}
?>
```

# Do-while loop

---

```
do{
 loop body
}while(condition);
```

```
<?php
$i = 1;
do{
 $i++;
 echo "The number is " . $i . "
";
}
while($i <= 3);
?>
```

## More examples;

---

```
<?php
$count=0;
while($count<3)
{
 Print "hello PHP. ";
 $count += 1;
 // $count = $count + 1;
 // or
 // $count++;
?>
```

- ▶ break
  - ▶ ends execution of the current for, foreach, while, do-while or switch structure.

---

- ▶ accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.
- ▶ continue
  - ▶ used within looping structures to skip the rest of the current loop iteration
  - ▶ execution continues at the condition evaluation and then the beginning of the next iteration
- ▶ return
  - ▶ If called from within a function, the return statement immediately ends execution of the current function, and returns its argument as the value of the function call
  - ▶ If return is called from within the main script file, then script execution ends.

```
<html>
<body>

<?php
 $i = 0;

 while($i < 10) {
 $i++;
 if($i == 3)break;
 }
 echo ("Loop stopped at i = $i");
?>

</body>
</html>
```

```
<html>
<body>

<?php
 $array = array(1, 2, 3, 4, 5);

 foreach($array as $value) {
 if($value == 3)continue;
 echo "Value is $value
";
 }
?>

</body>
</html>
```

---

- ▶ **File Inclusion**

- ▶ You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to include one PHP file into another PHP file.

- ▶ The include() Function

- ▶ The require() Function

- ▶ helps in creating functions, headers, footers, or elements that can be reused on multiple pages.

## include / require

- The include() and require() statement allow you to include the code contained in a PHP file within another PHP file.
- Including a file produces the same result as copying the script from the file specified and pasted into the location where it is called.
- A typical example is including the header, footer and menu file within all the pages of a website.

The basic syntax of the include() and require() statement can be given with:

```
include("path/to/filename");
-Or- include "path/to/filename";
require("path/to/filename");
-Or- require "path/to/filename";
```

- ▶ The following example will show you how to include the common header, footer and menu codes which are stored in separate 'header.php', 'footer.php' and 'menu.php' files respectively,

```
<!DOCTYPE html>
<html lang="en">
 <head>
 <title>Tutorial Republic</title>
 </head>
 <body>
 <?php include "header.php"; ?>
 <?php include "menu.php"; ?>
 <h1>Welcome to Our Website!</h1>
 <p>Here you will find lots of useful information.</p>
 <?php include "footer.php"; ?> </body> </html>
```

## Difference Between include and require Statements

---

- You might be thinking if we can include files using the `include()` statement then why we need `require()`.
- Typically the `require()` statement operates like `include()`.
- The only difference is — the `include()` statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the `require()` statement will generate a fatal error and stops the script execution.

## The `include_once` and `require_once` Statements

- If you accidentally include the same file (typically functions or classes files) more than one time within your code using the `include` or `require` statements, it may cause conflicts.
- To prevent this situation, PHP provides `include_once` and `require_once` statements.
- These statements behave in the same way as `include` and `require` statements with one exception.
- The `include_once` and `require_once` statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again.

```
<?php function
multiplySelf($var){ $var *=
$var; // multiply variable by
itself echo $var; }
?>
```

'my\_functions.php' file.

```
<?php
// Including file
require "my_functions.php"; // Calling the
function multiplySelf(2); // Output: 4
echo "
";
// Including file once again
require "my_functions.php"; // Calling the
function multiplySelf(5); // Doesn't execute
?>
```

- When you run the above script, you will see the error message something like this: "Fatal error: Cannot redeclare multiplySelf()". This occurs because the 'my\_functions.php' included twice, this means the function multiplySelf() is defined twice, which caused PHP to stop script execution and generate fatal error. Now rewrite the above example with require\_once.

```
<?php
```

```

// Including file
require_once "my_functions.php";
// Calling the function multiplySelf(2);
// Output: 4 echo "
";
// Including file once again
require_once "my_functions.php";
// Calling the function
```

```

multiplySelf(5); // Output: 25
?>
```

- As you can see, by using `require_once` instead of `require`, the script works as we expected.

# Example

---

vars.php

<?php

```
$color = 'green';
$fruit = 'apple';
```

?>

test.php

<?php

```
echo "A $color $fruit"; // A
```

```
include 'vars.php';
```

```
echo "A $color $fruit"; // A green apple
```

?>

## More examples:

include “opendb.php”;

include “closedb.php”;

- ▶ This inserts files; the code in files will be inserted into current code.
- ▶ This will provide useful and protective means once you connect to a database, as well as for other repeated functions.

Include (“footer.php”);

The file footer.php might look like:

```
<hr SIZE=11 NOSHADE WIDTH=“100%”>
```

```
<i>Copyright © 2001-2012 gsu</i>

```

```
<i>ALL RIGHTS RESERVED</i>

```

```
<i>URL: http://www.gsu.edu.edu</i>

```



## PHP functions

---

► A function is a self-contained block of code that performs a specific task.

### ► Why Functions?

- Functions reduces the repetition of code within a program
- Functions makes the code much easier to maintain
- Functions makes it easier to eliminate the errors
- Functions can be reused in other application

### • PHP Built-in Functions

- like `gettype()`, `print_r()`, `var_dump`, etc.
- See PHP office site for a complete list of useful PHP built-in functions.

### • PHP User-Defined Functions

- PHP also allows you to define your own functions.
- It is a way to create reusable code packages that perform specific tasks and can be kept and maintained separately from main program.



# Built-In Functions...

---

- ▶ Much of the power of PHP comes from its built-in functions

```
echo strrev(" .dlrow olleH");
echo str_repeat("Hip ", 2);
echo strtoupper("hooray!");
echo "\n";
```

**Hello world.  
Hip Hip  
HOORAY!**



# PHP Documentation - Google

The screenshot shows a web browser with two tabs. The first tab, titled 'php string replace function - Google Search', shows a Google search results page for the query 'php string replace function'. The second tab, titled 'PHP: str\_replace - Manual', shows the official PHP Manual page for the `str_replace` function. The browser's address bar in the second tab displays the URL `http://php.net/manual/en/function.str-replace.php`. The PHP Manual page features a navigation menu on the left with links to 'PHP Manual', 'Function Reference', 'Text Processing', 'Strings', and 'String Functions'. The main content area of the manual page includes a search bar, a list of related functions like `str_repeat` and `str_rot13`, and a detailed description of the `str_replace` function. The description states: 'str\_replace — Replace all occurrences of the search string with the replacement string'. It also provides the function signature: `mixed str_replace ( mixed $search , mixed $replace , mixed $subject [, int &$count ] )`. A 'Report a bug' link is visible next to the description.

php string replace function - Google Search

http://www.google.com/search?client=safari&rls=en&q=php+string+replace+function&ie=UTF-8

php string replace functi

http://localhost/si572/php-05/f... PHP: str\_replace - Manual

http://php.net/manual/en/function.str-replace.php

php string replace functi

Google

Search

Everything

Images

Maps

Videos

News

Shopping

More

PHP Manual

Function Reference

Text Processing

Strings

String Functions

addslashes

addslashes

bin2hex

chop

chr

chunk\_split

convert\_cyr\_string

convert\_uudecode

convert\_uuencode

«str\_repeat str\_rot13»

view this page in Brazilian Portuguese

[edit] Last updated: Fri, 16 Sep 2011

**str\_replace**

(PHP 4, PHP 5)

str\_replace — Replace all occurrences of the search string with the replacement string

**Description** [Report a bug](#)

`mixed str_replace ( mixed $search , mixed $replace , mixed $subject [, int &$count ] )`

This function returns a string or an array with all occurrences of *search* in *subject* replaced with the given *replace* value.



- 
- ▶ PHP is a very configurable system and has lots of capabilities that can be plugged in.
  - ▶ The `phpinfo()` function prints out the internal configuration capabilities of your particular PHP installation

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



# Defining Your Own Functions

---

- ▶ We use the function keyword to define a **function**, we **name the function** and take **optional argument variables**. The body of the function is in a block of code **{ }**
- ▶ User defined functions may be declared as follows:

```
<?php
function function_name($arg_1, $arg_2, ..., $arg_n)
{
 function body
 [return $retval;]
}
?>
```



# Return Values

---

- ▶ Often a function will take its arguments, do some computation and **return** a value to be used as the value of the function call in the calling expression. The **return** keyword is used for this.

```
function greeting() {
 return "Hello";
}
```

```
print greeting() . "Abeb\n"; Hello Abebe
print greeting() . " Aster\n"; Hello Aster
```



# Arguments

---

- Functions can choose to accept optional **arguments**. Within the function definition the variable names are effectively "**aliases**" to the **values passed in** when the function is called.

```
function sayHi($lang) {
 if ($lang == 'amh')
 return "Selam";
 if ($lang == 'en')
 return "Hi";
 return "Hello";
}
```

Selam Aster  
Hi Evryone

```
print sayHi('amh') . " Aster\n";
print sayHi('en') . " Everyone\n";
```

---



## More examples on returning a value:

```
<?php
```

```
// returns true if $str begins with $sub
function beginsWith($str, $sub) {
 return (substr($str, 0,
 strlen($sub)) == $sub);
}
```

```
<?php
```

```
// returns true if $str ends with $sub
function endsWith($str, $sub) {
 return (substr($str,
 strlen($str)-strlen($sub)) ==
 $sub);
}
```

# Optional Arguments

---

- Arguments can have defaults and so can be omitted

**// functions.php**

**<?php**

```
function sayHi($lang="amh") {
 if($lang=="amh")
 return "Selam ";
 if($lang=="en")
 return "Hi";
}
```

**?>**

```
<?php
include "functions.php";
echo sayHi(). " Aster
";
echo sayHi(). "everyone
";
?>
```



- ▶ Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

## Example

```
<?php
function makeyogurt($type = "acidophilus",
$flavour)
{
 return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt("raspberry"); // won't work,
error
?>
```

- ▶ To define an argument with a default value, the default value can be assigned to the argument in the function definition.

```
function sample($arg1, $arg2 = "initial value"){
 ...
}
```

when calling:

```
sample($var1, $var2);
sample($var1);
 //second argument takes default value
```

# Choosing Function Names

---

- ▶ Much like variable names - but do not start with a dollar sign
- ▶ Start with a letter or underscore - consist of letters, numbers, and underscores ( \_ )
- ▶ Avoid built in function names



# Call By Value

---

- ▶ The argument variable within the function is an "alias" to the actual variable
- ▶ But even further, the alias is to a \*copy\* of the actual variable in the function call

```
function double($alias) {
 $alias = $alias * 2;
 return $alias;
}
```

```
$val = 10;
```

Value = 10 Doubled = 20

```
$dval = double($val);
```

```
echo "Value = $val Doubled = $dval\n";
```



# Call By Reference

---

- ▶ Sometimes we want a function to change one of its arguments - so we indicate that an argument is "by reference" using ( & )

```
function triple(&$realthing) {
 $realthing = $realthing * 3;
}
```

```
$val = 10;
triple($val);
echo "Triple = $val\n";
```

Triple = 30

---



- ▶ To pass an argument to a function by reference, you can prepend an ampersand (&) to the argument name in the function definition.

## Examples

1)

```
<?php
function add_five(&$value) {
 $value += 5;
}
```

```
$num = 2;
add_five($num);
echo $num;
```

```
?>
```

...

---

```
<?php
function getDate(&$arg1){
 $arg1 = date("d-m-Y");
}
getDate($date);
echo $date;

?>
```

# Conditional functions

---

```
<?php
 $defined= true;

 // We can't call f1() from here since it doesn't exist yet, but we can
 call f2()

 f2();

 if ($defined) {
 function f1() {
 echo "I don't exist until program execution reaches me.\n";
 }
 }

 // Now we can safely call f1() since $defined evaluated to true

 if ($defined) f1();

 function f2() {
 echo "I exist immediately upon program start.\n";
 }
?>
```

# Functions within functions

```
<?php
 function f1(){
 function f2(){
 echo "I don't exist until f1() is called.\n";
 }
 }

 /* We can't call f2() yet since it doesn't exist. */

 f1();

 /* Now we can call f2(), f1()'s processing has made it accessible.
 */

 f2();
?>
```

- ▶ All functions in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.
- ▶ PHP does not support function overloading.

## Recursive functions

A recursive function is a function that calls itself again and again until a condition is satisfied.

Example

```
<?php
function recursion($a){
 if ($a < 20) {
 echo "$a \r\n";
 recursion($a + 1);
 }
}
?>
```

# Returning values

---

- ▶ Values are returned by using the optional return statement.
- ▶ Any type may be returned, including lists and objects.
- ▶ This causes the function to end its execution immediately and pass control back to the line from which it was called.
- ▶ To return a reference from a function, you have to use the reference operator & in both the function declaration and when assigning the returned value to a variable

# Returning values

---

## Example

```
<?php
function &returns_reference(){
 return $someref;
}

$newref =& returns_reference();
?>
```

## ► Redirect to another page

---

Use the PHP header() Function

```
<?php
header("Location: http://www.google.com/");
/* Redirect browser */
```

```
/* Make sure that code below does not get
executed when we redirect. */
```

```
exit;
```

```
?>
```

- Remember that header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

# Form Handling

**Any form element is automatically available via one of the built-in PHP variables (provided that element has a “name” defined with it).**

---

```
<html>
<-- form.html -->
<body>
<form action="welcome.php" method="POST">
Enter your name: <input type="text" name="name" />

Enter your age: <input type="text" name="age" />

<input type="submit" /> <input type="reset" />
</form>
</body>
</html>
```

```
<html>
<!-- welcome.php -->
<body>

Welcome <?php echo $_POST["name"].".".""; ?>

You are <?php echo $_POST["age"]; ?> years old!

</body>
</html>
```

**\$\_POST**  
contains all POST data.

**\$\_GET**  
contains all GET data.



# Input

---

- ▶ Input to server side scripts comes from clients through forms.
- ▶ Two methods of sending data: GET & POST
  - ▶ GET
    - ▶ Search queries and small amounts of data
    - ▶ Also generated when a user clicks on a link
    - ▶ Non secure (displayed in address bar)
  - ▶ POST
    - ▶ Large and secure data
- ▶ The default method for HTML forms is GET

## Input (cont'd)

---

- ▶ To access form field values in PHP, use the built-in PHP arrays: `$_GET` and `$_POST` respectively for GET and POST request methods
- ▶ The names of the form fields will be used as indices in the respective arrays.
- ▶ For example, to access the value of an input box named 'first\_name' in a form whose method is POST, we'd write:  
`$_POST[ 'first_name' ]`
- ▶ If the form method is GET,  
`$_GET[ 'first_name' ]`

# Input (cont'd)

---

► Example:

//login.html

...

```
<form method='POST' action="login.php">
 <input type='text' name='username'>

 <input type='password' name='password'>

 <input type='submit' value='login'>
</form>
```

...

# Input (cont'd)

---

//login.php

<?php

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

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

    if(\$username == "user" && \$password == "pass"){

        //login successful

        header( 'Location: home.php' );

        exit();

    }else{

        //login failed

        header( 'Location: login.html' );

        exit();

    }

} ?>

username:

password:

sex: ☒ Male

☐ Female

☒ Red

☒ Green

☒ Blue

Eth

Hi guys,

send

```
<form action="process.php?id=25&name=koke"
method="post">
```

---

```
username:<input type="text" name="username"/>

```

```
password:<input type="password"
name="password"/>

```

```
sex:<input type="radio" name="sex"
value="male"/>Male

```

```
<input type="radio" name="sex"
value="female"/>Female

```

```
<input type="checkbox" name="red"
value="red"/>Red

```

```
<input type="checkbox" name="green"
value="green"/>Green

```

```
<input type="checkbox" name="blue"
value="blue"/>Blue

```

---

```
<select name="country">
```

```
 <option value="-1" selected="selected">(Select your
country)</option>
```

```
 <option value="et" >Eth</option>
```

```
 <option value="ken" >Keny</option>
```

```
</select>

```

```
 <textarea name="comment" cols="20"
rows="10"></textarea>

```

```
<input type="submit" value="send" />
```

```
</form>
```

---

► Process.php

```
<pre>
<?php
print_r($_POST);

?>
</pre>
```

## ► Form Processing on the same page

Form.php

---

```
<?php
 if(isset($_POST["name"]) && isset($_POST["age"])) {
 if (preg_match("/^[A-Za-z'-]$/",$_POST['name'])) {
 die ("invalid name and name should be alpha");
 }

 echo "Welcome ". $_POST['name']. "
";
 echo "You are ". $_POST['age']. " years old.";

 exit();
 }
?>
```

```
<html>
```

```
<body>
```

---

```
 <form action = "<?php
 echo $_SERVER['PHP_SELF']?>" method = "POST">
```

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

```
 Age: <input type = "text" name = "age" />
```

```
 <input type = "submit" />
```

```
 </form>
```

```
</body>
```

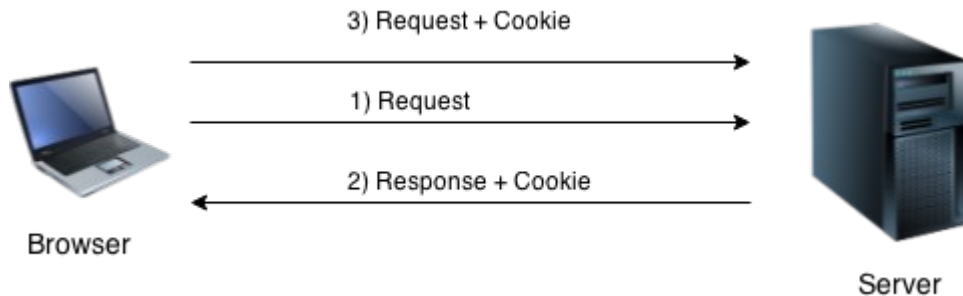
```
</html>
```

# cookies

- ▶ Another way of getting input from client can be using cookies.
- ▶ PHP cookie is a small piece of information which is stored at client browser. It is used to recognize the user.
- ▶ Cookie is information stored on the client by the server.
- ▶ Cookies stored on a client associated to a particular server are sent to the server every time that same client computer requests for a page.
- ▶ Cookies are often used to identify a user.
- ▶ A cookie is a piece of attribute/value data. A server can send cookies as value of a HTTP header Set-Cookie:. Multiple headers may be sent.
- ▶ When the client visits the web site again, it will send the cookie back to the server with a HTTP header Cookie:



- Cookie is created at server side and saved to client browser.
- Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.



In short, cookie can be created, sent and received at server end.

**Note: PHP Cookie must be used before <html> tag.**

## PHP setcookie() function

---

- PHP **setcookie()** function is used to set cookie with HTTP response.
- Once cookie is set, you can access it by **\$\_COOKIE** **superglobal** variable.

### Syntax

```
bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path
[, string $domain [, bool $secure = false [, bool $httponly = false]]]]])
```

# Set-Cookie

```
setcookie: name=value; [expires= date];
[path=path;] [domain= domain] [secure]
```

---

where

- ▶ `name=` is the variable name set in the cookie
- ▶ `value=` is the variable's value
- ▶ `date=` is a date when the cookie expires
- ▶ **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=** restricts the sending of the cookie to a certain domain
  - ▶ 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

The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).



**Following example will create two cookies name and age these cookies will be expired after one hour.**

---

```
<?php
 setcookie("name", "John ", time()+3600, "/", "", 0);
 setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>
```

```
 <head>
 <title>Setting Cookies with PHP</title>
 </head>
```

```
 <body>
 <?php echo "Set Cookies"?>
 </body>
```

## Accessing Cookies with PHP

- PHP provides many ways to access cookies. Simplest way is to use either **`$_COOKIE`** or **`$HTTP_COOKIE_VARS`** variables.
- Following example will access all the cookies set in above example.

```
<html>
 <head>
 <title>Accessing Cookies with PHP</title>
 </head>
 <body>
 <?php
 echo $_COOKIE["name"]. "
";
 /* is equivalent to */
 echo $HTTP_COOKIE_VARS["name"]. "
";
 echo $_COOKIE["age"] . "
";
 /* is equivalent to */
 echo $HTTP_COOKIE_VARS["age"] . "
";
 ?>
 </body>
</html>
```

**You can use `isset()` function to check if a cookie is set or not.**

```
<html>

<head>
 <title>Accessing Cookies with PHP</title>
</head>

<body>

 <?php
 if(isset($_COOKIE["name"]))
 echo "Welcome " . $_COOKIE["name"] . "
";

 else
 echo "Sorry... Not recognized" . "
";
 ?>

</body>
</html>
```

`setcookie(name,value,expire,path,domain)` creates cookies.

---

```
<?php
setcookie("uname", $_POST["name"], time()+36000);
?>
```

**NOTE:**

**setcookie() must appear  
BEFORE <html> (or any output) as it's part of  
the header information sent with the page.**

```
<html>
<body>
<p>
```

Dear <?php echo \$\_POST["name"] ?>, a cookie was set on  
this

page! The cookie will be active when the client has sent the  
cookie back to the server.

```
</p>
</body>
</html>
```

---

```
<html>
<body>
<?php
if (isset($_COOKIE["uname"]))
echo "Welcome " . $_COOKIE["uname"] . "!

";
else
echo "You are not logged in!
";
?>
</body>
</html>
```

**`$_COOKIE`**  
**contains all COOKIE data.**

**`isset()`**  
**finds out if a cookie is set**

## PHP Delete Cookie

- If you set the expiration date in past, cookie will be deleted.

```
<?php
setcookie ("CookieName", "", time() - 3600); // set the expiration date to one hour ago
?>
```

---

## Modify a Cookie Value

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

## Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the `$_COOKIE` array variable:

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>

<html>
<body>

<?php
if(count($_COOKIE) > 0) {
 echo "Cookies are enabled.";
} else {
 echo "Cookies are disabled.";
}
?>

</body>
</html>
```

## PHP Session

- A session is a way to store information (in variables) to be used across **multiple pages**.
- **Unlike a cookie, the information is not stored on the users computer.**
- PHP session is used to store and pass information from one page to another temporarily (until user close the website).
- PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.
- PHP session creates unique user id for each browser to recognize the user and avoid conflict between multiple browsers.

- 
- ▶ **By default**, **session variables** last until the user closes the browser.
  - ▶ So; Session variables hold information about one single user, and are available to all pages in one application.  
**Tip: If you need a permanent storage, you may want to store the data in a database.**
  - The location of the temporary file is determined by a setting in the **php.ini** file called **session.save\_path**. Before using any session variable make sure you have setup this pat

## Why PHP-Sessions?

---

- Whenever you want to create a [website](#) that allows you to store and display information about a user, determine which user groups a person belongs to, utilize permissions on your [website](#) or you just want to do something cool on your site, [PHP's Sessions](#) are vital to [each](#) of these features.

- Cookies are about 30% unreliable right now and it's getting worse every day.

- More and more web browsers are starting to come with security and privacy settings and people browsing the net these days are starting to frown upon Cookies because they store information on their local computer that they do not want stored there.

---

- 
- PHP has a great set of functions that can achieve the same results of Cookies and more without storing information on the user's computer. PHP Sessions store the information on the web server in a location that you chose in special files.
  - These files are connected to the user's web browser via the server and a special ID called a "Session ID".
  - This is nearly 99% flawless in operation and it is virtually invisible to the user

## Start a PHP Session

- A session is started with the `session_start()` function.
- Session variables are set with the **PHP global variable: `$_SESSION`**.
- When a session is started following things happen –
  - PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
  - A cookie called PHPSESSID is automatically sent to the user's computer to store unique session identification string.
  - A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess\_ ie sess\_3c7foj34c3jj973hjkop2fc937e3443.

- When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from **the PHPSESSID** cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.
- A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

## PHP \$\_SESSION

PHP **\$\_SESSION** is an **associative array** that contains all session variables. It is used to set and get session variable values.

### Example: Store information

```
$_SESSION["user"] = "John";
```

Example: Get information

```
echo $_SESSION["user"];
```

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

# Get PHP Session Variable Values

---

notice that all session variable values are stored in the global `$_SESSION` variable:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "
";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

- Sessions store their identifier in a cookie in the client's browser
- Every page that uses session data must be preceded by the session\_start() function**
- Session variables are then set and retrieved by accessing the global `$_SESSION[]`

**Note: The session\_start() function must be the very first thing in your document. Before any HTML tags**



# Modify a PHP Session Variable

---

To change a session variable, just overwrite it:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
```

```
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
```

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

## PHP Session Counter Example

---

```
<?php
 session_start();

 if (!isset($_SESSION['counter'])) {
 $_SESSION['counter'] = 1;
 } else {
 $_SESSION['counter']++;
 }
 echo ("Page Views: " . $_SESSION['counter']);
?>
```

## Destroying a PHP Session

- A PHP session can be destroyed by **session\_destroy()** function. This function does not need any argument and a single call can destroy all the session variables.
- If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable –

Here is the call which will destroy all the session variables –

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

---

**Here is the example to unset a single variable –**

```
<?php
 unset($_SESSION['counter']);
?>
```

# Avoid Error PHP-Sessions

---

```
PHP Example: <?php
echo "Look at this nasty error below:
";
session_start();
?>
```

## Error!

- ▶ **Warning: Cannot send session cookie - headers already sent by (output started at session\_header\_error/session\_error.php:2) in session\_header\_error/session\_error.php on line 3**
- ▶ **Warning: Cannot send session cache limiter - headers already sent (output started at session\_header\_error/session\_error.php:2) in session\_header\_error/session\_error.php on line 3**

## PHP Example:

```
<?php
session_start();
echo "Look at this nasty error below:";
```

---

▶ ?>

## Correct

---

# Relational Databases and MySQL



# Starting mysql using Command Line

---

- ▶ After Control Panel is Running...
- ▶ Macintosh
  - ▶ `/Applications/MAMP/Library/bin/mysql -uroot -p`
- ▶ Windows
  - ▶ `c:\xampp\mysql\bin\mysql.exe -u root -p`

## Logging to mysql using command line

Syntax:

> `mysql -h hostname -u username -p`

Example:

`C:\wamp\bin\mysql\mysql5.1.36\bin>mysql -u root`

Result:

`mysql>`

---



# Your first MySQL Command

---

- ▶ Showing all databases in mysql;
- ▶ `> show databases;`

## Creating a Database

```
Mysql>CREATE DATABASE People;
USE People;
```



# Start Simple - A Single Table

---

- ▶ Lets make a table of Users in our People database

- ▶ Two columns - Name and an E-Mail

```
CREATE TABLE Users(
 name VARCHAR(128),
 email VARCHAR(128)
)
```

```
DESCRIBE Users;
```



# SQL Insert

---

```
INSERT INTO Users (name, email) VALUES
('Chuck', 'csev@umich.edu')
INSERT INTO Users (name, email) VALUES
('Sally', 'sally@umich.edu')
INSERT INTO Users (name, email) VALUES
('Somesh', 'somesh@umich.edu')
INSERT INTO Users (name, email) VALUES
('Caitlin', 'cait@umich.edu')
INSERT INTO Users (name, email) VALUES ('Ted',
'ted@umich.edu')
```

- ▶ The Insert statement inserts a row into a table



# SQL Delete

---

- ▶ Deletes a row in a table based on a selection criteria

```
DELETE FROM Users WHERE email='ted@umich.edu'
```



# SQL: Update

---

- ▶ Allows the updating of a field with a where clause

```
UPDATE Users SET name='Charles' WHERE
email='csev@umich.edu'
```



# Retrieving Records: Select

---

- ▶ The select statement retrieves a group of records - you can either retrieve all the records or a subset of the records with a WHERE clause

```
SELECT * FROM Users
```

```
SELECT * FROM Users WHERE email='csev@umich.edu'
```



# Sorting with ORDER BY

---

- ▶ You can add an **ORDER BY** clause to **SELECT** statements to get the results sorted in ascending or descending order

```
SELECT * FROM Users ORDER BY email
```

```
SELECT * FROM Users ORDER BY name
```



# The LIKE clause

---

- ▶ We can do wildcard matching in a WHERE clause using the LIKE operator

```
SELECT * FROM Users WHERE name LIKE '%e%'
```



# The LIMIT Clause

---

- ▶ The LIMIT clause can request the first "n" rows, or the first "n" rows after some starting row. Note: the first row is zero, not one
- ▶ WHERE and ORDER BY clauses happen \*before\* the LIMIT is applied
- ▶ The limit can be a count or a starting row and count (starts from 0)

```
SELECT * FROM Users ORDER BY email DESC LIMIT 2;
SELECT * FROM Users ORDER BY email LIMIT 1,2;
```



# Counting Rows with SELECT

---

- ▶ You can request to receive the count of the rows that would be retrieved instead of the rows

```
SELECT COUNT(*) FROM Users;
```

```
SELECT COUNT(*) FROM Users WHERE email='csev@umich.edu'
```



# SQL Summary

---

```
INSERT INTO Users (name, email)
VALUES ('Ted', 'ted@umich.edu')
```

```
DELETE FROM Users WHERE email='ted@umich.edu'
```

```
UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
```

```
SELECT * FROM Users WHERE email='csev@umich.edu'
```

```
SELECT * FROM Users ORDER BY email
```

```
SELECT * FROM Users WHERE name LIKE '%e%'
```

```
SELECT * FROM Users ORDER BY email LIMIT 1,2;
```

```
SELECT COUNT(*) FROM Users WHERE email='csev@umich.edu'
```

---



# Looking at Data Types

---

- ▶ Text fields (small and large)
- ▶ Binary fields (small and large)
- ▶ Numeric fields
- ▶ AUTO\_INCREMENT fields



# String Fields

---

- ▶ Understand character sets and indexable for searching
- ▶ CHAR allocates entire space (faster for small strings where length is known)
- ▶ VARCHAR allocates variable amount of space depending on the data length (less space)



# Text Fields

---

- ▶ Have a character set - paragraphs or HTML pages
  - ▶ TINYTEXT up to 255 characters
  - ▶ TEXT up to 65K
  - ▶ MEDIUMTEXT up to 16M
  - ▶ LONGTEXT up to 4G
- ▶ Generally not used with indexing or sorting
  - and only then limited to a prefix



# Binary Types (rarely used)

---

- ▶ Character = 8 - 32 bits of information depending on character set
- ▶ Byte = 8 bits of information
  - ▶ BYTE(n) up to 255 bytes
  - ▶ VARBINARY(n) up to 65K bytes
- ▶ Small Images - data
- ▶ Not indexed or sorted



---

## Binary Large Object (BLOB)

- ▶ Large raw data, files, images, word documents, PDF, Movies, etc etc..
- ▶ No translation, indexing or character set
  - ▶ TINYBLOB(n) - up to 255
  - ▶ BLOB(n) - up to 65K
  - ▶ MEDIUMBLOB(n) - up to 16M
  - ▶ LONGBLOB(n) - up to 4G



---

## Integer Numbers

- ▶ Numbers are very efficient, take little storage and are easy to process because CPU's can compare them often with a single instruction
  - ▶ TINYINT (-128, 128)
  - ▶ SMALLINT (-32768, +32768)
  - ▶ INT or INTEGER (2 Billion)
  - ▶ BIGINT - ( $10^{18}$  ish)



---

# Floating Point Numbers

- ▶ Floating point numbers can represent a wide range of values but accuracy is limited
  - ▶ FLOAT (32-bit)  $10^{38}$  with seven digits of accuracy
  - ▶ DOUBLE (64-bit)  $10^{308}$  with 14 digits of accuracy



---

## Dates

- ▶ TIMESTAMP - 'YYYY-MM-DD HH:MM:SS' (1970, 2037)
- ▶ DATETIME - 'YYYY-MM-DD HH:MM:SS'
- ▶ DATE - 'YYYY-MM-DD'
- ▶ TIME - 'HH:MM:SS'
- ▶ Built-in MySql function NOW()



# AUTO\_INCREMENT

---

- ▶ Often as we make multiple tables and need to JOIN them together we need an integer, primary key for each row so we can efficiently add a reference to a row in a table, in some other table as a foreign key

```
DROP TABLE Users;
```

```
CREATE TABLE Users (
 user_id INT UNSIGNED NOT
 NULL
 AUTO_INCREMENT KEY,
 name VARCHAR(128),
 email VARCHAR(128)
)
```

```
DESCRIBE Users;
```



# Summary

---

- ▶ SQL allows us to describe the shape of data to be stored and give many hints to the database engine as to how we will be accessing or using the data
- ▶ SQL is a language that provides us operations to Create, Read, Update, and Delete (CRUD) our data in a database



## Example :

---

- 1) Create a database called people
- 2) Under people database, create a table called users

With the following fields:

id    int    auto\_increment    primary key

Name varchar(25)

Email varchar(25)

id	name	email
1	Abebe	abebe@yahoo.com
2	mohammed	mohammed@yahoo.com
3	aster	<a href="mailto:aster@yahoo.com">aster@yahoo.com</a>
4	almaz	almaz@yahoo.com

# Database Manipulation

- ▶ One of the defining features of PHP is the ease with which you can connect to and manipulate databases.
- ▶ PHP implements functions for connecting to a wide range of databases systems: MySQL, Oracle, MSSQL, Interbase, dBase, and many more.
- ▶ Many hosting companies use MySQL,
- ▶ MySQL is free software

# Steps in Establishing Connection and work with mysql

---

Connecting to MySQL

Use the php function

```
mysqli_connect("domain_name/or IP
address","username","password") or die("Unable
to connect to the database
server!").mysqli_connect_error(); //terminates the
script, and
outputs the error
```

---

Example:

```
$conn = mysqli_connect("localhost", "user", "password",
"database-name");
```

## 2. Write query

---

insert

update

select

delete

## 3. Execute the query using

**mysql\_query()** function performs a query against a database.

**mysql\_query(connection, query)**

```
$result=mysql_query(($conn, ,$query)
```

```
<?php
$con = mysqli_connect("localhost","root","","university");
```

---

```
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
```

```
// Perform query
if ($result = mysqli_query($con, "SELECT * FROM users")) {
 echo "Returned rows are: " . mysqli_num_rows($result);
 // Free result set
 mysqli_free_result($result);
}
```

```
mysqli_close($con);
?>
```

## 5) Output the result of the query if it is select query.

---

► `$result=mysqli_query($mysqli, $query)`  
`mysqli_fetch_assoc()` function fetches a result row as an associative array.

Note: Fieldnames returned from this function are case-sensitive.

```
$row = mysqli_fetch_assoc($result);
echo $row['_msg'];
```

```
<?php
$con = mysqli_connect("localhost","my_user","my_password","my_db");
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
```

```
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = mysqli_query($con, $sql);
```

```
// Associative array
$row = mysqli_fetch_assoc($result);
echo $row["Lastname"];
Echo $row["Age"]);
```

```
// Free result set
mysqli_free_result($result);
```

```
mysqli_close($con);
?>
```

**mysqli\_fetch\_array()** function fetches a result row as an associative array, a numeric array, or both.

Note: Fieldnames returned from this function are case-sensitive.

**mysqli\_fetch\_array(result,resulttype)**

- **resulttype** Optional. Specifies what type of array that should be produced. Can be one of the following values:

**MYSQLI\_ASSOC**

**MYSQLI\_NUM**

**MYSQLI\_BOTH (this is default)**

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
```

---

```
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
```

```
$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = mysqli_query($con,$sql);
```

```
// Numeric array
$row = mysqli_fetch_array($result, MYSQLI_NUM);
echo $row[0];
echo $row[1];
```

```
// Associative array
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);
Echo $row["Lastname"];
echo $row["Age"];
```

```
// Free result set
```

---

```
mysqli_free_result($result);
```

Internet Programming

---

## PHP mysqli connect\_errno() Function

mysqli\_connect\_errno() function returns the error code from the last connection error, if any.

```
<?php
$con =
mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
?>
```

---

**mysqli\_connect\_error()** function returns the error description from the last connection error, if any.

```
<?php
$con = mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
?>
```

## Example

1. Create university database
2. Create student(id int primary key, firstname varchar(15), age int)
3. Fill student table with some data
4. Display the content of student table

## ► Solution

```
<?php
```

---

```
// Connecting, selecting database<?php
$con =
mysqli_connect("localhost","root","","university");

// Check connection
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " .
mysqli_connect_error();
 exit();
}
?>
```

---

## PHP mysqli affected\_rows Function

- **mysqli\_affected\_rows()** function returns the number of affected rows in the previous SELECT, INSERT, UPDATE, REPLACE, or DELETE query.

```
<?php
```

```
$con = mysqli_connect("localhost","my_user","my_password","my_db");
```

---

```
if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}
```

```
// Perform queries and print out affected rows
```

```
mysqli_query($con, "SELECT * FROM Persons");
```

```
echo "Affected rows: " . mysqli_affected_rows($con);
```

```
mysqli_query($con, "DELETE FROM Persons WHERE Age>32");
```

```
echo "Affected rows: " . mysqli_affected_rows($con);
```

```
mysqli_close($con);
```

```
?>
```

**mysqli\_field\_count()** function returns the number of columns for the most recent query.

---

```
<?php
$con =
mysqli_connect("localhost","my_user","my_password","my_db");

if (mysqli_connect_errno()) {
 echo "Failed to connect to MySQL: " . mysqli_connect_error();
 exit();
}

mysqli_query($con, "SELECT * FROM Friends");
// Get number of columns - will return 3
mysqli_field_count($con);

mysqli_close($con);
?>
```

- 
- ▶ Example
  - ▶ Create the student table under university database

<b>Idno</b>	<b>fname</b>
1	abebe
2	kebede

---

## menu.html

```
<html>
<head>
 <title></title>
</head>
<body>
<h1>Menu</h1>
Register

View All

Update

Delete

</body>
</html>
```

## connection.php

```
<?php
$HOST="localhost";
$USERNAME="root";
$PASSWORD="";
$DATABASE="aau";
$conn=mysqli_connect($HOST,$USERNAME,$PASSWORD,
$DATABASE);

if(mysqli_connect_errno()){
 echo "COnnection Failed:".mysqli_connect_error();
 exit;
}

?>
```

```
<html>
<head>
 <title></title>
</head>
<body>
<h1>Registration Form</h1>
<form method="post" action="register.php">
Id No :<input type="number" name="idno">

First name:<input type="text" name="fname">

<input type="submit" value="Submit">
</body>
</html>
```

<?php

register.php

```
include "connection.php";
$idno=$_POST['idno'];
$name=$_POST['fname'];

$query="INSERT INTO student
VALUES($idno,'$name')";
$result=mysqli_query($conn,$query);
if($result){
 echo "One Row is affected!!!";
}

?>
```

<a href="menu.html">Back to Menu</a>

<?php

view.php

include "connection.php";

```
$query="SELECT * FROM student";
$result=mysqli_query($conn,$query);
```

```
echo "<table border=1>";
while($row = mysqli_fetch_assoc($result)){
 echo "<tr><td> $row[idno] </td> <td> $row[fname] </td>
</tr>";
}
?>
</table>

Back to Menu
</body>
</html>
```

```
<html>
<head>
 <title></title>
</head>
<body>
<form method="post" action="update1.php">
Search By Id : <input type="number" name="idno">

New Name : <input type="text" name="fname">

<input type="submit" value="submit">

</form>
</body>
</html>
```

<?php

update.php

include "connection.php";

\$idno=\$\_POST['idno'];

\$fname=\$\_POST['fname'];

\$query="UPDATE student SET fname='\$fname' WHERE  
idno=\$idno";

\$result=mysqli\_query(\$conn,\$query);

if(\$result){

    echo "One Row is Updated!!!";

}

?>

<a href="menu.html">Back to Menu</a>

```
<html>
```

```
<head>
```

---

```
 <title></title>
```

```
</head>
```

```
<body>
```

```
<h1>Delete By Id </>
```

```
 <form method="post" action="delete.php">
```

```
 Enter Id :<input type="number" name="idno">

```

```
 <input type="submit" value="submit">
```

```
 </form>
```

```
</body>
```

```
</html>
```

<?php

## delete.php

---

```
include "connection.php";
$idno=$_POST['idno'];
```

```

$query="DELETE FROM student WHERE idno=$idno";
$result=mysqli_query($conn,$query);
if($result){
 echo "One Row is DELETED!!!";
}
```

```
?>
```

```
Back to Menu
```