WEB PROGRAMMING SCV1223

PHP Programming

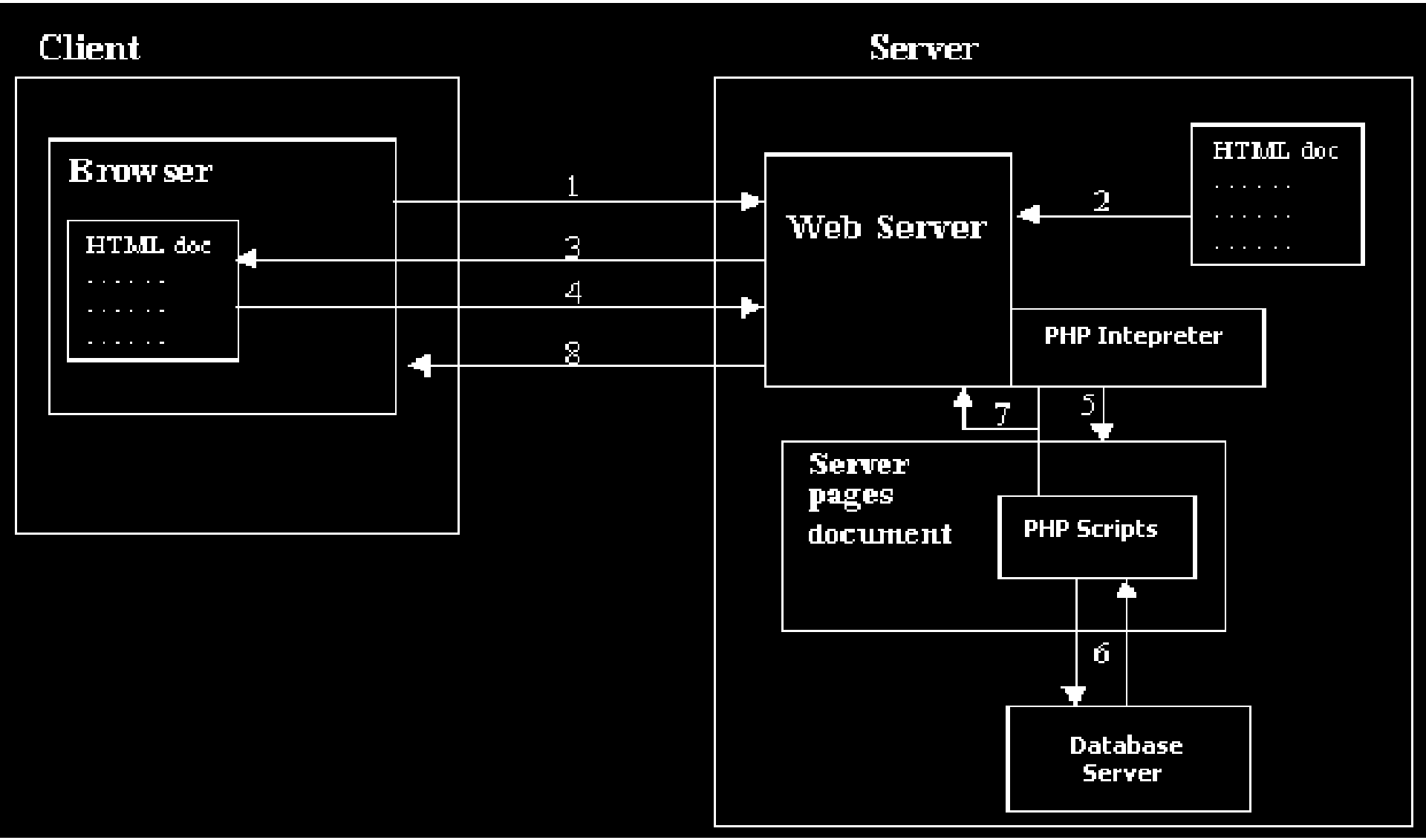
Dr. Md Sah bin Hj Salam

En. Jumail bin Taliba

Introduction

- PHP *(PHP Hypertext Preprocessor)* is a web scripting language.
- Specifically designed for web-based application development.
- It is a server-pages technology. PHP script is embedded into HTML.

Web-based Architecture Using PHP



Embedding PHP into HTML

- There are different ways to embed PHP scripts into HTML
  - `<?php echo "Hello World"; ?>`
  - `<? echo "Hello World"; ?>`
  -
```
<script language="php">
   echo "Hello World";
</script>
```
  - `<% echo "Hello World"; %>`

    *(this format needs the PHP.ini option **asp_tag** to be turn on)*

- Switching between HTML and PHP can be done at any time.

- Example: print tag
 100 times.

```
<?php for($i=0; $i<100; $i++) { ?>
  <BR>
<?php } ?>
```

• Including Files

- Example:

```
<?php
  $message="Hello World";
  include "hello.php";
?>
```

```
<html>
  <head> <title></title>
  </head>
  <body>
    <?php echo $message ?>
  </body>
 </html>
```

File: ***hello.php***

Output in the Web Browser:

```
Hello World
```

Syntax

- Identifiers
  - Variable names are **case-sensitive.**
  - Function names (both built-in and user-defined functions) are **NOT case-sensitive.**

- Comments

```
/* C style comments */
// C++ style comments
# Bourne shell style comments
```

- PHP Statements are terminated by semicolon.

Variables

- Preceded by a dollar sign ($).
- There is no limit on the length of a variable name.
- Variable names in PHP are case-sensitive.

Example:

Valid Variable Names:

```
$counter
$_COUNTER
$user_name
$user100
```

Invalid Variable Names:

```
$1counter
$#counter
```

- In PHP, we do not have to declare variables.
- Variables are automatically declared by the PHP intepreter, the first time a value is assigned to them.
- PHP variables are untyped; you can assign a value of any type to a variable.

Example:

```
$var = "Hello World";
echo "$var\n";
$var = 1001;
echo "The number is $var\n";
```

Output:

```
Hello World
The number is 1001
```

- In PHP, uninitialized variables have the value undef, which evaluates to different values, depending on its context.
- undef value for a numeric context is *0*
- undef value for a string context is an *empty string* (*“”*)

Example:

```
$a=100;
$str="World";
$msg= $hello . $str;
$c = $a + $b;

echo $msg, "\n";
echo $c;
```

Output:

```
World
100
```

Dynamic Variables

- Also known as *variable variables*.
- Syntax: `$$var` or `${$var}`
- Meaning: obtain the value of the variable whose name is equal to the value of `$var`.

Example 1:

```
$msg = "Hello World";
$var = "msg";
echo $$var;
```

Output:

```
Hello World
```

How does it work?

$$var => $"msg" or $msg => "Hello World";

Example 2:

```
$var ="hello";
$$var = "World";

print $hello;
```

Output:

```
World
```

How does it work?

We created the variable `$hello` indirectly, where `$$var` means `$"hello"`, or just `$hello`.

The assignment statement `$$var="World"` is equal to `$hello="World"`

Constants

- Constant is a value that cannot be modified once it is declared.
- Constants are created using function `define`.
- Constant names by default are case-sensitive.

Example:

```
define ("HELLO","Hi World");
define ("NUMBER", 5);

echo HELLO, "\n";
echo "The number is ", NUMBER;
```

Output:

```
Hi World
The number is 5
```

Data Types

- PHP provides 4 primitive data types
  - Integer numbers
  - Floating-point numbers
  - String
  - Boolean
- PHP provides 2 compound data types
  - Array
  - Object

Integers, Floating-Point Numbers and Booleans

- Example:

```
$decimal=16;
$hex=0x10;
$octal=020;

$float=0.234;

$bool=true;
```

Strings

Three ways for creating a string in PHP

- Double quatation mark ("")
- Single quatation mark (')
- Here document

Double-quoted strings are subject to ***variable substitution*** and ***escape sequence handling***, while single quotes are not.

- Example 1: *double and single quatation*

```
$a="World";
echo "1. Hello \t$a\n";
echo '2. Hi \t$a\n';
echo "\n";

$b=' Peace';
$c = $a.$b;
echo "3. $a.$b = $c\n";
echo "4. \$a.\$b = $c\n";
```

Output:

```
1. Hello     World
2. Hi \t$a\n
3. World. Peace = World Peace
4. $a.$b = World Peace
```

- Example 2: *here document*

```
$a ='World';
$str = <<<EOT
  Hello $a
  This is a "multiline" string
  assigned using the 'heredoc' syntax.
EOT;

echo $str;
```

Output:

```
Hello World
This is a "multiline" string
assigned using the 'heredoc' syntax.
```

Type Casting

- Type casting is process that converts a data type to another type.

- In PHP, type casting can be performed by using
  - C-style syntax
  - Function `settype`

Example 1: *Using C-Style*

```
// create two strings $a and $b
$a = "12";
$b = "10";

$c = $a.$b;  // string concatenation
$d = (int)$a + (int)$b; // arithmetic operation

echo "The value of \$c is $c \n";
echo "The value of \$d is $d \n";
```

Output:

```
The value of $c is 1210
The value of $d is 22
```

Example 2: *Using the function* `settype`

```
$a = "12";
$b = "10";

settype ($a,"integer");
settype ($b,"integer");

$c = $a + $b;
echo "The value of \$c is $c \n";
```

Output:

```
The value of $c is 22
```

Selection Statements

`if` statement

- Format 1:

```
if(expr)
{
   statements
} elseif(expr)
   {
     statements
   } else
     {
      statements
     }
```

Format 2:

```
if(expr):
   statements
elseif(expr):
   statements
else:
   statements
endif;
```

switch statement

- Format 1:

```
switch(expr)
{
   case expr1:
      statements
   break;

   case expr2:
      statements
   break;

   default:
      statements
   break;
}
```

- Format 2:

```
switch(expr):
   case expr1:
      statements
   break;

   case expr2:
      statements
   break;

   default:
      statements
   break;
endswitch;
```

Repetition Statements

`while` statement

- Format 1:

```
while(expr)
{
   statements
}
```

Format 2:

```
while(expr):
   statements
endwhile;
```

do-while statement

```
do
{
   statements
} while(expr);
```

`for` statement

- Format 1:

```
for(start_expr; cond_expr; iter_expr)
{
   statements
}
```

Format 2:

```
for(start_expr; cond_expr; iter_expr):
   statements
endfor;
```

foreach statement

- Format 1:

```
foreach(array_expression as $value)
{
   statements
}
```

Format 2:

```
foreach(array_expression as $value):
   statements
endforeach;
```

Format 3: *associative array*

```
foreach(hash_array as $key=>$value){
 statements
}
```

foreach statements

Example 1:

```
$brands = array("National", "MEC", "Khind", "Toshiba");

foreach ($brands as $value)
    echo $value,"\n";
```

Output:

```
National
MEC
Khind
Toshiba
```

foreach statements

Example 2:

```
$month["jan"] = "January";
$month["feb"] = "February";
$month["mar"] = "March";
$month["apr"] = "April";

foreach ($month as $key => $value)
   echo "Key: $key,  Value: $value \n";
```

Output:

```
Key: jan,  Value: Janury
Key: feb,  Value: February
Key: mar,  Value: March
Key: apr,  Value: April
```

`break` and `continue` statements

- `break` is used to stop a loop.
- `continue` is used to skip the current iteration and go to the next iteration in a loop.

`break` and `continue` statements

Example:

```
for ($i=1; $i<99; $i++)
{
  if (($i%2)==0) continue;
  print ("$i is is an odd number\n");
  if ($i>5) break;
}
```

Output:

```
1 is an odd number
3 is an odd number
5 is an odd number
7 is an odd number
```

Functions

- Function names in PHP are NOT case-sensitive.
- Functions are defined using the keyword `function`.

Example 1:

```
function HelloMessage() // define a function without parameter
{
   return "Hello World!"; // keyword "return" returns the result
}

echo HelloMessage(); // call the function; to execute it
```

Output: `Hello World!`

Passing parameters by value

Example 2:

default value

```
function PrintValues($a, $b=10)
{
    echo "a=$a, b=$b\n";
}

PrintValues(5);
PrintValues(5,15);
```

Output:

```
a=5, b=10
a=5, b=15
```

Parameters that are set with a *"default value"* are ***optional*** when the function is called. Otherwise, they are ***compulsory***.

Passing parameters by reference

Example 3:

& indicates passing by reference

```
function Change($x, &$y)
{
   $x++;
   $y++;
}

$x = 10;
$y = 5;

echo "Before: x=$x, y=$y\n";
Change($x, $y);
echo "After: x=$x, y=$y\n";
```

Output:

```
Before: x=10, y=5
After: x=10, y=6
```

Variable Scope

- The scope of a variable is the context within which a variable is available.
- There are two types of scope:
  - *Global scope*; variables are created outside from functions.
  - *Local* or *function scope*; variables are created inside a function.

- Variable scoping in PHP is different from most other languages.

- In most other languages such as Perl, variables in a function are treated as global if they are not declared in that function.

- In PHP, default scope for a varibale in a function is "*local*". So, to turn it to global, we have to set it as *global*, using the keyword `global`.

Example 1:

```
$var1="Hello World";

function PrintValue()
{
  echo $var1;
}

PrintValue();
```

There is no output. Why?

Function `PrintValue` has its own copy of variable `$var1`.

The variable is different from global variable `$var1="Hello World";`

By default all variables in a function are treated as local, unless they are defined as global.

Example 2:

```
$var1="Hello World";

function PrintValue()
{
  global $var1;
  echo $var1;
}

PrintValue();
```

Output:

```
Hello World
```

Another way to use global variables, using the superglobal array named $GLOBALS

Example:

```
echo $GLOBALS["var1"];
```

Static Variables

Example:

```
function PrintCounter()
{
  static $counter=0;

  $counter++;

  echo "Counter=$counter\n";
}

PrintCounter();
PrintCounter();
PrintCounter();
```

Output:

```
Counter=1
Counter=2
Counter=3
```

Arrays

- Two types of array in PHP
  - **normal array** *(using numerical index)*
  - **hash or associative array** *(using string index*)
- Elements in an array may consist different data types.
- There are different ways for creating an array

Example 1: *Creating an array using function* `array()`

```
$mixed = array ("Zero", "One", 2, 3);

for ($i=0; $i<count($mixed); $i++)
   print ("Element in $i is $mixed[$i]\n");

print("\n\nUsing foreach loop\n");

foreach ($mixed as $element)
   print ("$element\n");
```

Function `count()` returns the size of an array (number of elements in that array)

Output:

```
Element in 0 is Zero
Element in 1 is One
Element in 2 is 2
Element in 3 is 3

Using foreach loop
Zero
One
2
3
```

Example 2:

```
$list[0]="Zero";
$list[1]="One";
$list[]="Two";
$list[]="Three";

for ($i=0; $i<count($list); $i++)
   print ("Element in $i is $list[$i]\n");
```

Output:

```
Element in 0 is Zero
Element in 1 is One
Element in 2 is Two
Element in 3 is Three
```

The statement *$list[] = "Two" appends the value "Two" to the end of the array $list*

Example 3: *Creating an hash array*

```
$subject["SCK3633"]="Web Programming";
$subject["SCK3032"]="Project I";
$subject["SCK3134"]="Project II";

for ( reset($subject); $index = key ($subject); next($subject) )
   print ("$index is $subject[$index]\n");
```

Output:

```
SCK3633 is Web Programming
SCK3032 is Project I
SCK3134 is Project II
```

Example 4: *Using `foreach` loop*

```
$subject["SCK3633"]="Web Programming";
$subject["SCK3032"]="Project I";
$subject["SCK3134"]="Project II";

foreach ($subject as $index => $value)
   print ("$index is $value\n");
```

Output:

```
SCK3633 is Web Programming
SCK3032 is Project I
SCK3134 is Project II
```

Example 5: *Creating an hash array using function* `array()`

```
$subject= array ("SCK3633" => "Web Programming",
                 "SCK3032" => "Project I",
                 "SCK3134" => "Project II");

print_r($subject);
```

Function `print_r()` is used to print information about the specified variable

Output:

```
Array
(
    [SCK3633] => Web Programming
    [SCK3032] => Project I
    [SCK3134] => Project II
)
```

OOP in PHP

Introduction

OOP comes with some concepts such as

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

All those concepts are supported by PHP.

But, we are only going to explore for the encapsulation and abstraction.

Creating classes

```
<?php
 class ClassName
 { var $property1, $property2,…., $propertyn;

   function ClassName($prop1, $prop2, …, $propn);  //constructor
   { $this->property1 = $prop1;
     $this->property2 = $prop2;
      ……..
     $this->propertyn =$propn;
   }

   function method1()
   {
    ……..
   }

   function method_n($param1, $param2)
   {
     . . .
   }
  }
?>
```

Corrections:

(as marked in red color)

- When referring a property, DON'T put a dollar sign ($). The sign is only for the variable `$this`

Example.

```
$this->property1 = $prop1;//(ok)
$this->$property1 = $prop1;//(wrong)
```

- ***Class name*** and ***method name*** (including the constructor) are **NOT case-sensitive**
- ***Property name*** is **case-sensitive**

Creating classes (cont.)

Example:

```
<?php
  class Person
  { var $name, $age;

    function Person($name, $age)
    { $this->name = $name;
      $this->age = $age;
    }

    function displayInfo()
    { echo "Name: $this->name \n";
      echo "Age : $this->age \n\n";
    }

    function getAge()
    { return $this->age;
    }
  }
?>
```

Corrections:
(as marked in red color)

- A property name doesn't need a dollar sign ($). The sign is only for the variable `$this`.

Creating object and accessing its members

Example:

```
<?php
   // creating an object of class Person
   $person1 = new Person("Ali",20);

   // accessing method of person1
   $person1->displayInfo();

   // accessing property of person1
   echo "Five years from now, $person1->name is ",
        $person1->getAge() + 5, " years old\n";
?>
```

Output:

Name: Ali

Age: 20

Five years from now, Ali is 25 years old

Correction:
The property `name` doesn't need a dollar sign ($).

Array of objects

Example:

```
<?php
   $person_list = array(); // creating array

   // inserting objects into the array elements
   $person_list[0]= new Person("Ali",20);
   $person_list[] = new Person("Aminah",24);
   $person_list[] = new Person("Bakar",19);
?>
```

```php
<?php

  // displaying info of all persons - using for loop
      for ($i=0; $i<count($person_list); $i++)
          $person_list[$i]->displayInfo();
?>
```

```php
<?php

  // displaying info of all persons - using foreach loop
      foreach ($person_list as $person)
         $person->displayInfo();

?>
```

```php
<?php

    // calculating the average age of all persons
    $sum=0;
    foreach ($person_list as $person)
        $sum += $person->getAge();

    $average = $sum / count($person_list);
?>
```

Regular Expressions

Today's Topics

- Introduction
- Functions
- Syntax
- Examples

Introduction

- Regular expression is a formatted pattern that can be used to find instances of a string in another.
- PHP supports two type of regular expression:
  - POSIX
  - Perl-compatible

Regular Expression Functions

POSIX Regular Expression functions:

- **`ereg`** - searches a *string* for matches of a *pattern* in a case-sensitive way
- **`ereg_replace`** - searches a *string* of a *pattern*, then replaces the matched text with a new string.
- **`split`**- splits a string into array by regular expression.
- **`eregi`** – performs the same as `ereg`, but this ignores case distinction
- **`eregi_replace`** - performs the same as `ereg_replace`, but this ignores case distinction
- **`spliti`**- performs the same as `split`, but this ignores case distinction.

Syntax:

```
int ereg ( string pattern, string string [, array &matches] )
```

Returns: **true** if pattern is found, **false** if pattern is not found

Example:

```
if (ereg("Wo","Hello World"))
  print("The pattern is found");

//Ouput: The pattern is found
```

Syntax:

```
string ereg_replace ( string pattern,
                      string replacement, string string)
```

Returns: if pattern is found, it returns the **replaced** string. Otherwise, it returns the original string

Example:

```
$s1="Hello World";
$s2 = ereg_replace("l", "L", $s1);

//result: $s2="HeLLo WorLd".  $s1 remains unchanged
```

Syntax:

```
array split ( string pattern, string string [, int limit])
```

Example 1:

```
$a="Hello World again";
$b = split(" ",$a);

//result: $b[0]="Hello"
//         $b[1]="World"
//         $b[2]="again"
//            $a remains unchanged
```

Example 2:

```
$a="Hello World again";
$b = split(" ",$a, 2);

//result: $b[0]="Hello"
//         $b[1]="World again"
//            $a remains unchanged
```

Regular Expression Syntax

Metacharacters:

- `^` *(caret)* - searches at the beginning of the string
- `$` *(dollar sign)* - searches at the end of string
- `.` *(dot)* - searches for any character

Example:

```
$a1 = ereg("^hello","hello world");          // $a1=true
$a2 = ereg("^hello","I say hello world");    // $a2=false

$b1 = ereg("bye$","goodbye");                // $b1=true
$b2 = ereg("bye$","goodbye my friend");      // $b2=false

$c1 = ereg(".", "hello");                     // $c1=true
$c2 = ereg(".", "");                          // $c2=false
```

Quantifiers:

`{n}`	matches exactly n times
`{m,n}`	matches between m and n times inclusive
`{n,}`	matches n or more times
`+`	matches one or more times (same as `{1,}`)
`*`	matches zero or more times (same as `{0,}`)
`?`	matches zero or one time (same as `{0,1}`)

Example 1:

```
$a1 = ereg("t*","tom");          // $a1=true
$a2 = ereg("t*","fom");          // $a2=true

$b1 = ereg("w+","hello world");          // $b1=true
$b2 = ereg("w+","hello wwwwwwwworld");   // $b2=true

$c1 = ereg("b?", "book");                 // $c1=true
$c2 = ereg("b?", "cook");                 // $c2=true
```

Example 2:

```
$a1 = ereg("l{2}","hello world");          // $a1=true
$a2 = ereg("o{2}","hello world");          // $a2=false

$b1 = ereg("o{1,2}","hello world");        // $b1=true
$b2 = ereg("u{0,2}","hello world");        // $b2=true

$c1 = ereg("d{1,}", "hello world");        // $c1=true
$c2 = ereg("d{2,}", "hello world");        // $c2=false
```

Escape sequence characters:

Use a *backslah,* `\` before the character

`\^`	caret character
`\$`	dollar sign
`\(`	opening parenthesis
`\)`	closing parenthesis
`\.`	dot character
`|`	or symbol
`\*`	asterisk character
`\?`	question mark
`\\`	backslash
`\{`	opening bracket
`\}`	closing parenthesis

Example:

```
//search for a string p^ at the beginning
$a = ereg("^p\^","p^2 means p power of 2"); // $a=true

//search for a dollar sign at the end
$b = ereg("\$$","hello world$");            // $b=true

//replace dot characters to a string ' dot '
$c = ereg_replace("\.", " dot ", "world.com.my");
// result: $c="world dot com dot my

$d = ereg_replace(".", " dot ", "com.my");
// result: $d=" dot dot dot dot dot dot

//search for a question mark in a string
$e = ereg("\?", "hello"); // $e=false

$f = ereg("?", "hello");  // error - because what to repeat?
$g = ereg(".?", "hello"); // $g=true - repeat 0 or 1 time of
                          //    any character
```

Grouping patterns:

Related patterns can be grouped together between square brackets `[]`

Example 1:

```
//check whether all characters in a string are in lower case
$a1 = ereg("^[a-z]+$","hello");        // $a1=true
$a2 = ereg("^[a-z]+$","hello world"); // $a2=false, because the space


//check whether all characters are in a string in upper case
$b = ereg("^[A-Z]+$","Hello"); // $b=false


//check whether a string is a numerical string
$c = ereg("^[0-9]+$","123"); // $c=true
```

Example 2:

```
//check whether a string contains alphabets but not numbers
$d1 = ereg("^[a-zA-Z]+$","HeLLo");     // $d1=true
$d2 = ereg("^[a-zA-Z]+$","123Hello"); // $d2=false


//the first character must be a number followed by alphabets
$a1 = ereg("^[0-9][a-zA-Z]+$","2abC");    // $a1=true
$a2 = ereg("^[0-9][a-zA-Z]+$","22abC");   // $a2=false
$a3 = ereg("^[0-9][a-zA-Z]+$","2abC2");   // $a3=false
```

Grouping terms:

Related terms can be grouped together using parentheses **()**

Example:

```
// check whether a string starts with hello or world
$a1 = ereg("^(hello|world)","hello world");     // $a1=true
$a2 = ereg("^(hello|world)","world wide web"); // $a2=true


//check whether a string contains repeated uppercase alphabet and
//number. eg. A2C4D1
$b1 = ereg("^([A-Z][0-9])+$","A2C4D1");   // $b1=true
$b2 = ereg("^([A-Z][0-9])+$","AAC4D1");   // $b2=false
$b3 = ereg("^([A-Z][0-9])+$","A222222"); // $b3=false

//if (b3 above) doesn't use parentheses
$c = ereg("^[A-Z][0-9]+$","A222222");    // $c=true
```

Predefined character classes:

`[[:alpha:]]`	Alphabet characters (same as `[a-zA-Z]`)
`[[:digit:]]`	Digit characters (same as `[0-9]`)
`[[:alnum:]]`	Alphanumeric characters (i.e, letters [a-zA-Z] or digits [0-9])
`[[:lower:]]`	Lowercase letters (same as `[a-z]`)
`[[:upper:]]`	Uppercase letters (same as `[A-Z]`)
`[[:space:]]`	Whitespaces (i.e, space, tab or newline characters)

Example:

```
// check whether the string is a number

// both regular expressions below are equivalent
$a = ereg("^[[:digit:]]$","5467"); // $a=true
$b = ereg("^[0-9]$","5467");       // $b=true
```

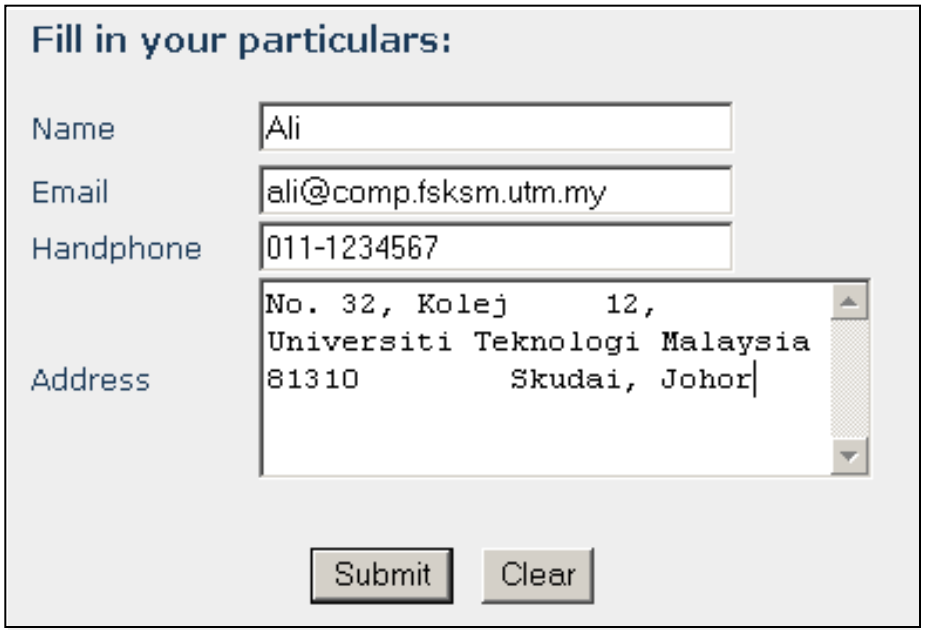
Applications of Regular Expression

- Validating formats or patterns
- Pre-processing strings
- Counting the occurrences of pattern
- Filtering records

Example

We'll use the form below for the following examples.

```
<form name='form' method='GET' action='form.php' >
```



form.html

Validating Format:

Example 1: *Validating the IP address of accessing users*

Let say, students are only allowed to fill in the form from their residential college. Assume the IP address of each college follows this format: ***161.139.X.Y****, where X = 70, 71, 75 or 80 and Y is a number.*

form.php

```
<?php
// Firstly, read the IP address of the client PC from an environment variable
 $ip = $_SERVER['REMOTE_ADDR'];

// then checks whether the ip address follows the format.
//    If it does not, terminate the page
 if (!ereg("161\.139\.(70|71|75|80)\.[0-9]+",$ip))
      die("You are not allowed to access this page");
?>
<html>
  <head> </head>
  <body>
        If the IP address has been validated, this
        part and beyond are accessible to the user
</body>
</html>
```

Validating Format:

Example 2: *Validating the user's Name*

Let say, the Name must contain at least three characters and at least two words.

```
<body>
<?php
// Firstly, read the Name
 $Name = $_GET['Name'];

// check whether the Name contains at least 3 characters
if (!ereg(".{3,}",$Name))
      print("You must enter at least 3 characters");

// check whether the Name contains at least 2 words
 if (!ereg("[[:alpha:]]+[[:space:]]([[:alpha:]]+[[:space:]]*)+",$Name))
      print("You must enter at least 2 words");
?>
</body>
```

form.php

Validating Format:

Example 3: *Validating Email address*

eg valid email address: ali@hotmail.com, ali@utm.my, ali@comp.fsksm.utm.my

eg invalid email address: ali@com, utm.my

```
<?php

 // Firstly, read the Email address
 $Email = $_GET['Email'];

// check whether the Email address is valid
if (!ereg(".+@(.+\.)+.+",$Email))
      print("Invalid email address");

?>
```

form.php

Validating Format:

Example 4: *Validating Handphone numbers*

Let say the valid mobile phone numbers are in this format: 01X-1234567

where 01X is 012, 013, 016, 017 or 019

```
<?php

 // Firstly, read the Handphone number
 $Handphone = $_GET['Handphone'];

// check whether the Handphone number is valid
if (!ereg("01(0|2|3|6|7|8|9)-[0-9]{7}",$Handphone))
      print("Invalid Phone number");

?>
```

form.php

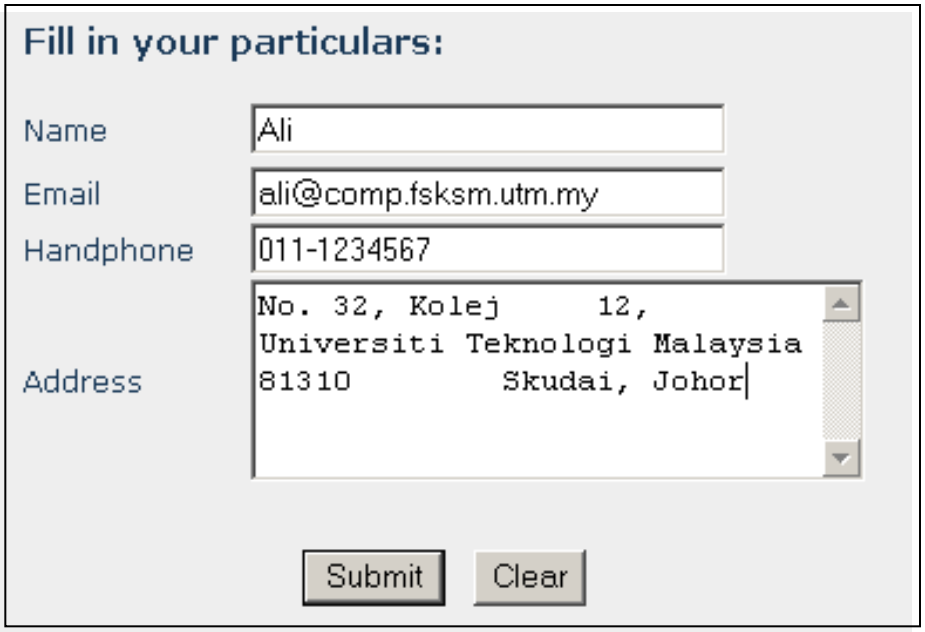
Pre-processing String:

Sometimes we need to process a string before we use it.

Example:

*In the following example, the program reads the Address which is entered by a user, then removes the **new line** characters and lastly removes the **redundant spaces**. Let say the input is as the figure below. The program will receive the Address as*

"No. 32, Kolej 12,\nUniversiti Teknologi Malaysia\n81310 Skudai, Johor"



Pre-processing String:

```php
<?php

 // Firstly, read the Address
 $Address = $_GET['Address'];

// Let say the $Address is
//"No. 32,  Kolej      12,\nUniversiti Teknologi Malaysia\n81310        Skudai, Johor"

// Replacing newline characters with spaces
$Address = ereg_replace("\n", " ", $Address);

// The $Address becomes
//"No. 32,  Kolej      12, Universiti Teknologi Malaysia 81310        Skudai, Johor"

// Removing redundant spaces
$Address = ereg_replace("[[:space:]]{2,}", " ", $Address);

// The $Address becomes
//"No. 32, Kolej 12, Universiti Teknologi Malaysia 81310 Skudai, Johor"

?>
```

form.php

Counting the number of occurrences:

Regular expression does not provide any function to count the number occurrences of a pattern in a string. The trick for doing this is, firstly splits the string and then counts the result.

```
<?php

// Let say the string is
 $s = "There is no counting function in regular expression.".
      "To count the number of occurrences of a pattern in ".
      "a string, we can use functions split() and count(). ";

// Counting the number of words
$words = split("[[:space:]]",$s);
print ("Number of words:" .count($words)); // Output: 27

// Counting the number of sentences
$sentences = split("\.",$s);
print ("Number of sentences:" .count($sentences)); // Output: 3

?>
```

form.php

Counting the number of occurrences:

Another trick is by using a loop. In this technique, we remove the occurrence of a string for each iteration. Then do the searching again until there is no more matched string.

```
<?php
// Let say the string is
 $s = "There is no counting function in regular expression.".
      "To count the number of occurrences of a pattern in ".
      "a string, we can use functions split() and count(). ";

// Counting the number of string the in $s
$count=0;
while ( eregi("the",$s, $match) )
{ $count++;

  // remove the first occurrence of a word beginning with 'the' to
  // find another intances in the string
  $s = ereg_replace($match[0],"", $s);
}
print ("Number of string 'the': " .$count ); // Output: 2
?>
```

form.php

Filtering records:

Filtering means, what records should be or should not be selected. Usually, we use SQL statements for doing this.

Example: *Filtering records using SQL*

Let say we have a table named 'person' as below and we want to display all persons who his/her name starts with 'A'

Table: person

name	gender
Syatilla Binti Yusoff	F
Abdul Ghani Bin Rustam	M
Amzar Bin Kamaruddin	M
Jamilah Binti Muhammad Hakim	F
Muaz Bin Abdul Jalil	M
Noraini Binti Kamaluddin	F
Aryani Binti Ghazali	F

Expected Output

Name	Gender
Abdul Ghani Bin Rustam	M
Amzar Bin Kamaruddin	M
Aryani Binti Ghazali	F

Filtering records:

Example: Using SQL statements

SQL has its own regular expression. It uses the operator ***like***

```
<table width="300" border="2">
  <tr><th>Name</th><th>Gender</th> </tr>

<?php
  $conn = mysql_connect('localhost','example','abc123');
  $db=mysql_select_db('db_example',$conn);
  $query = mysql_query("select * from person where name like 'A%'");

  while ( $row=mysql_fetch_row($query) )
  {
    $name   = $row[0];
    $gender = $row[1];

    print("<tr><td>".$name."</td><td>".$gender."</td></tr>\n");
  }
?>

</table>
```

form.php

Filtering records:

Example: Filtering using PHP regular expression

In this technique, the SQL statement is still needed. But it is only used for retrieving records from the database. The filtering process is done by the PHP regular expression.

```
<table width="300" border="2">
  <tr><th>Name</th><th>Gender</th> </tr>
<?php
  $conn = mysql_connect('localhost','example','abc123');
  $db=mysql_select_db('db_example',$conn);
  $query = mysql_query("select * from person");

  while ( $row=mysql_fetch_row($query) )
  {
    $name   = $row[0];
    $gender = $row[1];

   if (!ereg("^A",$name)) continue; // do not display the record if it is not
                                    //   matched with the pattern

   print("<tr><td>".$name."</td><td>".$gender."</td></tr>\n");
  }
?>
</table>
```

PHP: Web Variables

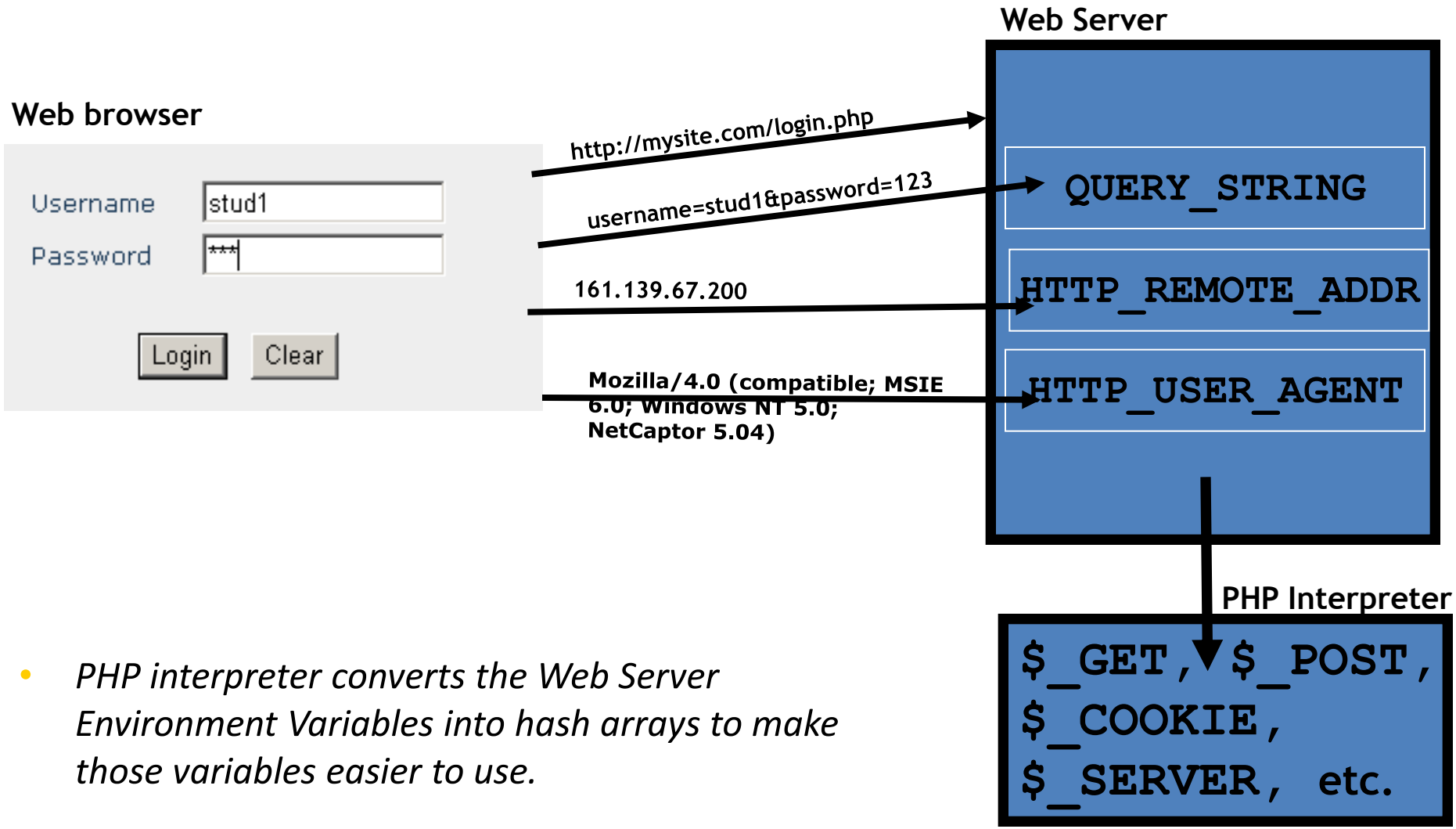
Topics

- $_GET and $_POST
- $_COOKIE
- $_ENV and $_SERVER

What is an environment variable?

- When accessing a page, web browser sends an URL to web server.
- Web browser also sends some other information such as IP address, browser's information, cookies and so on.
- These information will be received by the Web Server and stored into variables called "CGI Environment Variables".

Example:



- *PHP interpreter converts the Web Server Environment Variables into hash arrays to make those variables easier to use.*

Web Variables

- Web variables are automatically created, when the PHP interpreter receives an HTTP request.
- Web variables are stored in *hash arrays*
- Types of web variables:
  - Form Data
  - Cookies
  - Environment Variables

Form Data

- `$_GET`

- Contains form data that are sent using the CGI method *GET*
- or data that are supplied directly in the URL.

- Example:

- URL: *http://comp.fsksm.utm.my/~nometrik/myscript.php?a=5&b=10*

```
$a=$_GET["a"];
$b=$_GET["b"];
echo "a=$a, b=$b";
```

Output: `a=5, b=10`

- `$_POST`

- Contains form data that are sent using the CGI method *POST*

Creating form data variables indirectly using the function `extract()`

Example:

***File**: form.html*

```
<form method="POST" name="form1" action="myscript.php">
  Name: <input type="text" name="txtName"><br>
  Age: <input type="text" name="txtAge"><br>
</form>
```

***File**: myscript.php*

```
extract($_POST);
echo "Name: $txtName,  Age: $txtAge";
```

Cookies

- `$_COOKIE`
- Contains all cookies that sent by the browser.
- The name of the cookie is the key and the cookie value becomes the array value.

- Example:

File: sendcookies.php

```
<?php
setcookie("name", "Ali", time() + 3600);
setcookie("age", "20",0);
?>

<html>
 <head></head>
 <body>
   <a href="receivecookies.php"> Click me </a>
 </body>
</html>
```

You may also use the `extract` function :

```
extract($_COOKIE);
```

File: receivecookies.php

```
<html>
 <head></head>
 <body>
   <?php
      $name = $_COOKIE["name"];
      $age  = $_COOKIE["age"];
      echo "Name: $name <br>\n";
      echo "Age  : $age <br>\n";
   ?>
 </body>
</html>
```

Output:

Name: Ali
Age : 20

Environment Variables

- `$_ENV`
- Contains data about the client's environment such as the type of operating system used, the path of the temporary directory used in the client and so on. The contents of this hash array will vary from system to system.

Example: ***getting the OS used by the client***

```
$os_type = $_ENV["OS"];
```

Example

$_ENV variables

Name	Value
SELINUX_INIT	YES
CONSOLE	/dev/pts/0
TERM	linux
INIT_VERSION	sysvinit-2.85
PATH	/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
RUNLEVEL	5
runlevel	5
PWD	/
LANG	en_US.UTF-8
PREVLEVEL	N
previous	N
SHLVL	2
_	/sbin/initlog

Environment Variables (cont.)

- `$_SERVER`
- Mostly contains data about the server such as `DOCUMENT_ROOT`, `SERVER_NAME`, `SERVER_PORT`, and some data about client such as `REMOTE_ADDR`, `REMOTE_PORT`.

Example: ***getting the IP address of the client***

```
$userip = $_SERVER["REMOTE_ADDR"];
```

Example

$_SERVER variables

Name	Value
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
HTTP_ACCEPT_LANGUAGE	en-us
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; NetCaptor 5.04)
HTTP_HOST	comp.fsksm.utm.my
HTTP_CONNECTION	Keep-Alive
HTTP_COOKIE	PHPSESSID=b359e45900abe4df95e97f5f6c19f0af
PATH	/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
SERVER_SIGNATURE	*Apache/2.0.51 (Fedora) Server at comp.fsksm.utm.my Port 80*
SERVER_SOFTWARE	Apache/2.0.51 (Fedora)
SERVER_NAME	comp.fsksm.utm.my
SERVER_ADDR	161.139.68.248
SERVER_PORT	80
REMOTE_ADDR	10.1.0.124
DOCUMENT_ROOT	/var/www/html
SERVER_ADMIN	root@comp.fsksm.utm.my
SCRIPT_FILENAME	/home/lecturers/jumail/public_html/examples/env/env_vars.php
REMOTE_PORT	2656
GATEWAY_INTERFACE	CGI/1.1
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	GET
QUERY_STRING	txt=Hello+World
REQUEST_URI	/~jumail/examples/env/env_vars.php?txt=Hello+World
SCRIPT_NAME	/~jumail/examples/env/env_vars.php
PHP_SELF	/~jumail/examples/env/env_vars.php
argv	Array
argc	2

- $GLOBALS

- Contains all global variables

Example:

```
$var1=10;

function printNumber()
{
  echo $GLOBALS[“var1”];
}
```

Reference

- Sebesta, R. W., Programming the World Wide Web, (2009), 5th Edition, Pearson.
- Deitel P. J, Deitel H. M., Internet & World Wide Web: How To Program, (2007), 5th Edition, Prentice Hall.
- Anderson-Freed S., (2001), Weaving A Website: Programming in HTML, JavaScript, PHP and Java. Prentice Hall