# Web Technologies — Week 12

Mikheil Rukhaia

International Black Sea University,
Tbilisi, Georgia

December 24, 2015

# Outline

# Regular Expressions

# What is regular expression?

- Regular expressions provide a syntax to search for and match pattern of strings.
- Regular expressions are placed within delimiters, usually slashes.
- `preg_match()` function takes a regular expression and a string, and outputs true if (sub) string matches the pattern specified, or false otherwise.
- Example:

```
$myString = "Hello, world!";
preg_match("/world/", $myString);   // 1
preg_match("/WORLD/", $myString);   // 0
preg_match("/WORLD/i", $myString); // 1
```

## Syntax

- Literal strings are matched as it is.
- There are nineteen symbols
  `.` `\` `+` `*` `?` `[` `^` `]` `$` `(` `)` `{` `}` `=` `!` `<` `>` `|` `:`
  that have different meaning and must be escaped, using backslash (`\`), to get literal value.
- If the delimiter symbol is needed as a literal value, than it should be also escaped.
- Example: `preg_match("/http\:\/\//", $url);`

## Syntax (ctd.)

- To avoid too much escaping, it is possible to change the delimiter symbol.
- Example: `preg_match("#http\://#", $url);`
- `preg_quote()` takes a string and optionally the delimiter symbol, and returns the same string with any special characters escaped (useful in run-time).
- Example:

```
preg_quote("$3.5");              // "\$3\.5"
preg_quote("http://", "/");      // "http\:\/\/"
```

## Syntax (ctd.)

- Square brackets can be used to match characters from a set.
- Ranges of characters can be specified using hyphen (−).
- To match everything except the specified set, place the caret (^) symbol after the opening bracket.
- When ^ is placed after the opening delimiter, it indicates start of the matching string.
- $ placed before the closing delimiter indicates the end of the matching string.

## Syntax (ctd.)

- There are some predefined sets of characters:

| Character | Meaning |
|---|---|
| `.` | Any character. |
| `\d` | A digit. |
| `\D` | Any non-digit character. |
| `\w` | A word character (letter, digit, or underscore). |
| `\W` | Any non-word character. |
| `\s` | A whitespace character (space, tab, line feed, carriage return, or form feed). |
| `\S` | Any non-whitespace character. |

# Syntax (ctd.)

- It is also possible to quantify the patterns:

| Quantifier | Meaning |
|---|---|
| `*` | Can occur zero or more times. |
| `+` | Can occur one or more times. |
| `?` | Can occur exactly once, or not at all. |
| `{n}` | Must occur exactly `n` times. |
| `{n,}` | Must occur at least `n` times. |
| `{n,m}` | Must occur at least `n` times but no more than `m` times. |

## Syntax (ctd.)

- Complex groups can be enclosed in parenthesis.
- Alternative matches can be connected via vertical bar (`|`).
- Long regular expression can be spread on multiple lines, but then `x` modifier should be given to ignore whitespace.
- Example:

```
$url = "cu.edu.ge";
echo preg_match("/^([\w-]+\.)+[\w-]+
| \[(\d{1,3}\.){3}\d{1,3}\]$/x", $url); // 1
echo preg_match("/^([\w-]+\.)+[\w-]+
| \[(\d{1,3}\.){3}\d{1,3}\]$/", $url); // 0
```

# Syntax (ctd.)

- Lookaheads and lookbehinds can be used to assert the presence or absence of characters in a string.

- Example: what happens for `$xbc = "xbc"`, if

```
$abc = "abc";
echo preg_match("/(?<=a)b(?=c)/", $abc);    // 1
echo preg_match("/(?<!a)b(?!c)/", $abc);    // 0
echo preg_match("/(?<!a)b|b(?!c)/", $abc); // 0

$abx = "abx";
echo preg_match("/(?<=a)b(?=c)/", $abx);    // 0
echo preg_match("/(?<!a)b(?!c)/", $abx);    // 0
echo preg_match("/(?<!a)b|b(?!c)/", $abx); // 1
```

# Replacing text

- `preg_replace()` can be used to replace text using regular expressions.
- This function is very important in form data processing.
- Example:

```
$myString="Hello, World!";
preg_replace("/[^a-zA-Z]/", "", $myString);
                              // HelloWorld
preg_replace("/./", "*", $myString);
                              // ************
```

# Handling HTML Forms

## Get and Post methods

- To receive a (form) data from a user PHP has special associative arrays `$_GET` and `$_POST` corresponding to get and post methods.
- Content of these arrays depend on the data user has sent.
- Since both methods are sending data as a `name=value` pairs, the arrays contain, respectively, `name=>value` bindings.
- Example:

  `http://www.example.com/index.php?page=5`

```
$page = $_GET['page']; // returns 5
```

## Get and Post methods (ctd.)

- To check whether a field was passed to the server, use the `isset()` function.
- `isset()` takes a variable name and returns true if it has a value, otherwise returns false.
- If a form and its processing code is in the same .php file, then it is necessary to check whether form was submitted before starting the processing.
- Example:

```
if (isset($_GET['page'])) echo $_GET['page'];
else echo "Please specify page number in URL";
```

## Processing Form Data

- To check inputs, there are two kind of techniques: whitelisting and blacklisting.
- Whitelisting means that you match the content to allowed characters and if it contains any other character, output an error message.
- Blacklisting is opposite, where you check for unwanted characters.
- It is always better to use whitelisting (whenever applicable), since you do not create a security hole by forgetting some special characters in a blacklist.

# Processing Form Data (ctd.)

- Another technique is filtering, which means that you do not reject the input, but remove unwanted characters.

- Variation of filtering is to type cast, to ensure the correct input:
  `$page = (int) $_GET['page']`

- Example:

```
$page = $_GET['page'];
if (preg_match("/^[0-9]*$/", $page)
                              echo "whitelisting";
if (preg_match("/[<>&%]/", $page)
                              echo "blacklisting";
$page = preg_replace("/[^0-9]/", "", $page);
echo "filtering $page";
```

# Processing Form Data (ctd.)

- Example: HTML part

```
<form action="" method="post">
 Username:
 <input type="text" name="username"/><br />
 Password:
 <input type="password" name="password"/><br />
 <input type="submit" name="login"/>
</form>
```

# Processing Form Data (ctd.)

- Example: PHP part

```
<?php
 if (isset($_POST['login'])) {
  if ($_POST['name'] == "mrukhaia" and
      $_POST['username'] == "1234")
      echo "Welcome Mikheil";
  else echo "Sorry $_POST['name'],
    you cannot log in";
 }
?>
```

## Handling empty fields

- If checkboxes, radiobuttons, listboxes, and the like, are not selected, no data is sent to the server for these fields.
- Consequently, there is no corresponding elements in `$_GET` or `$_POST` and accessing them will generate an error message.
- Thus it is always best to use `isset()` on these controls as well.
- Example:

```
if (isset($_POST['checkbox']))
                echo "checkbox selected";
if (isset($_POST['radiobutton']))
                echo $_POST['radiobutton'];
```

## Multi-value fields

- Multi-value fields send several values for the same name.
- To process such fields, the trick is needed: add `[]` symbols at the end of control name.
- PHP will then generate an array for the field to access all the values.
- Example:

```
<form action="" method="post">
 <input type="checkbox" name="cb[]" value="1"/>
 <input type="checkbox" name="cb[]" value="2"/>
 <input type="submit" value="OK" name="ok"/>
</form>
<?php foreach ($_POST['cb'] as $cb) echo $cb;?>
```

# File uploads

- Recall: the method must be `post` and `enctype="multipart/form-data"`
- When file is uploaded, `$_FILES` are created, that is an array of associative arrays.
- For each file element there is `name`, `size`, `type`, `tmp_name` and `error` values.
- To restrict the size of files to be uploaded, use hidden control, directly before file upload, with `name="MAX_FILE_SIZE"` and set `value` to maximum allowed size in bytes.
- Use `move_uploaded_file()` to keep the file on the server.

# File uploads (ctd.)

- Example:

```
<form action="" method="post"
      enctype="multipart/form-data">
 <input type="hidden" name="MAX_FILE_SIZE"
                     value="100000"/>
 <input type="file" name="photo"/>
 <input type="submit" name="ok" value="OK"/>
</form>
<?php
 if (move_uploaded_file(
            $_FILES['photo']['tmp_name'],
        basename($_FILES['photo']['name'])))
   echo "Successfully uploaded.";
 else
   echo "Problem when uploading, try again.";
?>
```

# Rewrite Rules

# URL vs URI

- URL is used to locate a resource on the Internet.
- URL has several parts:

$$\underbrace{\texttt{http://}}_{\text{scheme}}\underbrace{\texttt{www.logic.at}}_{\text{host address}}\underbrace{\texttt{/staff/mrukhaia/}}_{\text{URI}}$$

- URI is anything after the host address, consisting of

$$\underbrace{\texttt{/asap/index.php}}_{\text{file path}}\underbrace{\texttt{?page=1}}_{\text{query string}}$$

## Purpose

- Hiding the underlying functionality of PHP and thus exposing less of the site internals.
- Creating clean and readable URLs that are easier to remember and aid in search engine optimization.
- It makes illusion of security.

# Expressions

- `mod_rewrite` uses regular expressions to match URI patters.
- There are several syntax difference in regular expressions used in `mod_rewrite`:
  - No delimiter is needed before/after expression.
  - Exclamation mark (`!`) placed in front of an expression negates it.

## `RewriteRule` command

- `RewriteRule` directive lets you rewrite a requested URI into some other URI.
- The rules can appear in Apache main configuration file, or in `.htaccess` file in your directory.
- In the first case, pattern is applied to the `REQUEST_URI`, which is the URL in the address bar, without schema, hostname and query string parts.
- If a rule is defined in `.htaccess` file, then it is relative to the current directory.

## `RewriteRule` command (ctd.)

- Rewrite rules have the following syntax:

```
RewriteRule pattern target_uri [flag,flag,...]
```

- Example:

```
RewriteEngine on
RewriteRule  ^index\.html$  ./index.php [L]
```

- Flags are not mandatory, but it is recommended to always use them to avoid unexpected default behavior.

# Flags

- Most frequently used flags are:

| Flag | Meaning |
|---|---|
| `F` | forces an HTTP 403 Forbidden status code. |
| `G` | forces an HTTP 410 Gone status code. |
| `L` | indicates the end of the rewriting process. |
| `NC` | makes the rule case insensitive. |
| `PT` | treats the rewrite target as URL |
| `QSA` | append query string back to URL. |
| `R` | redirects the browser to the new URL (HTTP 302 code). |

## `RewriteCond` command

- Rewrite rules have the following syntax:

```
RewriteCond test_string pattern [flags]
```

- Example:

```
RewriteEngine On
RewriteCond %{REQUEST_URI} !^/index.php [NC]
RewriteRule (.*) ./index.php?file=$1 [PT,L]
```

- In the example above, `$1` is a backreference, storing the original value of the requested URL.

## `RewriteCond` command (ctd.)

- It is possible to specify several conditions for one rule.
- In this case, the conditions are connected by AND logical operator.
- To change this behavior the `OR` flag should be provided.
- Example:

```
RewriteEngine On
RewriteCond %{REQUEST_URI} !^/index.php [OR]
RewriteCond %{REQUEST_URI} !^/index.html
RewriteRule (.*) ./index.php?file=$1 [PT,L]
```

# Laboratory Work

# Exercises

- Write a regular expression matching proper e-mails only.
- Write a regular expression matching proper URLs only.
- Assemble your previous lab works as one web site, create an `.htaccess` file and experiment with rewrite rules.

# Discussion?!