Foreword
This lecture will only discuss the basic concepts of PHP programming language.
How to access MySQL database through PHP script will also be introduced.
However, you need advanced knowledge on PHP and its MySQL extension to complete your project.
If you encounter difficulties in your project, please consult the PHP document at:
http://www.php.net
The instructor will guide you to solve the problems in your project but will not help you to write the program.

What is PHP?
PHP ("PHP: Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP is a powerful server-side scripting language for creating dynamic and interactive websites.
Example:
```html
<html>
<head>
<title>Example</title>
</head>
<body>
<?php
  echo "Hi, I'm a PHP script!";
?>
</body>
</html>
```

What is a web server?
Program that understands the HTTP protocol and generates appropriate responses
- Clients "connect" to the web server
- Clients send a "request"
- Server reads request, generates "response"
- Client interprets response appropriately

Web Server Basics
- What is a web server?
  - Program that responds to requests for documents
    - "http daemon"
  - Uses the Hypertext Transfer Protocol (HTTP) to communicate
  - Physical machine which runs the program
- Duties
  - Listen to a port
  - When a client is connected, read the HTTP request
  - Perform some lookup function
  - Send HTTP response and the requested data

A simplified web server
- Client asks for a file through network request
- Server finds appropriate file based on client’s request
- Server sends back a response header followed by the requested file’s data through the network
- Server closes connection
How do we identify web server?
- Domain names/IP address and ports
- `http://127.0.0.1:8080/index.html`
  - Refers to local box (127.0.0.1 is me)
  - Port # is 8080
  - File is named index.html

How do we identify files?
- File name is specified in Request Message
- Server maps that name to a real file
- Mapping can be done in whichever way server wants
- For example, `/~vivek/index.html` is actually `/nfs/fac/vivek/public_html/index.html`
- Request and response follow HTTP protocol

What is HTTP?
- The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems.
- It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands).
- HTTP is the protocol that supports communication between web browsers and web servers.
- HTTP also serves as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP, NNTP, FTP, Gopher, and WAIS protocols.

HTTP Request - Response
- The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server.
- The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.
- HTTP can support multiple request-reply exchanges over a single TCP connection.
- The "well known" TCP port for HTTP servers is port 80.
- Other ports can be used as well...

Server side resources

Server Side Script
Consider the Apache web server as the intermediary that interprets HTTP requests that arrive through a network port and decides how to handle the request, which often requires working in conjunction with PHP.

Databases provide a way to implement one of the most important software design principles: one should separate that which varies from that which stays the same.

The program (PHP) determines which data to display, often from information in the GET or POST query string, and then uses a database API to interact with the database. Although the same separation could be achieved by storing content in files on the server, databases offer intuitive and optimized systems that do far more than a file-based design that would require custom-built reading, parsing, and writing functions.

PHP is a server-side scripting language. Therefore, you cannot see the PHP code in your web browser. Instead, you will see the HTML code generated by your PHP code.

Basic PHP Syntax:
- A PHP scripting block always starts with `<?php` and ends with `?>`. A PHP scripting block can be placed anywhere in the HTML document.

Comments in PHP:
- In PHP, we use `//` to make a single-line comment or `/*` and `*/` to make a large comment block.

Alternative PHP Syntax:
- `<script language="php">
  echo 'some editors (like FrontPage) don\'t like processing instructions';
</script>`
PHP Data Types

- PHP supports eight primitive types.
  - Four scalar types: boolean, integer, float (double), string.
  - Two compound types: array, object.
  - Two special types: resource, NULL.

- The type of a variable is usually not set by the programmer; rather, it is decided at runtime by PHP depending on the context in which that variable is used.

- If you would like to force a variable to be converted to a certain type, you may either cast the variable or use the settype() function on it.

```php
<?php
$a_bool = TRUE; // a boolean
$a_str = "foo"; // a string
$a_int = 12; // an integer

echo gettype($a_bool); // prints out: boolean
echo gettype($a_str); // prints out: string
// If this is an integer, increment it by four
if (is_int($a_int)) {
    $a_int += 4;
}
// If $bool is a string, print it out
// (does not print out anything)
if (is_string($a_bool)) {
    echo "String: $a_bool";
}
?>
```

PHP Variables

- Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

- Variable names follow the same rules as other labels in PHP.
- A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

```php
<?php
$var = 'Bob';
$Var = '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.
?>
```

PHP Constants

- A constant is an identifier (name) for a simple value, which won't change during the execution of the script.

- A constant is case-sensitive. By convention, constant identifiers are always uppercase.

```php
<?php
// Valid constant names
define("FOO", "something");
define("FOO2", "something else");
define("FOO_BAR", "something more");
// Invalid constant names
define("_F00", "something");
// This is valid, but should be avoided
// PHP may one day provide a magical constant
// That will break your script
define("__F00__", "something");
?>
```
PHP Expressions

In PHP, almost anything you write is an expression, or “anything that has a value” is an expression.

```php
function double($i)
{
    return $i*2;
}
$b = $a = 5; /* assign the value five into the variable $a and $b */
$c = $a++ /* post-increment, assign original value of $a (5) to $c */
$e = $d = ++$b; /* pre-increment, assign the incremented value of $b (6) to $d and $e; at this point, both $d and $e are equal to 6 */
$f = double($d++ /* assign twice the value of $d before the increment, 2*12 to $1 */
$g = double($d++ /* assign twice the value of $a after the increment, 2*14 to $g */
$h = $g += 10; /* first, $g is incremented by 10 and ends with the value of 24, the value of the assignment (24) is then assigned into $h, and $h ends with the value of 24 as well; */
```
Error Control Operators
- PHP supports one error control operator: the at sign (@).
- When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.
- If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg. This variable will be overwritten on each error, so check early if you want to use it.

```
<?php
/* Intentional file error */
$my_file = @file('non_existent_file') or die("Failed opening file: error was '$php_errormsg'");
// this works for any expression, not just functions:
$value = @$cache[$key];
// will not issue a notice if the index $key doesn't exist.
?>
```

Execution Operators
- PHP supports one execution operator: backticks (``).
- Note that these are not single-quotes!
- PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable).
- Use of the backtick operator is identical to shell_exec().

```
<?php
$output = `ls -al`;
echo "<pre>$output</pre>";
?>
```

Incrementing/Decrementing Operators
- PHP follows Perl's convention when dealing with arithmetic operations on character variables and not C's. For example, in Perl 'Z'+1 turns into 'AA', while in C 'Z'+1 turns into '[' (ord('Z') == 90, ord('[') == 91).
- Note that character variables can be incremented but not decremented and even so only plain ASCII characters (a-z and A-Z) are supported.

```
<?php
$i = 'W';
for ($n=0; $n<6; $n++) {
    echo ++$i . "\n";
}
?>
```

Logical Operators
```
<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a and $b</td>
<td>And</td>
<td>TRUE if both $a and $b are TRUE.</td>
</tr>
<tr>
<td>$a or $b</td>
<td>Or</td>
<td>TRUE if either $a or $b is TRUE.</td>
</tr>
<tr>
<td>$a xor $b</td>
<td>Xor</td>
<td>TRUE if either $a or $b is TRUE, but not both.</td>
</tr>
<tr>
<td>$a != $b</td>
<td>Inequality</td>
<td>TRUE if $a is not equal to $b.</td>
</tr>
<tr>
<td>$a &lt;&gt; $b</td>
<td>Inequality</td>
<td>TRUE if $a is not equal to $b.</td>
</tr>
<tr>
<td>$a !== $b</td>
<td>Non-identity</td>
<td>TRUE if $a is not identical to $b.</td>
</tr>
</tbody>
</table>
```

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

```
<?php
  $a = "Hello ";
  $b = $a . "World!"; // now $b contains "Hello World!"
  $a = "Hello ";
  $a = "World!"; // now $a contains "Hello World!"
?>
```

Array Operators
```
<table>
<thead>
<tr>
<th>Example</th>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + $b</td>
<td>Union</td>
<td>Union of $a and $b.</td>
</tr>
<tr>
<td>$a == $b</td>
<td>Equality</td>
<td>TRUE if $a and $b have the same key/value pairs.</td>
</tr>
<tr>
<td>$a === $b</td>
<td>Identity</td>
<td>TRUE if $a and $b have the same key/value pairs in the same order and of the same types.</td>
</tr>
<tr>
<td>$a &lt; $b</td>
<td>Inequality</td>
<td>TRUE if $a is less than $b.</td>
</tr>
<tr>
<td>$a &lt;&gt; $b</td>
<td>Inequality</td>
<td>TRUE if $a is not equal to $b.</td>
</tr>
<tr>
<td>$a !== $b</td>
<td>Non-identity</td>
<td>TRUE if $a is not identical to $b.</td>
</tr>
</tbody>
</table>
```
- The + operator appends elements of remaining keys from the right handed array to the left handed, whereas duplicated keys are NOT overwritten.
Control Structures

- **if, else, elseif:**
  ```php
  <?php
  if ($a > $b) {
    echo "a is bigger than b";
  } elseif ($a == $b) {
    echo "a is equal to b";
  } else {
    echo "a is smaller than b";
  }
  ?>
  ```

- **While:**
  ```php
  <?php
  $i = 1;
  while ($i <= 10) {
    echo $i++; /* the printed value would be $i before the increment (post-increment) */
  }
  ?>
  ```

- **do … while:**
  ```php
  <?php
  $i = 0;
  do {
    echo $i++;
  } while ($i < 5);
  ?>
  ```

- **for**
  ```php
  for (expr1; expr2; expr3) statement
  ```

  The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop.

  In the beginning of each iteration, expr2 is evaluated. If it evaluates to TRUE, the loop continues and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop ends.

  At the end of each iteration, expr3 is evaluated (executed).

- **foreach**
  ```php
  foreach (array_expression as $value) statement
  ```

  The first form loops over the array given by array_expression. On each loop, the value of the current element is assigned to $value and the internal array pointer is advanced by one (so on the next loop, you’ll be looking at the next element).

  ```php
  $arr = array(1, 2, 3, 4);
  foreach ($arr as &$value) {
    $value = $value * 2;
  }
  // $arr is now array(2, 4, 6, 8)
  ```

- **break**
  ```php
  $arr = array('one', 'two', 'three', 'four', 'stop', 'five');
  while (list(, $val) = each($arr)) {
    if ($val == 'stop') {
      break; /* You could also write 'break 1;' here. */
    }
    echo "$val<br />
  }
  $i = 0;
  while (++$i) {
    switch ($i) {
      case 5:
        echo "At 5<br />
        break 1; /* Exit only the switch. */
      case 10:
        echo "At 10; quitting<br />
        break 2; /* Exit the switch and the while. */
      default:
        break;
      }
  }
  ?>
  ```

- **switch**
  ```php
  $arr = array(1, 2, 3, 4);
  foreach ($arr as $key => $value) {
    switch ($key) {
      case 0:
        echo "$value is equal 0";
      case 1:
        echo "$value is equal 1";
      case 2:
        echo "$value is equal 2";
      }
  }
  ```
continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration. continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

```php
<?php
while ($i++ < 5) {
    echo "Outer<br />
";
    while (1) {
        echo "&nbsp;&nbsp;Middle<br />
";
        while (1) {
            echo "&nbsp;&nbsp;&nbsp;&nbsp;Inner<br />
";
            continue 3;
        }
        echo "This never gets output.<br />
";
    }
    echo "Neither does this.<br />
";
}
?>
```

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. return() will also end the execution of an eval() statement or script file.

If called from the global scope, then execution of the current script file is ended. If the current script file was include() or require(), then control is passed back to the calling file. Furthermore, if the current script file was include()ed, then the value given to return() will be returned as the value of the include() call.

If return() is called from within the main script file, then script execution ends. If the current script file was named by the auto_prepend_file or auto_append_file configuration options in php.ini, then that script file's execution is ended.

require() and include()

The require() or include() statement includes and evaluates the specific file.

require() and include() are identical in every way except how they handle failure.

They both produce a Warning, but require() results in a Fatal Error. In other words, don't hesitate to use require() if you want a missing file to halt processing of the page.

include() does not behave this way, the script will continue regardless. Be sure to have an appropriate include_path setting as well.

```php
<?php
require 'prepend.php';
include 'vars.php';
?>
```

require_once() and include_once()

The require_once() or include_once() statement includes and evaluates the specific file during the execution of the script.

This is a behavior similar to the require() or include() statement, with the only difference being that if the code from a file has already been included, it will not be included again.

```php
<?php
require_once("a.php"); // this will include a.php
require_once("a.php"); // this will not include a.php again
include_once("b.php"); // this will include a.php
include_once("b.php"); // this will not include a.php again
?>
```

Functions

User defined functions:

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.'\n";
    return $retval;
}
?>
```

Function arguments:

Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.

```php
<?php
function add_some_extra(&$string)
{
    $string .= 'and something extra.';
}
?>
```
Returning values

- Values are returned by using the optional return statement. Any type may be returned, including lists and objects. You can't return multiple values from a function, but similar results can be obtained by returning a list.

```php
<?php
function small_numbers()
{
    return array(0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
?>
```

- 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.

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

Variable functions

- PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

```php
<?php
function foo() {
    echo "In foo()<br />
";
}
function bar($arg = '') {
    echo "In bar(); argument was '" . $arg . "'.<br />
";
}$func = 'foo';
$func(); // This calls foo()
$func = 'bar';
$func('test'); // This calls bar()
?>
```

Variable functions (cont.)

- Among other things, variable functions can be used to implement callbacks, function tables, and so forth.
- Variable functions won't work with language constructs such as echo(), print(), unset(), isset(), empty(), include(), require() and the like. You need to use your own wrapper function to utilize any of these constructs as variable functions.

```php
<?php
    // This is a wrapper function around echo
    function echoit($string)
    {
        echo $string;
    }
$func = 'echoit';
$func('test'); // This calls echoit()
?>
```

Internal (built-in) functions

- PHP comes standard with many functions and constructs. There are also functions that require specific PHP extensions compiled in otherwise you'll get fatal "undefined function" errors.
- For example, to use image functions such as imagecreatetruecolor(), you'll need your PHP compiled with GD support. Or, to use mysql_connect() you'll need your PHP compiled in with MySQL support.
- There are many core functions that are included in every version of PHP like the string and variable functions.
- A call to printf() or get_loaded_extensions() will show you which extensions are loaded into your PHP.
- Also note that many extensions are enabled by default and that the PHP manual is split up by extension. See the configuration, installation, and individual extension chapters, for information on how to setup your PHP.

Exceptions

- PHP 5 and later versions have an exception model similar to that of other programming languages.
- An exception can be thrown, and caught ("catched") within PHP. Code may be surrounded in a try block, to facilitate the catching of potential exceptions.
- Each try must have at least one corresponding catch block. Multiple catch blocks can be used to catch different classes of exceptions.
- Normal execution (when no exception is thrown within the try block, or when a catch matching the thrown exception's class is not present) will continue after that last catch block defined in sequence.
- Exceptions can be thrown (or re-thrown) within a catch block.

```php
<?php
    function inverse($x) {
        if (!$x) {
            throw new Exception('Division by zero.');
        } else return 1/$x;
    }
    try {
        echo inverse(5) . "\n";
        echo inverse(0) . "\n";
    } catch (Exception $e) {
        echo 'Caught exception: ', $e->getMessage(), "\n";
    }
// Continue execution
echo 'Hello World';
?>
```
The \$_GET variable is used to collect values from a form with method="get".
- The \$_GET variable is an array of variable names and values sent by the HTTP GET method.
- The \$_GET variable is used to collect values from a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and it has limits on the amount of information to send (max. 100 characters).

Example:
```html
<form action="welcome.php" method="get">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
</form>
```
When use click on submit button, address bar will have:
```
http://webserver/welcome.php?name=Peter&age=37
```

Your server side scripting file, "welcome.php", now must use the \$_GET variable to catch the form data.
```
Welcome <?php echo \$_GET["name"];?>.<br />
You are <?php echo \$_GET["age"];?> years old!
```

Pros and Cons:
- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

The HTTP GET method is not suitable on large variable values; the value cannot exceed 100 characters.

The \$_POST variable is an array of variable names and values sent by the HTTP POST method.
- The \$_POST variable is used to collect values from a form with method="post".
- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

For example, for the same form, your script will look like:
```
Welcome <?php echo \$_POST["name"];?>.
You are <?php echo \$_POST["age"];?> years old!
```

Pros and Cons:
- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

The PHP \$_REQUEST variable contains the contents of both \$_GET, \$_POST, and \$_COOKIE.
- The PHP \$_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.
- For the same example, your script will look like:
```
Welcome <?php echo \$_REQUEST["name"];?>.
You are <?php echo \$_REQUEST["age"];?> years old!
```

PHP MySQL Extensions
- PHP MySQL extensions include a set of functions that allow you to access MySQL database servers.
- More information about MySQL can be found at http://www.mysql.com/.
- Documentation for MySQL can be found at http://dev.mysql.com/doc/.

Example
```
<?php
//Connecting, selecting database
$link = mysqli_connect('mysql_host', 'mysql_user', 'mysql_password', 'my_database')
or die('Could not connect : ' . mysqli_error($link));
//Send query
$query = 'SELECT * FROM my_table';
$result = mysqli_query($link, $query) or die("Query error: " . mysqli_error($link));
// Printing results in HTML
while ($line = mysqli_fetch_array($result, MYSQLI_ASSOC)) {
    echo "<tr>
    
    " . $col_value . "<td>$col_value</td>
    </tr>
```

```
// Free resultset
mysqli_free_result($result);
// Closing connection
mysqli_close($link);
```
MySQL Functions

- mysqli_affected_rows
  - Get number of affected rows in previous MySQL operation
- mysqli_change_user
  - Change logged in user of the active connection
- mysqli_client_encoding
  - Returns the name of the character set
- mysqli_close
  - Close MySQL connection
- mysqli_connect
  - Open a connection to a MySQL Server
- mysqli_data_seek
  - Move internal result pointer
- mysqli_db_query
  - Send a MySQL query

MySQL Functions (cont.)

- mysqli_errno
  - Returns the numerical value of the error message from previous MySQL operation
- mysqli_error
  - Returns the text of the error message from previous MySQL operation
- mysqli_escape_string
  - Escapes a string for use in a mysqli_query
- mysqli_fetch_assoc
  - Fetch a result row as an associative array
- mysqli_fetch_field
  - Get column information from a result and return as an object
- mysqli_fetch_lengths
  - Get the length of each output in a result
- mysqli_fetch_object
  - Fetch a result row as an object
- mysqli_fetch_row
  - Get a result row as an enumerated array
- mysqli_field_seek
  - Set result pointer to a specified field offset
- mysqli_free_result
  - Free result memory
- mysqli_get_client_info
  - Get MySQL client info
- mysqli_get_host_info
  - Get MySQL host info
- mysqli_get_proto_info
  - Get MySQL protocol info

MySQL Functions (cont.)

- mysqli_get_server_info
  - Get MySQL server info
- mysqli_info
  - Get information about the most recent query
- mysqli_insert_id
  - Get the ID generated from the previous INSERT operation
- mysqli_thread_id
  - List MySQL processes
- mysqli_field_count
  - Get number of fields in result
- mysqli_num_rows
  - Get number of rows in result
- mysqli_stat
  - Get current system status
- mysqli_ping
  - Ping a server connection or reconnect if there is no connection
- mysqli_query
  - Send a MySQL query
- mysqli_real_escape_string
  - Escapes special characters in a string for use in a SQL statement
- mysqli_select_db
  - Select a MySQL database
- mysqli_set_charset
  - Sets the client character set
- mysqli_thread_id
  - Return the current thread ID

References

- http://www.w3.org/TR/REC-html40/interact/forms.html
- http://www.w3schools.com/php/