Chapter 15

How to use regular expressions, handle exceptions, and validate data

Objectives

Applied

1. Create and use regular expressions.
2. Create and throw exceptions.
3. Catch and handle exceptions and errors.

Objectives (continued)

Knowledge

1. Describe the creation of a regular expression, and the processing that's done by the preg_match() function.
2. Describe the use of case-insensitive, multiline, and global regular expressions.
3. Describe the use of the preg_replace() and preg_split() functions that work with regular expressions.
4. Describe how regular expressions can be used for data validation such as validating a social security number.
5. Describe the way exceptions are created, thrown, and handled.
6. Describe the type of errors that PHP 7 lets you catch and how you catch and handle them.

A function for matching a regular expression

```
preg_match($pattern, $string)
```

How to create a regular expression

```
$pattern = '/Harris/';
```

Two strings to test

```
$author = 'Ray Harris';
$editor = 'Joel Murach';
```

How to search for the pattern

```
$author_match = preg_match($pattern, $author);
// $author_match is 1

$editor_match = preg_match($pattern, $editor);
// $editor_match is 0
```


How to test for errors in a regular expression

```
if ($author_match === false) {
    echo 'Error testing author name.';
} else if ($author_match === 0) {
    echo 'Author name does not contain Harris.';
} else {
    echo 'Author name contains Harris.';
}
```


A case-insensitive regular expression

```
$pattern = '/murach/i';
```

How to use a case-insensitive regular expression

```
$editor_match = preg_match($pattern, $editor);
// $editor_match is 1
```

Key terms

- regular expression
- pattern

Patterns for special characters

Pattern	Matches
\\	Backslash character
\/	Forward slash
\t	Tab
\n	New line
\r	Carriage return
\f	Form feed
\xhh	The Latin-1 character whose value is the two hexadecimal digits

Matching special characters

```
$string =
"© 2017 Mike's Music. \ All rights reserved (5/2017).";

preg_match('/\xA9/', $string)
// Matches © and returns 1

preg_match('//', $string)
// Returns FALSE and issues a warning

preg_match('/\//', $string)
// Matches / and returns 1

preg_match('/\\\\/', $string)
// Matches \ and returns 1
```


Patterns for character types

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

Matching character types

```
$string = 'The product code is MBT-3461.';

preg_match('/MB./', $string)
// Matches MBT and returns 1

preg_match('/MB\d/', $string)
// Matches nothing and returns 0

preg_match('/MBT-\d/', $string)
// Matches MBT-3 and returns 1
```


Using the character class

```
$string = 'The product code is MBT-3461.';

preg_match('/MB[TF]/', $string)
// Matches MBT and returns 1

preg_match('/[.]/', $string)
// Matches . and returns 1

preg_match('/[13579]/', $string)
// Matches 3 and returns 1
```

Using metacharacters

```
preg_match('/MB[^TF]/', $string)
// Matches nothing and returns 0

preg_match('/MBT[^^]/', $string)
// Matches MBT- and returns 1

preg_match('/MBT-[1-5]/', $string)
// Matches MBT-3 and returns 1

preg_match('/MBT[_*-]/', $string)
// Matches MBT- and returns 1
```


Using bracket expressions

```
preg_match('/MBT[[:punct:]]/', $string)
// Matches MBT- and returns 1

preg_match('/MBT[[:digit:]]/', $string)
// Matches nothing and returns 0

preg_match('/MB[[:upper:]]/', $string)
// Matches MBT and returns 1
```


Patterns for string positions

Pattern	Matches
^	The beginning of the string (use \^ to match a caret)
$	The end of the string (use \$ to match a dollar sign)
\b	The beginning or end of a word (must not be inside brackets)
\B	A position other than the beginning or end of a word

Matching string positions

```
$author = 'Ray Harris';

preg_match('/^Ray/', $author)
// Returns 1

preg_match('/Harris$/', $author)
// Returns 1

preg_match('/^Harris/', $author)
// Returns 0

$editor = 'Anne Boehm';

preg_match('/Ann/', $editor)
// Returns 1

preg_match('/Ann\b/', $editor)
// Returns 0
```


Matching subpatterns

```
$name = 'Rob Robertson';

preg_match('/^(Rob)|(Bob)\b/', $name)
// Returns 1

preg_match('/^(\w\w\w) \1/', $name)
// Returns 1
```


Matching repeating patterns

```
$phone = '559-555-6627';

preg_match('/^\d{3}-\d{3}-\d{4}$/', $phone)
// Returns 1

$fax = '(559) 555-6635';

preg_match('/^\(\d{3}\) ?\d{3}-\d{4}$/', $fax)
// Returns 1

$phone_pattern  =
    '/^(\d{3}-)|(\(\d{3}\) ?)\d{3}-\d{4}$/';

preg_match($phone_pattern, $phone)
// Returns 1

preg_match($phone_pattern, $fax)
// Returns 1
```

Look-ahead assertions

```
(?=[[:digit:]])
// The next character in the pattern must be a digit

(?=.*[[:digit:]])
// The pattern must contain at least one digit
```

A look-ahead assertion

```
$pattern = '/^(?=.*[[:digit:]])[[:alnum:]]{6}$/';

preg_match($pattern, 'Harris')
// Assertion fails and returns 0

preg_match($pattern, 'Harri5')
// Matches and returns 1
```


A negative look-ahead assertion

```
$pattern = '/^(?!3[2-9])[0-3][[:digit:]]$/';

preg_match($pattern, '32')
// Assertion fails and returns 0

preg_match($pattern, '31')
// Matches and returns 1
```


A pattern to enforce password complexity

```
$pw_pattern =
'/^(?=.*[[:digit:]])(?=.*[[:punct:]])[[:print:]]{6,}$/';
```

The parts of the pattern

```
^                   // start of the string
(?=.*[[:digit:]])   // at least one digit
(?=.*[[:punct:]])   // at least one punctuation character
[[:print:]]{6,}     // six or more printable characters
$                   // nothing else
```

Using the pattern

```
$password1 = 'sup3rsecret';
$password2 = 'sup3rse(ret';

preg_match($pw_pattern, $password1)
// Assertion fails and returns 0

preg_match($pw_pattern, $password2)
// Matches and returns 1
```


How to work with a multiline regular expression

```
$string   = "Ray Harris\nAuthor";

$pattern1 = '/Harris$/';
preg_match($pattern1, $string);
// Does not match Harris and returns 0

$pattern2 = '/Harris$/m';
preg_match($pattern2, $string);
// Matches Harris and returns 1
```


How to work with a global regular expression

```
$string  = 'MBT-6745 MBT-5712';
$pattern = '/MBT-[[:digit:]]{4}/';

$count = preg_match_all($pattern, $string, $matches);
// Count is 2

foreach ($matches[0] as $match) {
    echo '<div>' . $match . '</div>';
    // Displays MBT-6745 and MBT-5712
}
```


How to use the preg_replace() function to replace a pattern with a string

```
$items = 'MBT-6745 MBS-5729';
$items = preg_replace('/MB[ST]/', 'ITEM', $items);

echo $items;       // Displays ITEM-6745 ITEM-5729
```

How to use the preg_split() function to split a string on a pattern

```
$items =
    'MBT-6745 MBS-5729, MBT-6824, and MBS-5214';
$pattern = '/[, ]+(and[ ]*)?/';
$items = preg_split($pattern, $items);

// $items contains:
// 'MBT-6745', 'MBS-5729', 'MBT-6824', 'MBS-5214'

foreach ($items as $item) {
    echo '<li>' . $item . '</li>';
}
```


Regular expressions for testing validity

Phone numbers as: 999-999-9999

```
/^[[:digit:]]{3}-[[:digit:]]{3}-[[:digit:]]{4}$/
```

Credit card numbers as: 9999-9999-9999-9999

```
/^[[:digit:]]{4}(-[[:digit:]]{4}){3}$/
```

Zip codes as either: 99999 or 99999-9999

```
/^[[:digit:]]{5}(-[[:digit:]]{4})?$/
```

Dates as: mm/dd/yyyy

```
/^(0?[1-9]|1[0-2])\/(0?[1-9]|[12][[:digit:]]|3[01])\/
[[:digit:]]{4}$/     // on one line with no spaces
```


Testing a phone number for validity

```
$phone = '559-555-6624';
$phone_pattern =
  '/^[[:digit:]]{3}-[[:digit:]]{3}-[[:digit:]]{4}$/';
$match = preg_match($phone_pattern, $phone);
// Returns 1
```


Testing a date for a valid format, but not for a valid month, day, and year

```
$date = '8/10/209';         // invalid date
$date_pattern = '/^(0?[1-9]|1[0-2])\/'
      . '(0?[1-9]|[12][[:digit:]]|3[01])\/'
      . '[[:digit:]]{4}$/';
$match = preg_match($date_pattern, $date);
// Returns 0
```


Complete email address validation

```
function valid_email ($email) {
    $parts = explode("@", $email);
    if (count($parts) != 2 ) return false;
    if (strlen($parts[0]) > 64) return false;
    if (strlen($parts[1]) > 255) return false;

    $atom = '[[:alnum:]_!#$%&\'*+\/=?^`{|}~-]+';
    $dotatom = '(\.' . $atom . ')*';
    $address = '(^' . $atom . $dotatom . '$)';
    $char = '([^\\\\"])';
    $esc  = '(\\\\[\\\\"])';
    $text = '(' . $char . '|' . $esc . ')+';
    $quoted = '(^"' . $text . '"$)';
    $local_part = '/' . $address . '|' . $quoted . '/';
    $local_match = preg_match($local_part, $parts[0]);
    if ($local_match === false
        || $local_match != 1) return false;
```


Complete email address validation (continued)

```
    $hostname =
        '([[:alnum:]]([-[:alnum:]]{0,62}[[:alnum:]])?)';
    $hostnames = '(' . $hostname .
                 '(\.' . $hostname . ')*)';
    $top = '\.[[:alnum:]]{2,6}';
    $domain_part = '/^' . $hostnames . $top . '$/';
    $domain_match = preg_match($domain_part, $parts[1]);
    if ($domain_match === false
        || $domain_match != 1) return false;

    return true;
}
```

The syntax for creating a new Exception object

```
new Exception($message [, $code])
```

The syntax for the throw statement

```
throw $exception;
```


A function that may throw an exception

```
function calculate_future_value(
       $investment, $interest_rate, $years) {
    if   ($investment <= 0 ||
          $interest_rate <= 0 ||
          $years <= 0 ) {
      throw new Exception(
              "All arguments must be greater than zero.");
    }

    $future_value = $investment;
    for ($i = 1; $i <= $years; $i++) {
        $future_value +=
                $future_value * $interest_rate * .01;
    }
    return round($futureValue, 2);
}
```

A statement that causes an exception

```
$futureValue =
    calculate_future_value(10000, 0.06, 0);
```


Methods of Exception objects

```
getMessage()
getCode()
getFile()
getLine()
getTrace()
getTraceAsString()
```


The syntax for a try/catch statement

```
try { statements }
catch (ExceptionClass $exceptionName) { statements }
[ catch (ExceptionClass $exceptionName) { statements } ]...
```

A statement that catches an Exception object

```
try {
    $fv =  calculate_future_value(10000, 0.06, 0);
    echo 'Future value was calculated successfully.';
} catch (Exception $e) {
    echo 'Exception: ' . $e->getMessage();
}
```


A statement that rethrows an Exception object

```
try {
    $fv = calculate_future_value(
        $investment, $annual_rate, $years);
} catch (Exception $e) {
    throw $e;
}
```

A statement that catches two types of exceptions

```
try {
    $db =
        new PDO($dsn, 'mmuser', 'pa55word', $options);
    // other statements
} catch (PDOException $e) {
    echo 'PDOException: ' . $e->getMessage();
} catch (Exception $e) {
    echo 'Exception: ' . $e->getMessage();
}
```


Code that catches all errors and exceptions (PHP 7 and later)

```
try {
    $fv = calc_future_value(10000, 0.06, 9);
} catch (Exception $e) {
    echo 'Error: ' . $e->getMessage();
} catch (Error $e) {
    echo 'Error: ' . $e->getMessage();
}
```

The message for the error above

```
Error: Call to undefined function calc_future_value()
```

A more concise way to catch all errors and exceptions

```
try {
    $fv = calculate_future_value(10000, 0.06, 9);
} catch (Throwable $e) {
    echo 'Error: ' . $e->getMessage();
}
```

Code that catches a ParseError (PHP 7 and later)

```
try {
    require 'calculations.php';
} catch (ParseError $e) {
    echo 'Required file not included: '
        . $e->getMessage();
}
```

The message for the error above

```
Required file not included: syntax error, unexpected '}',
expecting ';'
```

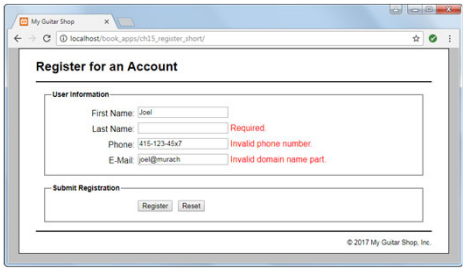

Code that catches a TypeError (PHP 7 and later)

```
try {
    $average = avg_of_3(5.1, 2.7, 8.2);
} catch (TypeError $e) {
    echo 'Error: ' . $e->getMessage();
}
```

The message for the error above

```
Error: Argument 1 passed to avg_of_3() must be of the
type integer, float given
```


The user interface for the Registration application



The file structure

```
app_root/
  model/
    fields.php      The Field and Fields classes
    validate.php    The Validate class
  view/
    header.php      The HTML and PHP for the header
    register.php    The HTML and PHP for the form view
    success.php     The HTML and PHP for the success view
    footer.php      The HTML and PHP for the footer
  index.php         The PHP for the controller
  main.css          The CSS for the application
```


model/fields.php

```
<?php
class Field {
    private $name;
    private $message = '';
    private $hasError = false;

    public function __construct($name, $message = '') {
        $this->name = $name;
        $this->message = $message;
    }

    public function getName()
        { return $this->name; }
    public function getMessage()
        { return $this->message; }
    public function hasError()
        { return $this->hasError; }
```


model/fields.php (continued)

```
    public function setErrorMessage($message) {
        $this->message = $message;
        $this->hasError = true;
    }

    public function clearErrorMessage() {
        $this->message = '';
        $this->hasError = false;
    }

    public function getHTML() {
        $message = htmlspecialchars($this->message);
        if ($this->hasError()) {
            return '<span class="error">' .
                   $message . '</span>';
        } else {
            return '<span>' . $message . '</span>';
        }
    }
}
```

model/fields.php (continued)

```
class Fields {
    private $fields = array();

    public function addField($name, $message = '') {
        $field = new Field($name, $message);
        $this->fields[$field->getName()] = $field;
    }

    public function getField($name) {
        return $this->fields[$name];
    }

    public function hasErrors() {
        foreach ($this->fields as $field) {
            if ($field->hasError()) {return true; }
        }
        return false;
    }
}
?>
```


model/validate.php

```
<?php
class Validate {
    private $fields;

    public function __construct() {
        $this->fields = new Fields();
    }

    public function getFields() {
        return $this->fields;
    }
```


model/validate.php (continued)

```
    // Validate a generic text field
    public function text($name, $value, $required = true,
                         $min = 1, $max = 255) {
        // Get Field object
        $field = $this->fields->getField($name);

        // If not required and empty, clear errors
        if (!$required && empty($value)) {
            $field->clearErrorMessage();
            return;
        }
        // Check field and set or clear error message
        if ($required && empty($value)) {
            $field->setErrorMessage('Required.');
        } else if (strlen($value) < $min) {
            $field->setErrorMessage('Too short.');
        } else if (strlen($value) > $max) {
            $field->setErrorMessage('Too long.');
        } else {
            $field->clearErrorMessage(); }
    }
```


model/validate.php (continued)

```
    // Validate a field with a generic pattern
    public function pattern($name, $value, $pattern,
            $message, $required = true) {
        // Get Field object
        $field = $this->fields->getField($name);

        // If not required and empty, clear errors
        if (!$required && empty($value)) {
            $field->clearErrorMessage();
            return;
        }
        // Check field and set or clear error message
        $match = preg_match($pattern, $value);
        if ($match === false) {
            $field->setErrorMessage('Error testing field.');
        } else if ( $match != 1 ) {
            $field->setErrorMessage($message);
        } else {
            $field->clearErrorMessage();
        }
    }
```


model/validate.php (continued)

```
    public function phone(
    $name, $value,$required = false) {
        $field = $this->fields->getField($name);

        // Call the text method
        // and exit if it yields an error
        $this->text($name, $value, $required);
        if ($field->hasError()) { return; }

        // Call the pattern method
        // to validate a phone number
        $pattern =
        '/^[[:digit:]]{3}-[[:digit:]]{3}-[[:digit:]]{4}$/';

        $message = 'Invalid phone number.';
        $this->pattern(
            $name, $value, $pattern, $message, $required);
    }
```


model/validate.php

```
    public function email($name, $value, $required = true) {

        $field = $this->fields->getField($name);

        // If not required and empty, clear errors
        if (!$required && empty($value)) {
            $field->clearErrorMessage();
            return;
        }

        // Call the text method
        // and exit if it yields an error
        $this->text($name, $value, $required);
        if ($field->hasError()) { return; }
```

model/validate.php (continued)

```
// Split email address on @ sign and check parts
$parts = explode('@', $value);
if (count($parts) < 2) {
    $field->setErrorMessage('At sign required.');
    return;
}
if (count($parts) > 2) {
    $field->setErrorMessage(
        'Only one at sign allowed.');
    return;
}
$local = $parts[0];
$domain = $parts[1];
```


model/validate.php (continued)

```
// Check lengths of local and domain parts
if (strlen($local) > 64) {
    $field->setErrorMessage('Username too long.');
    return;
}
if (strlen($domain) > 255) {
    $field->setErrorMessage(
        'Domain name part too long.');
    return;
}
```


model/validate.php (continued)

```
// Patterns for address formatted local part
$atom = '[[:alnum:]_!#$%&\'*+\/=?^`{|}~-]+';
$dotatom = '(\.' . $atom . ')*';
$address = '(^' . $atom . $dotatom . '$)';

// Patterns for quoted text formatted local part
$char = '([^\\\\"])';
$esc  = '(\\\\[\\\\"])';
$text = '(' . $char . '|' . $esc . ')+';
$quoted = '(^"' . $text . '"$)';

// Combined pattern for testing local part
$localPattern =
    '/' . $address . '|' . $quoted . '/';

// Call the pattern method and exit if error
$this->pattern($name, $local, $localPattern,
               'Invalid username part.');
if ($field->hasError()) { return; }
```


model/validate.php (continued)

```
        // Patterns for domain part
        $hostname =
            '([[:alnum:]]([-[:alnum:]]{0,62}[[:alnum:]])?)';
        $hostnames =
            '(' . $hostname . '(\.' . $hostname . ')*)';
        $top = '\.[[:alnum:]]{2,6}';
        $domainPattern = '/^' . $hostnames . $top . '$/';

        // Call the pattern method
        $this->pattern($name, $domain, $domainPattern,
                'Invalid domain name part.');
    }
}
?>
```


The controller (index.php)

```
<?php
require_once('model/fields.php');
require_once('model/validate.php');

// Add fields with optional initial message
$validate = new Validate();
$fields = $validate->getFields();
$fields->addField('first_name');
$fields->addField('last_name');
$fields->addField('phone', 'Use 888-555-1234 format.');
$fields->addField('email', 'Must be a valid email.');

$action = filter_input(INPUT_POST, 'action');
if ($action === NULL) {
    $action = 'reset';
} else {
    $action = strtolower($action);
}
```


The controller (index.php) (continued)

```
switch ($action) {
    case 'reset':

        // Reset values for variables
        $first_name = '';
        $last_name = '';
        $phone = '';
        $email = '';

        // Load view
        include 'view/register.php';
        break;
```

The controller (index.php) (continued)

```
    case 'register':

        // Copy form values to local variables
        $first_name = trim(filter_input(INPUT_POST,
        'first_name'));
        $last_name = trim(filter_input(INPUT_POST,
        'last_name'));$phone = trim(filter_input(INPUT_POST,
        'phone'));
        $email = trim(filter_input(INPUT_POST, 'email'));

        // Validate form data
        $validate->text('first_name', $first_name);
        $validate->text('last_name', $last_name);
        $validate->phone('phone', $phone);
        $validate->email('email', $email);
```


The controller (index.php) (continued)

```
        // Load appropriate view based on hasErrors
        if ($fields->hasErrors()) {
            include 'view/register.php';
        } else {
            include 'view/success.php';
        }
        break;
}
?>
```


The view (view/register.php)

```
<?php include 'header.php'; ?>
<main>
    <form action="." method="post" >
    <fieldset>
        <legend>User Information</legend>

        <label>First Name:</label>
        <input type="text" name="first_name"
               value="<?php echo
                   htmlspecialchars($first_name);?>">
        <?php echo $fields->getField('first_name')
                        ->getHTML(); ?><br>

        <label>Last Name:</label>
        <input type="text" name="last_name"
               value="<?php echo
                   htmlspecialchars($last_name);?>">
        <?php echo $fields->getField('last_name')
                        ->getHTML(); ?><br>
```


The view (view/register.php) (continued)

```
        <label>Phone:</label>
        <input type="text" name="phone"
               value="<?php echo
                   htmlspecialchars($phone);?>">
        <?php echo $fields->getField('phone')
                        ->getHTML(); ?><br>

        <label>E-Mail:</label>
        <input type="text" name="email"
               value="<?php echo
                   htmlspecialchars($email);?>">
        <?php echo $fields->getField('email')
                        ->getHTML(); ?><br>
    </fieldset>
```


The view (view/register.php) (continued)

```
    <fieldset>
        <legend>Submit Registration</legend>

        <label> </label>
        <input type="submit" name="action"
               value="Register">
        <input type="submit" name="action"
               value="Reset"><br>
    </fieldset>
    </form>
</main>
<?php include 'footer.php'; ?>
```


A long version of the Registration application