Author: Lokang Jackson

Book Name: PHP and MySQL

declaration

In PHP, declaring variables and constants follows specific syntax and rules. Here's an overview of the various declarations you can perform in PHP:

Variable Declaration

Variables in PHP are declared by assigning a value to a variable name prefixed with a dollar sign ($). PHP is a loosely typed language, so you don't need to declare the type of the variable explicitly.

```
<?php
// String variable
$name = "John Doe";
// Integer variable
$age = 30;
// Float variable
$height = 5.9;
// Boolean variable
$is_student = true;
// Array variable
$colors = array("red", "green", "blue");
// Associative array variable
$user = array("name" => "Jane Doe", "age" => 25);
// Null variable
$unknown = null;
?>
```

Constant Declaration

Constants are declared using the define() function or the const keyword. Constants are case-sensitive by default and cannot be changed once they are set.

Using define()

```
<?php
define("SITE_NAME", "My Website");
define("MAX_USERS", 100);
// Accessing constants
echo SITE_NAME; // Outputs "My Website"
echo MAX_USERS; // Outputs 100
?>
```

Using const

```
<?php
```

```
const SITE_URL = "https://www.example.com";
const TIMEOUT = 30;
// Accessing constants
echo SITE_URL; // Outputs "https://www.example.com"
echo TIMEOUT;  // Outputs 30
?>
```

Declaring Constants in Classes

Constants can also be declared within classes.

```
<?php
class MyClass {
   const CONSTANT_VALUE = 'A constant value';
   public function showConstant() {
       echo self::CONSTANT_VALUE;
   }
}
// Accessing class constants
echo MyClass::CONSTANT_VALUE; // Outputs "A constant value"
$myObject = new MyClass();
$myObject->showConstant(); // Outputs "A constant value"
?>
```

Type Declarations

From PHP 7 onwards, you can specify the type of function arguments, return values, and properties.

Function Argument Type Declarations

```
<?php
function addNumbers(int $a, int $b): int {
   return $a + $b;
}
echo addNumbers(5, 10); // Outputs 15
?>
```

Return Type Declarations

```
<?php
function getGreeting(): string {
   return "Hello, World!";
}
echo getGreeting(); // Outputs "Hello, World!"
?>
```

Nullable Type Declarations

You can allow a function argument or return type to be null by prefixing the type with a question mark (?).

```php
<?php
function setAge(?int $age) {
   if (is_null($age)) {
       echo "Age is not provided.";
   } else {
       echo "Age is $age.";
   }
}
setAge(25);  // Outputs "Age is 25."
setAge(null); // Outputs "Age is not provided."
?>
```

Class Property Type Declarations (PHP 7.4+)

```php
<?php
class Person {
   public string $name;
   public int $age;
   public function __construct(string $name, int $age) {
       $this->name = $name;
       $this->age = $age;
   }
}
$person = new Person("Alice", 30);
echo $person->name; // Outputs "Alice"
echo $person->age;  // Outputs 30
?>
```

These examples cover various declarations in PHP, demonstrating how to declare and use variables, constants, and type hints in functions and classes.

Index

In PHP, an array index refers to the location of a value within an array. The index is used to identify the position of an element within the array.

In PHP, arrays are zero-indexed, which means that the first element of the array has an index of 0. For example, if you have an array called **$fruits**, the first element of the array would be **$fruits[0]**. The second element would be **$fruits[1]**, and so on.

Here is an example of an array in PHP with three elements:

```php
$fruits = array("apple", "banana", "cherry");
echo $fruits[0];  // Outputs "apple"
echo $fruits[1];  // Outputs "banana"
echo $fruits[2];  // Outputs "cherry"
```

You can also use negative indices to access elements of the array, starting from the end. For example, **$fruits[-1]** would refer to the last element of the array, **$fruits[-2]** would refer to the second to last element, and so on.

```
$fruits = array("apple", "banana", "cherry");
echo $fruits[-1];  // Outputs "cherry"
echo $fruits[-2];  // Outputs "banana"
echo $fruits[-3];  // Outputs "apple"
```

You can also use numeric indices that are larger than the size of the array. In this case, PHP will create empty elements in the array to fill the gap between the end of the array and the specified index.

```
$fruits = array("apple", "banana", "cherry");
$fruits[5] = "grapes";
print_r($fruits);
// Outputs: Array ( [0] => apple [1] => banana [2] => cherry [5] =>
grapes )
```

Associated

In PHP, an associated array is an array with string keys rather than numeric keys. An associated array can be created by using the **array** function and specifying the key-value pairs as arguments. For example:

```
$student = array(
  "first_name" => "John",
  "last_name" => "Doe",
  "age" => 25
);
```

This creates an associated array with three elements, each of which has a string key and a value. The keys are "first_name", "last_name", and "age", and the corresponding values are "John", "Doe", and 25, respectively.

You can access the elements of an associated array using the keys as indices. For example:

```
echo $student["first_name"];  // Outputs "John"
echo $student["last_name"];  // Outputs "Doe"
echo $student["age"];  // Outputs 25
```

You can also use a loop to iterate over the elements of an associated array and process each element in turn. For example:

```
foreach ($student as $key => $value) {
  echo "$key: $value\n";
}
```

This would output something like:

```
first_name: John
last_name: Doe
```

```
age: 25
```

Multidimensional

In PHP, a multidimensional array is an array that contains other arrays as its elements. It is an array of arrays, where each element of the main array is itself an array.

Here is an example of a multidimensional array in PHP:

```
$students = array(
  array(
     "first_name" => "John",
     "last_name" => "Doe",
     "age" => 25
  ),
  array(
     "first_name" => "Jane",
     "last_name" => "Doe",
     "age" => 23
  ),
  array(
     "first_name" => "Bob",
     "last_name" => "Smith",
     "age" => 30
  )
);
```

This creates a multidimensional array called **$students** that contains three elements, each of which is itself an associated array.

You can access the elements of a multidimensional array using multiple indices. For example:

```
echo $students[0]["first_name"];  // Outputs "John"
echo $students[1]["last_name"];  // Outputs "Doe"
echo $students[2]["age"];  // Outputs 30
```

You can also use loops to iterate over the elements of a multidimensional array. For example:

```
foreach ($students as $student) {
  foreach ($student as $key => $value) {
     echo "$key: $value\n";
  }
  echo "\n";
}
```

This would output something like:

```
first_name: John
last_name: Doe
```

```
age: 25
first_name: Jane
last_name: Doe
age: 23
first_name: Bob
last_name: Smith
age: 30
```

Arithmatics

In PHP, you can use the following arithmetic operators:

Addition (+): This operator is used to add two operands. For example, $a + $b will add the value of $a to the value of $b.

Subtraction (-): This operator is used to subtract one operand from another. For example, $a - $b will subtract the value of $b from the value of $a.

Multiplication (*): This operator is used to multiply two operands. For example, $a * $b will multiply the value of $a by the value of $b.

Division (/): This operator is used to divide one operand by another. For example, $a / $b will divide the value of $a by the value of $b.

Modulus (%): This operator is used to find the remainder when one operand is divided by another. For example, $a % $b will give the remainder when the value of $a is divided by the value of $b.

Exponentiation (**): This operator is used to raise one operand to the power of another. For example, $a ** $b will raise the value of $a to the power of $b.

Example:

```
<?php
$a = 10;
$b = 20;
$c = $a + $b; // 30
$d = $b - $a; // 10
$e = $a * $b; // 200
$f = $b / $a; // 2
$g = $b % $a; // 0
$h = $a ** $b; // 100000000000000000000
echo $c . "\n";
echo $d . "\n";
echo $e . "\n";
echo $f . "\n";
echo $g . "\n";
echo $h . "\n";
?>
```

Assignment

In PHP, you can use the assignment operator (=) to assign a value to a variable. For example:

```
$x = 10;
```

This will assign the value 10 to the variable **$x**.

You can also use compound assignment operators to perform an operation and assign the result to a variable in a single statement. For example:

```
$x = 10;
$x += 5; // equivalent to $x = $x + 5;
```

This will add 5 to the value of **$x**, which is 10, and assign the result (15) to **$x**.

Here are the compound assignment operators available in PHP:

- **+=**: Addition assignment
- **-=**: Subtraction assignment
- ***=**: Multiplication assignment
- **/=**: Division assignment
- **%=**: Modulus assignment
- **.=**: Concatenation assignment

Example:

```
<?php
$x = 10;
$y = 20;
$x += 5; // 15
$y -= 5; // 15
$x *= 2; // 30
$y /= 2; // 10
$x %= 3; // 0
$y .= " apples"; // "10 apples"
echo $x . "\n";
echo $y . "\n";
?>
```

Comparison

In PHP, you can use comparison operators to compare two values and determine whether they are equal, greater than, or less than each other.

Here are the comparison operators available in PHP:

- **==**: Equal to
- **!=**: Not equal to
- **>**: Greater than

- **<**: Less than
- **>=**: Greater than or equal to
- **<=**: Less than or equal to

These operators return a boolean value of **true** or **false** depending on the result of the comparison.

Example:

```
<?php
$x = 10;
$y = 20;
$result1 = $x == $y; // false
$result2 = $x != $y; // true
$result3 = $x > $y; // false
$result4 = $x < $y; // true
$result5 = $x >= $y; // false
$result6 = $x <= $y; // true
echo $result1 . "\n";
echo $result2 . "\n";
echo $result3 . "\n";
echo $result4 . "\n";
echo $result5 . "\n";
echo $result6 . "\n";
?>
```

You can also use the **===** operator to compare two values not only for equality, but also for their data types. This operator will return **true** only if both values are equal and of the same data type.

Example:

```
<?php
$x = 10;
$y = "10";
$result1 = $x == $y; // true
$result2 = $x === $y; // false
echo $result1 . "\n";
echo $result2 . "\n";
?>
```

Logical

In PHP, you can use logical operators to combine multiple conditions and perform different actions based on those conditions.

Here are the logical operators available in PHP:

- **&&**: Logical AND
- **||**: Logical OR
- **!**: Logical NOT

The **&&** operator will return **true** only if both operands are **true**. The **||** operator will return **true** if

at least one of the operands is **true**. The **!** operator will negate the value of the operand, turning **true** into **false** and vice versa.

You can use these operators to create complex conditional statements, such as:

```
if (($x > 0 && $y > 0) || !$z) {
   // do something
} else {
   // do something else
}
```

This statement will execute the first block of code if either **$x** and **$y** are both greater than 0, or if **$z** is **false**. If none of these conditions are met, it will execute the second block of code.

Example:

```
<?php
$x = 10;
$y = 20;
$z = false;
$result1 = $x > 0 && $y > 0; // true
$result2 = $x > 0 || $z; // true
$result3 = !$z; // true
echo $result1 . "\n";
echo $result2 . "\n";
echo $result3 . "\n";
?>
```

Spaceship

The spaceship operator (**<=>**) is a comparison operator introduced in PHP 7 that returns -1, 0, or 1 depending on the relative values of its operands.

If the operand on the left is less than the operand on the right, the operator returns -1. If the operands are equal, it returns 0. If the operand on the left is greater than the operand on the right, it returns 1.

This operator can be useful when sorting arrays or performing other kinds of comparison tasks.

Example:

```
<?php
$x = 10;
$y = 20;
$result1 = $x <=> $y; // -1
$result2 = $y <=> $x; // 1
$result3 = $x <=> $x; // 0
echo $result1 . "\n";
echo $result2 . "\n";
echo $result3 . "\n";
```

```
?>
```

You can also use the spaceship operator when sorting arrays with the **usort()** function:

```
<?php
$array = [3, 2, 1];
usort($array, function($a, $b) {
   return $a <=> $b;
});
print_r($array); // [1, 2, 3]
?>
```

if

The **if** statement in PHP allows you to execute a block of code conditionally, based on the truth value of a given expression.

Here is an example of an **if** statement in PHP:

```
$x = 10;
if ($x > 5) {
   echo "x is greater than 5";
}
```

In this example, the code inside the curly braces will be executed if the value of **$x** is greater than 5.

You can also include an **else** clause to specify a block of code to be executed if the condition is not met:

```
$x = 10;
if ($x > 5) {
   echo "x is greater than 5";
} else {
   echo "x is not greater than 5";
}
```

You can also use the **elseif** clause to specify additional conditions to be tested:

```
$x = 10;
if ($x > 15) {
   echo "x is greater than 15";
} elseif ($x > 10) {
   echo "x is greater than 10 but not greater than 15";
} else {
   echo "x is not greater than 10";
}
```

In this example, the code inside the first **if** clause will not be executed because the condition is not met. The code inside the second **elseif** clause will also not be executed because the condition is not

met. The code inside the final **else** clause will be executed because it is the last option and all previous conditions have not been met.

Switch

The **switch** statement in PHP allows you to execute a block of code based on a specific value or expression.

Here is an example of a **switch** statement in PHP:

```
$x = 10;
switch ($x) {
   case 10:
       echo "x is equal to 10";
       break;
   case 11:
       echo "x is equal to 11";
       break;
   default:
       echo "x is not equal to 10 or 11";
       break;
}
```

In this example, the code inside the first **case** block will be executed because the value of **$x** is equal to 10. The **break** statement at the end of each case block causes the **switch** statement to exit and stop executing any further code. If the **break** statement is omitted, the **switch** statement will continue to execute the code in the next case block, even if the condition has not been met.

The **default** case is optional, and it specifies a block of code to be executed if none of the other case conditions are met.

You can also use the **switch** statement to compare the type of a variable, rather than its value:

```
$x = 10;
switch (gettype($x)) {
   case "integer":
       echo "x is an integer";
       break;
   case "string":
       echo "x is a string";
       break;
   default:
       echo "x is not an integer or a string";
       break;
}
```

In this example, the code inside the first case block will be executed because the type of $x is an integer.

For

The **for** loop in PHP allows you to execute a block of code repeatedly for a given number of iterations.

Here is the general syntax for a **for** loop in PHP:

```
for (initialization; condition; increment) {
   code to be executed;
}
```

The **initialization** expression is executed before the loop starts, and it is usually used to initialize a loop counter. The **condition** expression is evaluated at the beginning of each iteration, and the loop continues to run as long as the condition is **true**. The **increment** expression is executed at the end of each iteration, and it is usually used to update the loop counter.

Here is an example of a **for** loop that counts from 1 to 10:

```
for ($i = 1; $i <= 10; $i++) {
   echo "$i ";
}
```

In this example, the value of **$i** is initialized to 1 before the loop starts. The loop will continue to run as long as **$i** is less than or equal to 10. At the end of each iteration, **$i** is incremented by 1. The code inside the loop will print the value of **$i** on each iteration.

You can also nest **for** loops to create a loop within a loop. Here is an example of a nested **for** loop that counts from 1 to 3 in the outer loop, and from 4 to 6 in the inner loop:

```
for ($i = 1; $i <= 3; $i++) {
   for ($j = 4; $j <= 6; $j++) {
       echo "$i, $j ";
   }
}
```

In this example, the inner loop will execute 3 times for each iteration of the outer loop, resulting in a total of 9 iterations. The code inside the inner loop will print the values of **$i** and **$j** on each iteration.

While

The **while** loop in PHP allows you to execute a block of code repeatedly as long as a given condition is **true**.

Here is the general syntax for a **while** loop in PHP:

```
while (condition) {
   code to be executed;
}
```

The **condition** expression is evaluated at the beginning of each iteration, and the loop continues to run as long as the condition is **true**.

Here is an example of a **while** loop that counts from 1 to 10:

```
$i = 1;
while ($i <= 10) {
   echo "$i ";
   $i++;
}
```

In this example, the value of **$i** is initialized to 1 before the loop starts. The loop will continue to run as long as **$i** is less than or equal to 10. At the end of each iteration, **$i** is incremented by 1. The code inside the loop will print the value of **$i** on each iteration.

It is important to include a way to update the condition within the loop, or the loop will become an infinite loop and will continue to run indefinitely.

You can also nest **while** loops to create a loop within a loop. Here is an example of a nested **while** loop that counts from 1 to 3 in the outer loop, and from 4 to 6 in the inner loop:

```
$i = 1;
while ($i <= 3) {
   $j = 4;
   while ($j <= 6) {
       echo "$i, $j ";
       $j++;
   }
   $i++;
}
```

In this example, the inner loop will execute 3 times for each iteration of the outer loop, resulting in a total of 9 iterations. The code inside the inner loop will print the values of **$i** and **$j** on each iteration.

Do While

The **do...while** loop in PHP is similar to the **while** loop, but it guarantees that the code inside the loop will be executed at least once, even if the condition is not met.

Here is the general syntax for a **do...while** loop in PHP:

```
do {
   code to be executed;
} while (condition);
```

The code inside the loop is executed first, and then the **condition** expression is evaluated. If the condition is **true**, the loop continues to run. If the condition is **false**, the loop exits and the code after the loop is executed.

Here is an example of a **do...while** loop that counts from 1 to 10:

```
$i = 1;
do {
   echo "$i ";
   $i++;
} while ($i <= 10);
```

In this example, the value of **$i** is initialized to 1 before the loop starts. The code inside the loop will be executed at least once, regardless of the value of **$i**. At the end of each iteration, **$i** is incremented by 1. The loop will continue to run as long as **$i** is less than or equal to 10. The code inside the loop will print the value of **$i** on each iteration.

It is important to include a way to update the condition within the loop, or the loop will become an infinite loop and will continue to run indefinitely.

You can also nest **do...while** loops to create a loop within a loop. Here is an example of a nested **do...while** loop that counts from 1 to 3 in the outer loop, and from 4 to 6 in the inner loop:

```
$i = 1;
do {
   $j = 4;
   do {
       echo "$i, $j ";
       $j++;
   } while ($j <= 6);
   $i++;
} while ($i <= 3);
```

In this example, the inner loop will execute 3 times for each iteration of the outer loop, resulting in a total of 9 iterations. The code inside the inner loop will print the values of **$i** and **$j** on each iteration.

foreach

The **foreach** loop in PHP allows you to iterate over the elements of an array and execute a block of code for each element.

Here is the general syntax for a **foreach** loop in PHP:

```
foreach ($array as $value) {
   code to be executed;
}
```

You can also specify the key of each element in the array by using the following syntax:

```
foreach ($array as $key => $value) {
   code to be executed;
}
```

Here is an example of a **foreach** loop that iterates over an array of numbers and prints each element:

```
$numbers = array(1, 2, 3, 4, 5);
foreach ($numbers as $number) {
   echo "$number ";
}
```

In this example, the **foreach** loop will iterate over the elements of the **$numbers** array, assigning each element to the **$number** variable in turn and executing the code inside the loop. The code inside the loop will print the value of **$number** on each iteration.

Here is an example of a **foreach** loop that iterates over an associative array and prints each key and value:

```
$students = array(
   "student1" => "John",
   "student2" => "Jane",
   "student3" => "Bob"
);
foreach ($students as $student_id => $name) {
   echo "Student ID: $student_id, Name: $name <br>";
}
```

In this example, the **foreach** loop will iterate over the elements of the **$students** array, assigning the key of each element to the **$student_id** variable and the value of each element to the **$name** variable, and executing the code inside the loop. The code inside the loop will print the value of **$student_id** and **$name** on each iteration.

Continue and continue

The **break** and **continue** statements in PHP can be used to control the flow of a loop.

The **break** statement allows you to exit a loop prematurely, while the **continue** statement allows you to skip the rest of the current iteration and move on to the next one.

Here is an example of a **for** loop that uses the **break** statement to exit the loop when a certain condition is met:

```
for ($i = 1; $i <= 10; $i++) {
   if ($i == 5) {
       break;
   }
   echo "$i ";
}
```

In this example, the **for** loop will iterate 10 times, but the **break** statement will cause the loop to exit after the fifth iteration, so the numbers 1 through 4 will be printed.

Here is an example of a **for** loop that uses the **continue** statement to skip the rest of the current

iteration and move on to the next one:

```
for ($i = 1; $i <= 10; $i++) {
   if ($i % 2 == 0) {
       continue;
   }
   echo "$i ";
}
```

In this example, the **for** loop will iterate 10 times, but the **continue** statement will cause the loop to skip the rest of the current iteration and move on to the next one if the value of **$i** is even. This means that only the odd numbers 1, 3, 5, 7, and 9 will be printed.

You can also use the **break** and **continue** statements in **while**, **do...while**, and **foreach** loops in a similar way.

cookies

Cookies in PHP are a way to store small amounts of data on the client side, which can be used to track user activity or remember user preferences. Below is an example of how to set, retrieve, and delete cookies in PHP.

Setting a Cookie

To set a cookie in PHP, use the setcookie() function. Here's an example of setting a cookie:

```
<?php
// Set a cookie
$cookie_name = "user";
$cookie_value = "John Doe";
$cookie_expiry = time() + (86400 * 30); // 86400 = 1 day, so the cookie
will expire in 30 days
setcookie($cookie_name, $cookie_value, $cookie_expiry, "/"); // "/" means
the cookie is available in the entire domain
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Set Cookie</title>
</head>
<body>
   <?php
   if(!isset($_COOKIE[$cookie_name])) {
       echo "Cookie named '" . $cookie_name . "' is not set!";
   } else {
       echo "Cookie '" . $cookie_name . "' is set!<br>";
       echo "Value is: " . $_COOKIE[$cookie_name];
   }
```

```
   ?>
</body>
</html>
```

Retrieving a Cookie

To retrieve a cookie in PHP, you use the $_COOKIE superglobal array. Here’s an example:

```
<?php
// Retrieve the cookie
$cookie_name = "user";
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Retrieve Cookie</title>
</head>
<body>
   <?php
   if(!isset($_COOKIE[$cookie_name])) {
       echo "Cookie named '" . $cookie_name . "' is not set!";
   } else {
       echo "Cookie '" . $cookie_name . "' is set!<br>";
       echo "Value is: " . $_COOKIE[$cookie_name];
   }
   ?>
</body>
</html>
```

Deleting a Cookie

To delete a cookie, you set its expiration date to a past time. Here’s an example:

```
<?php
// Delete a cookie
$cookie_name = "user";
setcookie($cookie_name, "", time() - 3600, "/"); // set the expiration
date to one hour ago
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Delete Cookie</title>
</head>
<body>
   <?php
```

```
    echo "Cookie '" . $cookie_name . "' is deleted.";
    ?>
</body>
</html>
```

Explanation

Setting a Cookie:

- setcookie(name, value, expire, path): Sets a cookie.
- time() + (86400 * 30): Sets the cookie to expire in 30 days (86400 seconds in a day).
- "/": The cookie is available within the entire domain.

Retrieving a Cookie:

- $_COOKIE['name']: Retrieves the value of the cookie.

Deleting a Cookie:

- setcookie(name, "", time() - 3600, "/"): Sets the cookie's expiration date to one hour ago, effectively deleting it.

These examples show the basic operations of setting, retrieving, and deleting cookies in PHP. Place these scripts in your web server directory and access them through your browser to see the results.

SQL

To run the script, you need the MySQL script below. Create a database and run the below MySQL script.

```
CREATE TABLE `user` (
    `id` int(10) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `fName` varchar(150) NOT NULL,
        `lName` varchar(150) NOT NULL,
     `email` varchar(150) NOT NULL,
     `password` varchar(150) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Connecting

Database class sets user and database credentials for users to access the server database. It uses prepare to run SQL queries. It is also the connection to the SQL database.

```
<?php
class Database{
    public $conn;

    public function __construct(){
        $servername = "localhost";
```

```
        $username = "root";
        $password = "root";

        try {
            $this->conn = new
PDO("mysql:host=$servername;dbname=yourDatabase", $username, $password);
            // set the PDO error mode to exception
            $this->conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
            //echo "Connected successfully";
        }
        catch(PDOException $e) {
            echo "Connection failed: " . $e->getMessage();
        }
    }
}
```

Classes

Other classes extend the Database class to access the server. In this example class, the User extends Database. It has methods such as getAll, get, create, update, and destroy which are used to carry actions on the user data. This is the PHP for all SQL commands that run the queries for selecting, creating, updating and deleting. The methods getAll and get selects database rows from SQL and display the result, the create methods create the user to the database, the update method updates it and destroy method to delete the user.

Selecting
This has two methods getAll(public function getAll()) and get(public function get($id). The getAll method selects all users from the database and output the result in the window and the get method select the result of one of the user and output it on the screen.

Create
The create(public function Create()) method creates user detail on the database table. The details are entered and stored on the database for later users. We use this in select and destroy methods to select and destroy a particular user from the database table.

Update
The update(public function update()) methods updates user details from the database. It changed user detail if they typed the wrong name, email, or password. When this is changed it will update detail in the database and the user will have new detail. This is useful for a user when they have forgotten a password and want to change it.

Destroy
Destroy(public function destroy()) delete a user from the database table. It erases the user totally from the database. If the user needs to get to the website again, they will have to create their details on the website again.

```php
<?php

class User extends Database{
    public function auth(){
        if(!empty($_COOKIE['id']) && !empty($_COOKIE['password'])){
            $user = $this->get($_COOKIE['id']);
            if($user && $user['password'] == $_COOKIE['password']){
                return $user;
            }
        }
        return false;
    }
    public function getAll()
    {
        $prepare = $this->conn->prepare("SELECT * FROM user");
        $prepare->execute();
        return $prepare->fetchAll();
    }

    public function get($id)
    {
        $prepare = $this->conn->prepare("SELECT * FROM user WHERE id =
?");
        $prepare->execute([$id]);
        return $prepare->fetch();
    }

    public function getByEmail($email)
    {
        $prepare = $this->conn->prepare("SELECT * FROM user WHERE email =
?");
        $prepare->execute([$email]);
        return $prepare->fetch();
    }

    public function Create($gender, $fName, $lName, $email, $password)
    {
        $prepare = $this->conn->prepare("INSERT INTO user(gender, fName,
lName, email, password) VALUES (?, ?, ?, ?, ?)");
        $prepare->execute([$gender, $lName, $fName, $email, $password]);
    }

    public function update($gender, $lName, $fName, $email, $id)
    {
        $prepare = $this->conn->prepare("UPDATE user SET gender = ?,
fName = ?, lName = ?, email = ? WHERE id = ?");
        $prepare->execute([$gender, $fName, $lName, $email, $id]);
    }

    public function destroy($id)
    {
```

```
        $prepare = $this->conn->prepare("DELETE FROM user WHERE id = ?");
        $prepare->execute([$id]);
    }
}
```

Select

To select all users include User and Database classes and use methods (e.g. getAll) inside User. We also use method foreach(<?php foreach ($users as $user):?>) to call all users from the database and display them onto the screen. Because we used session to login, we have to put session start at the header to avoid users who do not login to access the page.

```
<?php
include_once 'Database.php';
include_once 'User.php';

$insertModel = new User();
if ($_POST) {
    $insertModel->Create($_POST['gender'], $_POST['lName'],
$_POST['fName'], $_POST['email'], password_hash($_POST['password'],
PASSWORD_DEFAULT));
}
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>insert</title>
</head>
<body>
<main>
    <form action="" method="post">
        <p>
            <label>Gender</label>
            <select name="gender" required>
                <option value="male">Male</option>
                <option value="female">Female</option>
            </select>
        </p>
        <p>
            <label for="name">First Name</label>
            <input name="fName" type="text" placeholder="Enter first
name">
        </p>
```

```
        <p>
            <label for="lName">First Name</label>
            <input name="lName" type="text" placeholder="Enter last
name">
        </p>
        <p>
            <label for="email">Email</label>
            <input name="email" type="text" placeholder="Enter email">
        </p>
        <p>
            <label for="password">Password</label>
            <input name="password" type="password" placeholder="Enter
password">
        </p>
        <button type="submit">Submit</button>
    </form>
</main>
</body>
</html>
```

Insert

Insert includes the Database and User classes to access methods from them. Then it enters information about a user into the database.

```
<?php
ini_set('display_startup_errors', 1);
ini_set('display_errors', 1);
error_reporting(-1);

include_once 'Database.php';
include_once 'User.php';

$insertModel = new User();
if($_POST){
    $insertModel->Create($_POST['gender'], $_POST['lName'],
$_POST['fName'], $_POST['email'], password_hash($_POST['password'],
PASSWORD_DEFAULT));
}
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no,
initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>insert</title>
```

```
</head>
<body>
    <main>
        <form action="" method="post">
            <p>
                <label>Gender</label>
                <select name="gender" required>
                    <option value="male">Male</option>
                    <option value="female">Female</option>
                </select>
            </p>
            <p>
                <label for="name">First Name</label>
                <input name="fName" type="text" placeholder="Enter first
name">
            </p>
            <p>
                <label for="lName">First Name</label>
                <input name="lName" type="text" placeholder="Enter last
name">
            </p>
            <p>
                <label for="email">Email</label>
                <input name="email" type="text" placeholder="Enter
email">
            </p>
            <p>
                <label for="password">Password</label>
                <input name="password" type="password" placeholder="Enter
password">
            </p>
            <button type="submit">Submit</button>
        </form>
    </main>
</body>
</html>
```

Delete

It includes User and Database classes and call for the function destroy to delete a user from database.

```
<?php
include_once 'Database.php';
include_once 'User.php';
$destroy = new User();
$destroy->destroy($_GET['id']);
```

```
header('Location: select.php');
```

Update

The update includes User and Database classes and checks if a form is submitted. It uses two functions update and gets to update the information of users in the database and prints updates on the screen. Because we used session to login, we have to put session start at the header to avoid users who do not login to access the page.

```
<?php
include_once 'Database.php';
include_once 'User.php';
$userModel = new User();

if(!$userModel->auth()){
    header('Location: login.php');
    exit;
}
$updateModel = new User();
if ($_POST) {
    $updateModel->update($_POST['gender'], $_POST['fName'],
$_POST['lName'], $_POST['email'], $_GET['id']);
}

$updates = $updateModel->get($_GET['id']); // comes from the URL
?>

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <title>Update</title>
</head>
<body>
<a href="logout.php">Logout</a>
<main>
    <form action="" method="post">
        <p>
            <label>Gender</label>
            <select name="gender" required>
                <option value="male">Male</option>
                <option value="female">Female</option>
            </select>
        </p>
        <p>
```

```
            <label for="fName">First Name</label>
            <input name="fName" type="text" value="<?php echo
$updates['fName']; ?>" placeholder="Enter first name">
        </p>

        <p>
            <label for="lName">Name</label>
            <input name="lName" type="text" value="<?php echo
$updates['lName']; ?>" placeholder="Enter name">
        </p>
        <p>
            <label for="email">Email</label>
            <input name="email" type="text" value="<?php echo
$updates['email']; ?>" placeholder="Enter email">
        </p>
        <button type="submit">Submit</button>
    </form>
</main>
</body>
</html>
```

Increment and Decrement

In PHP, you can use the increment operator (**++**) to increase the value of a variable by 1, and the decrement operator (**--**) to decrease the value of a variable by 1.

The increment operator can be placed before or after the variable:

```
$x = 10;
$x++; // $x is now 11
++$x; // $x is now 12
```

The decrement operator can also be placed before or after the variable:

```
$x = 10;
$x--; // $x is now 9
--$x; // $x is now 8
```

You can use these operators to perform simple arithmetic operations, such as counting the number of times a loop has iterated:

```
for ($i = 0; $i < 10; $i++) {
   // do something
}
```

You can also use the increment and decrement operators in expressions:

```
$x = 10;
```

```
$y = ++$x + 5; // $y is 16
$x = 10;
$y = $x++ + 5; // $y is 15
```

In the first example, the value of **$x** is incremented before it is used in the expression, so **$y** is equal to **$x + 5** after **$x** has been incremented. In the second example, the value of **$x** is used in the expression first, and then it is incremented, so **$y** is equal to **$x + 5** before **$x** is incremented.

String

In PHP, a string is a sequence of characters, represented by enclosing the characters in single or double quotes:

```
$string1 = 'This is a string';
$string2 = "This is also a string";
```

You can use the concatenation operator (.) to combine multiple strings into a single string:

```
$string1 = 'Hello';
$string2 = 'world';
$string3 = $string1 . ' ' . $string2; // 'Hello world'
```

You can also use the concatenation assignment operator (**.=**) to append a string to the end of another string:

```
$string = 'Hello';
$string .= ' world'; // 'Hello world'
```

You can access individual characters in a string using array notation, with the index of the character starting at 0 for the first character:

```
$string = 'Hello';
echo $string[0]; // 'H'
echo $string[4]; // 'o'
```

You can use the **strlen()** function to get the length of a string, and the **strpos()** function to find the position of a substring within a string:

```
$string = 'Hello world';
echo strlen($string); // 11
echo strpos($string, 'world'); // 6
```

PHP also provides a number of string manipulation functions, such as **substr()**, **strtolower()**, and **strtoupper()**, that you can use to manipulate strings in various ways.

Example:

```
<?php
```

```
$string = 'Hello world';
echo strtoupper($string) . "\n"; // 'HELLO WORLD'
echo strtolower($string) . "\n"; // 'hello world'
echo substr($string, 6) . "\n"; // 'world'
echo substr($string, 0, 5) . "\n"; // 'Hello'
echo str_replace('world', 'Earth', $string) . "\n"; // 'Hello Earth'
?>
```

Session

Sessions in PHP are used to store user information across different pages. Unlike cookies, session data is stored on the server. Here's a basic example of how to start a session, store data in a session, retrieve session data, and destroy a session.

Starting a Session and Storing Data

Create a file called start_session.php:

```
<?php
// Start the session
session_start();
// Store data in session variables
$_SESSION["username"] = "JohnDoe";
$_SESSION["email"] = "john.doe@example.com";
echo "Session variables are set.";
?>
```

Retrieving Session Data

Create a file called retrieve_session.php:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Retrieve Session</title>
</head>
<body>
   <?php
   // Retrieve session data
   if(isset($_SESSION["username"]) && isset($_SESSION["email"])) {
       echo "Username: " . $_SESSION["username"] . "<br>";
       echo "Email: " . $_SESSION["email"];
   } else {
       echo "No session data found.";
```

```
    }
    ?>
</body>
</html>
```

Destroying a Session

Create a file called destroy_session.php:

```
<?php
// Start the session
session_start();
// Unset all session variables
$_SESSION = array();
// Destroy the session
session_destroy();
echo "Session is destroyed.";
?>
```

Explanation

Starting a Session and Storing Data (start_session.php):

- session_start(): Starts a new session or resumes an existing session.
- $_SESSION["key"] = value: Stores data in session variables.

Retrieving Session Data (retrieve_session.php):

- session_start(): Resumes the session started in start_session.php.
- $_SESSION["key"]: Retrieves the value stored in a session variable.

Destroying a Session (destroy_session.php):

- session_start(): Resumes the session.
- $_SESSION = array(): Clears all session variables.
- session_destroy(): Destroys the session.

Usage

Start a session:

- Access start_session.php in your browser. This will set session variables.

Retrieve session data:

- Access retrieve_session.php in your browser. This will display the session data set in start_session.php.

Destroy the session:

- Access destroy_session.php in your browser. This will destroy the session and clear all session data.

By using these scripts, you can manage sessions effectively in PHP.

Form

Creating a form in PHP involves both HTML for the form structure and PHP to process the form data. Below is an example of a simple form that collects a user's name and email, then processes and displays the input.

HTML Form

Create a file called form.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Simple Form</title>
</head>
<body>
   <form action="process.php" method="post">
       <label for="name">Name:</label>
       <input type="text" id="name" name="name" required><br><br>
       <label for="email">Email:</label>
       <input type="email" id="email" name="email" required><br><br>
       <input type="submit" value="Submit">
   </form>
</body>
</html>
```

PHP Processing Script

Create a file called process.php:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // Collect the form data
   $name = htmlspecialchars($_POST['name']);
   $email = htmlspecialchars($_POST['email']);
   // Display the collected data
   echo "<h1>Form Submission</h1>";
   echo "<p>Name: " . $name . "</p>";
   echo "<p>Email: " . $email . "</p>";
}
?>
```

Explanation

HTML Form (form.html):

- The form has two input fields: one for the user's name and one for their email.
- The action attribute of the form specifies that the form data will be sent to process.php.
- The method attribute is set to post, meaning the form data will be sent via the HTTP POST method.

PHP Processing Script (process.php):

- The script first checks if the form was submitted using the POST method.
- It then collects the form data using $_POST and sanitizes it with htmlspecialchars() to prevent XSS (Cross-Site Scripting) attacks.
- Finally, it displays the submitted data on the web page.

Make sure both files are saved in the same directory on your server. When you open form.html in your browser and submit the form, process.php will display the submitted data.

Class

Creating a class in PHP is straightforward. PHP is an object-oriented programming language, so it supports classes and objects. Below is an example of how you can define and use a class in PHP.

Defining a Class

```
<?php
class Car {
    // Properties
    public $make;
    public $model;
    public $year;

    // Constructor
    public function __construct($make, $model, $year) {
        $this->make = $make;
        $this->model = $model;
        $this->year = $year;
    }

    // Methods
    public function display() {
        return "This car is a $this->year $this->make $this->model.";
    }
}

// Create an instance of the class
$myCar = new Car("Toyota", "Corolla", 2021);

// Access the method
echo $myCar->display(); // Outputs: This car is a 2021 Toyota Corolla.
?>
```

Explanation

Class Definition:

- class Car { ... } defines a class named Car.
- Properties ($make, $model, $year) are defined within the class.

Constructor:

- The __construct method is a special method called a constructor. It initializes the object when it is created.
- The constructor takes three parameters ($make, $model, $year) and assigns them to the corresponding properties of the class.

Methods:

- display() is a method of the class that returns a string describing the car.

Creating an Instance:

- $myCar = new Car("Toyota", "Corolla", 2021); creates an instance of the Car class.

Accessing Methods:

- echo $myCar->display(); calls the display method of the $myCar object and prints the result.

Adding More Features

You can add more methods and properties as needed. For example, you can add a method to start the car:

```
<?php
class Car {
   // Properties
   public $make;
   public $model;
   public $year;
   public $isRunning = false;
   // Constructor
   public function __construct($make, $model, $year) {
       $this->make = $make;
       $this->model = $model;
       $this->year = $year;
   }
   // Methods
   public function display() {
       return "This car is a $this->year $this->make $this->model.";
   }
   public function start() {
       $this->isRunning = true;
       return "The car has started.";
   }
```

```
}
// Create an instance of the class
$myCar = new Car("Toyota", "Corolla", 2021);
// Access the method
echo $myCar->display(); // Outputs: This car is a 2021 Toyota Corolla.
echo $myCar->start(); // Outputs: The car has started.
?>
```

Summary

- **Class**: Defines a blueprint for objects.
- **Properties**: Variables within a class.
- **Methods**: Functions within a class.
- **Constructor**: Special method to initialize objects.
- **Instance**: An individual object created from a class.

Feel free to ask if you have any more specific requirements or questions!

Call

Calling or using a class in PHP involves creating an instance of the class and then accessing its properties and methods. Here's a detailed guide on how to do this.

Step-by-Step Guide

1. **Define the Class**:
    - Define a class with properties and methods.
2. **Create an Instance of the Class**:
    - Use the new keyword to create an object (instance) of the class.
3. **Access Properties and Methods**:
    - Use the -> operator to access properties and methods of the object.

Example

Let's define a class Car and then create an instance of it and call its methods

```
<?php
// Step 1: Define the Class
class Car {
   // Properties
   public $make;
   public $model;
   public $year;
   // Constructor
   public function __construct($make, $model, $year) {
       $this->make = $make;
       $this->model = $model;
       $this->year = $year;
   }
   // Methods
   public function display() {
```

```
        return "This car is a $this->year $this->make $this->model.";
    }
    public function start() {
        return "The car has started.";
    }
}
// Step 2: Create an Instance of the Class
$myCar = new Car("Toyota", "Corolla", 2021);
// Step 3: Access Properties and Methods
echo $myCar->display(); // Outputs: This car is a 2021 Toyota Corolla.
echo "\n"; // New line for better readability
echo $myCar->start();   // Outputs: The car has started.
?>
```

Explanation

Define the Class:

- class Car { ... }: This defines the Car class.
- Properties: $make, $model, $year to hold the car's make, model, and year.
- Constructor: __construct($make, $model, $year) to initialize the properties when a new object is created.
- Methods: display() and start() to provide functionality to the class.

Create an Instance:

- $myCar = new Car("Toyota", "Corolla", 2021);: This creates a new instance of the Car class, passing the make, model, and year to the constructor.

Access Properties and Methods:

- echo $myCar->display();: Calls the display method of the $myCar object.
- echo $myCar->start();: Calls the start method of the $myCar object.

Additional Example

You can also define a class with more properties and methods to demonstrate different functionalities:

```
<?php
class Person {
    // Properties
    public $firstName;
    public $lastName;
    public $age;
    // Constructor
    public function __construct($firstName, $lastName, $age) {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
        $this->age = $age;
    }
    // Methods
```

```
    public function getFullName() {
        return "$this->firstName $this->lastName";
    }
    public function isAdult() {
        return $this->age >= 18 ? "Adult" : "Not an Adult";
    }
}
// Create an instance of the class
$person = new Person("John", "Doe", 25);
// Access the methods
echo $person->getFullName(); // Outputs: John Doe
echo "\n"; // New line for better readability
echo $person->isAdult();     // Outputs: Adult
?>
```

Summary

1. **Define the Class**: Use class ClassName { ... }.
2. **Create an Instance**: Use new ClassName(parameters);.
3. **Access Properties and Methods**: Use $object->property and $object->method().

Class and Methods

Defining and Using Methods Inside a Class

Let's define a class called Calculator with methods for basic arithmetic operations: addition, subtraction, multiplication, and division. We'll then create an instance of the class and call these methods.

```
<?php
// Define the Calculator class
class Calculator {
    // Method to add two numbers
    public function add($a, $b) {
        return $a + $b;
    }
    // Method to subtract two numbers
    public function subtract($a, $b) {
        return $a - $b;
    }
    // Method to multiply two numbers
    public function multiply($a, $b) {
        return $a * $b;
    }
    // Method to divide two numbers
    public function divide($a, $b) {
        if ($b == 0) {
            return "Error: Division by zero";
        }
        return $a / $b;
```

```
    }
}
// Create an instance of the Calculator class
$calculator = new Calculator();
// Call the methods of the Calculator class and display the results
echo "Addition: " . $calculator->add(10, 5) . "\n";         // Outputs:
Addition: 15
echo "Subtraction: " . $calculator->subtract(10, 5) . "\n"; // Outputs:
Subtraction: 5
echo "Multiplication: " . $calculator->multiply(10, 5) . "\n"; //
Outputs: Multiplication: 50
echo "Division: " . $calculator->divide(10, 5) . "\n";         //
Outputs: Division: 2
echo "Division by zero: " . $calculator->divide(10, 0) . "\n"; //
Outputs: Error: Division by zero
?>
```

Explanation

Class Definition:

- The class Calculator defines a new class named Calculator.
- Methods add, subtract, multiply, and divide are defined within the class to perform arithmetic operations.

Methods:

- add($a, $b): Adds two numbers.
- subtract($a, $b): Subtracts the second number from the first number.
- multiply($a, $b): Multiplies two numbers.
- divide($a, $b): Divides the first number by the second number, with a check to prevent division by zero.

Creating an Instance:

- $calculator = new Calculator(); creates an instance of the Calculator class.

Calling Methods:

- echo $calculator->add(10, 5); calls the add method on the $calculator object and prints the result.
- Similarly, other methods (subtract, multiply, divide) are called and their results are printed.

Calling Methods with Parameters

When calling a method, you pass arguments to it, which the method uses to perform its operations. In the example above, each arithmetic method takes two parameters ($a and $b).

```
<?php
// Call the add method
$result = $calculator->add(8, 3);
echo "8 + 3 = $result\n"; // Outputs: 8 + 3 = 11
```

```
// Call the subtract method
$result = $calculator->subtract(8, 3);
echo "8 - 3 = $result\n"; // Outputs: 8 - 3 = 5
// Call the multiply method
$result = $calculator->multiply(8, 3);
echo "8 * 3 = $result\n"; // Outputs: 8 * 3 = 24
// Call the divide method
$result = $calculator->divide(8, 3);
echo "8 / 3 = $result\n"; // Outputs: 8 / 3 = 2.66666666667
```

Summary

Defining a Class:

- Use class ClassName { ... } to define a class.
- Define methods inside the class using public function methodName($parameters) { ... }.

Creating an Instance:

- Use new ClassName() to create an instance of the class.

Calling Methods:

- Use $object->methodName(arguments) to call methods on the object.

This example demonstrates how to define methods inside a class, create an instance of the class, and call the methods with parameters.

Creating strings

Here are some examples of how to create strings in PHP using various methods.

Using Double Quotes

```
<?php
$string1 = "Hello, World!";
echo $string1; // Outputs: Hello, World!
?>
```

Using Single Quotes

```
<?php
$string2 = 'Hello, World!';
echo $string2; // Outputs: Hello, World!
?>
```

Using Heredoc Syntax

Heredoc syntax allows the creation of strings that span multiple lines and support variable interpolation.

```
<?php
$name = "John";
$string3 = <<<EOT
Hello, $name!
This is a string
created using Heredoc syntax.
EOT;
echo $string3;
// Outputs:
// Hello, John!
// This is a string
// created using Heredoc syntax.
?>
```

Using Nowdoc Syntax

Nowdoc syntax is similar to Heredoc but does not parse variables.

```
<?php
$string4 = <<<'EOT'
Hello, $name!
This is a string
created using Nowdoc syntax.
EOT;
echo $string4;
// Outputs:
// Hello, $name!
// This is a string
// created using Nowdoc syntax.
?>
```

Creating Empty Strings

```
<?php
$emptyString1 = "";
$emptyString2 = '';
echo $emptyString1; // Outputs nothing
echo $emptyString2; // Outputs nothing
?>
```

These are the primary ways to create strings in PHP.

DateTime

Using the DateTime class in PHP provides a more object-oriented way to handle dates and times. Here is an example of how to get the current date using the DateTime class:

```
<?php
$date = new DateTime();
```

```
echo $date->format('Y-m-d'); // Outputs the current date in the format:
2024-05-30
?>
```

In this example, we create a new DateTime object representing the current date and time. We then use the format() method to output the date in the desired format.

Here are some additional examples of how to use the DateTime class to format dates:

To display the date in d/m/Y format:

```
<?php
$date = new DateTime();
echo $date->format('d/m/Y'); // Outputs: 30/05/2024
?>
```

To display the date and time:

```
<?php
$date = new DateTime();
echo $date->format('Y-m-d H:i:s'); // Outputs: 2024-05-30 14:30:00
(example time)
?>
```

To create a DateTime object for a specific date and time:

```
<?php
$date = new DateTime('2024-05-30 14:30:00');
echo $date->format('l, F j, Y H:i:s'); // Outputs: Thursday, May 30, 2024
14:30:00
?>
```

To modify the date and time:

```
<?php
$date = new DateTime();
$date->modify('+1 day');
echo $date->format('Y-m-d'); // Outputs the date for tomorrow
?>
```

The DateTime class provides many methods to manipulate and format dates and times, making it a powerful tool for handling date and time in PHP. You can find more details in the PHP documentation for the DateTime class.

Declare

To declare something in PHP means to introduce it to the interpreter, usually by creating a new variable or defining a new function or class.

Here are some examples of declaring variables in PHP:

```
$name = "John";  // declares a new string variable called $name and
assigns it the value "John"
$age = 30;  // declares a new integer variable called $age and assigns it
the value 30
$employee = true;  // declares a new boolean variable called $employee
and assigns it the value true
```

Here is an example of declaring a function in PHP:

```
function sayHello($name) {
 echo "Hello, $name!";
}
```

And here is an example of declaring a class in PHP:

```
class Person {
 public $name;
 public $age;

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

 public function sayHello() {
   echo "Hello, my name is $this->name and I am $this->age years old.";
 }
}
```

Declaring variables, functions, and classes is an important part of writing PHP code, as it allows you to use and manipulate these entities in your scripts.

create

To create a function in PHP, you can use the **function** keyword followed by the name of the function and a set of parentheses. The code that makes up the function should be enclosed in curly braces. Here is an example of a simple function in PHP:

```
function greet($name) {
 echo "Hello, $name!";
}
```

This function takes a single argument, **$name**, and outputs a greeting using the value of **$name**. To call this function, you can use the function name followed by a set of parentheses and the required arguments:

```
greet('John');  // Outputs: "Hello, John!"
```

You can also specify a default value for function arguments by using the **=** operator in the function definition. If a default value is provided, the argument is optional when calling the function. For example:

```
function greet($name = 'world') {
 echo "Hello, $name!";
}
greet();  // Outputs: "Hello, world!"
greet('John');  // Outputs: "Hello, John!"
```

You can also return a value from a function using the **return** keyword. For example:

```
function add($x, $y) {
 return $x + $y;
}
$sum = add(1, 2);  // $sum is now 3
```

Function

To create a function in PHP, you can use the **function** keyword followed by the name of the function and a set of parentheses. The code that makes up the function should be enclosed in curly braces. Here is an example of a simple function in PHP:

```
function greet($name) {
 echo "Hello, $name!";
}
```

This function takes a single argument, **$name**, and outputs a greeting using the value of **$name**. To call this function, you can use the function name followed by a set of parentheses and the required arguments:

```
greet('John');  // Outputs: "Hello, John!"
```

You can also specify a default value for function arguments by using the **=** operator in the function definition. If a default value is provided, the argument is optional when calling the function. For example:

```
function greet($name = 'world') {
 echo "Hello, $name!";
}
greet();  // Outputs: "Hello, world!"
greet('John');  // Outputs: "Hello, John!"
```

You can also return a value from a function using the **return** keyword. For example:

```
function add($x, $y) {
```

```
 return $x + $y;
}
$sum = add(1, 2);  // $sum is now 3
```

File

Creating a PHP and MySQL application that includes file upload functionality with CRUD (Create, Read, Update, Delete) operations can be broken down into several steps. Below is a step-by-step guide to help you build this application.

Step 1: Set Up Your Database

Create a MySQL database and a table to store the file information.

```
CREATE DATABASE file_upload;
USE file_upload;
CREATE TABLE files (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   path VARCHAR(255) NOT NULL,
   uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Step 2: Create a Database Connection

Create a file called db.php to handle the database connection.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "file_upload";
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
?>
```

Step 3: Create the File Upload Form

Create a file called index.php for the file upload form.

```
<!DOCTYPE html>
<html>
<head>
   <title>File Upload</title>
</head>
<body>
   <h2>Upload File</h2>
   <form action="upload.php" method="post" enctype="multipart/form-data">
```

```
        <label for="file">Select file:</label>
        <input type="file" name="file" id="file" required>
        <button type="submit">Upload</button>
    </form>
</body>
</html>
```

Step 4: Handle File Upload

Create a file called upload.php to handle the file upload.

```
<?php
include 'db.php';
if ($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_FILES['file'])) {
    $fileName = $_FILES['file']['name'];
    $fileTmpName = $_FILES['file']['tmp_name'];
    $uploadDirectory = 'uploads/';
    $filePath = $uploadDirectory . basename($fileName);
    if (move_uploaded_file($fileTmpName, $filePath)) {
        $sql = "INSERT INTO files (name, path) VALUES ('$fileName',
'$filePath')";
        if ($conn->query($sql) === TRUE) {
            echo "File uploaded successfully.";
        } else {
            echo "Error: " . $sql . "<br>" . $conn->error;
        }
    } else {
        echo "There was an error uploading your file.";
    }
}
$conn->close();
?>
```

Step 5: Display Uploaded Files

Create a file called list.php to display the uploaded files.

```
<?php
include 'db.php';
$sql = "SELECT * FROM files";
$result = $conn->query($sql);
?>
<!DOCTYPE html>
<html>
<head>
    <title>Uploaded Files</title>
</head>
<body>
    <h2>Uploaded Files</h2>
    <table border="1">
```

```
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Path</th>
            <th>Uploaded At</th>
            <th>Action</th>
        </tr>
        <?php
        if ($result->num_rows > 0) {
            while($row = $result->fetch_assoc()) {
                echo "<tr>
                    <td>{$row['id']}</td>
                    <td>{$row['name']}</td>
                    <td>{$row['path']}</td>
                    <td>{$row['uploaded_at']}</td>
                    <td>
                        <a
href='download.php?id={$row['id']}'>Download</a> |
                        <a href='delete.php?id={$row['id']}'>Delete</a>
                    </td>
                </tr>";
            }
        } else {
            echo "<tr><td colspan='5'>No files found.</td></tr>";
        }
        ?>
    </table>
</body>
</html>
<?php
$conn->close();
?>
```

Step 6: File Download

Create a file called download.php to handle file downloads.

```
<?php
include 'db.php';
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $sql = "SELECT * FROM files WHERE id = $id";
    $result = $conn->query($sql);
    if ($result->num_rows == 1) {
        $row = $result->fetch_assoc();
        $filePath = $row['path'];
        if (file_exists($filePath)) {
            header('Content-Description: File Transfer');
            header('Content-Type: application/octet-stream');
            header('Content-Disposition: attachment;
filename="'.basename($filePath).'"');
```

```
            header('Expires: 0');
            header('Cache-Control: must-revalidate');
            header('Pragma: public');
            header('Content-Length: ' . filesize($filePath));
            readfile($filePath);
            exit;
        } else {
            echo "File not found.";
        }
    } else {
        echo "Invalid file ID.";
    }
}
$conn->close();
?>
```

Step 7: File Deletion

Create a file called delete.php to handle file deletion.

```
<?php
include 'db.php';
if (isset($_GET['id'])) {
    $id = $_GET['id'];
    $sql = "SELECT * FROM files WHERE id = $id";
    $result = $conn->query($sql);
    if ($result->num_rows == 1) {
        $row = $result->fetch_assoc();
        $filePath = $row['path'];
        if (file_exists($filePath)) {
            unlink($filePath);
            $sql = "DELETE FROM files WHERE id = $id";
            if ($conn->query($sql) === TRUE) {
                echo "File deleted successfully.";
            } else {
                echo "Error deleting file: " . $conn->error;
            }
        } else {
            echo "File not found.";
        }
    } else {
        echo "Invalid file ID.";
    }
}
$conn->close();
?>
```

Step 8: Combine Everything

Ensure all files are in the same directory and accessible. Create an uploads directory with

appropriate permissions for file uploads.

This completes a basic CRUD application with file upload functionality in PHP and MySQL. You can extend and customize this further based on your requirements.

Foreach

In PHP, the **foreach** loop is used to iterate over the elements of an array. It allows you to loop through each element of an array and perform a set of actions on each element.

Here is the basic syntax of the **foreach** loop:

```
foreach ($array as $value) {
  // code to be executed
}
```

The **$array** variable is the array that you want to loop through, and the **$value** variable is a placeholder for the current element of the array. The loop will execute once for each element of the array, with the value of the current element being assigned to **$value** on each iteration.

Here is an example of using the **foreach** loop to loop through an array of numbers and print each element to the screen:

```
$numbers = array(1, 2, 3, 4, 5);
foreach ($numbers as $number) {
  echo $number . "\n";
}
```

This would output:

```
1
2
3
4
5
```

You can also use the **foreach** loop to iterate over the elements of an associated array, by specifying two variables: one for the key and one for the value. The key variable will hold the key of the current element, and the value variable will hold the value.

Here is an example of using the **foreach** loop to loop through an associated array and print the key-value pairs:

```
$student = array(
  "first_name" => "John",
  "last_name" => "Doe",
  "age" => 25
);
foreach ($student as $key => $value) {
  echo "$key: $value\n";
```

```
}
```

```
This would output:
first_name: John
last_name: Doe
age: 25
```

built-in functions

PHP has a number of built-in functions that you can use in your code. These functions are available to you automatically, and you don't need to define them yourself. Here are a few examples of built-in functions in PHP:

- **strlen()**: This function returns the length of a string.
- **count()**: This function counts the number of elements in an array.
- **time()**: This function returns the current Unix timestamp.
- **rand()**: This function returns a random number.

To use a built-in function, you can simply call it by name and pass it the required arguments. For example:

```
$str = 'Hello, world!';
echo strlen($str);  // Outputs: 13
$arr = [1, 2, 3, 4];
echo count($arr);  // Outputs: 4
echo time();  // Outputs: 1623456789
echo rand();  // Outputs a random number
```

email

Email requires two things, mail to send the form and form to process the inputs. Mail requires email, subject, message and from email to send the email. And those the inputs require at the inputs.

```
<?php
if($_POST){
    mail('info@lokang.com', 'email from website', $_POST['message'], [
        'From' => $_POST['email']
    ]);
}

?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
```

```
    <title>email | Lokang</title>
</head>
<body>
<header></header>

<form action="" method="post">
    <p>
        <label for="name" class="form-label">Name</label>
        <input name="name" type="text" id="name" class="form-control"
required>
    </p>
    <p>
        <label for="email" class="form-label">email</label>
        <input name="email" id="email" type="text" class="form-control"
required>
    </p>
    <p>
        <label for="message" class="form-label">message</label>
        <textarea name="message" id="message" class="form-
control"></textarea>
    </p>
    <button class="btn btn-secondary mt-2" type="submit">submit</button>
</form>

<footer>
    &copy; Lokang <?php echo date("Y") ?>
</footer>

</body>
</html>
```

Access Modifiers

In the access modifiers AccessModified, we created three access modifiers namely private, protected and public. The access differently depending on which it is used in the class.
Private: This is only called within a class.
Protect: Is call inside a class and the class that extends it.
Public: Call anywhere the class.

For a private and protected we called write functions to called them and we called them outside of the class using the echo echo $accessModified->name(); and echo $accessModified->age();. For the public we call it from outside the class and it is fine with it.

```
<?php
class AccessModifier{
    private $name = 'Lokang';
    protected $age = 40;
    public $color = "black";
    public function age(){
```

```
        return $this->age;
    }
    public function name(){
        return $this->name;
    }
}

$accessModifier = new AccessModifier();
echo $accessModifier->name();
echo '<br>';
echo $accessModifier->age();
echo '<br>';
echo $accessModifier->color;
```

Constructor

Constructor runs the process of calling access modifiers. It is a function in PHP which is name __constructor. For this example, we an access modifier private and the methods name(). The constructor has a double underscore mark. It is then echoed the name of the function to output 'Lokang'.

```
<?php
class Constructor{
    private $name = 'Lokang';
    public function __constructor($name){
        $this->age = $age;
    }

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

$constructor = new Constructor();
echo $constructor->name();
```

Inheritance

Inheritance in OOP = When a class derives from another class. The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods. An inherited class is defined by using the extends keyword. Class Age extends Inheritance as its initial class. It the called the result name and myAge. When you write the name of inheritance same name of the method, it assumpts it to be a constructor.

```
<?php
class Inheritance{
```

```
    public $name = "lokang";

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

class Age extends Inheritance {
    private $age = 40;
    public function myAge(){
        return $this->age;
    }
}

$inheritance = new Inheritance();
echo $inheritance->name();
echo '<br>';
$age = new Age();
echo $age->myAge();
```

login

Login allows users to login to the website the access other information that not a login has.

```
<?php
include_once 'Database.php';
include_once 'User.php';
//echo hash('sha512', 'password');

if ($_POST) {
    $usersModel = new User();
    $user = $usersModel->getByEmail($_POST['email']);

    if ($user && password_verify($_POST['password'], $user['password']))
{
        setcookie('id', $user['id'], time()+3600*24*30, '/');
        setcookie('password', $user['password'], time()+3600*24*30, '/');
        header('Location: select.php');
    } else {
        echo 'cannot login';
    }
}
?>

<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>login</title>
</head>
<body>
<main>
    <form action="" method="post">
        <p>
            <label for="email">Email</label>
            <input name="email" type="text" placeholder="Enter email">
        </p>
        <p>
            <label for="password">Password</label>
            <input name="password" type="password" placeholder="Enter
password">
        </p>
        <button type="submit">Submit</button>
    </form>
</main>
</body>
</html>
```

logout

Logout ends a users access to the website if he/she is login before.

```
<?php
setcookie('id', 0, time()-1, '/');
setcookie('password', '', time()-1, '/');
header('Location: login.php');
?>
```

primitive

In PHP, variables can store data of different types, and the type of data a variable holds can change during the execution of a script. PHP is a loosely typed language, which means you don't have to declare the data type of a variable when you create it. PHP determines the data types automatically.

Basic Data Types

1. Integer

An integer is a whole number without a fractional part. It can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format.

```
$int1 = 123;    // decimal number
```

```
$int2 = -123;   // a negative decimal number
$int3 = 0x1A;   // hexadecimal number
$int4 = 0123;   // octal number
```

2. Float (or Double)

A float (floating-point number) is a number with a decimal point or a number in exponential form.

```
$float1 = 1.234;
$float2 = 1.2e3;
$float3 = 7E-10;
```

3. String

A string is a sequence of characters. It can be defined using single or double quotes.

```
$string1 = 'Hello, World!';
$string2 = "Hello, World!";
```

4. Boolean

A boolean represents two possible states: **TRUE** or **FALSE**.

```
$bool1 = true;
$bool2 = false;
```

cookies

This code is an example of setting and using cookies in PHP. A cookie is a small piece of data that a website stores on the client's computer. Cookies are commonly used for authentication, tracking user preferences, and for storing session data.

The code uses the **setcookie()** function to set a cookie named "user" with a value "John Doe" and a expiration time of 1 hour (time()+3600) in the future. The cookie will be stored on the client's computer and sent back to the server with every request.

The code then uses an if-else statement to check if the cookie "user" is set using the **isset()** function. If the cookie is set, the code will display a welcome message with the value of the cookie using the $_COOKIE superglobal array. If the cookie is not set, the code will display a welcome message to the guest.

Cookies can be accessed and modified via the $_COOKIE superglobal array, which contains the values of all cookies sent by the client. It is important to note that cookies are stored on the client's browser, so the data stored in cookies is accessible to the user, and can be deleted or modified by the user. So, it is not recommended to store sensitive information in cookies.

```
<?php
// set a cookie variable
```

```
setcookie("user", "John Doe", time()+3600);// set the cookie for 1 hour
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<?php
// check if the cookie is set
if(isset($_COOKIE["user"]))// if the cookie is set
    echo "Welcome " . $_COOKIE["user"] . "!<br />";// print the cookie
else
    echo "Welcome guest!<br />";// print a message
?>
</body>
</html>
```

built in errors

The code is using the built-in error handling feature in PHP. The **require** statement is attempting to include a file called 'test.txt'. If the file does not exist, it will throw an error, specifically a warning that states "Failed to open stream: No such file or directory." This error message will also indicate the file and line number where the error occurred.

```
<?php
// built in error handling
require ('test.txt');
// if the file does not exist, it will throw an error
/*
 * Warning: require(test.txt): Failed to open stream: No such file or
directory in /Users/user/Desktop/Web/php/errorHandling/index.php on line
3
 * This is built in error handling in php
 */
```

get

This code is an example of using the $_GET superglobal array in PHP to retrieve data sent in the URL as query parameters. The $_GET array is used to collect data from an HTML form using the "get" method or to retrieve data from the URL.

The code includes a link with an anchor tag that points to a file called "get.php" and includes a query parameter named "page" with a value of "1". When the user clicks on this link, it sends a GET request to the server, which includes the "page" parameter and its value.

The PHP code then uses an if statement to check if the "page" parameter is set using the **isset()** function. If the "page" parameter is set, the code will display its value using the $_GET array.

When the user clicks on the link, the page get.php will be loaded and the query parameter page=1 is passed to this page and the PHP script will print the value of the page variable on the page.

It is worth noting that the $_GET array is not secure and should not be used to pass sensitive data, as the data can be easily seen and modified by the user.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p>Please click on the following link to see all get variables</p>
<a href="get.php?page=1">Click here</a>
<?php
if (isset($_GET['page'])) {
    echo $_GET['page'];// print the page get variable
}
?>
</body>
</html>
```

global

This code is an example of global variables in PHP. Global variables are variables that are defined outside of any function and can be accessed from anywhere in the script, including inside functions.

The code creates a global variable named $color and assigns it the value "red". Then, it defines a function called **myTest()** that uses the global variable $color. Inside the function, the code uses the **global** keyword to indicate that it wants to access the global $color variable, rather than a local variable with the same name.

The code then includes an HTML document that includes a PHP script that calls the **myTest()** function. This function will print "My car is red" on the page.

It is worth noting that global variables can be accessed and modified from anywhere in the script, including inside functions, which can make code less readable, and harder to maintain. So, it is recommended to use global variables only when necessary.

```
<?php
//create an example of global variables
//create a variable in the global scope
$color = "red";
```

```
//create a function that uses the global variable
function myTest() {
    //use x inside this function
    global $color;
    echo "My car is " . $color . "<br>";
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<?php
myTest();
?>
</body>
</html>
```

post

This code is an example of using the $_POST superglobal array in PHP to handle data sent from an HTML form using the "post" method. The $_POST array is used to collect data from an HTML form using the "post" method or to retrieve data from an API.

The code includes an HTML form with two input fields, one for the name and one for the email, and a submit button. When the user fills in the form and clicks the submit button, the form sends a POST request to the server, which includes the input field data as part of the request body.

The PHP code then uses an if statement to check if the form is submitted by checking if the submit button is set using the **isset()** function. If the form is submitted, the code will use the $_POST array to get the values of the name and email fields and save them to variables.

Finally, the script will display the name and email on the page by echoing the values of the variables.

It is worth noting that the $_POST array is more secure than the $_GET array as the data sent in the request body is not visible in the URL and is less likely to be modified by the user.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<p>html form</p>
<form action="post.php" method="post">
    <input type="text" name="name" placeholder="name">
```

```
    <input type="text" name="email" placeholder="email">
    <input type="submit" name="submit" value="submit">
</form>

<?php
// check if the form is submitted
if(isset($_POST['submit'])){
    // get the form data
    $name = $_POST['name'];
    $email = $_POST['email'];
    // display the form data
    echo "Your name is: $name and your email is: $email";
}
?>

</body>
</html>
```

server

This code is an example of using the $_SERVER superglobal array in PHP to access information about the server and the current request. The $_SERVER array contains information about the server, the current request, and the current script.

The code uses the **echo** statement to print the value of the **SERVER_NAME** key in the $_SERVER array. This key contains the name of the server that is currently running the script.

The $_SERVER array contains a lot of information about the current request and the server, such as the server name, the server software, the request method, the script name, the client's IP address, the browser's user-agent, and more.

It is worth noting that the information in the $_SERVER array may vary depending on the server configuration and the PHP version used, and some keys may not be available on all servers.

```
<!-- print a server name -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<?php
// print the server name
echo $_SERVER['SERVER_NAME'];
?>
</body>
</html>
```

session

This code is an example of using PHP sessions to store data on the server that can persist across multiple requests. A session is a way to store data on the server that is associated with a specific user.

The code starts the session using the **session_start()** function, this function creates a new session or resumes an existing one. Then it sets a session variable named "favcolor" to the value "green" using the $_SESSION superglobal array.

The code then includes an HTML document that includes a PHP script that prints the value of the "favcolor" session variable using the $_SESSION array. The script will display "Favorite color is green." on the page.

It is worth noting that sessions use a session ID that is stored in a cookie on the client's browser, and the data is stored on the server. This allows the server to associate the data with a specific user across multiple requests. Also, it is important to call the **session_start()** function before any output is sent to the browser, otherwise the session will not be created.

```
<?php
session_start();
// set session variables
$_SESSION["favcolor"] = "green";
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<?php
// print a session variable that was set before
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
?>
</body>
</html>
```

custom error

The code is an example of custom error handling in PHP. It defines a custom error handling function called **customError()** which will be called when an error occurs. The function takes two parameters: **$errno**, which is the error number, and **$errstr**, which is the error message. The function then echoes a message with the error number and error message.

The code then sets this custom error handler function to be used with the **set_error_handler()** function. This function takes two parameters: the name of the function to be used as the error handler and the error level to be handled by the error handler. In this case, it is set to handle E_WARNING level errors.

After this, the code triggers an error by calling a **require** statement to include a file that does not exist, 'fileDoesNotExist.txt', this will throw an error because the file does not exist. The error handler function **customError()** will be called, displaying the message "unfortunately an error happened: [2] fileDoesNotExist.txt: No such file or directory"

```
<?php
//example of custom error handling
// this function will be called when an error occurs
function customError($errno, $errstr) {
    echo "<b>unfortunately an error happened:</b> [$errno] $errstr";
}

// set error handler
set_error_handler("customError", E_WARNING);

// The above function activates the custom error handler,
// and the error handler will be called when an error occurs

// trigger error on false require
require ('fileDoesNotExists.txt'); // this will throw an error because
the file does not exist
```

custom error die

The code is checking if a file called 'test.txt' exists, using the **file_exists()** function. If the file does not exist, the script calls the **die()** function and prints the string 'Error: this file does not exist' and exits the script. If the file does exist, the script continues to execute the code that follows this block.

The **die()** function is similar to the **exit()** function in that it terminates the execution of the script. However, **die()** also sends a message to the browser. The message parameter is optional, but when provided it is used as the error message. In this case, the message 'Error: this file does not exist' is displayed to the browser.

```
<?php
// if file test.txt does not exist, it will throw an error by using die
function
if (!file_exists('test.txt')) {
    die('Error: this file does not exist');
}
```

die

The code is checking if the value 10 is numeric using the **is_numeric()** function. If the value is not numeric, the script calls the **die()** function and prints the string 'This is not a number' and exits the script. If the value is numeric, the script continues to execute the code that follows this block.

The **is_numeric()** function is used to check if a variable is a number or a numeric string. It returns true if the variable is numeric, otherwise, it returns false. In this case, the variable is a number, so the function returns true, and the code inside the if statement will not execute.

die() function is similar to the **exit()** function in that it terminates the execution of the script. However, **die()** also sends a message to the browser. The message parameter is optional, but when provided it is used as the error message. In this case, the message 'This is not a number' is displayed to the browser.

```
<?php
// You can also trigger or display your own error by using one of the
built in functions
// such as die()

// if number 10 is not a number, it will throw an error
if (!is_numeric(10)) {
    die('This is not a number');
}
```

error to db

The code is an example of custom error handling in PHP that save the error messages into a database. It connects to a MySQL database using the PHP Data Objects (PDO) extension, which allows for a more secure and flexible way to interact with databases. The script sets the connection details such as the host, database name, user and password.

It then defines a function **saveError()** that takes two parameters: **$errno**, which is the error number and **$errstr**, which is the error message. The function then uses a prepared statement and the PDO object to insert the error message into the 'errors' table in the database. The function also echoes a message with the error number and error message.

The script sets the custom error handler function **saveError()** to be used with the **set_error_handler()** function. This function takes two parameters: the name of the function to be used as the error handler and the error level to be handled by the error handler. In this case, it is set to handle E_WARNING level errors.

After this, the code triggers an error by calling a **require** statement to include a file that does not exist, 'fileDoesNotExist.txt', this will throw an error because the file does not exist. The error handler function **saveError()** will be called, it will save the error message "fileDoesNotExist.txt" into the errors table and also display the message 'Error: fileDoesNotExist.txt' to the browser.

It is important to note that, the table **errors** should be created before running the script. The code provided a query to create the table.

```
<?php
/*
 * run this query in your database
```

```
 *  CREATE TABLE `errors` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `errstr` text NOT NULL,
   PRIMARY KEY (`id`)
 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
 */
// Save error messages into database

// connect to database pdo
$dsn = 'mysql:host=localhost;dbname=php';
$user = 'root';
$pass = 'root';

$con = new PDO($dsn, $user, $pass);

//  function to save error messages into database
function saveError($errno, $errstr) {
    global $con;
    $sql = 'INSERT INTO errors (errstr) VALUES (:errstr)';
    $stmt = $con->prepare($sql);
    $stmt->execute(array(
        ':errstr' => $errstr,
    ));
    echo 'Error: ' . $errstr;
}

// set error handler
set_error_handler('saveError', E_WARNING);

// trigger error on false require
require ('fileDoesNotExists.txt'); // this will throw an error because
the file does not exist

// create a table with auto increment id and errstr and utf8
```

exception

The code is an example of error handling using the try and catch statements in PHP. It attempts to connect to a MySQL database using the PHP Data Objects (PDO) extension but with wrong credentials.

The **try** block contains the code that may throw an exception, in this case, the code is trying to connect to the database but with a wrong password. If an exception is thrown, the code in the **catch** block will be executed.

The **catch** block receives an exception object as its parameter, in this case, it's **$e** which is an instance of the **Exception** class. It then uses the **getMessage()** method of the exception object to get the error message and echo it.

In this case, the script will throw an exception when the PDO object is created because the provided

password is wrong. The catch block will catch this exception and the message "Caught exception: SQLSTATE[HY000] [1045] Access denied for user 'root'@'localhost' (using password: YES)" will be displayed.

Using try-catch blocks allows you to handle errors in a specific way, in this case, it is to display the error message. It also allows you to separate the error-prone code from the error-handling code, making it more readable and maintainable.

```
<?php
// We can also handle errors by using the try and catch statements
// this is useful when you want to handle errors in a different way
// trigger error on false require
// this works usually with oop functions

try {
    // connect to database pdo with the WRONG password
    $dsn = 'mysql:host=localhost;dbname=php';
    $user = 'root';
    $pass = 'wrongpassword';

    $con = new PDO($dsn, $user, $pass);
} catch (Exception $e) {
    echo 'Caught exception: '. $e->getMessage();
}
```

logPHP

The code is an example of custom error handling in PHP that logs errors to a file. It defines a custom error handling function called **customError()** which will be called when an error occurs. The function takes two parameters: **$errno**, which is the error number, and **$errstr**, which is the error message.

The function then uses the **error_log()** function to log the error message to a file called "error.log". The **error_log()** function takes three parameters: the message, the message type, and the destination. In this case, the message is "Error: [$errno] $errstr", the message type is 3, which means that the message is being logged to a file and the destination is "error.log".

The script also sets this custom error handler function to be used with the **set_error_handler()** function. This function takes two parameters: the name of the function to be used as the error handler and the error level to be handled by the error handler. In this case, it is set to handle E_WARNING level errors.

After this, the code triggers an error by calling a **require** statement to include a file that does not exist, 'fileDoesNotExist.txt', this will throw an error because the file does not exist. The error handler function **customError()** will be called, it will log the error message "fileDoesNotExist.txt" to the "error.log" file and also display the message 'Error: 2 fileDoesNotExist.txt' to the browser.

This method allows you to keep track of errors that occur on a live website, and can also be useful for debugging. You can also send an email to the admin when an error occurs by using the same

error_log function by providing the email address as the destination.

```
<?php
// you cam also log php errors to a file, when they happen
// this is useful when you have a live website and you want to keep track
of errors
// you can also send an email to the admin when an error occurs
// this is done by using the error_log() function
// the error_log() function sends an error to a file or email


// this function will be called when an error occurs
function customError($errno, $errstr) {
    // this will log the error to a file
    error_log("Error: [$errno] $errstr", 3, "error.log");
    echo "Error: [$errno] $errstr";
}

// set error handler
set_error_handler("customError", E_WARNING);

// trigger error on false require

require ('fileDoesNotExists.txt'); // this will throw an error because
the file does not exist
```

mailError

The code is an example of custom error handling in PHP that sends errors to an email address. It defines a custom error handling function called **customError()** which will be called when an error occurs. The function takes two parameters: **$errno**, which is the error number, and **$errstr**, which is the error message.

The function then uses the **mail()** function to send an email to the admin. The **mail()** function takes three parameters: the recipient email address, the subject of the email, and the message. In this case, the recipient email address is 'admin@website.com', the subject is 'Error' and the message is $errstr.

The script also sets this custom error handler function to be used with the **set_error_handler()** function. This function takes two parameters: the name of the function to be used as the error handler and the error level to be handled by the error handler. In this case, it is set to handle E_WARNING level errors.

After this, the code triggers an error by calling a **require** statement to include a file that does not exist, 'fileDoesNotExists.txt', this will throw an error because the file does not exist. The error handler function **customError()** will be called, it will send an email to the admin with the error message "fileDoesNotExist.txt" and also display the message 'Error: 2 fileDoesNotExist.txt' to the browser.

This method allows you to keep track of errors that occur on a live website, and can also be useful for debugging. It also allows the administrator to receive an email with the error message and take necessary actions. Please note that the mail function uses the server's local mail transfer agent to deliver the mail and it may not work in all server configurations and also it is not recommended to use this function for a large number of emails as it may overload the server.

```
<?php
// You can also email errors to the admin
// this is done by using the error_log() function
// the error_log() function sends an error to email

// this function will be called when an error occurs
function customError($errno, $errstr)
{
    // send email to admin
    mail('admin@website.com', 'Error', $errstr);
    echo "Error: [$errno] $errstr";
}

// set error handler
set_error_handler("customError", E_WARNING);

// trigger error on false require
require ('fileDoesNotExists.txt'); // this will throw an error because
the file does not exist
```

2d

To draw 2D shapes in PHP, you can use the GD library, which is a built-in library for creating and manipulating images in PHP.

Here is an example of how to draw a square using the GD library in PHP:

```
$image = imagecreatetruecolor(100, 100); // Allocate a color for the
square
$color = imagecolorallocate($image, 255, 0, 0); // Draw the square
imagefilledrectangle($image, 0, 0, 99, 99, $color); // Output the image
header('Content-Type: image/png');
imagepng($image); // Free up memory
imagedestroy($image);
```

This will create a 100x100 pixel image with a red square drawn on it.

To draw other shapes, you can use the various drawing functions provided by the GD library. For example, to draw a triangle, you can use the **imageline** function to draw three lines connecting the vertices of the triangle:

```
$image = imagecreatetruecolor(100, 100); // Allocate a color for the
triangle
$color = imagecolorallocate($image, 255, 0, 0); // Draw the triangle
imageline($image, 50, 0, 0, 100, $color);
imageline($image, 50, 0, 100, 100, $color);
imageline($image, 0, 100, 100, 100, $color); // Output the image
header('Content-Type: image/png');
imagepng($image); // Free up memory
imagedestroy($image);
```

You can also use the **imagepolygon** function to draw more complex shapes. For example, to create a star shape, you can use the following code:

```
$image = imagecreatetruecolor(200, 200); // Allocate a color for the
star
$color = imagecolorallocate($image, 255, 255, 0); // Define the points
for the star shape
$points = array( 100, 10, 130, 30, 170, 30, 120, 50, 140, 90, 100, 70,60,
90, 80, 50, 30, 30, 70, 30 ); // Draw the star imagepolygon($image,
$points, 8, $color); // Output the image
header('Content-Type: image/png');
imagepng($image); // Free up memory
imagedestroy($image);
```

3d

To draw 3D shapes in PHP, you can use a 3D graphics library such as OpenGL or 3D.js.

OpenGL is a cross-platform API for rendering 2D and 3D graphics. It is widely used in computer graphics and is supported by many operating systems and hardware platforms. To use OpenGL in PHP, you can use an extension such as PHP-GL or use a wrapper library such as GLFW or SDL.

Here is an example of how you can use the PHP-GL extension to draw a 3D cube in PHP:

```
<?php


// Set the dimensions of the cube
$width = 200;
$height = 200;
$depth = 200;

// Create a blank image
$im = imagecreatetruecolor($width, $height);
```

```
// Allocate colors for the cube
$color1 = imagecolorallocate($im, 255, 0, 0);
$color2 = imagecolorallocate($im, 0, 255, 0);
$color3 = imagecolorallocate($im, 0, 0, 255);
$color4 = imagecolorallocate($im, 255, 255, 0);
$color5 = imagecolorallocate($im, 0, 255, 255);
$color6 = imagecolorallocate($im, 255, 0, 255);

// Draw the front face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2 + $depth, $height / 2,
    $width / 2 + $depth, $height / 2 + $height,
    $width / 2, $height / 2 + $height
), 4, $color1);

// Draw the back face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2, $height / 2 + $height,
    $width / 2 - $depth, $height / 2 + $height,
    $width / 2 - $depth, $height / 2
), 4, $color2);

// Draw the left face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2 - $depth, $height / 2,
    $width / 2 - $depth, $height / 2 - $height,
    $width / 2, $height / 2 - $height
), 4, $color3);

// Draw the right face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2, $height / 2 - $height,
    $width / 2 + $depth, $height / 2 - $height,
    $width / 2 + $depth, $height / 2
), 4, $color4);

// Draw the top face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2 - $height,
    $width / 2 + $depth, $height / 2 - $height,
    $width / 2 + $depth, $height / 2,
    $width / 2, $height / 2
), 4, $color5);

// Draw the bottom face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2 + $height,
```

```
    $width / 2, $height /
    2, $width / 2 + $depth, $height / 2,
    $width / 2 + $depth, $height / 2 + $height
), 4, $color6);

// Output the image
header('Content-Type: image/png');
imagepng($im);

// Free up memory
imagedestroy($im);

// Set the dimensions of the cube
$width = 200;
$height = 200;
$depth = 200;

// Create a blank image
$im = imagecreatetruecolor($width, $height);

// Allocate colors for the cube
$color1 = imagecolorallocate($im, 255, 0, 0);
$color2 = imagecolorallocate($im, 0, 255, 0);
$color3 = imagecolorallocate($im, 0, 0, 255);
$color4 = imagecolorallocate($im, 255, 255, 0);
$color5 = imagecolorallocate($im, 0, 255, 255);
$color6 = imagecolorallocate($im, 255, 0, 255);

// Draw the front face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2 + $depth, $height / 2,
    $width / 2 + $depth, $height / 2 + $height,
    $width / 2, $height / 2 + $height
), 4, $color1);

// Draw the back face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2, $height / 2 + $height,
    $width / 2 - $depth, $height / 2 + $height,
    $width / 2 - $depth, $height / 2
), 4, $color2);

// Draw the left face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2 - $depth, $height / 2,
    $width / 2 - $depth, $height / 2 - $height,
    $width / 2, $height / 2 - $height
), 4, $color3);
```

```
// Draw the right face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2,
    $width / 2, $height / 2 - $height,
    $width / 2 + $depth, $height / 2 - $height,
    $width / 2 + $depth, $height / 2
), 4, $color4);

// Draw the top face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2 - $height,
    $width / 2 + $depth, $height / 2 - $height,
    $width / 2 + $depth, $height / 2,
    $width / 2, $height / 2
), 4, $color5);

// Draw the bottom face of the cube
imagefilledpolygon($im, array(
    $width / 2, $height / 2 + $height,
    $width / 2, $height /
    2, $width / 2 + $depth, $height / 2,
    $width / 2 + $depth, $height / 2 + $height
), 4, $color6);

// Output the image
header('Content-Type: image/png');
imagepng($im);

// Free up memory
imagedestroy($im);

?>
```

Single line

In PHP, single-line comments are used to add brief notes or explanations directly within the code. These comments are ignored by the PHP engine during execution and are purely for the benefit of developers reading the code. There are two syntaxes for single-line comments in PHP:

Using Double Slashes (//)

You can create a single-line comment using double slashes. Everything following the // on that line will be considered a comment.

```
<?php
// This is a single-line comment using double slashes
echo "Hello, World!"; // This comment is after a line of code
?>
```

Using a Hash Symbol (#)

You can also create a single-line comment using a hash symbol. Everything following the # on that line will be considered a comment.

```
<?php
# This is a single-line comment using a hash symbol
echo "Hello, World!"; # This comment is after a line of code
?>
```

Examples

Here are some examples demonstrating both types of single-line comments:

```
<?php
// This is a comment explaining the next line of code
echo "Hello, World!";
// You can also place a comment after a statement
echo "PHP is fun!"; // This is an inline comment
# This is a comment using the hash symbol
echo "Learning PHP!";
# Inline comment with hash symbol
echo "Another example"; # This is also an inline comment
?>
```

Using comments effectively can help make your code more readable and maintainable by providing context and explanations where necessary.

Ai

Let's create a simple PHP script that interacts with a MySQL database to perform a basic AI function. In this example, we'll build a very rudimentary recommendation system. This system will recommend items to a user based on their previous selections.

Requirements

1. PHP environment (like XAMPP)
2. MySQL Database
3. PHP MySQLi extension (usually included in PHP installations)

Database Setup

Let's assume you have a MySQL database with the following tables:

- **searchItems**: Contains items that can be recommended.

Here's a simplified structure for these tables:

```
CREATE TABLE searchItems (
   id INT AUTO_INCREMENT PRIMARY KEY,
```

```
   name VARCHAR(100)
);
```

PHP Script

The following PHP script demonstrates how you might write a simple recommendation algorithm. This script will recommend items that have not been selected by the user but are popular among other users

```
<?php
class DatabaseOperations {
    private $conn;

    public function __construct($servername, $username, $password,
$dbname) {
        $this->conn = new mysqli($servername, $username, $password,
$dbname);
        if ($this->conn->connect_error) {
            die("Connection failed: " . $this->conn->connect_error);
        }
    }

    public function searchItems($searchTerm) {
        $searchTerm = $this->conn->real_escape_string($searchTerm);
        $sql = "SELECT * FROM articles WHERE description LIKE
'%$searchTerm%' ORDER BY RAND() LIMIT 1";
        $result = $this->conn->query($sql);
        $items = array();
        if ($result->num_rows > 0) {
            while($row = $result->fetch_assoc()) {
                $items[] = $row;
            }
        }
        return $items;
    }
    public function closeConnection() {
        $this->conn->close();
    }
}
?>
```

index.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Item Search</title>
```

```
</head>
<body>
<h1>Search for Items</h1>
<form method="get" action="index.php">
    <input type="text" name="searchTerm" placeholder="Enter search term"
required>
    <input type="submit" value="Search">
</form>

<?php
if (isset($_GET['searchTerm'])) {
    require_once 'DatabaseOperations.php';
    $db = new DatabaseOperations('localhost', 'root', 'root', 'db');
    $searchTerm = $_GET['searchTerm'];

    $results = $db->searchItems($searchTerm);
    if (count($results) > 0) {
        echo "<h2>Search Results:</h2>";
        echo "<ul>";
        foreach ($results as $item) {
            echo "<li>" . htmlspecialchars_decode($item['description']) .
"</li>";
        }
        echo "</ul>";
    } else {
        echo "<p>No items found.</p>";
    }

    $db->closeConnection();
}
?>
</body>
</html>
```

Explanation

- This script connects to a MySQL database.
- The function **searchItems** takes a user ID and finds the top 5 items that the user has not selected but are popular among other users.
- We test the function by passing a user ID (assumed to be 1 in this example) and output the recommendations.

Note

- Replace **localhost**, **your_username**, **your_password**, and **your_dbname** with your actual database details.
- This is a basic example. Real-world applications would need a more sophisticated approach, especially for more complex AI functionalities.
- Ensure your PHP environment is configured correctly to connect to MySQL.

constructor

A constructor in PHP is a special method that is automatically called when an object of a class is created. It is used to initialize the properties of the class. The constructor method in PHP is named __construct.

Example: Using a Constructor

Let's expand on the previous Calculator example by adding a constructor that initializes some properties.

```
<?php
// Define the Calculator class
class Calculator {
   // Properties
   private $a;
   private $b;
   // Constructor
   public function __construct($a, $b) {
       $this->a = $a;
       $this->b = $b;
   }
   // Method to add the properties
   public function add() {
       return $this->a + $this->b;
   }
   // Method to subtract the properties
   public function subtract() {
       return $this->a - $this->b;
   }
   // Method to multiply the properties
   public function multiply() {
       return $this->a * $this->b;
   }
   // Method to divide the properties
   public function divide() {
       if ($this->b == 0) {
           return "Error: Division by zero";
       }
       return $this->a / $this->b;
   }
}
// Create an instance of the Calculator class
$calculator = new Calculator(10, 5);
// Call the methods of the Calculator class and display the results
echo "Addition: " . $calculator->add() . "\n";         // Outputs:
Addition: 15
echo "Subtraction: " . $calculator->subtract() . "\n"; // Outputs:
Subtraction: 5
echo "Multiplication: " . $calculator->multiply() . "\n"; // Outputs:
Multiplication: 50
```

```
echo "Division: " . $calculator->divide() . "\n";         // Outputs:
Division: 2
?>
```

Explanation

Class Definition:

- The class Calculator defines a new class named Calculator.
- Properties $a and $b are defined as private properties of the class.

Constructor:

- The __construct($a, $b) method is defined to initialize the properties $a and $b when a new object is created.
- private $a; and private $b; define the properties that will hold the values.

Methods:

- add(), subtract(), multiply(), and divide() are methods defined within the class to perform arithmetic operations on the properties $a and $b.

Creating an Instance:

- $calculator = new Calculator(10, 5); creates an instance of the Calculator class and initializes the properties $a and $b with the values 10 and 5.

Calling Methods:

- echo $calculator->add(); calls the add method on the $calculator object and prints the result.
- Similarly, other methods (subtract, multiply, divide) are called and their results are printed.

Constructor with Default Values

You can also define a constructor with default values for the parameters:

```
<?php
class Calculator {
   // Properties
   private $a;
   private $b;
   // Constructor with default values
   public function __construct($a = 0, $b = 0) {
       $this->a = $a;
       $this->b = $b;
   }
   // Methods
   public function add() {
       return $this->a + $this->b;
   }
   public function subtract() {
       return $this->a - $this->b;
```

```
    }
    public function multiply() {
        return $this->a * $this->b;
    }
    public function divide() {
        if ($this->b == 0) {
            return "Error: Division by zero";
        }
        return $this->a / $this->b;
    }
}
// Create instances of the Calculator class
$calculator1 = new Calculator(10, 5);
$calculator2 = new Calculator();
// Call the methods of the Calculator class and display the results
echo "Calculator1 Addition: " . $calculator1->add() . "\n"; // Outputs:
Calculator1 Addition: 15
echo "Calculator2 Addition: " . $calculator2->add() . "\n"; // Outputs:
Calculator2 Addition: 0
?>
```

Explanation

Constructor with Default Values:

- The __construct($a = 0, $b = 0) method is defined to initialize the properties with default values if no arguments are passed when the object is created.

Creating Instances:

- $calculator1 = new Calculator(10, 5); creates an instance with specific values.
- $calculator2 = new Calculator(); creates an instance with default values.

Calling Methods:

- Methods are called on both instances to demonstrate the use of default values in the constructor.

Summary

- **Constructor**: A special method named __construct that initializes the properties of the class.
- **Creating an Instance**: Use new ClassName(parameters) to create an object and pass arguments to the constructor.
- **Default Values**: Constructors can have default values for parameters to handle cases where no arguments are provided.

Multi-line

In PHP, multi-line comments are used to provide longer explanations or documentation that spans multiple lines. Multi-line comments begin with /* and end with */. Everything between these delimiters is considered a comment and is ignored by the PHP engine during execution.

Syntax

The syntax for multi-line comments is as follows:

```
<?php
/*
This is a multi-line comment.
It can span multiple lines.
Use it to provide detailed explanations or documentation.
*/
?>
```

Example

Here's an example of a multi-line comment in a PHP script:

```
<?php
/*
This function calculates the sum of two numbers.
It takes two numeric parameters and returns their sum.
Make sure to provide valid numeric arguments when calling this function.
*/
function add($a, $b) {
   return $a + $b;
}
// Call the function with example values
echo add(5, 10);
?>
```

Usage Scenarios

Multi-line comments are particularly useful in the following scenarios:

- Providing detailed explanations or documentation for a block of code.
- Temporarily commenting out a block of code during development or debugging.
- Adding license information or author details at the beginning of a file.

Temporarily Commenting Out Code

Multi-line comments can be used to temporarily disable a block of code without deleting it:

```
<?php
/*
echo "This line is commented out and won't be executed.";
echo "This line is also commented out.";
*/
// This line will be executed
echo "Only this line will be executed.";
?>
```

Using multi-line comments effectively can help improve the readability and maintainability of your code by providing clear and concise documentation and explanations.

documentation

PHP documentation comments, also known as DocBlocks, are used to provide structured documentation for PHP code. These comments are typically used by tools like PHPDoc to generate API documentation automatically. DocBlocks start with /** and end with */, and they can include various tags to describe the code.

Basic Structure

Here's a basic example of a DocBlock for a PHP function:

```
/**
* Short description of the function.
*
* Longer description of the function, explaining its purpose and
behavior.
*
* @param type $paramName Description of the parameter.
* @return type Description of the return value.
*/
function exampleFunction($param) {
   // Function code here
}
```

Common Tags

Here are some of the most commonly used tags in PHP DocBlocks:

- @param - Describes a function parameter. Includes the type and a description.
- @return - Describes the return type and value of the function.
- @throws - Describes exceptions that the function might throw.
- @var - Describes the type of a variable.
- @deprecated - Indicates that the function or method is deprecated and should not be used.
- @see - References another element, like a related function or class.
- @since - Indicates the version of the project when the function was introduced.
- @author - Provides information about the author of the code.
- @version - Provides version information.

Example with Multiple Tags

Here’s a more comprehensive example showing various tags:

```
/**
* Adds two numbers together.
*
* This function takes two numeric parameters and returns their sum.
*
* @param int $a The first number.
```

```
* @param int $b The second number.
* @return int The sum of the two numbers.
* @throws InvalidArgumentException If non-numeric parameters are
provided.
* @since 1.0.0
* @see subtract()
*/
function add($a, $b) {
   if (!is_numeric($a) || !is_numeric($b)) {
       throw new InvalidArgumentException('Both parameters must be
numeric.');
   }
   return $a + $b;
}
```

Using DocBlocks with Classes and Properties

DocBlocks can also be used to document classes and their properties:

```
/**
* Represents a person.
*
* This class provides methods to get and set the person's name and age.
*
* @package MyPackage
*/
class Person {
   /**
    * @var string The name of the person.
    */
   private $name;
   /**
    * @var int The age of the person.
    */
   private $age;
   /**
    * Constructor for the Person class.
    *
    * @param string $name The name of the person.
    * @param int $age The age of the person.
    */
   public function __construct($name, $age) {
       $this->name = $name;
       $this->age = $age;
   }
   /**
    * Gets the name of the person.
    *
    * @return string The name of the person.
    */
   public function getName() {
```

```
        return $this->name;
    }
    /**
     * Sets the name of the person.
     *
     * @param string $name The new name of the person.
     * @return void
     */
    public function setName($name) {
        $this->name = $name;
    }
    /**
     * Gets the age of the person.
     *
     * @return int The age of the person.
     */
    public function getAge() {
        return $this->age;
    }
    /**
     * Sets the age of the person.
     *
     * @param int $age The new age of the person.
     * @return void
     */
    public function setAge($age) {
        $this->age = $age;
    }
}
```

By using DocBlocks, you can make your code more understandable and maintainable while also providing the necessary information for automatic documentation generation.

assignment

In PHP, variable assignment is straightforward. Variables in PHP are prefixed with a dollar sign ($). Here's a basic overview of how to assign values to variables in PHP:

Basic Variable Assignment

```
<?php
// Assigning a string value
$name = "John Doe";
// Assigning an integer value
$age = 30;
// Assigning a floating-point value
$height = 5.9;
// Assigning a boolean value
$is_student = true;
// Assigning an array
```

```
$colors = array("red", "green", "blue");
// Assigning an associative array
$user = array("name" => "Jane Doe", "age" => 25);
// Assigning a null value
$unknown = null;
?>
```

Variable Assignment by Reference

You can also assign variables by reference, which means that the new variable references the same value as the original variable. Any changes to one variable will affect the other.

```
<?php
$original = "Hello";
$reference = &$original;
$original = "Hi";
echo $reference; // Outputs "Hi"
?>
```

Variable Variables

PHP allows you to use the value of a variable as the name of another variable.

```
<?php
$a = 'hello';
$$a = 'world';
echo $hello; // Outputs "world"
?>
```

Constants

If you want to assign a value that cannot be changed, you use constants. Constants are defined using the define() function or the const keyword.

```
<?php
define("SITE_NAME", "My Website");
const SITE_URL = "https://www.example.com";
echo SITE_NAME; // Outputs "My Website"
echo SITE_URL;  // Outputs "https://www.example.com"
?>
```

Example with Conditional Assignment

You can assign values conditionally using the ternary operator.

```
<?php
$score = 85;
$result = ($score >= 60) ? 'Pass' : 'Fail';
echo $result; // Outputs "Pass"
```

```
?>
```

Assigning Values to Arrays

You can add values to arrays using different methods.

```
<?php
$fruits = array();
$fruits[] = "Apple";
$fruits[] = "Banana";
$fruits[] = "Cherry";
$fruits = array_merge($fruits, array("Date", "Elderberry"));
print_r($fruits); // Outputs the array with all the fruit names
?>
```

Concatenation

In PHP, you can concatenate strings using the dot operator (.). Here's a detailed explanation and some examples:

Basic Concatenation

```
<?php
$firstName = "John";
$lastName = "Doe";
// Concatenate first and last names
$fullName = $firstName . " " . $lastName;
echo $fullName; // Outputs "John Doe"
?>
```

Concatenation with Assignment

You can use the .= operator to append a string to an existing variable.

```
<?php
$greeting = "Hello";
$greeting .= " ";
$greeting .= "World!";
echo $greeting; // Outputs "Hello World!"
?>
```

Concatenation within Echo or Print

You can directly concatenate strings within an echo or print statement.

```
<?php
$city = "New York";
$state = "NY";
echo "I live in " . $city . ", " . $state . "."; // Outputs "I live in
```

```
New York, NY."
?>
```

Concatenation in Function Calls

You can concatenate strings when passing them as arguments to functions.

```
<?php
function welcomeMessage($name) {
   return "Welcome, " . $name . "!";
}
echo welcomeMessage("Alice"); // Outputs "Welcome, Alice!"
?>
```

Concatenation with Variables and Literals

You can mix variables and literals in concatenation.

```
<?php
$day = "Monday";
$activity = "coding";
$sentence = "On " . $day . "s, I enjoy " . $activity . ".";
echo $sentence; // Outputs "On Mondays, I enjoy coding."
?>
```

Concatenation in Loops

You can use concatenation within loops to build strings dynamically.

```
<?php
$items = array("apple", "banana", "cherry");
$list = "";
foreach ($items as $item) {
   $list .= $item . ", ";
}
// Remove the trailing comma and space
$list = rtrim($list, ", ");
echo $list; // Outputs "apple, banana, cherry"
?>
```

Concatenation with HTML

You can concatenate strings to create HTML content.

```
<?php
$title = "PHP Concatenation";
$body = "This is a simple example.";
$html = "<html><head><title>" . $title . "</title></head>";
$html .= "<body><p>" . $body . "</p></body></html>";
```

```
echo $html;
?>
```

Example with Variables and HTML

Combining variables with HTML for dynamic content:

```
<?php
$name = "John";
$age = 30;
echo "<div>Name: " . $name . "</div>";
echo "<div>Age: " . $age . "</div>";
?>
```

Functions

Echo

echo is a function in PHP that outputs one or more strings to the screen. It is often used to print out the value of a variable or the result of a calculation.

Here is an example of how to use the **echo** function in PHP:

```
$name = "John";
echo "Hello, " . $name;  // Outputs "Hello, John"
$a = 5;
$b = 10;
echo $a + $b;  // Outputs "15"
```

You can also use the **echo** function to output multiple strings by separating them with a comma. For example:

```
echo "Hello, ", $name;  // Outputs "Hello, John"
```

Note that the **echo** function does not add a newline character at the end of the string, so if you want to output multiple strings on separate lines, you will need to use the newline character (**\n**) or the **PHP_EOL** constant. For example:

```
echo "Hello, ", $name, "\n";  // Outputs "Hello, John" on a separate line
echo "Hello, ", $name, PHP_EOL;  // Same as above
```

Print

print is a function in PHP that outputs a string to the screen. It is similar to the **echo** function, but **print** can only output a single string, whereas **echo** can output multiple strings.

Here is an example of how to use the **print** function in PHP:

```
$name = "John";
print "Hello, " . $name;  // Outputs "Hello, John"
$a = 5;
$b = 10;
print $a + $b;  // Outputs "15"
```

Like the **echo** function, the **print** function does not add a newline character at the end of the string. If you want to output multiple strings on separate lines, you will need to use the newline character (**\n**) or the **PHP_EOL** constant.

```
print "Hello, " . $name . "\n";  // Outputs "Hello, John" on a separate
line
print "Hello, " . $name . PHP_EOL;  // Same as above
```

Note that **print** is a slower function than **echo**, so it is generally better to use **echo** when performance is important.

Print_r

print_r is a function in PHP that outputs the human-readable representation of a variable. It is often used for debugging purposes, as it allows you to see the contents of a variable, including arrays and objects.

Here is an example of how to use the **print_r** function in PHP:

```
$array = array(1, 2, 3, 4, 5);
print_r($array);  // Outputs "Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4
[4] => 5 )"
$object = new stdClass();
$object->name = "John";
$object->age = 30;
print_r($object);  // Outputs "stdClass Object ( [name] => John [age] =>
30 )"
```

You can also pass a second argument to **print_r** called **return**, which will cause the function to return the output as a string instead of printing it to the screen. This can be useful if you want to capture the output of **print_r** and use it in your code.

```
$output = print_r($array, true);
echo $output;  // Outputs "Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4
[4] => 5 )"
```

Printf

printf is a function in PHP that outputs a formatted string to the screen. It is similar to the **echo** and **print** functions, but **printf** allows you to use placeholders in the string and pass in values for those

placeholders as arguments.

Here is an example of how to use the **printf** function in PHP:

```
$name = "John";
$age = 30;
printf("Hello %s, you are %d years old.", $name, $age);  // Outputs
"Hello John, you are 30 years old."
```

In the example above, **%s** is a placeholder for a string, and **%d** is a placeholder for an integer. You can use different placeholders depending on the type of data you want to output. For example:

```
printf("The value of pi is approximately %.2f.", pi());  // Outputs "The
value of pi is approximately 3.14."
```

In the example above, **%.2f** is a placeholder for a floating-point number, and the **.2** specifies that the number should be formatted with two decimal places.

You can also use the **printf** function to output multiple strings by separating them with a comma. For example:

```
printf("Hello, %s\n", $name);  // Outputs "Hello, John" on a separate
line
```

Note that the **printf** function does not add a newline character at the end of the string, so if you want to output multiple strings on separate lines, you will need to use the newline character (**\n**) or the **PHP_EOL** constant.

var_dump is a function in PHP that outputs the type and value of a variable. It is often used for debugging purposes, as it provides more information about a variable than the **print_r** function.

Here is an example of how to use the **var_dump** function in PHP:

```
$array = array(1, 2, 3, 4, 5);
var_dump($array);  // Outputs "array(5) { [0]=> int(1) [1]=> int(2) [2]=>
int(3) [3]=> int(4) [4]=> int(5) }"
$object = new stdClass();
$object->name = "John";
$object->age = 30;
var_dump($object);  // Outputs "object(stdClass)#1 (2) { ["name"]=>
string(4) "John" ["age"]=> int(30) }"
```

As you can see, the **var_dump** function outputs the type of each value in the variable (e.g. "int" for integers, "string" for strings), as well as the actual value.

You can also pass multiple variables to the **var_dump** function, and it will output the type and value of each variable.

```
$a = 5;
$b = "hello";
var_dump($a, $b);  // Outputs "int(5) string(5) "hello""
```

Note that the **var_dump** function is generally more verbose than the **print_r** function, so it may produce a lot of output when used on large variables or data structures.

define and constants

In PHP, the **define()** function is used to define a constant. A constant is a value that cannot be changed once it has been set. Constants are global in scope, which means they can be accessed from anywhere in your PHP code.

Here is an example of how to use the **define()** function to define a constant:

```
define("MAX_WIDTH", 100);
echo MAX_WIDTH;  // prints 100
```

The **define()** function takes two arguments: the name of the constant and the value to be assigned to it. The constant name must be written in uppercase and it cannot be defined using a **$** symbol like a variable.

You can also define constants using the **const** keyword, which has been available since PHP 5.3.0. Here is an example of using the **const** keyword to define a constant:

```
const MAX_HEIGHT = 200;
echo MAX_HEIGHT;  // prints 200
```

Both **define()** and **const** are used to define constants in PHP, but **const** has some additional benefits. It is more intuitive to use, and it is also faster and more efficient because it is resolved at compile-time rather than runtime.

directory

Directory

In PHP, the magic constant **__DIR__** represents the directory of the current file. It is a built-in constant that is available in all PHP scripts and is automatically defined by the PHP interpreter.

Here is an example of how to use the **__DIR__** constant:

```
echo __DIR__;  // prints the directory of the current file
```

The **__DIR__** constant is often used to include files or set paths dynamically, as it allows you to reference the current directory without hardcoding the path.

For example, you can use **__DIR__** to include a configuration file in your script:

```
require_once __DIR__ . '/config.php';
```

Or you can use **__DIR__** to set a dynamic base path for your application:

```
define('BASE_PATH', __DIR__);
```

The **__DIR__** constant is a useful tool for working with files and directories in your PHP scripts. It can be particularly helpful when working with include paths or when creating dynamic file paths that need to be relative to the current directory.

Note that the **__DIR__** constant was introduced in PHP 5.3.0, so it may not be available in older versions of PHP. In earlier versions, you can use the **dirname(__FILE__)** expression to achieve the same result.

file

File

In PHP, the magic constant **__FILE__** represents the full path and filename of the current file. It is a built-in constant that is available in all PHP scripts and is automatically defined by the PHP interpreter.

Here is an example of how to use the **__FILE__** constant:

```
echo __FILE__;  // prints the full path and filename of the current file
```

The **__FILE__** constant is often used to include files or set paths dynamically, as it allows you to reference the current file or directory without hardcoding the path.

For example, you can use **__FILE__** to include a configuration file in your script:

```
require_once __DIR__ . '/config.php';
```

Or you can use **__FILE__** to set a dynamic base path for your application:

```
define('BASE_PATH', dirname(__FILE__));
```

The **__FILE__** constant is a useful tool for working with files and directories in your PHP scripts. It can be particularly helpful when working with include paths or when creating dynamic file paths that need to be relative to the current file.

Functions

Function

In PHP, a function is a block of code that can be called by name. Functions allow you to reuse code and modularize your scripts by breaking them up into smaller, self-contained units of functionality.

Here is an example of a simple function in PHP:

```
function sayHello($name) {
 echo "Hello, $name!";
}
```

To call a function, you simply use its name followed by parentheses and any necessary arguments:

```
sayHello("John");  // prints "Hello, John!"
```

Functions can also accept multiple arguments and return a value:

```
function add($x, $y) {
 return $x + $y;
}
echo add(2, 3);  // prints 5
```

Functions can also be defined inside classes, in which case they are called methods. Here is an example of a method in a class:

```
class Person {
 public $name;

 public function sayHello() {
   echo "Hello, my name is $this->name.";
 }
}
```

Functions are a powerful and important feature of PHP, as they allow you to reuse code and modularize your scripts. They also help to make your code more organized and easier to maintain.

Compound

5. Array

An array can hold multiple values at once.

```
$array1 = array("apple", "banana", "cherry");
$array2 = ["apple", "banana", "cherry"];
```

6. Object

An object is an instance of a class. You create objects from classes.

```
class Fruit {
   public $name;
   function __construct($name) {
       $this->name = $name;
   }
}
```

```
$apple = new Fruit("Apple");
```

Special Data Types

7. Resource

A resource is a special variable, holding a reference to an external resource, like a file, a database connection, image canvas areas, etc.

```
$file = fopen("text.txt", "r");
```

8. NULL

NULL is a special data type that only has one value: **NULL**.

```
$nullVar = NULL;
```

Checking Data Types

You can use various functions to check a variable's data type, like **is_int()**, **is_float()**, **is_string()**, **is_bool()**, **is_array()**, **is_object()**, **is_resource()**, and **is_null()**.

```
$var = "Hello, World!";
if(is_string($var)) {
   echo '$var is a string.';
}
```

Type Casting

You can also convert the data type of variables by type casting.

```
$int = 1;
$str = (string)$int; // convert integer to string
```

Type Juggling

PHP is capable of automatically converting a variable from one data type to another, depending on the context in which it is used. This is known as "type juggling".

```
$var = "3" + 4; // $var becomes integer 7
```

Data types in PHP allow the language to be quite flexible, but it's crucial to be mindful of the kind of data you're working with to prevent unexpected behavior, especially when dealing with different types of data simultaneously.

Time Zones

Handling time zones in PHP with the DateTime and DateTimeZone classes is straightforward. Here's how you can work with time zones:

Setting a Time Zone

You can set a specific time zone when creating a DateTime object by passing a DateTimeZone object to the constructor. Here is an example:

```
<?php
$date = new DateTime('now', new DateTimeZone('America/New_York'));
echo $date->format('Y-m-d H:i:s'); // Outputs the current date and time
in New York time zone
?>
```

Changing the Time Zone of an Existing DateTime Object

If you need to change the time zone of an existing DateTime object, you can use the setTimezone method:

```
<?php
$date = new DateTime('now', new DateTimeZone('America/New_York'));
echo $date->format('Y-m-d H:i:s'); // Current time in New York
$date->setTimezone(new DateTimeZone('Europe/London'));
echo $date->format('Y-m-d H:i:s'); // Current time in London
?>
```

Getting the Current Date and Time in Different Time Zones

You can easily create multiple DateTime objects for different time zones:

```
<?php
$dateNY = new DateTime('now', new DateTimeZone('America/New_York'));
$dateLondon = new DateTime('now', new DateTimeZone('Europe/London'));
$dateTokyo = new DateTime('now', new DateTimeZone('Asia/Tokyo'));
echo 'New York: ' . $dateNY->format('Y-m-d H:i:s') . "\n";
echo 'London: ' . $dateLondon->format('Y-m-d H:i:s') . "\n";
echo 'Tokyo: ' . $dateTokyo->format('Y-m-d H:i:s') . "\n";
?>
```

Handling UTC

Often, it's useful to work with UTC time to ensure consistency, especially when dealing with servers in different time zones. You can create a DateTime object in UTC like this:

```
<?php
$dateUTC = new DateTime('now', new DateTimeZone('UTC'));
echo $dateUTC->format('Y-m-d H:i:s'); // Current date and time in UTC
?>
```

Converting Between Time Zones

You can also convert between time zones by creating a DateTime object in one time zone and then

changing it to another:

```
<?php
$date = new DateTime('2024-05-30 14:30:00', new
DateTimeZone('America/New_York'));
echo 'Original (New York): ' . $date->format('Y-m-d H:i:s') . "\n";
$date->setTimezone(new DateTimeZone('Europe/London'));
echo 'Converted (London): ' . $date->format('Y-m-d H:i:s') . "\n";
?>
```

Listing Available Time Zones

You can list all available time zones using the timezone_identifiers_list function:

```
<?php
$timezones = DateTimeZone::listIdentifiers();
foreach ($timezones as $timezone) {
   echo $timezone . "\n";
}
?>
```

By leveraging the DateTime and DateTimeZone classes, you can effectively manage dates and times across different time zones in PHP. For more details, refer to the PHP documentation for the DateTimeZone class.

Timestamp

In PHP, you can get the current timestamp using the time() function, which returns the current time as a Unix timestamp (the number of seconds since the Unix Epoch, January 1 1970 00:00:00 UTC).

Using the time() Function

```
<?php
$timestamp = time();
echo $timestamp; // Outputs the current Unix timestamp
?>
```

Using the DateTime Class

You can also get the current timestamp using the DateTime class. Here is how you can do it:

```
<?php
$date = new DateTime();
$timestamp = $date->getTimestamp();
echo $timestamp; // Outputs the current Unix timestamp
?>
```

Getting the Timestamp for a Specific Date and Time

If you need the timestamp for a specific date and time, you can create a DateTime object for that date and then get the timestamp:

```
<?php
$date = new DateTime('2024-05-30 14:30:00', new
DateTimeZone('America/New_York'));
$timestamp = $date->getTimestamp();
echo $timestamp; // Outputs the Unix timestamp for the specified date and
time in New York
?>
```

Converting a Timestamp Back to a Date

You can convert a Unix timestamp back to a date using the DateTime class:

```
<?php
$timestamp = 1717085400; // Example timestamp
$date = new DateTime();
$date->setTimestamp($timestamp);
echo $date->format('Y-m-d H:i:s'); // Outputs the date and time
corresponding to the timestamp
?>
```

Example: Current Timestamp in Different Time Zones

Here is a complete example that demonstrates getting the current timestamp and converting it to different time zones:

```
<?php
// Get current timestamp
$currentTimestamp = time();
echo "Current Timestamp: " . $currentTimestamp . "\n";
// Create DateTime object from timestamp
$dateUTC = new DateTime("@$currentTimestamp");
$dateUTC->setTimezone(new DateTimeZone('UTC'));
echo "UTC Time: " . $dateUTC->format('Y-m-d H:i:s') . "\n";
// Convert to New York time
$dateNY = clone $dateUTC;
$dateNY->setTimezone(new DateTimeZone('America/New_York'));
echo "New York Time: " . $dateNY->format('Y-m-d H:i:s') . "\n";
// Convert to London time
$dateLondon = clone $dateUTC;
$dateLondon->setTimezone(new DateTimeZone('Europe/London'));
echo "London Time: " . $dateLondon->format('Y-m-d H:i:s') . "\n";
// Convert to Tokyo time
$dateTokyo = clone $dateUTC;
$dateTokyo->setTimezone(new DateTimeZone('Asia/Tokyo'));
echo "Tokyo Time: " . $dateTokyo->format('Y-m-d H:i:s') . "\n";
```

```
?>
```

This code snippet first gets the current timestamp, then creates a DateTime object from it, and finally converts the timestamp to different time zones (UTC, New York, London, and Tokyo) and prints the corresponding date and time for each.

Formatting

PHP provides extensive support for date and time formatting, allowing you to display dates and times in various formats. Here are some common examples of how to format dates and times using the DateTime class and the date() function.

Using the DateTime Class

The DateTime class provides a format() method that you can use to format dates and times. Here are some examples:

```
<?php
$date = new DateTime();
// Format as 'Y-m-d' (e.g., 2024-05-30)
echo $date->format('Y-m-d') . "\n";
// Format as 'd/m/Y' (e.g., 30/05/2024)
echo $date->format('d/m/Y') . "\n";
// Format as 'l, F j, Y' (e.g., Thursday, May 30, 2024)
echo $date->format('l, F j, Y') . "\n";
// Format as 'H:i:s' (e.g., 14:30:00)
echo $date->format('H:i:s') . "\n";
// Format as 'Y-m-d H:i:s' (e.g., 2024-05-30 14:30:00)
echo $date->format('Y-m-d H:i:s') . "\n";
?>
```

Using the date() Function

The date() function is useful for formatting the current date and time or a specific timestamp. Here are some examples:

```
<?php
// Current date in 'Y-m-d' format
echo date('Y-m-d') . "\n";
// Current date in 'd/m/Y' format
echo date('d/m/Y') . "\n";
// Current date and time in 'Y-m-d H:i:s' format
echo date('Y-m-d H:i:s') . "\n";
// Format a specific timestamp (e.g., 1717085400) as 'l, F j, Y'
$timestamp = 1717085400;
echo date('l, F j, Y', $timestamp) . "\n";
?>
```

Custom Date and Time Formats

You can create custom date and time formats using various format characters:

- Y - Four-digit year (e.g., 2024)
- y - Two-digit year (e.g., 24)
- m - Two-digit month (e.g., 05)
- n - Month without leading zeros (e.g., 5)
- d - Two-digit day of the month (e.g., 30)
- j - Day of the month without leading zeros (e.g., 30)
- H - 24-hour format of an hour with leading zeros (e.g., 14)
- G - 24-hour format of an hour without leading zeros (e.g., 14)
- i - Minutes with leading zeros (e.g., 30)
- s - Seconds with leading zeros (e.g., 00)
- l - Full textual representation of the day of the week (e.g., Thursday)
- F - Full textual representation of a month (e.g., May)

Formatting Date and Time with Time Zones

You can also include time zone information when formatting dates and times:

```
<?php
$date = new DateTime('now', new DateTimeZone('America/New_York'));
// Format date and time with time zone abbreviation
echo $date->format('Y-m-d H:i:s T') . "\n"; // e.g., 2024-05-30 14:30:00
EDT
// Format date and time with full time zone name
echo $date->format('Y-m-d H:i:s e') . "\n"; // e.g., 2024-05-30 14:30:00
America/New_York
// Format date and time with time zone offset
echo $date->format('Y-m-d H:i:s P') . "\n"; // e.g., 2024-05-30 14:30:00
-04:00
?>
```

These examples demonstrate how to format dates and times in various ways using PHP. By combining different format characters, you can customize the output to match your specific requirements. For more details on date and time formatting options, refer to the PHP documentation.

Concatenating Strings

In PHP, you can concatenate strings using the dot (.) operator. This operator combines two or more strings into a single string. Here are some examples to illustrate string concatenation in PHP:

Basic Concatenation

```
<?php
$string1 = "Hello";
$string2 = "World!";
$combinedString = $string1 . " " . $string2;
echo $combinedString; // Outputs: Hello World!
```

```
?>
```

Concatenating Multiple Strings

```
<?php
$part1 = "This ";
$part2 = "is ";
$part3 = "a ";
$part4 = "test.";
$combinedString = $part1 . $part2 . $part3 . $part4;
echo $combinedString; // Outputs: This is a test.
?>
```

Using Concatenation in Variables

```
<?php
$firstName = "John";
$lastName = "Doe";
$fullName = $firstName . " " . $lastName;
echo $fullName; // Outputs: John Doe
?>
```

Concatenation with Assignment Operator

PHP provides a convenient concatenation assignment operator (.=) to append a string to an existing variable.

```
<?php
$string = "Hello";
$string .= ", ";
$string .= "World!";
echo $string; // Outputs: Hello, World!
?>
```

Concatenation in Function Calls

You can also use concatenation directly in function calls.

```
<?php
function greet($name) {
   return "Hello, " . $name . "!";
}
echo greet("Alice"); // Outputs: Hello, Alice!
?>
```

Combining Strings and Variables

```
<?php
$greeting = "Hello";
```

```
$name = "John";
$combinedString = $greeting . ", " . $name . "!";
echo $combinedString; // Outputs: Hello, John!
?>
```

Example with HTML

String concatenation is often used in generating dynamic HTML content.

```
<?php
$title = "Welcome";
$content = "This is a dynamic webpage.";
$html = "<html><head><title>" . $title . "</title></head><body>";
$html .= "<h1>" . $title . "</h1>";
$html .= "<p>" . $content . "</p>";
$html .= "</body></html>";
echo $html;
// Outputs the complete HTML document with the dynamic content
?>
```

These examples show various ways to concatenate strings in PHP, demonstrating the flexibility and power of string manipulation in the language.

String Functions

PHP provides a wide range of built-in functions for manipulating strings. Here are some commonly used string functions along with examples:

strlen - Get the Length of a String

```
<?php
$string = "Hello, World!";
$length = strlen($string);
echo $length; // Outputs: 13
?>
```

strpos - Find the Position of the First Occurrence of a Substring

```
<?php
$haystack = "Hello, World!";
$needle = "World";
$position = strpos($haystack, $needle);
echo $position; // Outputs: 7
?>
```

str_replace - Replace All Occurrences of the Search String with the Replacement String

```
<?php
$text = "Hello, World!";
```

```
$newText = str_replace("World", "PHP", $text);
echo $newText; // Outputs: Hello, PHP!
?>
```

substr - Return Part of a String

```
<?php
$string = "Hello, World!";
$substring = substr($string, 7, 5);
echo $substring; // Outputs: World
?>
```

strtolower - Convert a String to Lowercase

```
<?php
$string = "Hello, World!";
$lowercase = strtolower($string);
echo $lowercase; // Outputs: hello, world!
?>
```

strtoupper - Convert a String to Uppercase

```
<?php
$string = "Hello, World!";
$uppercase = strtoupper($string);
echo $uppercase; // Outputs: HELLO, WORLD!
?>
```

ucfirst - Capitalize the First Character of a String

```
<?php
$string = "hello, world!";
$capitalized = ucfirst($string);
echo $capitalized; // Outputs: Hello, world!
?>
```

lcfirst - Make the First Character of a String Lowercase

```
<?php
$string = "Hello, World!";
$lowercased = lcfirst($string);
echo $lowercased; // Outputs: hello, World!
?>
```

ucwords - Capitalize the First Character of Each Word in a String

```
<?php
$string = "hello, world!";
```

```
$capitalizedWords = ucwords($string);
echo $capitalizedWords; // Outputs: Hello, World!
?>
```

trim - Strip Whitespace (or Other Characters) from the Beginning and End of a String

```
<?php
$string = "  Hello, World!  ";
$trimmed = trim($string);
echo $trimmed; // Outputs: Hello, World!
?>
```

ltrim - Strip Whitespace (or Other Characters) from the Beginning of a String

```
<?php
$string = "  Hello, World!  ";
$trimmed = ltrim($string);
echo $trimmed; // Outputs: Hello, World!
?>
```

rtrim - Strip Whitespace (or Other Characters) from the End of a String

```
<?php
$string = "  Hello, World!  ";
$trimmed = rtrim($string);
echo $trimmed; // Outputs:   Hello, World!
?>
```

explode - Split a String by a String

```
<?php
$string = "apple,banana,orange";
$array = explode(",", $string);
print_r($array);
// Outputs:
// Array
// (
//     [0] => apple
//     [1] => banana
//     [2] => orange
// )
?>
```

implode - Join Array Elements with a String

```
<?php
$array = ["apple", "banana", "orange"];
$string = implode(", ", $array);
echo $string; // Outputs: apple, banana, orange
```

```
?>
```

nl2br - Inserts HTML Line Breaks Before All Newlines in a String

```
<?php
$string = "Hello,\nWorld!";
$newString = nl2br($string);
echo $newString; // Outputs: Hello,<br />World!
?>
```

These examples cover many of the commonly used string functions in PHP. They can be used for various string manipulation tasks, from simple to complex operations.

Multiline Strings

In PHP, you can create multiline strings using different methods. The two most common ways are using Heredoc and Nowdoc syntaxes. Below are examples of each:

Heredoc Syntax

Heredoc syntax allows for the creation of multiline strings and supports variable interpolation.

```
<?php
$name = "John";
$heredocString = <<<EOD
This is a multiline string.
It can span multiple lines.
Hello, $name!
EOD;
echo $heredocString;
// Outputs:
// This is a multiline string.
// It can span multiple lines.
// Hello, John!
?>
```

Nowdoc Syntax

Nowdoc syntax is similar to Heredoc but does not parse variables. It is useful when you want to output a large block of text without any variable interpolation.

```
<?php
$nowdocString = <<<'EOD'
This is a multiline string.
It can span multiple lines.
Hello, $name!
EOD;
echo $nowdocString;
// Outputs:
// This is a multiline string.
```

```
// It can span multiple lines.
// Hello, $name!
?>
```

Multiline Strings Using Double Quotes

While not as clean as Heredoc or Nowdoc, you can also create multiline strings using double quotes and line breaks.

```
<?php
$multilineString = "This is a multiline string.
It can span multiple lines.
Hello, John!";
echo $multilineString;
// Outputs:
// This is a multiline string.
// It can span multiple lines.
// Hello, John!
?>
```

Example with HTML

Multiline strings are often used to generate dynamic HTML content.

```
<?php
$title = "Welcome";
$content = "This is a dynamic webpage.";
$html = <<<HTML
<!DOCTYPE html>
<html>
<head>
   <title>$title</title>
</head>
<body>
   <h1>$title</h1>
   <p>$content</p>
</body>
</html>
HTML;
echo $html;
// Outputs the complete HTML document with the dynamic content
?>
```

These methods allow you to handle multiline strings efficiently and are useful for embedding large blocks of text or code in your PHP scripts.

String Interpolation

String interpolation in PHP allows you to include variable values directly within a string. This is primarily done using double-quoted strings and Heredoc syntax. Here are examples of how to use

string interpolation in PHP:

Using Double-Quoted Strings

When you use double quotes, PHP will parse the string and replace any variables with their values.

```
<?php
$name = "John";
$age = 25;
// Simple interpolation
$greeting = "Hello, $name!";
echo $greeting; // Outputs: Hello, John!
// More complex interpolation
$info = "My name is $name and I am $age years old.";
echo $info; // Outputs: My name is John and I am 25 years old.
// Interpolating array values
$user = ['name' => 'Alice', 'age' => 30];
echo "User's name is {$user['name']} and age is {$user['age']}.";
// Outputs: User's name is Alice and age is 30.
?>
```

Using Curly Braces for Clarity

When embedding complex variables or arrays, it’s often clearer to use curly braces to delimit the variable names.

```
<?php
$name = "John";
$age = 25;
// Using curly braces for complex expressions
$greeting = "Hello, {$name}!";
$info = "My name is {$name} and I am {$age} years old.";
echo $greeting; // Outputs: Hello, John!
echo $info; // Outputs: My name is John and I am 25 years old.
?>
```

Using Heredoc Syntax

Heredoc syntax allows for multiline string interpolation and is particularly useful for embedding large blocks of text or code.

```
<?php
$name = "John";
$age = 25;
$heredocString = <<<EOD
Hello, $name!
You are $age years old.
This is a multiline string using Heredoc syntax.
EOD;
echo $heredocString;
```

```
// Outputs:
// Hello, John!
// You are 25 years old.
// This is a multiline string using Heredoc syntax.
?>
```

Interpolation with Object Properties

If you have an object, you can also interpolate its properties within strings.

```
<?php
class User {
   public $name = "John";
   public $age = 25;
}
$user = new User();
echo "User's name is $user->name and age is $user->age.";
// Outputs: User's name is John and age is 25.
?>
```

Interpolating Results of Function Calls

You can also interpolate the results of function calls directly in a string.

```
<?php
function getName() {
   return "John";
}
echo "Hello, " . getName() . "!"; // Outputs: Hello, John!
echo "Hello, {getName()}!"; // Will not work as PHP does not support
function calls inside string interpolation.
?>
```

In summary, string interpolation in PHP using double-quoted strings and Heredoc syntax is a powerful way to embed variable values directly within strings, making your code cleaner and more readable.

Formatting Strings

In PHP, you can format strings in various ways to make them more readable and structured. Below are some common methods and functions used for string formatting:

printf and sprintf

These functions are used to format strings using placeholders.

printf - Output a formatted string

```
<?php
$name = "John";
```

```
$age = 25;
// Using printf
printf("My name is %s and I am %d years old.", $name, $age);
// Outputs: My name is John and I am 25 years old.
?>
```

sprintf - Return a formatted string

```
<?php
$name = "John";
$age = 25;
// Using sprintf
$formattedString = sprintf("My name is %s and I am %d years old.", $name,
$age);
echo $formattedString;
// Outputs: My name is John and I am 25 years old.
?>
```

number_format - Format Numbers

This function formats a number with grouped thousands.

```
<?php
$number = 1234567.89;
$formattedNumber = number_format($number, 2, '.', ',');
echo $formattedNumber; // Outputs: 1,234,567.89
?>
```

str_pad - Pad a String to a Certain Length

This function pads a string to a certain length with another string.

```
<?php
$input = "Alien";
$padded = str_pad($input, 10, "*", STR_PAD_BOTH);
echo $padded; // Outputs: ***Alien***
?>
```

str_repeat - Repeat a String

This function repeats a string a specified number of times.

```
<?php
$input = "Hello";
$repeated = str_repeat($input, 3);
echo $repeated; // Outputs: HelloHelloHello
?>
```

strtoupper and strtolower - Change Case

These functions convert strings to uppercase or lowercase.

```
<?php
$input = "Hello, World!";
echo strtoupper($input); // Outputs: HELLO, WORLD!
echo strtolower($input); // Outputs: hello, world!
?>
```

ucfirst and lcfirst - Capitalize or Lowercase the First Character

These functions modify the first character of a string.

```
<?php
$input = "hello, World!";
echo ucfirst($input); // Outputs: Hello, World!
echo lcfirst("HELLO, WORLD!"); // Outputs: hELLO, WORLD!
?>
```

ucwords - Capitalize the First Character of Each Word

This function capitalizes the first character of each word in a string.

```
<?php
$input = "hello, world!";
echo ucwords($input); // Outputs: Hello, World!
?>
```

trim, ltrim, rtrim - Remove Whitespace

These functions remove whitespace from the beginning and/or end of a string.

```
<?php
$input = "  Hello, World!  ";
echo trim($input);  // Outputs: Hello, World!
echo ltrim($input); // Outputs: Hello, World!
echo rtrim($input); // Outputs:   Hello, World!
?>
```

Example with HTML Formatting

You can also format strings to generate HTML content dynamically.

```
<?php
$name = "John";
$age = 25;
$html = sprintf('<div class="user"><h2>Name: %s</h2><p>Age:
%d</p></div>', $name, $age);
```

```
echo $html;
// Outputs:
// <div class="user"><h2>Name: John</h2><p>Age: 25</p></div>
?>
```

Multiline String Formatting with heredoc

Using heredoc syntax for more complex formatting of multiline strings.

```
<?php
$name = "John";
$age = 25;
$text = <<<EOD
Name: $name
Age: $age
This is a formatted
multiline string.
EOD;
echo nl2br($text);
// Outputs:
// Name: John<br />
// Age: 25<br />
// This is a formatted<br />
// multiline string.
?>
```

These examples cover a variety of ways to format strings in PHP, helping you create more readable and structured output.

Character Encoding

Character encoding in PHP is an important aspect to ensure that your application correctly handles text in different languages and formats. PHP provides several functions and extensions for working with different character encodings, with UTF-8 being the most common encoding used for web applications. Here are some key concepts and functions related to character encoding in PHP:

Setting the Default Character Encoding

To ensure that your PHP script uses a specific character encoding, you can set the default character encoding using the default_charset directive in your php.ini file or at runtime:

In php.ini:

```
default_charset = "UTF-8"
```

At runtime:

```
<?php
ini_set('default_charset', 'UTF-8');
```

```
?>
```

Character Encoding Functions

mbstring Extension

The mbstring extension provides multibyte-specific string functions that help to properly handle character encoding, especially for UTF-8.

Check if a String is UTF-8 Encoded:

```
<?php
$string = "Hello, □□!";
if (mb_detect_encoding($string, 'UTF-8', true) === 'UTF-8') {
   echo "The string is UTF-8 encoded.";
} else {
   echo "The string is not UTF-8 encoded.";
}
?>
```

Convert Character Encoding:

```
<?php
$string = "Hello, World!";
$encodedString = mb_convert_encoding($string, 'UTF-8', 'ISO-8859-1');
echo $encodedString;
?>
```

Get the Internal Character Encoding:

```
<?php
echo mb_internal_encoding(); // Outputs the current internal character
encoding
?>
```

Set the Internal Character Encoding:

```
<?php
mb_internal_encoding('UTF-8');
?>
```

Get and Set HTTP Output Character Encoding:

```
<?php
// Get the current HTTP output character encoding
echo mb_http_output();
// Set the HTTP output character encoding
mb_http_output('UTF-8');
```

```
?>
```

Multibyte String Length:

```
<?php
$string = "Hello, □□!";
echo mb_strlen($string, 'UTF-8'); // Outputs the length of the string
?>
```

Substring for Multibyte Strings:

```
<?php
$string = "Hello, □□!";
echo mb_substr($string, 7, 3, 'UTF-8'); // Outputs "□□!"
?>
```

Detect Encoding of a String:

```
<?php
$string = "Hello, □□!";
echo mb_detect_encoding($string); // Outputs the detected encoding (e.g.,
UTF-8)
?>
```

Handling Character Encoding in HTML

To ensure that your HTML documents are correctly interpreted with the intended character encoding, include the following meta tag in the head section of your HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Character Encoding Example</title>
</head>
<body>
   <p>Hello, □□!</p>
</body>
</html>
```

Using iconv Functions

The iconv extension provides functions for converting between character sets.

Convert Character Encoding:

```
<?php
```

```
$string = "Hello, World!";
$encodedString = iconv('ISO-8859-1', 'UTF-8', $string);
echo $encodedString;
?>
```

Check Character Encoding:

```
<?php
$string = "Hello, World!";
if (iconv('ISO-8859-1', 'UTF-8', $string) !== false) {
   echo "Conversion successful.";
} else {
   echo "Conversion failed.";
}
?>
```

Example of Ensuring UTF-8 Encoding

Here's a complete example demonstrating how to ensure all inputs and outputs are in UTF-8:

```
<?php
// Set internal encoding to UTF-8
mb_internal_encoding('UTF-8');
mb_http_output('UTF-8');
// Convert incoming data to UTF-8
function convertToUtf8($data) {
   if (is_array($data)) {
       return array_map('convertToUtf8', $data);
   } elseif (is_string($data)) {
       return mb_convert_encoding($data, 'UTF-8', 'auto');
   } else {
       return $data;
   }
}
// Example usage
$input = "Hello, □□!";
$utf8Input = convertToUtf8($input);
echo $utf8Input;
// Ensure HTTP output is UTF-8
header('Content-Type: text/html; charset=UTF-8');
?>
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>UTF-8 Example</title>
</head>
<body>
   <p><?php echo $utf8Input; ?></p>
</body>
```

```
</html>
```

By using these methods and best practices, you can ensure that your PHP application correctly handles different character encodings, especially UTF-8.