GANPAT UNIVERSITY
U. V. PATEL COLLEGE OF ENGINEERING

2CEIT6PE1
WEB TECHNOLOGY

UNIT 4
Working with Forms

Prepared by: Prof. Megha Patel (Asst. Prof in C.E Dept.)

Regular Expression

- A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character or more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replace operations.

Advantages and uses of Regular expressions

- Regular expressions help in validation of text strings which are of programmer's interest.
- It offers a powerful tool for analyzing, searching a pattern and modifying the text data.
- It helps in searching specific string pattern and extracting matching results in a flexible manner.
- With the help of in-built regexes functions, easy and simple solutions are provided for identifying patterns.
- It effectively saves a lot of development time, which are in search of specific string pattern.
- It helps in important user information validations like email address and phone numbers.
- Regexes are mostly used for checking password strength and form validations.

Syntax

- In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

```
$exp = "/gnu/i";
```

- In the example above, / is the delimiter, gnu is the pattern that is being searched for, and i is a modifier that makes the search case-insensitive.
- The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

- **Note:** By default, regular expressions are case sensitive.

Regular Expression Modifiers

- Modifiers can change how a search is performed.

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line)
u	Enables correct matching of UTF-8 encoded patterns

Regular Expression Patterns

- Brackets are used to find a range of characters

xpression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Metacharacters

- Metacharacters are characters with a special meaning.

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

Quantifiers

- Quantifiers define 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

Quantifiers

- **Note:** If your expression needs to search for one of the special characters you can use a backslash (\ ) to escape them. For example, to search for one or more question marks you can use the following expression: $pattern = '/\?+/';

Grouping

- You can use parentheses ( ) to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.

Regular Expression	Matches
gnu	The string “gnu”
^gnu	The string starts with “gnu”
gnu$	The string ends with “gnu”
^gnu$	The string where “gnu” is alone on a string.
[abc]	a,b or c
[a-z]	Any lowercase letter
[^A-Z]	Any letter which is NOT a uppercase letter
(gif|png)	Either “gif” or “png”
[a-z]+	One or more lowercase letters
^[a-zA-Z0-9]{1, }$	Any word with at least one number or one letter
([ax])([by])	ab, ay, xb, xy
[^A-Za-z0-9]	Any symbol other than a letter or other than number
([A-Z]{3}|[0-9]{5})	Matches three letters or five numbers

Regular Expression Functions

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

Function	Definition
preg_match()	This function searches for a specific pattern against some string. It returns true if pattern exists and false otherwise.
preg_match_all()	This function searches for all the occurrences of string pattern against the string. This function is very useful for search and replace.
ereg_replace()	This function searches for specific string pattern and replace the original string with the replacement string, if found.
eregi_replace()	The function behaves like ereg_replace() provided the search for pattern is not case sensitive.
preg_replace()	This function behaves like ereg_replace() function provided the regular expressions can be used in the pattern and replacement strings.

Regular Expression Functions

preg_split()	The function behaves like the PHP split() function. It splits the string by regular expressions as its parameters.
preg_grep()	This function searches all elements which matches the regular expression pattern and returns the output array.
preg_quote()	This function takes string and quotes in front of every character which matches the regular expression.
ereg()	This function searches for a string which is specified by a pattern and returns true if found, otherwise returns false.
eregi()	This function behaves like ereg() function provided the search is not case sensitive.

Example-1

- **Use a regular expression to do a case-insensitive search for "gnu" in a string**

```
<?php
$str = "Visit Gnu";
$pattern = "/gnu/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

Example-2

- **Use a regular expression to do a case-insensitive count of the number of occurrences of "ain" in a string**

Using preg_match_all()

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

```
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // Outputs 4
?>
```

Example-3

- **Use a case-insensitive regular expression to replace Microsoft with GNU in a string:**

Using preg_replace()

The preg_replace() function will replace all of the matches of the pattern in a string with another string.

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

Example-4

- **Use grouping to search for the word "banana" by looking for ba followed by two instances of na:**

```
<?php
$str = "Apples and bananas.";
$pattern = "/ba(na){2}/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

Example-5

- **Name Validation**

```
<?php
$regex = '/^[a-zA-Z ]*$/';
$nameString = 'Hemant Shah';
// Use preg_match() function to search string pattern
if(preg_match($regex, $nameString)) {
    echo("Name string matching with regular expression");
}
else {
    echo("Only letters and white space allowed in name string");
}
?>
```

Example-6

- **Mobile Validation**

```
<?php
$regex = '/^[0-9]{10}$/';
$nameString = '9734567556';
// Use preg_match() function to search string pattern
if(preg_match($regex, $nameString)) {
    echo("Correct Number");
}
else {
    echo("Incorrect");
}
?>
```

Example-7

- **Password Validation**

```
<?php
$regex = '/^(?=.*[a-z])(?=.*[0-9])(?=.*[A-Z]).{6,8}$/';
$nameString = 'D2g6';
if(preg_match($regex, $nameString)) {
    echo("Correct");
}
else {
    echo("Incorrect");
}
?>
```

Example-8

- **Email Validation**

```
<?php
$regex='/^[a-z0-9_]+@[a-z]+.[a-z]{2,3}+$/';
$nameString = 'masswe13_2@as.com';
if(preg_match($regex, $nameString)) {
   echo("Correct Email");
}
else {
   echo("Incorrect");
}
?>
```

Example-9

```
<!DOCTYPE html>
<html>
<body>

<?php
$date = "1970-01-01 00:00:00";
$pattern = "/[-\s:]/";
$components = preg_split($pattern,
$date);
print_r($components);
?>

</body>
</html>
```

```
Array (
[0] =>1970
[1] => 01
[2] => 01
[3] => 00
[4] => 00
[5] => 00 )
```

Example-10

```
<!DOCTYPE html>
<html>
<body>

<?php
$input = [
  "Red",
  "Pink",
  "Green",
  "Blue",
  "Purple"
];

$result = preg_grep("/^p/i", $input);
print_r($result);
?>

</body>
</html>
```

```
Array ( [1] => Pink [4] => Purple )
```