Regular Expression

Regular Expressions

- A regular expression pattern is a symbolic representation of the rules that are used for matching and manipulating strings.
- Regular expression patterns are enclosed in delimiters. The first character in the pattern string is considered the opening delimiter. All characters after the opening delimiter are considered part of the pattern until the next occurrence of the opening delimiter character, called the closing delimiter. Any characters after the closing delimiter called the closing delimiter. Any characters after the closing delimiter are considered to be pattern modifiers. Here is the format:
  - / pattern / modifier
  - The forward slash (/) is the delimiter
  - e.g.
    - **/^[_a-z0-9-]+/**i

    where "^[_a-z0-9-]+" is the pattern and i is the modifier
- **Regular expression patterns consist of literal characters and metacharacters, which are special characters that define the pattern matching rules in a regular expression.**

Metacharacters

- . Matches any single character
- \ Identifies the next character as a literal value
- ^ Anchors characters to the beginning of a string
- $ Anchors characters to the end of a string
- () Specifies required characters to include in a pattern match
- [] Specifies alternate characters allowed in a pattern match
- [^] Specifies characters to exclude in a pattern match
- - Identifies a possible range of characters to match
- | Specifies alternate sets of characters to include in a pattern match

Regular expression

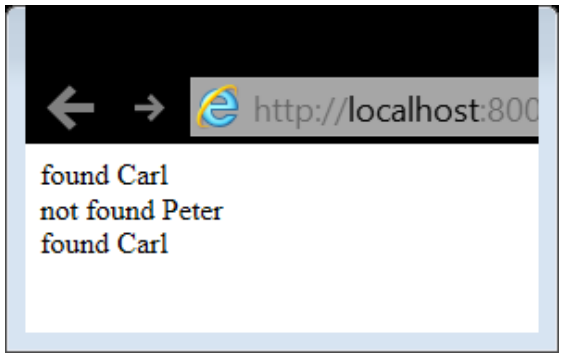
- A regular expression defines a pattern that can be searched for in a string. The pattern is case sensitive and enclosed in a forward slashes in a text string.
- To create a case-insensitive regular expression, add an I modifier to the end of the regular expression. Then, the case is ignores in both the pattern and string.
- The preg_match($pattern, $string) searches the specified string for a march to the specified regular expression. It returns 1 if the pattern is found and 0 if it's not found. If there is an error in the pattern, it returns FALSE.

The example of preg_match()

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Example of preg_match()</title>
   </head>
   <body>
      <?php
      // Create a regular expression
      $pattern = '/Buck/';
      $name1 = 'Carl Buck';
      $name2 = 'Peter Lee';

      // Search the pattern
      $match = preg_match($pattern, $name1);
      echo ($match == true)? "found Carl" : "not found Carl";
      echo '<br>';
      $match = preg_match($pattern, $name2);
      echo ($match == true)? "found Peter" : "not found Peter";
      echo '<br>';

      // Case-insensitive Search the pattern
      $match = preg_match('/carl/i', 'Carl Buck');
      echo ($match == true)? "found Carl" : "not found Carl";
      echo '<br>';
      ?>
   </body>
</html>
```



Example of metacharacters (metacharacters.php)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Metacharacters</title>
    </head>
    <body>
        <?php
        // Matching any character
        echo preg_match('/...../', "NPU") . '<br>'; // 0
        echo preg_match('/...../', "NPU76") . '<br>'; // 1
        echo preg_match('/...../', "Peter Lee") . '<br>'; // 1

        // Matching charactera at the beginning or end of a string
        $URL = "http://www.npu.edu";
        echo preg_match("/^http/", $URL) . '<br>'; // 1
        echo preg_match("/^https/", $URL) . '<br>'; // 0
        echo preg_match("/edu$/", $URL) . '<br>'; // 1
        echo preg_match("/com$/", $URL) . '<br>'; // 1
        ?>
    </body>
</html>
```

Quantifiers and subexpressions

- Quantifiers that specify the quantity of a match.

Quantifier	Description
?	Specifies that the preceding character is optional
+	Specifies that one or more of the preceding characters must match
*	Specifies that zero or more of the preceding characters can match
{n}	Specifies that the preceding character repeat exactly n times
{n,}	Specifies that the preceding character repeat at least n times
{,n}	Specifies that the preceding character repeat up to n times
{n1, n2}	Specifies that the preceding character repeat at least n1 times but no more than n2 times

- Subexpression or subpattern represents characters contained in a set of parentheses within a regular expression
- Subexpressions allow you to determine the format and quantities of the enclosed characters as a group.

Example of Quantifier and Subexpressions

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Quantifiers and Subexpressions</title>
   </head>
   <body>
      <?php
      // Using Quantifiers
      $URL = "http://www.npu.edu";
      echo preg_match("/^https?/", $URL) . '<br>'; // 1
      echo preg_match("/.+/", 'NPU') . '<br>'; // 1
      echo preg_match("/^0*/", '012345') . '<br>'; // 1
      echo preg_match("/^0*/", '12345') . '<br>'; // 1
      echo preg_match("/ZIP: .{5}$/", " ZIP: 01562") . '<br>'; // 1
      echo preg_match("/(ZIP: .{5,10})$/", "ZIP: 01562-2607") . '<br>'; // 1

      // using subexpressions
      $pattern = "/^(1 )?(\(.{3}\) )?(.{3})(\-.{4})$/";
      echo preg_match($pattern, '666-7777') . '<br>'; // 1
      echo preg_match($pattern, '(555) 666-7777') . '<br>'; // 1
      echo preg_match($pattern, '1 (555) 666-7777') . '<br>'; // 1
      ?>
   </body>
</html>
```

Matching special characters

Pattern	Matches
\\	Backslash character
\/	Forward slash
\t	Tab
\n	New line
\r	Carriage returns
\f	Form feed
\xhh	Two hexadecimal digital foe characters

- The backslash is the escape character in regular expressions. It gives special meaning to some characters and removes the special meaning from others.
- Since the backslash is the escape character in patterns, any backslash in your pattern must be preceded by another backslash. Then since the backslash is also the escape character in PHP strings, each backslash must be preceded by another backslash.
- When a character has a special meaning in a pattern, it‘s called a metacharacter.

Matching types of characters

Pattern	Matches
.	Any single character except a new line character
\w	Any letter, number, or the underscore
\W	Any letter, number, or the underscore
\d	Any digit
\D	Any character that's not a digit
\s	Any white space character (space, tab, new line, carriage return, form feed, or vertical tab)
\S	Any character that's not white space

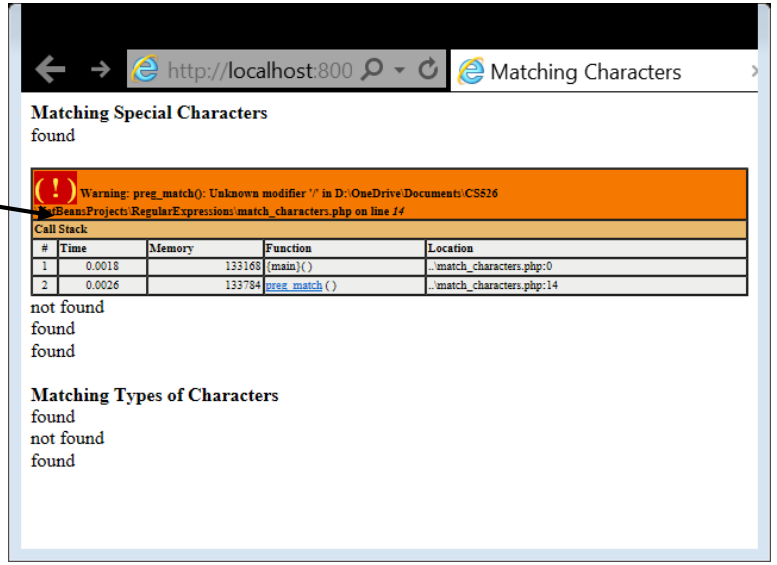
special meaning to some characters and removes the special meaning from others.

- Since the backslash is the escape character in patterns, any backslash in your pattern must be preceded by another backslash. Then since the backslash is also the escape character in PHP strings, each backslash must be preceded by another backslash.
- When a character has a special meaning in a pattern, it's called a metacharacter.

The match_characters.php

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Matching Characters</title>
  </head>
  <body>
    <?php
    echo '<strong>Matching Special Characters</strong><br>';
    $string = "© 2014 NPU. \ All student attended (10/2014).";
    $match = preg_match('/\xA9/', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    $match = preg_match('///', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    $match = preg_match('/\//', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    $match = preg_match('/\\\\/', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br><br>';

    echo '<strong>Matching Types of Characters</strong><br>';
    $string = "The part number is NPU-2015";
    $match = preg_match('/NPU./', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    $match = preg_match('/NPU\d/', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    $match = preg_match('/NPU-\d/', $string);
    echo ($match == true)? "found" : "not found";
    echo '<br>';
    ?>
  </body>
</html>
```



Using the character class

- A character class lets you match a single character from a list of possible characters.
- Most special characters lose their meaning inside the brackets. Except the caret and dash. And they are position dependent.
- You can use a bracket expression inside a character class for several predefined character range, and you can combine several bracket expressions in one character class.
- The | metacharacter that allows a string to contain an alternate set of patterns, you separate the strings in a regular expression pattern

The example of character classes

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Character Class</title>
   </head>
   <body>
      <?php
      // use [] to specify alternate characters that are allowed in a pattern match.
      echo preg_match("/analy[sz]e/", "analyse") . '<br>'; // 1
      echo preg_match("/analy[sz]e/", "analyze") . '<br>'; // 1

      // use a hyphen metacharacter (-) to specify a range of values
      $letterGrade = 'B';
      echo preg_match("/[A-DF]/", $letterGrade) . '<br>'; // 1

      // use ^ to specify optional characters to exclude in a pattern match
      echo preg_match("/[^EG-Z]/", $letterGrade) . '<br>'; // 1
      echo preg_match("/[0-9]/", "5") . '<br>'; // 1
      echo preg_match("/[^0-9]/", "5") . '<br>'; // 0

      // to use character classes to create a phone number regular expression pattern
      echo preg_match("/^(1 )?(\([0-9]{3}\) )?([0-9]{3})(\-[0-9]{4})$/", "1 (555) 666-7777") .
'<br>'; // 1

      // the e-mail validation regular expression
      $email = "peter@yahoo.com";
      echo preg_match("/^[\w-]+(\.[\w-]+)*@[\w- ]+(\.[\w-]+)*(\.[a-zA-Z]{2,})$/", $email) .
'<br>'; // 1

       // use | to have multiple choice pattern
      echo preg_match("/\.(com|org|net)$/i", $email) . '<br>'; // 1
      ?>
   </body>
</html>
```