PHP Development

- Classes
- Traits
- File system
- Form handling
- Project structure
- PHP MySQL

Classes

- Classes are the blueprint for objects.
- Objects are instances of classes.
- Classes can have properties and methods.

```
<?php
  class Car {
    public $color;
    public $model;
  }

  $car1 = new Car();
  $car1->color = "red";
  $car1->model = "Toyota";

  echo $car1->color; // red
  echo $car1->model; // Toyota
?>
```

Properties

- Properties are variables that hold data.
- Properties are declared with the `public`, `protected`, or `private` keyword.
- Properties can have default values.

```
<?php
  class Car {
    public $color = "red";
    public $model = "Toyota";
    protected $year = 2021;
    private $price = 10000;
  }

  $car1 = new Car();
  echo $car1->color; // red
  echo $car1->model; // Toyota
  echo $car1->year; // Fatal error: Uncaught Error: Cannot access protected property Car::$year
  echo $car1->price; // Fatal error: Uncaught Error: Cannot access private property Car::$price
?>
```

```php
<?php
  class Car {
    private $model;

    public function setModel($model) {
      $this->model = $model;
    }

    public function getModel() {
      return $this->model;
    }
  }

  $car1 = new Car();
  $car1->setModel("Toyota");
  echo $car1->getModel(); // Toyota
?>
```

```php
<?php
  class Car {
    protected $model;

    public function setModel($model) {
      $this->model = $model;
    }

    public function getModel() {
      return $this->model;
    }
  }

  class Toyota extends Car {
    public function showModel() {
      return $this->model;
    }
  }

  $toyota = new Toyota();
  $toyota->setModel("Toyota");
  echo $toyota->getModel(); // Toyota
  echo $toyota->showModel(); // Toyota
?>
```

- The `protected` keyword allows the property to be accessed by the class and its subclasses.
- The `private` keyword allows the property to be accessed only by the class.
- The `public` keyword allows the property to be accessed by any class.

Traits

- Traits are used to group functionality in a fine-grained and consistent way.
- Traits are similar to classes, but they are intended to group functionality in a fine-grained way.

```php
<?php
  trait Drivable {
    public function drive() {
      echo "Driving...";
    }
  }

  class Toyota {
    use Drivable;
  }

  $toyota = new Toyota();
  $toyota->drive(); // Driving...
?>
```

Form Handling

- Form handling is the process of capturing user input from a web form.
- The `$_GET` and `$_POST` superglobals are used to collect form data.
- The `$_GET` method is used to collect form data after submitting an HTML form with the `GET` method.
- The `$_POST` method is used to collect form data after submitting an HTML form with the `POST` method.

```php
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    $email = $_POST["email"];
    echo "Name: $name, Email: $email";
  }
?>

<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  Name: <input type="text" name="name"><br>
  Email: <input type="text" name="email"><br>
  <input type="submit">
</form>
```

- The `$_REQUEST` superglobal is used to collect form data after submitting an HTML form.
- The `$_REQUEST` method can collect data from both the `GET` and `POST` methods.

```php
<?php
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_REQUEST["name"];
    $email = $_REQUEST["email"];
    echo "Name: $name, Email: $email";
  }
?>

<form method="get" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  Name: <input type="text" name="name"><br>
  Email: <input type="text" name="email"><br>
  <input type="submit">
</form>
```

File System

- PHP has several functions for creating, reading, updating, and deleting files.
- The `fopen()` function is used to open a file.
- The `fwrite()` function is used to write to a file.
- The `fclose()` function is used to close a file.

```php
<?php
  $file = fopen("file.txt", "w");
  fwrite($file, "Hello, World!");
  fclose($file);
?>
```

- The `fread()` function is used to read a file.
- The `feof()` function is used to check if the end of a file has been reached.

```php
<?php
  $file = fopen("file.txt", "r");
  while (!feof($file)) {
    echo fgets($file);
  }
  fclose($file);
?>
```

Save array to csv

```php
<?php
  $data = array(
    array("John", "Doe", 25),
    array("Jane", "Smith", 30)
  );

  $file = fopen("file.csv", "w");
  foreach ($data as $row) {
    fputcsv($file, $row);
  }
  fclose($file);
?>
```

Read csv

```php
<?php
  $file = fopen("file.csv", "r");
  while (($row = fgetcsv($file)) !== false) {
    print_r($row);
  }
  fclose($file);
?>
```

Save array to json

```php
<?php
  $data = array(
    array("name" => "John", "age" => 25),
    array("name" => "Jane", "age" => 30)
  );

  $json = json_encode($data);
  file_put_contents("file.json", $json);
?>
```

Read json

```php
<?php
  $json = file_get_contents("file.json");
  $data = json_decode($json, true);
  print_r($data);
?>
```

Project Structure

- A typical PHP project structure might look like this:

```
project/
├── app/
│   ├── controllers/
│   ├── models/
│   ├── views/
├── public/
│   ├── css/
│   ├── js/
│   ├── index.php
├── vendor/
├── composer.json
├── composer.lock
```

- The `app` directory contains the application code.
- The `public` directory contains the public files (e.g., CSS, JavaScript, images).
- The `vendor` directory contains the third-party libraries.
- The `composer.json` file contains the project dependencies.
- The `composer.lock` file contains the exact versions of the dependencies.
- The `index.php` file is the entry point of the application.
- The `controllers` directory contains the controller classes.
- The `models` directory contains the model classes.
- The `views` directory contains the view files.
- The `css` directory contains the CSS files.
- The `js` directory contains the JavaScript files.

Use Composer

- Composer is a dependency manager for PHP.
- Composer is used to manage the project dependencies.
- Composer is used to autoload the classes.

```
composer require monolog/monolog
```

```php
<?php
  require 'vendor/autoload.php';

  use Monolog\Level;
  use Monolog\Logger;
  use Monolog\Handler\StreamHandler;

  $log = new Logger('name');
  $log->pushHandler(new StreamHandler('app.log', Level::Warning));

  $log->warning('Foo', ['foo' => 'bar']);
  $log->error('Bar', ['bar' => 'foo']);
?>
```

PHP MySQL

- PHP can be used to interact with MySQL databases.
- The `mysqli` extension is used to connect to MySQL databases.
- The `mysqli_connect()` function is used to connect to a MySQL database.
- The `mysqli_query()` function is used to execute a SQL query.
- The `mysqli_fetch_assoc()` function is used to fetch a result row as an associative array.

```php
<?php
  $conn = mysqli_connect("localhost", "root", "", "test");

  if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
  }

  $sql = "SELECT * FROM users";
  $result = mysqli_query($conn, $sql);

  if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_assoc($result)) {
      print_r($row);
    }
  }

  mysqli_close($conn);
?>
```

- The `mysqli_real_escape_string()` function is used to escape special characters in a string.
- The `mysqli_insert_id()` function is used to get the ID generated in the last query.

```php
<?php
  $name = mysqli_real_escape_string($conn, $_POST['name']);
  $email = mysqli_real_escape_string($conn, $_POST['email']);

  $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";

  if (mysqli_query($conn, $sql)) {
    $last_id = mysqli_insert_id($conn);
    echo "New record created successfully. Last inserted ID is: " . $last_id;
  } else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
  }
?>
```

- The `mysqli_error()` function is used to get the last error message.
- The `mysqli_close()` function is used to close the connection.
- The `mysqli_num_rows()` function is used to get the number of rows in a result set.
- The `mysqli_fetch_array()` function is used to fetch a result row as an associative array, a numeric array, or both.

```php
<?php
  $sql = "SELECT * FROM users";
  $result = mysqli_query($conn, $sql);

  if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_array($result)) {
      print_r($row);
    }
  }
?>
```

- The `mysqli_fetch_row()` function is used to fetch a result row as a numeric array.
- The `mysqli_fetch_object()` function is used to fetch a result row as an object.

```php
<?php
  $sql = "SELECT * FROM users";
  $result = mysqli_query($conn, $sql);

  if (mysqli_num_rows($result) > 0) {
    while ($row = mysqli_fetch_object($result)) {
      print_r($row);
    }
  }
?>
```

- The `mysqli_fetch_all()` function is used to fetch all result rows as an associative array.
- The `mysqli_fetch_assoc()` function is used to fetch a result row as an associative array.
- The `mysqli_fetch_array()` function is used to fetch a result row as an associative array, a numeric array, or both.

```php
<?php
  $sql = "SELECT * FROM users";
  $result = mysqli_query($conn, $sql);

  $rows = mysqli_fetch_all($result, MYSQLI_ASSOC);
  print_r($rows);
?>
```

Prepared Statements

- Prepared statements are used to execute the same SQL query repeatedly with high efficiency.
- Prepared statements are used to prevent SQL injection attacks.

```php
<?php
  $stmt = $conn->prepare("INSERT INTO users (name, email) VALUES (?, ?)");
  $stmt->bind_param("ss", $name, $email);

  $name = "John";
  $email = "john@doe.com";
  $stmt->execute();

  $name = "Jane";
  $email = "jane@doe.com";
  $stmt->execute();

  $stmt->close();
?>
```

- The `bind_param()` function is used to bind variables to a prepared statement.
- The `execute()` function is used to execute a prepared statement.
- The `close()` function is used to close a prepared statement.
- The `bind_result()` function is used to bind result variables to a prepared statement.

```php
<?php
  $stmt = $conn->prepare("SELECT name, email FROM users WHERE id = ?");
  $stmt->bind_param("i", $id);

  $id = 1;
  $stmt->execute();
  $stmt->bind_result($name, $email);

  while ($stmt->fetch()) {
    echo "Name: $name, Email: $email";
  }

  $stmt->close();
?>
```

MySQL Security

- Use prepared statements to prevent SQL injection attacks.
- Use `mysqli_real_escape_string()` to escape special characters in a string.
- Use `htmlspecialchars()` to prevent XSS attacks.

```
<?php
  $name = mysqli_real_escape_string($conn, $_POST['name']);
  $email = mysqli_real_escape_string($conn, $_POST['email']);

  $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
?>
```

```
<?php
  $name = htmlspecialchars($_POST['name']);
  $email = htmlspecialchars($_POST['email']);

  $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
?>
```

- Applying both `mysqli_real_escape_string()` and `htmlspecialchars()` is the best practice. Creating a function to escape special characters and prevent XSS attacks.

```
<?php
  function escape($conn, $value) {
    return htmlspecialchars(mysqli_real_escape_string($conn, $value));
  }

  $name = escape($conn, $_POST['name']);
  $email = escape($conn, $_POST['email']);

  $sql = "INSERT INTO users (name, email) VALUES ('$name', '$email')";
?>
```