Introduction

PHP(PHP: Hypertext Preprocessor), originally named Personal Homepage, is a general purpose scripting language developed by Rasmus Lerdorf in 1994. PHP is now widely-used as a general-purpose scripting language, however, it is mostly used for back-end web development and can be embedded into HTML.

It is a server-side web scripting language which means, it runs in a web server. The php script is executed first on the server, generating the HTML and sent to the client.

PHP is a very popular scripting language, however, it is not a very disciplined language in a way that there's a lot of inconsistencies in the language such as function names and there's no central organization governing the development of PHP.

In php, there are 2 ways of coding, the Object Oriented and the Procedural Paradigm.

PHP Fundamentals

Basic Syntax

PHP Tags

When PHP parses a file, it looks for opening and closing tags, which are <?php and ?> which tells the parser the beginning and end of a php code. This allows PHP to be embedded anywhere in the HTML. This also allows php to be embedded in different documents aside from HTML.

Example:

```
<?php
      Echo "Hello Webtech People!";
</php>
```

```
Escaping from HTML
PHP can skip HTML lines in between php scripts using conditions.
Example:
<php if ($expression == true){ ?>
<p>This will be shown if the condition is met.</p>
<?php }else{ ?>
<p>Otherwise this will be shown.</p>
<?php } ?>
```

Variables

Variables are represented by dollar sign($) followed by the name of the variable. PHP is a loosely typed language, meaning, that variable types are defined at run time, and can change at any time during execution.

Predefined Variables

Predefined variables are variables that are already defined by php.

Superglobals - are built-in variables that are always available in all scopes. Superglobals are named in capital letters and starts with `underscore('_')` after the dollar sign with the exception of the `$GLOBALS` variable.

1. `$GLOBALS` - references all the variables that are in global scope.
2. `$_SERVER` - an array containing information such as headers, paths, and script locations.
3. `$_GET` - an associative array containing variables passed through the url
4. `$_POST` - an associative array containing variables passed via the HTTP POST method.
5. `$_FILES` - an associative array of items uploaded via the HTTP POST method.
6. `$_REQUEST` - an associative array containing the value of HTTP Request variables `($_GET, $_POST` and `$_COOKIE`).
7. `$_SESSION` - an associative array which contains session variables available to the current script.
8. `$_ENV` - an associative array containing the variables passed via the environment method. These variables are coming from the environment under which the PHP parser is running.
9. `$_COOKIES` - an associative array containing the cookies passed via HTTP Cookies

Data Types

PHP has eight primitive data types namely: integers, floating point numbers, strings, Boolean, arrays, objects, resources and NULL.

Data Type	**Example**

Integers -	`<?php`
	`$x = 8; // decimal`
	`$y = -8; // negative`
	`$z = 0144; // octal`
	`$a = 0x3E8; // hexadecimal`
	`?>`

Floating Point Numbers -	`<?php`

	`$x = 1.5;`
	`$y = 1.5e5;`
	`$z = 1.5E-10;`
	`?>`
Strings -	`<?php`
	`$hi = ‘Hello world!’;`
	`$hi2 = “Hello World!”;`
	`?>`
Boolean -	`<?php`
	`$isValid = true;`
	`?>`
Arrays -	`<?php`
	`$members = array(“Juan”, “Maria”,`
	`“Pedro”);`
	`?>`
Objects -	`<?php`
	`Class dog{`
	`    public $dialogue = “Woof!”;`
	`    function bark(){`
	`        return $this->str;`
	`    }`
	`}`
	`?>`
Resources -	`<?php`
	`$file = fopen(“data.txt”, “r”);`
	`$db = mysql_connect(“localhost”,`
	`“root”, “”);`
	`?>`
NULL -	`<?php`
	`$empty = NULL;`
	`?>`

Conditional Statements

Conditional Statements in PHP are closely similar to loops in other programming/scripting languages such as Java and JavaScript and it has almost the same syntax as the aforementioned languages.

	Basic Syntax	**Sample Code**
If	`if(condition){` `    // code` `}`	`<?php` `$x = 1;` `if($x < 10){` `    echo "I have enough things to carry.";` `}` `?>`
If-else	`if(condition){` `    // code if condition is satisfied` `} else{` `    // code if condition is not satisfied` `}`	`<?php` `$x = 1;` `if($x < 10){` `    echo "I have enough things to carry.";` `} else{` `    echo "I have too much things to carry.";` `}` `?>`
If-elseif-else	`if(condition1){` `    // code if condition1 is satisfied` `} elseif(condition2){` `    // code if condition1 is not satisfied but condition2 is` `} else{` `    // code if both condition1 and condition2 are not satisfied` `}`	`<?php` `$x = 1;` `if($x == 0){` `    echo "I have nothing to carry.";` `} elseif($x > 10){` `    echo "I have too much things to carry.";` `} else {` `    echo "I have enough things to carry.";` `}` `?>`
Switch	`switch(n){` `    case label1:` `        // code when n=label1` `        break;` `    case label2:` `        // code when n=label2` `        break;`	`<?php` `$x = 1;` `switch($x){` `    case 1:` `        echo "I have something to carry";` `    case 2:` `        echo "I have two things to carry";`

	`...`
	`default:`
	`// default code if n is different from all labels`
	`}`
		`case 3:`
		`echo "I have three things to carry";`
		`default:`
		`echo "I have things to carry";`
		`}`
		`?>`

References:

https://www.tutorialrepublic.com/php-tutorial/php-if-else-statements.php
https://www.tutorialrepublic.com/php-tutorial/php-switch-case-statements.php

Loops

Like conditional statements, loops in PHP are also closely similar to loops in other programming/scripting languages such as Java and JavaScript and it has almost the same syntax as the aforementioned languages.

	Basic Syntax	**Sample Code**
While	`while(condition){` `// code` `}`	`<?php` `$i = 1;` `while($i <= 3){` `$i++;` `echo $i . " ";` `}` `?>`
For	`for(initialization; condition; increment){` `// code` `}`	`<?php` `for($i=1; $i<=3; $i++){` `echo $i . " ";` `}` `?>`
Foreach	`foreach($array as $var){` `// code` `}`	`<?php` `$numbers = array(1, 2, 3);` `foreach($numbers as $value){` `echo $value . " ";` `}` `?>`

Do-while	`do{`
	`// code`
		`<?php`
		`$i = 1;`
		`do{`

	`}while(condition);`	`    $i++;` `    echo $i . " ";` `}` `while($i <= 3);` `?>`

References:

https://www.tutorialrepublic.com/php-tutorial/php-loops.php

Functions

Functions, also called as methods in the context of other programming/scripting languages, are blocks of code that perform specific tasks. Although PHP has its own collection of libraries that can call directly in your PHP scripts, you can also create user-defined functions for your specific needs.

Basic Syntax:	**Sample Code**

`function funcName(){`
`     // code to be executed`
`}`
 	
NOTE: Function names are case-sensitive thus it must start with either a letter or an underscore.
	`function sayHello(){`
	`    echo "Hello dear user!";`
	`}`

You can also specify parameters to pass on to a function to be used as arguments for your block of code during run-time.

Basic Syntax:	**Sample Code**

`function funcName($param1,`
`$param2){`
`     // code to be executed`
`}`
	`function multiply($factor1,`
	`$factor2){`
	`    $product = $factor1 *`
	`$factor2;`
	`    return $sum;`
	`}`

You can also pass optional parameters with default values into a function. This way, a parameter may or may not be specified when calling its associated function.

Basic Syntax:

```
function funcName($param1,
$param2){
      // code to be executed
}
```

Sample Code

```
function incrementBy($num1,
$incrementNum2=1){
    $result = $num1 + $num22;
    return $result;
}
```

Handling Error Messages

PHP has built in functions for handling errors. Custom error handling functions can also be made with also exception error handling. When handling errors within your PHP code, you can either display, log, act on and/or ignore the errors that occur.

Error reports generated can be contained in different possible locations. It may be in the php.ini file, the .htaccess fil in the web browser, or in your own PHP code. You can set the error reporting level using the `error_reporting` variable in the php.ini file such as `E_ALL` or `E_STRICT`. You can also set it during run-time using the `error_reporting()` function (i.e. `error_reporting(E_ALL)`). Different error levels are also represented by an integer and an associated constant. Some of the most common error levels are the following:

Error Level	Value	Description
`E_ERROR`	1	A fatal run-time error that stops the execution of the script immediately.
`E_WARNING`	2	A non-fatal run-time error where most errors fall into this category. This, however, doesn't does not the execution of the script.
`E_NOTICE`	8	A run-time notice that indicates a possible error that had been encountered.
`E_STRICT`	2048	Although this is not strictly an error, it is triggered when PHP encounters any code that may cause future problems or forward incompatibilities.
`E_ALL`	8191	This indicates that all errors and warnings are to be displayed except for `E_STRICT` prior to PHP 5.4.0

The most basic way of handling errors in PHP is by using the `die()` function. This function simply displays a custom error message and terminates the currently running script.

This is a basic file opener error handler that handles errors regarding non-existent files.

```
<?php
if(file_exists("dummy.txt")){
    $file = fopen("dummy.txt", "wr");
} else{
    die("File does not exist. Terminating script……");
}
?>
```

References:
https://www.tutorialrepublic.com/php-tutorial/php-error-handling.php
http://www.hostingadvice.com/how-to/php-show-errors/

Sessions

Normally, accessing a website causes data to be stored using cookies which, in turn, are stored in the user's computer. Cookies are basically small pieces of data sent by the server to be stored in a user's browser which may send it back with the next request to the same server. It is typically used to tell if two requests came from the same browser (MDN Webdocs). Since cookies are susceptible to web attacks and can negatively affect a website's performance, PHP sessions solve these issues since it stores data in the server instead. In a session based environment, every user is identified through a unique number called session identifier or SID which is used to link each user with their own information in the server like emails and the like (TutorialRepublic).

One example of the use of sessions is in login forms. The basic idea of this is that after a user submits a login form and the password is verified by the server, the server creates a session variable for the user. For every page load that the user does within the website, the server will keep checking the session variable. Once the user logs out of the website, the session is destroyed (Morris, J., 2017).

Data are stored in between requests in the `$_SESSION` superglobal array.

Sample code:

```
<?php
// Starting session
session_start();

// Storing session data
$_SESSION["username"] = "user1";
?>
```

When a site with session support is accessed, PHP will check automatically or on request whether a specific session ID has been sent with the request if `session.auto_start` is set to 1 or explicitly through `session_start()` respectively (php.net).

To remove certain session data, you can simply use the unset function with the corresponding key of the `$_SESSION` array.

Sample Code:

```
<?php
// Starting session
session_start();

// Removing session data
if(isset($_SESSION["username"])){
    unset($_SESSION["username"]);
}
?>
```

However, if you want to completely remove the session data for a user, simply call the `session_destroy()` function to destroy the session.

Sample Code:

```
<?php
// Starting session
session_start();

// Storing session data
$_SESSION["username"] = "user1";

// Removing session data
if(isset($_SESSION["username"])){
    unset($_SESSION["username"]);
}

// Destroying session
session_destroy();
?>
```

References:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
https://www.tutorialrepublic.com/php-tutorial/php-sessions.php
http://php.net/manual/en/intro.session.php
https://www.johnmorrisonline.com/build-php-login-form-using-sessions/