

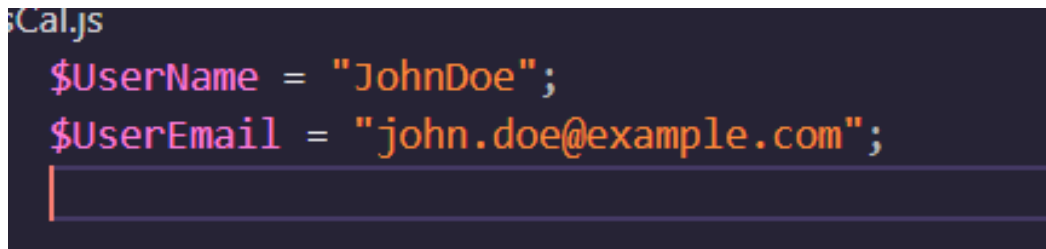
Coding Conventions

1 Consistency

Adopt **PascalCase** across the entire codebase to ensure clarity and uniformity. Follow consistent standards in naming, indentation, and overall structure in PHP, HTML, CSS, JavaScript, and SQL.

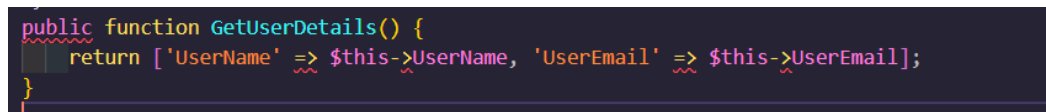
2 Naming Conventions

- **Variables (PHP and JavaScript):** Use **PascalCase** for variable names. Ensure names are descriptive and meaningful.



```
Cal.js
$username = "JohnDoe";
$email = "john.doe@example.com";
```

- **Functions and Methods:** Name functions and methods using **PascalCase**, with each word starting with a capital letter. The function name should clearly describe the action it performs.



```
public function getUserDetails() {
    return ['UserName' => $this->UserName, 'Email' => $this->Email];
}
```

- **Classes:** Class names should follow **PascalCase**.

```
class User {  
    // class content here  
}
```

- **Database Tables and Columns (MySQL):** Use **PascalCase** for naming tables and columns.

```
CREATE TABLE Users (  
    UserId INT AUTO INCREMENT PRIMARY KEY,  
    UserName VARCHAR(100) NOT NULL,  
    UserEmail VARCHAR(100) NOT NULL  
);
```

- **Files and Directories:** All files and directories should be named using **PascalCase**.

```
ProjectRoot/  
├── HealthMonitoring/  
│   ├── DatabaseConfig.php  
│   ├── UtilityFunctions.php  
│   ├── PatientRecords.php  
│   └── HealthReports.php  
├── Education/  
│   ├── DatabaseConfig.php  
│   ├── UtilityFunctions.php  
│   ├── StudentProfile.php  
│   └── CourseManager.php  
├── Security/  
│   ├── DatabaseConfig.php  
│   ├── UtilityFunctions.php  
│   ├── LoginHandler.php  
│   └── UserProfile.php  
├── Css/  
│   └── MainStyles.css  
└── Js/  
    ├── AttendanceScript.js  
    ├── HealthScript.js  
    ├── EducationScript.js  
    └── SecurityScript.js
```

3 Comments and Documentation

- **PHP DocBlocks:** Every function, class, and method must be documented using DocBlocks. The documentation should use **PascalCase** and include details on the purpose, parameters, and return values.

```

/**
 * GetUserDetails returns user details.
 * @return array Returns user details like UserName and UserEmail.
 */
public function GetUserDetails() {
    return ['UserName' => $this->UserName, 'UserEmail' => $this->UserEmail];
}

```

- **Inline Comments:** Use inline comments to clarify complex logic in the code.

```

1 // Check if the user is logged in before fetching details
2 if ($IsLoggedIn) {
3     $details = $user->GetUserDetails();
4 }
5

```

4 HTML Format

- **HTML Elements:** Use **PascalCase** for HTML element IDs and class names.

```

<div id="MainContainer">
    <div id="PageHeader" class="Header">Welcome to the Sample Page</div>
    <div id="UserDetails">
        <p id="UserName">JohnDoe</p>
        <p id="UserEmail">john.doe@example.com</p>
    </div>/#UserDetails
</div>/#MainContainer

```

5 CSS Format

- **CSS Selectors:** All CSS selectors should follow **PascalCase**.

```

.MainContainer {
    padding: 20px;
    background-color: #f9f9f9;
}

.Header {
    font-size: 24px;
    color: #333;
}

```

6 Exception Handling

Use try-catch blocks to handle exceptions in a graceful and organized manner.

```

try {
    if (document.getElementById('UserName').textContent === "") {
        throw new Error("UserName is empty.");
    }
} catch (error) {
    console.error("Error:", error.message);
}

```

```

try {
    if (document.getElementById('UserName').textContent === "") {
        throw new Error("UserName is empty.");
    }
} catch (error) {
    console.error("Error:", error.message);
}

```

7 Routing in PHP

Use **PascalCase** for routing endpoints to maintain consistency with the naming conventions.

```
// Routing endpoint example
$router->get('/UserDetails', function() {
    // handle the route
});
```

8 Indentation

- Use **4 spaces** per indentation level to ensure readability across all languages (JavaScript, PHP, HTML, CSS, and MySQL).
- **Spaces** are preferred over tabs for consistency across different editors and platforms.

```
def CalculateTotal(Amount, TaxRate):
    Total = Amount + (Amount * TaxRate)
    return Total
```

9 Maximum Line Length and Line Breaks

- Limit all lines to a maximum of **79 characters** to ensure readability on smaller screens.
- Indent continued lines appropriately to maintain logical structure.
- When breaking a line, ensure the break occurs in a logical place (preferably before or after an operator or other meaningful divider), maintaining readability.

- For new code, breaking lines before operators is recommended when possible, to improve clarity.

```
let result = price + tax + discount + shippingCost;
```

10 Blank Lines

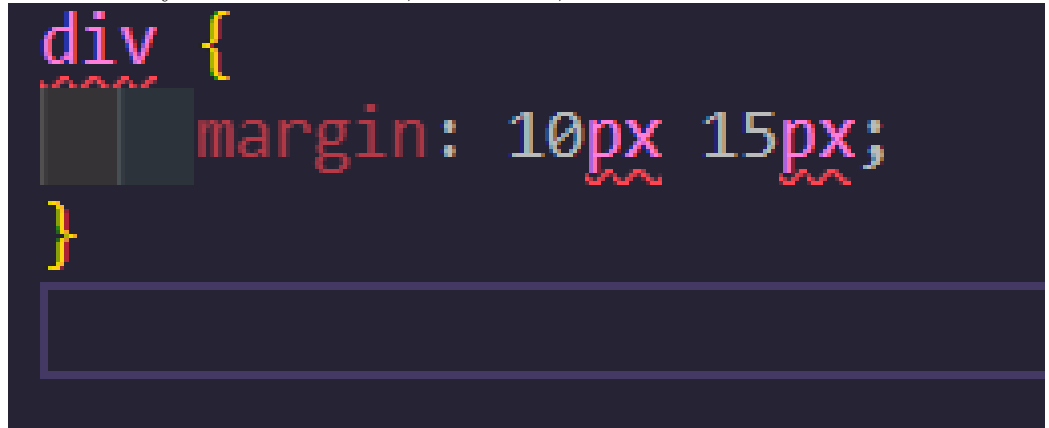
- Use **two blank lines** around top-level function and class definitions to visually separate them from other code.
- Use **one blank line** between method definitions inside a class to separate methods logically.
- Use blank lines sparingly to separate groups of related functions. Avoid excessive blank lines, especially between closely related functions.
- Use blank lines inside functions to indicate logical sections but do so sparingly

```
/**
 * Calculate discount for the given price.
 *
 * @param float $price The original price.
 * @return float The discounted price.
 */
function calculateDiscount($price) {
    return $price * 0.9;
}
```

11 Whitespace in Expressions and Statements

- Avoid extraneous whitespace in the following situations

- Immediately inside parentheses, brackets, or braces.
- Between a trailing comma and a following close parenthesis.
- Immediately before a comma, semicolon, or colon.



- If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority. Never use more than **one space** around an operator, and ensure that the amount of whitespace is consistent on both sides of a binary operator.

12 Documentation Strings (Docstrings)

- Write **docstrings** for all public modules, functions, classes, and methods in **PHP, JavaScript, and MySQL**. These should describe their purpose, parameters, and return values.
- Non-public methods should have comments that describe what they do.
- For one-liner docstrings, the closing `"""` should be kept on the same line.


```

/**
 * Calculate discount for the given price.
 * @param float $Price The original price.
 * @return float The discounted price.
 */
function calculateDiscount($Price) {
    return $Price * 0.9;
}

```

13 Comments

- Keep comments up-to-date with the code. Comments that contradict the code are worse than no comments at all.
- Always ensure comments are meaningful and describe the logic or reasoning behind code, especially if the logic is complex.
- Use comments to clarify sections of code that might be non-intuitive or tricky.

```

// Calculate the final price including tax
let FinalPrice = Price * (1 + TaxRate); // Add tax to price

```

14 JavaScript Specific Guidelines

- Use **PascalCase** for class names and constructor functions.
- Use **PascalCase** for variables, functions, and methods.
- Always use **Const** or **Let** for variable declarations instead of **Var**.
- Avoid using **Eval()** and **With()** for better security and maintainability.
- Add a space after **If**, **For**, **While**, etc., before the opening parenthesis.

```
class Product {  
    constructor(Name, Price) {  
        this.Name = Name; // PascalCase for properties  
        this.Price = Price;  
    }  
  
    calculatePriceWithTax() {  
        return this.Price * 1.1;  
    }  
}
```

15 PHP Specific Guidelines

- Use **PascalCase** for class names.
- Use **PascalCase** for functions, methods, and variables.
- Always use `<?php` for opening PHP tags.
- Use `Echo` or `Print` for output, and prefer `Echo` for simple outputs.
- Indent all code blocks consistently, and never mix spaces and tabs.

```

<?php
class Product {
    public $ProductName;

    public function SetProductName($Name) {
        $this->ProductName = $Name;
    }

    public function GetProductName() {
        return $this->ProductName;
    }
}
?>

```

16 HTML Specific Guidelines

- Use **PascalCase** for HTML element IDs and class names.
- Attribute names should be in lowercase (e.g., `class="my-class"`).
- Always quote attribute values, even if they are numeric.
- Use semantic tags (`Header`, `Footer`, `Article`, `Section`) wherever possible for better accessibility and SEO.
- Avoid inline styling; use external CSS files instead.

```

<div id="MainContainer" class="ProductDetails">
    <p class="ProductName">Product A</p>
</div>/#MainContainer.ProductDetails

```

17 CSS Specific Guidelines

- Use **PascalCase** for class names and IDs.
- Use **BEM** (Block, Element, Modifier) naming convention for classes.

- Indent nested CSS rules using 4 spaces.
- Always include `Box-Sizing: Border-Box;` for all elements for consistent layout calculations.
- Avoid using `!Important` unless absolutely necessary.

```
.ProductContainer {  
    padding: 20px;  
    background-color: #f9f9f9;  
}  
  
.ProductName {  
    font-size: 24px;  
}
```

18 MySQL Specific Guidelines

- Use **PascalCase** for table names and column names.
- Always use `Inner Join` or `Left Join` for joining tables explicitly.
- Write `Select` statements in uppercase.
- Prefer `Limit` and `Offset` over `Select *` for performance optimization.
- Always include proper indexing on frequently searched columns.

```
CREATE TABLE Products (  
    ProductId INT AUTO_INCREMENT PRIMARY KEY,  
    ProductName VARCHAR(100) NOT NULL,  
    ProductPrice DECIMAL(10, 2) NOT NULL  
);  
  
SELECT ProductName, ProductPrice FROM Products WHERE ProductName = 'Laptop';
```