INTRODUCTION TO WEB PROGRAMMING II

A

LECTURE NOTE

PREPARED

BY

MUHAMMAD SALISU ALI

FEDERAL UNIVERSITY DUTSE

16 APRIL, 2021

PHP Data Types

- PHP provides for **six** data types. Each data type is used to store a different type of data.
- An I**nteger** is a whole number that can start with a positive or negative sign.
- A **Double** value consists of a positive or negative sign, digits, an optional decimal point, and optional decimal digits.
- The **Boolean** data type is used to represent a Boolean value that is either TRUE or FALSE.
- The **String** data type contains strings that are made up of any characters. A String is coded within a single or double quotation marks.
- **Exponents** can be used for double values that represent extremely large or small values.

The Six Data types

- Integer – whole numbers that range from *-2,147,483,648* to *2,147,483,647*
- Double – Numbers with decimal places that range from *-1.7E308* to *1.7E308* with up to 16 significant digits. This data type is also known as the floating-point or decimal data type.
- Boolean – A TRUE or FALSE value.
- String – Text that consists of any characters.
- Array – A container that holds multiple values of one or more data types.
- Object – A container that contains data (properties) and functions (methods).

Examples

- **Integer values** (whole numbers)

```
15                  // an integer
-21                 // a negative integer
```

- **Double values**

```
21.5                // a floating-point number
-124.82             // a negative floating-point number
```

- **String values**

```
'Habeeb Idris'      // a string with single quotes
"Habeeb Idris"      // a string with double quotes
‘ ‘                 // an empty string
null                // a NULL value
```

- **Double values that use scientific notation**

```
3.7e9               // equivalent to 37000000000
4.5e-9              // equivalent to 0.0000000045
-3.7e9              // equivalent to -37000000000
```

How to declare variables and constants

- A variable stores a value that can change as the application executes.
- To declare a variable, code the dollar sign($) followed by the variable name. Then, to assign a value to the variable, code the assignment operator (=) followed by the value for the variable.
- PHP assigns a data type to the variable depending on the value that's assigned to the variable.
- A literal is a value that is expressed as itself. To code a numerical literal, code the value within quotation marks. To code a Boolean literal, code TRUE or FALSE. These keywords are not case-sensitive.
- A constant value does not change when the program executes.
- To declare a constant, you can use the define function to specify the name and value of the constant. A dollar sign is not used since the value of constants can't change when the program executes. By convention, many programmers use capital letters for the names of constants.

Rules for declaring variables

- Variable names are case-sensitive.
- Variable names can contain letters, numbers, and underscore.
- Variable names cannot contain special characters.
- Variable names cannot begin with a digit or two underscores.
- Variable names cannot use names that are reserved by PHP such as the variable named $this that's reserved for use with objects.

```
$count = 10;                    // an integer literal
$list_price = 9.50;             // a double literal
$first_name = 'Muhammad';       // a string literal - single quotes
$first_name = "Muhammad";       // a string literal - double quotes
$is_valid = false;              // a Boolean literal - lowercase


$product_count = $count         // $product_count is 10
$price = $list_price;           // $price is 9.50
$name = $first_name;            // $name is "Muhammad"
$is_new = $is_valid;            // $is_new is FALSE
```

How to declare a constant

- define('MAX_QTY', 100); // an integer constant
- define('PI', 3.14159265); // a double constant
- define('MALE', 'm'); // a string constant

How to get data from a request

- Most PHP applications operate on data that they get from an HTTP request.
- When you use the **HTTP GET** method with a form, the parameters are appended to the URL when the form is submitted. Then, these parameters are stored in the **$_GET** array.
- The $_GET variable is an array that contains the keys and values for the data that is passed with the HTTP request.
- When you code or enter a URL that requests a PHP page, you can add a parameter list to it starting with a **question** mark and with not intervening spaces. Then, each parameter consists of its name, an equal sign(=), and its value. To code multiple parameters, use ampersand(&) to separate the parameters.
- The <a> tag always uses the HTTP GET method when it passes parameters to a page.
- The $_GET variable is a superglobal variable, which means it's always available to the PHP code for a page.
- It's a best practice to use the PHP function like ***filter_input*** when retrieving values from a superglobal variable. This helps protect the application from malicious data from a user.

- The built-in $_GET array
- $_GET - A built-in array that contains the values passed by the GET method (default).
- **An HTML form that performs an HTTP GET request**

```
<form action="display.php" method="get">
  <label>First name:</label>
  <input type="text" name="first_name"><br >
  <label>Last name:</label>
  <input type="text" name="last_name"><br >
  <label> </label>
  <input type="submit" value="Submit">
</form>
```

- **The URL for the HTTP GET request with appended key/value pairs**

  http://localhost/.../display.php?first_name=Ray&last_name=Harris

- **The $_GET array that's created when the GET request is received**

key	Value
first_name	Ray
last_name	Harris

- **The PHP code that gets the data from the array and stores it in variables**

```
$first_name = $_GET['first_name'];
$last_name = $_GET['last_name'];
```

- **An <a> tage that performs an HTTP GET request**

```
<a href="display.php?first_name=Joel&last_name="John">Display Name</a>
```

How to use the built-in $_POST array

- When you use a <form> tag to request a PHP file, there are times when you will want to use the *HTTP POST* method for the request. When you use the *POST* method, the parameters passed to the PHP file are not shown in the URL.
- The built-in $_POST array

  $_POST - A built-in array that contains the values passed by the POST method.
- An HTML form that specifies the POST method

  ```
  <form action="display.php" method="post">
  ```
- A PHP code that gets the data from the $_POST array

  ```
  $first_name = $_POST['first_name'];

  $last_name = $_POST['last_name'];
  ```

When to use GET or POST methods

- **When to use the HTTP GET method**

  when the request is for a page that gets data from a database server.

  when the request can be executed multiple times without causing any problems.

- **When to use the HTTP POST method**

  when the request is for a page that writes data to a database server.

  when executing the request multiple times may cause problems.

  when you don't want include the parameters in the URL for security reasons.

  when you don't want users to be able to include the parameters when they bookmark a page.

  when you need to transfer more than 4 KB data.

How to work with data

- How to code string expressions

  Using single quotation marks to code a string processes more efficiently.

  An empty string is a string that doesn't contain any characters. To assign an empty string to a variable, code a set of quotation marks with not characters between.

  A null value indicates that the value is unknown. To assign a null value to a string, you can use the NULL keyword. This keyword is not *case-sensitive*.

  If you code a variable name within double quotes, it will be converted to a string.

  To join, or concatenate, two ore more strings, you can use the *concatenation operator (.)*.

  To send data to the browser, you can use the ***echo*** *statement*.

  To protect against *cross-site scripting(XSS)* attacks, you can use the ***htmlspecialchars*** function with *echo statements*.

- **How to assign string expressions**

  Use single quotes for simple strings to improve PHP efficiency

  ```
  $first_name = 'Muhammad';
  $last_name = 'Salisu';
  ```

- How to assign NULL values and empty strings

  ```
  $address2 = ‘’;              // an empty string
  $address2 = null;            // a NULL value
  ```

- How to use double quotes to insert a variable into a string

  ```
  $name = "Name:    $first_name";          // Name: Muhammad
  $name = "$first_name $last_name";       // Muhammad Salisu
  ```

- How to use single and double quotes for special purposes

  ```
  $last_name = "O'Brien";             // O'Brien
  $line = 'She said, "Hi."';          // She said, "Hi."
  ```

- How to use the concatenation operator (.) to join strings

  How to use the concatenation operator for simple joins

  ```
  $first_name = 'Muhammad';
  $last_name = 'Salisu';
  $name = 'Name: ' . $first_name;              // Name: Muhammad
  $name = $first_name . ' ' . $last_name;      // Muhammad Salisu
  ```

- How to join a number to a string

  ```
  $price = 19.99;
  $price_string = 'Price: ' . $price;        // Price: 19.99
  ```

- The syntax for the echo statement

  echo *string_expression*;

- How use an echo statement

  ```
  <p>Name: <?php echo $name;  ?></p>
  ```

- How to use an echo statement with the htmlspecialchars function

  ```
  <p>Name: <?php echo htmlspecialchars($name); ?></p>
  ```

How to code numeric expressions

- Common arithmetic operators

Operator	Description	Example	Result
+	Addition	5 + 7	12
-	Subtraction	5 - 12	-7
*	Multiplication	6 * 7	42
/	Division	13 / 4	3.25
%	Modulus	13 % 4	1
++	Increment	$counter++	adds 1 to counter
--	Decrement	$counter--	subtracts 1 from counter

- **Some simple numeric expressions**

```
$x= 14;
$y = 8;
$result = $x + $y;                  // 22
 $result = $x - $y;                 // 6
$result = $x * $y;                  // 112
 $result = $x / $y;                 // 1.75
$result = $x % $y;                  // 6
 $x++;                              // 15
 $y--;                              // 7
```

- **Statements that calculate a discount**

```
$list_price = 19.95;
$discount_percent = 20;
$discount_amount = $list_price * $discount_percent * 0.01;
$discount_price = $list_price - $discount_amount;          // 15.96
```

The order of precedence for arithmetic expressions

Order	Operators	Direction	Description
1	++	Left to right	Increment operator
2	--	Left to right	Decrement operator
3	* / %	Left to right	Multiplication, division, modulus
4	+ -	Left to right	Addition, subtraction

Examples of precedence and the use of parentheses

```
3 + 4 * 5       // 23 since the multiplication is done first
(3 + 4) * 5     // 35 since the addition is done first
```

How to use the compound assignment operators

The compound assignment operators

Operator	Description
.=	Appends a string expression to the variable.
+=	Adds the result of the numeric expression to the variable.
-=	Subtracts the result of the numeric expression from the variable.
*=	Multiplies the variable by the result of the numeric expression.
/=	Divides the variable by the result of the numeric expression.
%=	Stores the modulus of the variable and the result of the numeric expression in the variable.

To perform an operation on a variable and assign the result to the same variable, you can use a compound assignment operator(.=, +=, *=, /=, or %=).

Two ways to append string data to a variable

The standard assignment operator

```
$name = 'Ray ';
$name = $name . 'Harris';        // 'Ray Harris'
```

- A compound assignment operator

```
$name = 'Ray ';
$name .= 'Harris';               // 'Ray Harris'
```

- Three ways to increment a counter variable

The standard assignment operator

```
$counter = 1;
$counter = $counter + 1;
```

- The compound assignment operator

```
$counter = 1;
$counter += 1;         //2
```

- The increment operator

```
$counter = 1;
$counter++;          //2
```

- More examples

  How to append numeric data to a string variable

  ```
  $message = 'Months: ';
  $months = 120;
  $message .= $months;          // 'Months: 120‘
  ```

- How to work with numeric data

  ```
  $subtotal = 24.50;
  $subtotal += 75.50;          // 100
  $subtotal *= .9;             // 90(100 * .9)
  ```

How to use some built-in functions

- PHP provides many functions that you can use to work with data, this functions are often referred to as built-in functions or internal functions because they are part of PHP.
- **A function for formatting numbers**

  number_format($number [, $decimals]) - Return a number that's formatted with a comma (,). If the second parameter is specified, this function also rounds the number to the specified number of decimal places.

- **Statement that format numbers**

```
$nf = number_format(12345);          // 12,345
$nf = number_format(12345, 2);       // 12,345.00
$nf = number_format(12345.674, 2);   // 12,345.67
$nf = number_format(12345.675, 2);   // 12,345.68
```

- A function for getting the current date

  date($format) - Returns the current date with the specified format string. Common format strings characters are 'Y'(four-digit year), 'y'(two-digit year), 'm'(numeric month), and 'd'(numeric day).

- Statement that format a date

```
$date = date('Y-m-d');          //2017-10-128
$date = date('m/d/y');          //10/28/17
$date = date('m.d.Y');          //10.28.2017
$date = date('Y');              // 2017
```

Three functions for checking variable values

isset($var) - Returns a TRUE value if the variable has been set and is not a NULL value.

empty($var) - Returns a TRUE value if the variable hasn't been set, contains a NULL value, or contains an empty string.

is_numberic($var) - Returns a TRUE value if the variable is a number or a string that can be converted to a number.

Function calls that check variable values

```
isset($name)          // TRUE if $name has been set and is not NULL
empty($name)          // TRUE if $name is empty
is_numeric($price)    // TRUE if $price is a number
```

Two functions for converting user-entered data for display

htmlspecialchars($string) - Converts certain HTML special characters(&, ', ", <, and >) to their corresponding HTML entities and returns the resulting string. For example, this function converts the ampersand character(&) to the ampersand entity (&amp).

htmlspecialentities($string) - Converts all HTML characters that have corresponding HTML entities and return the resulting string.

- **The filter_input function**

filter_input($type, $variable_name [, $filter]) - Gets a value from a superglobal variable and optionally filters it. Returns the requested value on success, a FALSE value if the filter fails, or a NULL value if the requested value is not set.

The first three arguments

type - Specifies the superglobal variable to access. Common values include INPUT_GET, INPUT_POST, and INPUT_COOKIE.

variable_name - The name of the value to retrieve.

filter - Optional. The constant for the filter to apply.

- Common constants for filters

FILTER_VALIDATE_INT - validates an integer value.

FILTER_VALIDATE_FLOAT - validates a floating-point (double) value.

FILTER_VALIDATE_EMAIL - validates an email address.

FILTER_VALIDATE_URL - validates a URL.

FILTER_VALIDATE_BOOLEAN - Returns a TRUE value for "1", "true", "on", or "yes". Otherwise, it returns a FALSE value.

Statements that retrieve values from the superglobal variables

```
$product_description = filter_input(INPUT_GET, 'product_desc');
  // NULL if 'product_description' has not been set in the $_GET array
$investment = filter_input(INPUT_POST, 'investment', FILTER_VALIDATE_FLOAT);
// NULL if 'investment' has not been set in the $_POST array
// FALSE if 'investment' is not a valid float(double) value
$years = filter_input(INPUT_POST, 'years', FILTER_VALIDATE_INT);
// NULL if 'years' has not been set in the $_POST array
// FALSE if 'years' is not a valid integer value
```

How to code control statements

- PHP control statements, let you control how the statements in an application are executed.
- A conditional expression uses the relational operators to compare the results of two expressions.
- A compound conditional expression joins two ore more conditional expressions using the logical operators.
- If you use a relational operator to compare two different data types, PHP Automatically converts the data types to the same data type and attempts to perform the comparison.
- Confusing the assignment operator (=) with the equality operator (==) is a common programming error. Remember to use == for comparisons. When you use the equality operator (==), PHP converts values to the same type if necessary.
- You can use the identical operator (===) to test whether a value is both equal to another value and the same type as another value.

The relational operators

Operator	Name	Example
==	Equal	`$last_name == "Harris"` `$test_score == 10`
!=	Not equal	`$first_name != "Ray"` `$months != 0`
<	Less than	`$age < 18`
<=	Less than or equal	`$investment <= 0`
>	Greater than	`$test_score > 100`
>=	Greater than or equal	`$rate / 100 >= 0.1`
===	Identical	`$investment === FALSE` `$years === NULL`
!==	Not identical	`$investment !== FALSE` `$years !== NULL`

The logical operators in order of precedence

Operator	Name	Example
!	NOT	`!is_numeric($age)`
&&	AND	`$age > 17 && $score < 70`
||	OR	`!is_numeric($rate) || $rate < 0`

How the logical operators work

- Both tests with the *AND* operator must be TRUE for the overall test to be TRUE.
- At least one test with the *OR* operator must be TRUE for the overall test to be TRUE.
- The *NOT* operator switches the result of the expression to the other Boolean value. For example, if an expression is TRUE, the *NOT* operator converts it to FALSE.
- To override the order of precedence when two or more logical operators are used in a conditional expression, you can use *parentheses*.

How to code if statements

- An *if statement* lets you control the execution of statements based on the results of one or more conditional expressions.
- An *if statement* starts with an *if clause* and can include multiple *else if clauses* and one *else clause* at the end.
- The if clause of the statement executes one or more statements if its condition is TRUE.
- The *else if clause* is executed when the previous condition or conditions are FALSE. The statement or statements within an *else if clause* are executed if its condition is TRUE.
- The *else clause* is executed when all previous conditions are FALSE.
- You can code one *if statement* within the *if*, *else*, *if*, or *else clause* of another *if statement*. Those statements are referred to as ***nested if statements***.

- An if statement with no other clauses

```
if($price <= 0) {
   $message = 'Price must be greater than zero.';
}
```

- An if statement with an else clause

```
if( empty($first_name)) {
    $message = 'You must enter your first name.';
} else {
     $message = 'Hello' . $first_name . '!';
}
```

- An if statement with else if and else clauses

```
if( empty($investment) ) {
    $message = 'Investment is a required field.';
} else if( !is_numeric($investment) ) {
      $message = 'Investment must be a valid number.';
 } else if( $investment <= 0 ) {
  $message = 'Investment must be greater than zero.';
} else {
    $message = 'Investment is valid!';
}
```

- An if statement with a compound conditional expression

```
if ( empty($investment) || !is_numeric($investment) || $investment <= 0) {
        $message = 'Investment must be a valid number greater than zero.';
}
```

- A nested if statement

```
if( empty($months) || !is_numeric($months) || $months <= 0) {
        $message = 'Please enter a number of months greater than zero.';
} else {
        $years = $months / 12;
        if($years > 1) {
                $message = 'A long-term investment.';
        } else {
                $message = 'A short-term investment.';
        }
}
```

How to code while and for statements

- The *while* and *for statements* let you code loops that repeat a block of statements one or more times. It contains a block of code that is executed as long as a condition is TRUE. This condition is tested at the beginning of the loop. When the condition becomes FALSE, PHP skips to the code after the *while loop*.
- The *for statement* is used to create a for loop that contains a block of code that is executed a specific number of times.
- Other looping structures would be discussed in the subsequent lessons.

- A while loop that stores the numbers from 1 through 5 in a string

```
$counter = 1;
while($counter <= 5) {
    $message = $message . $counter . '|';    // appends the counter value
    $counter++;                              // adds 1 to the counter
}
```

- A for loop that stores the numbers from 1 through 5 in a string

```
for($counter = 1; $counter <= 5; $counter++ ) {
    $message = $message . $counter . '|';    // appends the counter value
 }
```

- A while loop that calculates the future value of a one-time investment

```
$nvestment = 1000;                  //$investment is $1000
$interest_rate = .1;                // yearly interest rate is 10%
$years = 25;                        // years is 25
$future_value = $investment;        // future value starts at $1000

$i = 1;
while($i <= $years) {
    $future_value = ($future_value + ($future_value * $interest_rate));
    $i++;
}
```

- A for loop that calculates the future value of a one-time investment

```
$nvestment = 1000;                    //$investment is $1000
$interest_rate = .1;                  // yearly interest rate is 10%
$years = 25;                          // years is 25
$future_value = $investment;          // future value starts at $1000

for($i = 1; $i <= $years; $i++) {
        $future_value = ($future_value + ($future_value * $interest_rate));
}
```

- The include function

```
include 'index.php';                    // parentheses are optional
include ('index.php';                   // index.php in the current directory
```

- The require function

```
require('index.php');                   // index.php in the current directory
```

- The exit function

```
exit;                                   // parentheses are optional
exit();
exit('Unable to connect to DB.');     // passes a message to the browser
```

- How to pass control to another PHP file in the current directory

```
if($is_valid) {
        include('process_data.php');              // $is_valid is true
}
```

- How to navigate up and down directories

```
include('view/header.php');         // navigate down one directory
include('./error.php');             // in the current directory
include('../error.php');            // navigate up one directory
include('../../error.php');         // navigate up two directories
```

How to code control statements

- A conditional expression uses the relational operators to compare the results of two expressions.
- A compound conditional expression joins two or more conditional expressions using the logical operators.
- If you use a relational operator to compare two different data types, PHP automatically converts the data types to the same data type and attempts to perform the comparison.
- Confusing the assignment operator (=) with the equality operator (==) is a common programming error so remember to use == for comparisons. When you use the equality operator (==), PHP converts values to the same type if necessary.
- You can use the identical operator (===) to test whether a value is both equal to another value and the same type as another value.

- Relational Operators

Operator	Name	Example
==	Equal	$last_name == "Harisu“ $test_score == 10
!=	Not equal	$first_name != "Rabiu" $months != 0
<	Less than	$age < 18
<=	Less than or equal to	$investment <= 0
>	Greater than	$test_score > 100
>=	Greater than or equal to	$rate / 100 >= 0.1
===	Identical	$investment === FALSE $years === NULL
!==	Not identical	$investment !== FALSE $years !== NULL

- How the logical operators work

  Both tests with the **AND** operator must be **TRUE** for the overall test to be TRUE. At least one test with the **OR** operator must be TRUE for the overall test to be

  TRUE.

  The **NOT** operator switches the result of the expression to the other Boolean value. For example, if an expression is TRUE, the NOT operator converts it to FALSE.

  To override the order of precedence when two or more logical operators are used in a conditional expression, you can use ***parentheses***.

The logical operators in order of precedence

Operator	Name	Example
!	Not	!is_numeric($age)
&&	And	$age > 17 && $score < 70
||	Or	!is_numeric($rate) || $rate < 0

How to code if statements

- An **if** statement lets you control the execution of statements based on the results of one or more conditional expressions.
- An if statement starts with an if clause and can include multiple else if clauses and one else clause at the end.
- The if clause of the statement executes one or more statements if its condition is TRUE.
- The **else if** clause is executed when the previous condition or conditions are FALSE. The statement or statements within an else if clause are executed if its condition is TRUE.
- The else clause is executed when all previous conditions are FALSE. You can code one if statement within the if, else if, or else clause of another if statement. Those statements are referred to as **nested if statements**.

- Examples

An if statement with no other clauses

```
if ( $price <= 0 ) {
        $message = 'Price must be greater than zero.';}
```

An if statement with an else clause

```
if ( empty($first_name) ) {
        $message = 'You must enter your first name.';
} else { $message = 'Hello ' . $first_name.'!';}
```

An if statement with else if and else clauses

```
if ( empty($investment) ) {
        $message = 'Investment is a required field.';
} else if ( !is_numeric($investment) ) {
        $message = 'Investment must be a valid number.';
} else if ( $investment <= 0 ) {
        $message = 'Investment must be greater than zero.';}
else { $message = 'Investment is valid!';}
```

An if statement with a compound conditional expression

```
if ( empty($investment) || !is_numeric($investment) || $investment <= 0 ) {
    $message = 'Investment must be a valid number greater than zero.';
}
```

A nested if statement

```
if ( empty($months) || !is_numeric($months) || $months <= 0 ) {
    $message = 'Please enter a number of months greater than zero.';
} else {
    $years = $months / 12; if ( $years > 1 ) {
    if ( $years > 1 ) {
        $message = 'A long-term investment.';
    } else {
        $message = 'A short-term investment.';
    }
}
```

How to code while and for statements

- The while and for statements let you code loops that repeat a block of statements one or more times.
- The while statement is used to create a while loop that contains a block of code that is executed as long as a condition is TRUE. This condition is tested at the beginning of the loop. When the condition becomes FALSE, PHP skips to the code after the while loop.
- The for statement is used to create a for loop that contains a block of code that is executed a specific number of times.
- A while loop that stores the numbers from 1 through 5 in a string

```
$counter = 1;
while ($counter <= 5) {
    $message = $message . $counter . '|';
    $counter++;
}
```

- A for loop that stores the numbers from 1 through 5 in a string

```
for ($counter = 1; $counter <= 5; $counter++) {
    $message = $message . $counter . '|'; // appends the counter value
}
```

How to pass control to another page

- As a PHP application executes, it moves from one web page to another. To do that, you can use the **include** and **require** functions. When that page finishes, control returns to the statement after the include or require function.
- You can use the **exit** or **die** function to exit from the PHP script that is running.

Function	Description
include($path)	Inserts and runs the specified file. if this function fails, it causes a warning that allows the script to continue. Parentheses are optional
include_once($path)	Same as include, but it makes sure that the file is included only once.
require($path)	Works the same as the include function. However, if this function fails, it causes a fatal error that stops the script.
require_once($path)	Same as require, but it makes sure that the file is only required once.
exit([$status])	Exits the current PHP script. If $status isn't supplied, the parentheses are optional. If $status is supplied, this function sends the $status string to the browser before it exits.
die([$status])	Works the same as the exit function.

- Examples

The include function

```
include 'index.php';                // parenthesis are optional
include('index.php');               // index.php in the current directory
```

The require function

```
require('index.php');           // index.php in the current directory
```

The exit function

```
exit;               // parenthesis are optional
exit();
exit('Unable to connect to DB.')    // passes a message to the browser
```

How to pass control to another PHP file in the current directory

```
if ($is_valid) {
        // if $is_valid is true
        include('process_data.php');
         exit();
}
```