Web Technology II (BIT301)

Instructor: Prakash Neupane

Strings and Arrays

Strings

- Strings in PHP:
  - Strings are sequences of characters.
  - Used to store various types of data like names, passwords, addresses, and more.
- PHP String Functions:
  - PHP provides a rich set of functions for string manipulation and handling.

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - using single quotes,
  - double quotes,
  - the here document (heredoc) format and
  - now document (nowdoc).

Strings and Arrays

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - using single quotes,

<?php *$name = 'CCT';* *echo 'Hello, $name!';* *?>* *Output:*	Properties: • Variables inside single-quoted strings are not interpolated; • they are treated as literal text.

Strings and Arrays

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - double quotes,

<?php *$name = ''CCT'';* *echo ''Hello, $name!'';* *?>* *Output:*	Properties: • Variables inside double-quoted strings are interpolated, and • their values are inserted into the string.

Strings and Arrays

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - the here document (heredoc) format

```
<?php
$heredoc = <<< EndOfQuote
Hello!
Class!
EndOfQuote;
echo $heredoc;
?>
Output:
```

Properties:

- Enclosed within <<< followed by a label (often in uppercase) and concluded with the same label on a line by itself.
- Recognizes escape sequences and interpolates variables.
- Ideal for creating multiline strings or

Strings and Arrays

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - now document (nowdoc)

```
<?php
$name = 'cct';
$nowdoc = <<<'EOD'
Hello, $name!
This is a nowdoc string.
EOD;

echo $nowdoc;
?>
Output:
```

Properties:

- Similar to heredoc but enclosed within <<< followed by a label (often in uppercase) with single quotes, i.e., <<<'EOD'.
- Does not recognize escape sequences and does not interpolate variables.
- Useful when you want to preserve the exact content, similar to single quoted

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - Variable Interpolation:
  - When you define a string literal using double quotes or a heredoc, the string is subject to variable interpolation.
  - Interpolation is the process of replacing variable names in the string with their contained values.
  - There are two ways to interpolate variables into strings.
    - Putting the variable name in a double-quoted string
    - Surround the variable being interpolated with curly braces.

Strings and Arrays

Quoting String Constants

- Ways to write a string literal in your PHP code:
  - Variable Interpolation:
  - There are two ways to interpolate variables into strings.
    - Putting the variable name in a double-quoted string
    - Surround the variable being interpolated with curly braces.

```
<?php
$who = 'Web';
$fun = 'Fun!';
echo "$who is $fun";
?>
```

```
<?php
$position = 1;
echo 'You are the {$position}st person';
?>
```

Using second syntax ensures the correct variable is interpolated. The classic use of curly

Printing Strings

- ways to send output to the browser:
  - The **echo** construct
  - **print()**
  - The **printf()** function
  - The **print_r()** function

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - The **echo** construct : Lets evaluate the following

echo”Hello”; echo”Hello!”, “Hi”;	
echo(“Hi!”);	
echo("hi", "hello ");	

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - **print()**
    - **The print() function sends one value (its argument) to the browser:**

```
print("Hello, World!");
$name = "Alice";
print("Hello, $name!");
```

```
if (print("test\n")) {
print("It worked!");
}
```

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - The **printf()** function

- printf() is used for formatted string output.
- It allows you to build a formatted string by inserting values into a template using placeholders.
- Useful for displaying data in specific formats, such as numbers with decimal places or dates.

```
<?php
$num = 42;
printf("The answer is %d.", $num);
printf('%.2f%% Complete', 2.1);
?>
```

- ***Format modifiers?***
- ***Type Specifiers?***

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - The **printf()** function

```
<?php
$month = date('m');
$day = date('d');
$year = date('Y');
printf('%02d/%02d/%04d', $month, $day, $year);
?>
```

```
<?php
$var =20.00715;
printf('%.2f', $var);
?>
<?php
```

- ***Format modifiers?***
- ***Type Specifiers?***

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - The **print_r()** function
    - The print_r() function intelligently displays what is passed to it, rather than casting everything to a string, as echo and print() do.
    - Strings and numbers are simply printed.
    - Arrays appear as parenthesized lists of keys and values, prefaced by Array:

Strings and Arrays

Printing Strings

- ways to send output to the browser:
  - The **print_r()** function

```
<?php
$a = array('name' => 'RAM',
'age' => 35,
'job' => 'Student');

print_r($a);
?>
```

```
<?php
$a = array('name' => 'RAM',
'age' => 35,
'job' => 'Student');

foreach ($a as $key => $value) {
    print("$key: $value\n");
}
?>
```

Strings and Arrays

Accessing Characters in Strings

```
<?php

$string = 'Hello';

for ($i = 0; $i < strlen($string); $i++) {

    printf("The %dth character is %s\n", $i, $string[$i]);

}

?>
```

Cleaning Strings

- Strings we get from files or users need to be cleaned up before we can use them.
- Two common problems with raw data are:
  - the presence of extraneous whitespace and
  - incorrect capitalization

Cleaning Strings

- Removing Whitespace
  - We can remove leading or trailing whitespace with the
  - trim(),
  - ltrim(), and
  - rtrim() functions:

Cleaning Strings

- Removing Whitespace
  - Example

```
<?php
$title = " Good Afternoon Class! ";
$str1 = ltrim($title);
$str2 = rtrim($title);
$str3 = trim($title);
echo "$str1,\n $str2,\n $str3 \n";
?>
```

Cleaning Strings

- Changing Case

```
<?php
$string1 = "CCT web\n";
$string2 = "bit fifth\n";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
?>
```

Strings and Arrays

Encoding and Escaping

- PHP programs interacts with
  - HTML pages,
  - web addresses (URLs), and
  - databases, along with the need for different escaping methods.

Strings and Arrays

Encoding and Escaping

- HTML Interaction:
  - PHP frequently interacts with HTML pages, which contain special characters like <, >, &, and ".
  - To prevent these characters from being interpreted as HTML tags and entities, PHP provides the htmlspecialchars() function to encode them correctly.
  - Proper HTML escaping ensures safe rendering of HTML content within web pages.

Strings and Arrays

Encoding and Escaping

- HTML Interaction:

```
<?php
$html_content = '<p>This is <b>bold</b> text.</p>';
$safe_html = htmlspecialchars($html_content);
echo $safe_html;
?>
```

Encoding and Escaping

- Web Address (URL) Handling:
  - Web addresses (URLs) often include reserved characters such as :, /, ?, &, and more.
  - When working with URLs, it's essential to use the urlencode() function to escape special characters.
  - URL encoding ensures that URLs are properly formatted and can be safely used in PHP code or sent via HTTP requests.

Strings and Arrays

Encoding and Escaping

- Web Address (URL) Handling:

```
<?php
$base_url = 'https://www.cctdharan.edu.np/';
$escaped_url = urlencode($base_url);
echo $escaped_url;
?>
```

Output:

- In the encoded URL:
  https is encoded as https.
  : is encoded as %3A.
  / is encoded as %2F.
- The dot . and other characters in the domain are not encoded because they are part of the valid URL structure.
- This encoding ensures that the URL can be safely used in various contexts, such as in HTTP requests, to avoid issues with special characters that have special meanings in URLs.

Encoding and Escaping

- Databases:
  - Most database systems require that string literals in your SQL queries be escaped.
  - SQL's encoding scheme is pretty simple—single quotes, double quotes, NUL-bytes, and backslashes need to be preceded by a backslash.
  - The ***addslashes()*** function adds these slashes, and the ***stripslashes()*** function removes them:

Strings and Arrays

Encoding and Escaping

- Databases:

```
<?php
$string = <<< EOF
"It's never going to work," she cried,
as she hit the backslash (\) key. \n
EOF;
$string = addslashes($string);
echo $string;
echo stripslashes($string);
?>
```

- The purpose of this code appears to be demonstrating how to add and remove escape slashes using the addslashes() and stripslashes() functions
- However, it's important to note that addslashes() is typically used in the context of preparing data for SQL queries to prevent SQL injection, and its usage outside of that context may not always be necessary.

comparing strings

- PHP indeed provides various operators and functions for comparing strings.
  - Equality Operator (==):
  - Identity Operator (===):

Strings and Arrays

comparing strings

- Equality Operator (==) and Identity Operator (===):

```
$str1 = "5";
$str2 = 5;

if ($str1 == $str2) {
    echo "Equal";
} else {
    echo "Not Equal";
}
```

```
$str1 = "5";
$str2 = 5;
if ($str1 === $str2) {
    echo "Equal and Same Type";
} else {
    echo "Not Equal or Different Type";
}
```

Strings and Arrays

comparing strings

- commonly used functions in PHP for comparing strings,
  - strcmp() - String Comparison:

```
<?php
$str1 = "c";
$str2 = "b";

$result = strcmp($str1, $str2);
if ($result < 0) {
    echo "$str1 is less than $str2";
} elseif ($result > 0) {
    echo "$str1 is greater than $str2";
} else {
    echo "$str1 is equal to $str2";
}

?>
```

- Compares two strings lexicographically (based on their character codes).
- Returns an integer less than 0 if the first string is less than the second, 0 if they are equal, and an integer greater than 0 if the first string is greater.

Strings and Arrays

comparing strings

- commonly used functions in PHP for comparing strings,
  - strcasecmp() - Case-Insensitive String Comparison:

```
<?php
$str1 = "file101.txt";
$str2 = "file201.txt";

$result = strnatcmp($str1, $str2);
if ($result < 0) {
    echo "$str1 is less than $str2";
} elseif ($result > 0) {
    echo "$str1 is greater than $str2";
} else {
    echo "$str1 is equal to $str2";
}

?>
```

- Compares two strings using a "natural order" algorithm, often used for sorting filenames or version numbers.
- Returns an integer less than 0 if the first string is less than the second, 0 if they are equal, and an integer greater than 0 if the first string is greater.

Strings and Arrays

comparing strings

- commonly used functions in PHP for comparing strings,
  - strnatcasecmp() - Case-Insensitive Natural Order String Comparison:

```
<?php
$str1 = "file101.txt";
$str2 = "file201.txt";

$result = strnatcmp($str1, $str2);
if ($result < 0) {
    echo "$str1 is less than $str2";
} elseif ($result > 0) {
    echo "$str1 is greater than $str2";
} else {
    echo "$str1 is equal to $str2";
}

?>
```

- Case-insensitive version of strnatcmp().
- Compares two strings using a "natural order" algorithm without considering case differences.
- Returns the same integer comparison results as strnatcmp().

Strings and Arrays

comparing strings

- commonly used functions in PHP for comparing strings,
  - strstr() - String Search:

```
<?php
$str1 = "Hello! Central Campus of Technology, BIT fifth Sem";
$key = "BIT";

$result = strstr($str1, $key);
if ($result !== false) {
    echo "Found: $result";
} else {
    echo "Not Found";
}
?>
```

- Searches for a substring within a string and returns the portion of the string starting from the first occurrence of the substring.
- Returns false if the substring is not found.

Strings and Arrays

comparing strings

- commonly used functions in PHP for comparing strings,
  - stristr() - Case-Insensitive String Search:

```
<?php
$str1 = "Hello! Central Campus of Technology,
BIT fifth Sem";
$key = "BIT";

$result = strstr($str1, $key);
if ($result !== false) {
    echo "Found: $result";
} else {
    echo "Not Found";
}
?>
```

- Case-insensitive version of strstr().
- Searches for a substring within a string without considering case differences.
- Returns false if the substring is not found.

Strings and Arrays

Quick Implementation

- Suppose you are building a login system for a website. Explain how you would use PHP to validate whether a user's input matches their stored password.
  - **Question**: How can you compare a user's input password with the stored password in PHP? Provide a code example.
- Scenario: You are working on a content management system, and you want to count the number of words in a given text input by a user.
  - **Question**: Describe the steps involved in counting the number of words in a string using PHP. Provide a code example.

Regular Expression

- Regular expressions are indeed sequences of characters that form search patterns used for text searching and manipulation.
- They can represent simple patterns like a single character or complex patterns involving combinations of characters, quantifiers, and other constructs.
- Regular expressions provide a powerful and flexible way to describe and search for specific patterns within text data.
- They are widely used in programming and text processing tasks for tasks such as data validation, searching, and text extraction.

Regular Expression

- In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers
- Syntax
  - $regex = "/hello/i";
- In the example above, / is the delimiter, w3schools is the pattern that is being searched for, and i is a modifier that makes the search case-insensitive.

Regular Expression

- PHP provides a variety of functions that allow you to use regular expressions.
- ***preg_match(),***
- ***preg_match_all()*** and
- ***preg_replace()*** functions are some of the most commonly used ones:

Regular Expression

- ***preg_match(),***

```
<?php
$str = "Visit Microsoft!";
$pattern = "/microsoft/i";
echo preg_match($pattern, $str);
?>
```

- ***The preg_match() function will tell you whether a string contains matches of a pattern.***

Regular Expression

- ***preg_match_all()***

```
<?php
$str = "Here we are reading";
$pattern = "/re/i";
echo preg_match_all($pattern, $str);
?>
```

- ***The preg_match_all() function will tell you how many matches were found for a pattern in a string.***

Regular Expression

- ***preg_replace()***

```
<?php
$str = "Visit course page via url for daily update";
$pattern = "/url/i";
echo preg_replace($pattern, "neupaneprakash.github.io/web.html", $str);
?>
```

Regular Expression

- Modifier Description: Performs a case-insensitive search

```
<?php
$pattern = '/example/i'; // Case-insensitive pattern
$text = 'This is an Example sentence with an example.';
if (preg_match($pattern, $text)) {
    echo 'Pattern found (case-insensitive).';
} else {
    echo 'Pattern not found.';
}
?>
```

Regular Expression

- Modifier Description: Performs a multiline search

```
<?php
$pattern = '/^Start/m'; // Multiline pattern
$text = "Start of line 1.\nStart of line 2.";
if (preg_match($pattern, $text)) {
    echo 'Pattern found (multiline).';
} else {
    echo 'Pattern not found.';
}
?>
```

The /m modifier makes the ^ anchor match the start of each line in a multiline string

Regular Expression

- Modifier Description: Enables correct matching of UTF-8 encoded patterns

```
<?php
$pattern = '/café/u'; // Unicode pattern
$text = 'Visit café for a coffee.';
if (preg_match($pattern, $text)) {
    echo 'Pattern found (Unicode).';
} else {
    echo 'Pattern not found.';
}
?>
```

In this case, it matches "café" with the accented "é."

Regular Expression: Metacharacters

Metacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Regular Expression: Quantifiers

- Quantifiers defines quantities

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X* *n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

Strings and Arrays

Regular Expression: How to validate peoples name?

- Humans name contains:
  - Combinations of sets of alphabets A-Z and a-z
- So name pattern follows the following regex: **'/^[A-Za-z]+$/'**
  - /: The forward slashes are delimiters that enclose the regular expression pattern.
  - ^: This caret symbol is an anchor that matches the start of the string.
  - [A-Za-z]+: This part of the pattern is a character class.
  - [...]: Square brackets define a character class, which is a set of characters to match.
  - A-Z: Matches any uppercase letter from 'A' to 'Z'.
  - a-z: Matches any lowercase letter from 'a' to 'z'.
  - (space): Matches a space character.
  - +: The plus sign quantifier means "one or more of the preceding character class or group."
  - $: This dollar sign is an anchor that matches the end of the string.

Regular Expression: How to validate peoples name?

- In summary, the regular expression '/^[A-Za-z]+$/' checks for the following conditions:
  - ^: The string must start with the beginning.
  - [A-Za-z]+: The string must contain one or more uppercase letters, lowercase letters, or spaces.
  - $: The string must end with the end

Strings and Arrays

Regular Expression: How to validate peoples name?

```
<?php
$name_pattern = '/^[A-Za-z ]+$/';
$name = 'Prakash Neupane';

if (preg_match($name_pattern, $name)) {
    echo 'Valid name.';
} else {
    echo 'Invalid name.';
}
?>
```

Regular Expression: How to validate web url?

url_pattern = '/^(https?|ftp):\/\/[^\s\/$.?#].[^\s]*$/i';

- ***^: Start Anchor***
- *The regular expression starts with the ^ symbol, which is an anchor that specifies that the pattern must match from the beginning of the string.*

Regular Expression: How to validate web url?

*url_pattern = '/^**(https?|ftp):***\/\/[^\s\/$.?#].[^\s]*$/i';*

- ***(https?|ftp): Protocol Matching***
- *This part of the expression is enclosed in parentheses (...) and is a capturing group. It matches one of the following:*
  - *http: The protocol "http."*
  - *https: The protocol "https."*
  - *ftp: The protocol "ftp."*
- *The **?** following **s** makes the **s** character in "https" optional, allowing it to match "http" as well.*

Regular Expression: How to validate web url?

*url_pattern = '/^(https?|ftp)**:\W**[^\s\/$.?#].[^\s]*$/i';*

- ***:\W** (Protocol Separator)*
  - *This part matches the characters **"://"**, which are common in URLs and separate the protocol from the rest of the URL*

Regular Expression: How to validate web url?

*url_pattern = '/^(https?|ftp):\/\/**[^\s\/$.?#].[^\s]***$/i';*

- ***[^\s\/$.?#].[^\s]****: Host and Path Matching*
- ***[^\s\/$.?#]:*** *This part is a character class [...] that matches any character that is not a whitespace character (\s), a forward slash (/), a dollar sign ($), a period (.), a question mark (?), or a hash symbol (#). In other words, it matches characters valid in a URL host.*
- ***.****: This matches any single character.*
- ***[^\s]****: This matches zero or more characters that are not whitespace.*
- *Combined,* ***[^\s\/$.?#].[^\s]**** *matches a valid host and path portion of the URL.*

Regular Expression: How to validate web url?

*url_pattern = '/^(https?|ftp):\/\/**[^\s\/$.?#].[^\s]***$/i';*

- *$: End Anchor*
- *The regular expression ends with the $ symbol, which is an anchor that specifies that the pattern must match until the end of the string.*
- *Additionally, the /i modifier is used at the end of the regular expression to make the pattern case-insensitive, so it can match URLs with different capitalizations of the protocol (e.g., "http," "HTTP," "https," "HTTPS," etc.).*

Regular Expression: How to validate web url?

```
<?php
$url_pattern = '/^(https?|ftp):\/\/[^\s\/$.?#].[^\s]*$/i';
$url = 'https://www.cctdharan.edu.np';
if (preg_match($url_pattern, $url)) {
    echo 'Valid URL.';
} else {
    echo 'Invalid URL.';
}
?>
```

Regular Expression: How to validate email address?

- '/^**[a-zA-Z0-9._%+-]**+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/'
  - %: In this context, the percent symbol % is treated as a literal character and is included within the character class **[a-zA-Z0-9._%+-].**
  - It's used to allow the percent symbol as a valid character in the local part of the email address. Some email addresses, especially in older systems or with specialized use cases, may include the percent symbol as part of the local part.

Regular Expression: How to validate email address?

- '/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+**\.**[a-zA-Z]{2,}$/'
  - ****: The backslash \ is an escape character in regular expressions. In the regular expression pattern, it is used to escape the following period (.) character.
  - The escaped period **\.** matches a literal period in the email address. The period is a special character in regular expressions, and if you want to match it literally, you need to escape it.

Strings and Arrays

Regular Expression: How to validate email address?

- '/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+**\.**[a-zA-Z]{2,}$/'
  - The {2,} in the regular expression [a-zA-Z]{2,}$/ specifies that the preceding character class [a-zA-Z] must match at least two consecutive uppercase or lowercase letters in a row at the end of the input string.
  - This is commonly used to ensure that the top-level domain (TLD) part of an email address, like ".com" or ".org," or “.np” consists of at least two letters, which is a standard requirement for TLDs in email addresses.

Strings and Arrays

Regular Expression: How to validate email address?

```
<?php
$email_pattern = '/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/';
$email = 'example@email.com.np';

if (preg_match($email_pattern, $email)) {
    echo 'Valid email address.';
} else {
    echo 'Invalid email address.';
}

?>
```

Strings and Arrays

Regular Expression: How to validate phone number?

```
<?php
$mobile_pattern = '/^\+\d{1,3}\d{10}$/';
$mobile_number = '+9779843540809';

if (preg_match($mobile_pattern, $mobile_number)) {
    echo 'Valid mobile number.';
} else {
    echo 'Invalid mobile number.';
}
?>
```

Arrays

- PHP supports both scalar and compound data types
- An array is a collection of data values organized as an ordered collection of key-value pairs.
- An array is a special variable, which can hold more than one value at a time.

Strings and Arrays

Arrays

- In PHP, the array() function is used to create an array:
  - array();
- In PHP, there are three types of arrays:
  - Indexed arrays - Arrays with a numeric index
  - Associative arrays - Arrays with named keys
  - Multidimensional arrays - Arrays containing one or more arrays

Indexed Arrays

- The keys of an indexed array are integers, beginning at 0.
- Indexed arrays are used when you identify things by their position.

```
<?php
$arr = array(1, 2, 3, 4, 5);
print_r($arr);
?>
```

Indexed Arrays

```
<?php
$arr = array();          // Declaring an array

$arr[0] = 5;     // Assigning values
$arr[1] = 6;

print_r($arr);
?>
```

Indexed Arrays

```
<?php
$arr = array('First'=>1, 2, 3);          // Declaring an array
print_r($arr);
$arrlen =count($arr);



?>
```

Indexed Arrays

```
<?php
echo"EVs steal the show at NADA Auto Show 2023 \n";
$cars = array("Tata", "Hyundai", "Kia", "Great Wall Motors", "Nissan");
print_r($cars);
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo "\n";
}
?>
```

Associative Arrays

- Associative arrays have strings as keys and behave more like two-column tables.
- The first column is the key, which is used to access the value.

Associative Arrays

```
<?php
// Declaring an array
$arr = array(
"Java" => "Spring Boot",
"Python" => "Django",
"PHP" => "Laravel"
);

print_r($arr);

?>
```

Associative Arrays

```
<?php

// Declaring an array
$arr = array();

// Declaring key-value pairs
$arr['Python'] = "Django";
$arr['Java'] = "SpringBoot";
$arr['PHP'] = "Laravel";

print_r($arr);
echo"Python use " .$arr['Python']."frameworks for web development";
?>
```

Associative Arrays

```
<?php

// Declaring an array
$arr = array();

// Declaring key-value pairs
$arr['Python'] = "Django";
$arr['Java'] = "SpringBoot";
$arr['PHP'] = "Laravel";

// implement foreach loop here to print all values of associative array
……….
……..
?>
```

Strings and Arrays

Indexed Arrays vs Associative Arrays

Indexed Array	Associative Array
The keys of an indexed array are integers which start at 0.	Keys may be strings in the case of an associative array.
They are like single-column tables.	They are like two-column tables.
They are not maps.	They are known as maps.

Storing data in arrays

- Implement the following understanding the concepts at your own: (may use internet or any other mediums)
  - Appending values to an Array
  - Assigning Range of values to an Array
  - Getting the Size/length of an Array
  - Padding an Array

Multi dimensional Arrays

- The dimension of an array indicates the number of indices you need to select an element.
  - For a two-dimensional array you need two indices to select an element
  - For a three-dimensional array you need three indices to select an element

Multi dimensional Arrays

Year	Compulsory	Elective
First Year	*10*	*0*
Second Year	*11*	*1*
Third Year	*12*	*2*
Fourth Year	*5*	*4*

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]
[3][0]	[3][1]	[3][2]

Strings and Arrays

Multi dimensional Arrays

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]
[3][0]	[3][1]	[3][2]

```
<?php
$courses = array (
  array("First Year",10,0),
  array("Second Year",11,1),
  array("Third Year",12,2),
  array("Fourth Year",5,4)
);
 echo" Curriculam Breakdown: reperesnting the concepts in 2 D array\n";
echo $courses[0][0].": Compulsory: ".$courses[0][1].", Elective: ".$courses[0][2].".\n";
echo $courses[1][0].": Compulsory: ".$courses[1][1].", Elective: ".$courses[1][2].".\n";
echo $courses[2][0].": Compulsory: ".$courses[2][1].", Elective: ".$courses[2][2].".\n";
echo $courses[3][0].": Compulsory: ".$courses[3][1].", Elective: ".$courses[3][2].".\n";
?>
```

Multi dimensional Arrays

```
<?php
$courses = array (
  array("First Year",10,0),
  array("Second Year",11,1),
  array("Third Year",12,2),
  array("Fourth Year",5,4)
);
 for ($row = 0; $row < 4; $row++)
{
  echo "Row number:" .$row;
  echo "\n";
  for ($col = 0; $col < 3; $col++) {
    echo $courses[$row][$col]."\t";
  }
  echo "\n";
}
?>
```

Strings and Arrays
Multi dimensional Arrays: Extracting values

```php
<?php
$matrix = array(
    array(1, 2, 3),
    array(4, 5, 6),
    array(7, 8, 9)
);
$extractedValues = array();
foreach ($matrix as $row) {
    foreach ($row as $value) {
        $extractedValues[] = $value;
    }
}
echo $extractedValues[5];
?>
```

Strings and Arrays

Multi dimensional Arrays: Extracting values

```php
<?php
$students = array(
    array(
        'name' => 'Alice',
        'age' => 20
    ),
    array(
        'name' => 'Bob',
        'age' => 22
    )
);

// Extracting all the names
$extractedNames = array();
foreach ($students as $student) {
    $extractedNames[ ] = $student['name'];
}
// Displaying the extracted values
echo "Names: " . implode(", ", $extractedNames) . "\n";
?>
```

Strings and Arrays
Multi dimensional Arrays: Extracting values

- Implement the following understanding the concepts at your own: (may use internet or any other mediums)
  - Slicing an Array
  - Splitting an Array into chunks
  - Keys and values
  - Checking weather an elements exist or not
  - Removing and inserting elements

Multi dimensional Arrays

- Converting Between Arrays and Variables
  - PHP provides two functions,
    - extract( ) and
    - compact( ),
  - These functions convert between arrays and variables.

Multi dimensional Arrays

- **extract( ) function:** Takes an associative array and creates variables based on the keys and values in the array

```
<?php
$data = array(
    'name' => 'Santosh',
    'age' => 20,
    'city' => 'Dharan'
);

extract($data);
echo "Name: $name, Age: $age, City: $city";
?>
```

Multi dimensional Arrays

- **compact() Function**: Does the reverse of extract().

```
<?php
$name = 'Santosh';
$age = 20;
$city = 'Dharan';

$variablesToCompact = array('name', 'age', 'city');
$data = compact($variablesToCompact);

print_r($data);
?>
```

Strings and Arrays

- **Traversing Arrays:**
  - Traversing (or iterating through) arrays is a common operation in programming, and it involves accessing and processing each element in an array.
  - PHP provides several methods for traversing arrays, including using loops and array functions.
- Here are some common ways to traverse arrays in PHP:
  - foreach Construct
  - Iterator functions
  - Using loop (for, while, do while)
  - Calling a Function for Each Array Element
  - Reducing an Array
  - Searching for Values

- **Traversing Arrays:**
  - foreach Construct

Strings and Arrays

- **Traversing Arrays:**
- Iterator functions
  - Every PHP array keeps track of the current element you're working with;
  - the pointer to the current element is known as the iterator.

```
<?php
$fruits = array("apple", "banana", "cherry");
reset($fruits); // Sets the internal pointer to the first element

echo current($fruits)."\n";
echo key($fruits)."\n";
echo next($fruits)."\n";
echo key($fruits)."\n";
echo prev($fruits)."\n";
echo key($fruits)."\n";
echo end($fruits)."\n";
echo key($fruits)."\n";
?>
```

Strings and Arrays

- **Traversing Arrays:** Using loop (like for, while, do while)

```
<?php
$fruits = array("apple", "banana", "cherry");
reset($fruits);
// Loop through the array using a while loop
while (key($fruits) !== null) {
    $index = key($fruits); // Get the current key (index)
    $fruit = current($fruits); // Get the current value
    echo "Index: $index, Value: $fruit\n";
    // Move the pointer to the next element
    next($fruits);
}
?>
```

- **Traversing Arrays: Calling a Function for Each Array Element**
  - PHP provides a mechanism, array_walk(), for calling a user-defined function once per element in an array:
  - Syntax:
    - *array_walk(array, callable);*

Strings and Arrays

- **Traversing Arrays: Calling a Function for Each Array Element**

```
<?php
$fruits = array("apple", "banana", "cherry");
function processFruit(&$fruit, $key) {
    $fruit = strtoupper($fruit);     // Convert the fruit to uppercase
}
// Use array_walk() to apply the custom function to each element
array_walk($fruits, 'processFruit');
print_r($fruits);
?>
```

Strings and Arrays

- ***Traversing Arrays: Reducing an Array***

```
<?php
$numbers = [1, 2, 3, 4, 5, 10];
// Define a callback function to sum the values
$sumCallback = function ($a, $b) {
    return $a + $b;
};
// Use array_reduce() to apply the callback and reduce the array
$result = array_reduce($numbers, $sumCallback, -5);
echo "The sum of the array is: $result";
?>
```

Strings and Arrays

- ***Traversing Arrays: Searching for Values***

```
<?php
$sub = array("Web", "Crypto", "SE", "CG");
if (in_array("SE", $sub)) {
    echo "Found SE in the array!\n";
} else {
    echo "SE not found in the array.\n";
}
if (in_array("Algo", $sub)) {
    echo "Found Algo in the array!\n";
} else {
    echo "Algo not found in the array.\n";
}
?>
```

References

- Kevin Tatroe , Peter MacIntyre, Programming PHP: Creating Dynamic Web Pages, O′Reilly, 2021