

Functions

- Functions are blocks of code that perform specific tasks and can be reused throughout the program. Functions help with better code organization, reuse, and maintainability.

Advantages of Functions:

1. **Code Reusability:** Functions allow the reuse of code blocks without rewriting them, making development faster and more efficient.
2. **Modular Structure:** Functions help in breaking down complex problems into smaller, more manageable pieces, enhancing code readability.
3. **Ease of Maintenance:** Changes can be made in a single place (the function), and all the parts of the application that use the function will automatically benefit from the changes.
4. **Improved Debugging:** Functions isolate different tasks, making it easier to identify and fix bugs.

Key Points:

- **Static Properties and Methods:** Declared using the `static` keyword and are shared among all instances of the class.
- **Inheritance of Static Properties:** A child class can access static properties and methods from the parent class using `parent::`.
- **Accessing Static Properties:** You do not need to instantiate an object to access static properties or methods; they are accessed using the `::` operator.
- **Method Overloading:** Achieved using `__call()` for instance methods and `__callStatic()` for static methods to handle calls to undefined methods dynamically.
- **Encapsulation:** Control visibility using `private`, `protected`, and `public` access modifiers.
- **Polymorphism:** Implemented by allowing a child class to override a parent class method.
- **Abstraction:** Achieved using abstract classes and interfaces to provide a simplified interface to complex logic.
- **Functions:** Provide code reusability, modular structure, easier maintenance, and improved debugging.

Built-in Functions: These are the standard functions that are always available in PHP without any special requirement like.

strlen(): To get the length of a string.

array_merge(): To merge two or more arrays.

is_numeric(): To check if a variable is a number or a numeric string.

Build In Functions List

<https://www.php.net/manual/en/indexes.functions.php>

```
$string = "Hello,World!";
$length = strlen($string);
echo $length;

$value="12Z";
if (is_numeric($value)) {
    echo "The value '$value' is numeric";
}
else {
    echo "The value '$value' is NOT numeric";
}
```

User-defined Functions: These are functions created by the programmer. They're defined using the function keyword.

```
function sum() {
    $num1=20;
    $num2=30;
    echo $num1+$num2;
}
sum();
```

Function With Params

```
function sum($num1, $num2) {
    echo $num1+$num2;
}
sum(1,2);
```

Function With Params Default Value

One without a default value must be declared before the one with a default value.

```
function sum($num1, $num2=10) {
    echo $num1+$num2;
}
sum(2);
```

Type Hinting

- **Primitive types:** like int, float, bool, string.
- **Classes and interfaces:** Any class or interface name.
- **Arrays:** with the array keyword.
- **Callable:** for functions or objects that implement the invoke method.
- **Nullable types:** by prefixing with a ?.
- **Iterables:** with the iterable keyword, which includes both arrays and objects implementing the Traversable interface.
- **Union types:** introduced in PHP 8, where a parameter can accept multiple types.

```
function sum(int $num1,int $num2) {
    echo $num1+$num2;
}
sum(20,20);
```

Multiple type hinting in one parameter

```
function sum(int|string $num1,int|string $num2) {
    echo $num1+$num2;
}
```

```
sum("33","20");
```

Nullable type hints

```
function myName(?string $text): void {
    if ($text) {
        echo "Text is: $text<br>";
    } else {
        echo "No text provided.<br>";
    }
}

myName("Jack");
myName(null);
```

Variadic Functions

- Functions that accept an indefinite number of arguments.
- They use the (...) spread operator

```
function sum(...$numbers) {
    echo array_sum($numbers);
}
sum(1,2,3,4,5);

function sum(...$numbers) {
    echo $numbers[0];
}
sum(1,2,"3",4,5);
```

Anonymous Functions (or Closures)

These are functions without a name. They can be stored in variables or passed as arguments to other functions.

Creation & Execution Directly

```
(function () {
    echo "Hello from the anonymous function!";
})();
```

Assign to a Variable and Execute

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

$greet("OSTAD");
```

Arrow functions provide a more concise syntax for writing anonymous functions. are designed for simple, single-expression use-cases.

```
$sum = fn($num1,$num2) =>$num1 * $num2;
echo $sum(5,5);
```

Recursive Functions

Functions that call themselves to solve a problem.

```
function factorial($n) {
    if ($n == 0) {
        return 1;
    } else {
        return $n * factorial($n-1);
    }
}

echo factorial(5);
```

Callback Function

Function that is passed as an argument to another function and is executed after the completion of that function. This pattern is quite common in programming, especially in scenarios like event handling, asynchronous operations, or functions that require custom logic.

```
function sum($a, $b) {
    echo $a + $b;
}

function calculate($num1, $num2, $callback) {
    return $callback($num1, $num2);
}

$result = calculate(5, 3, 'sum');
```

Basic Return Types

```
function getAge(): int {
    return 25;
}

function isAdult(int $age): bool {
    return $age >= 18;
}

echo getAge();
echo isAdult(11)
```

Nullable Return Type

```
function findUsername(int $id): ?string {
    if ($id === 1) {
        return "JohnDoe";
    }

    return null; // It's valid because of the ? before string
}

echo findUsername(2);
```

Return Type of void

```
function logMessage(string $message): void {
    echo $message;
}

logMessage("Hello");
```

Union Return Types

```
function sum($num1,$num2):int|string{
    return $num1+$num2;
}

echo sum("33","20");
```

Strict mode in PHP

- Affects how type hints are enforced.
- By default, PHP will try to coerce values of the wrong type to match the expected type.
- In strict mode, PHP will throw a `TypeError` if the provided value does not exactly match the expected type.

```
declare(strict_types=1);

function add(int $a, int $b): int {
    return $a + $b;
}

echo add("5", "10");
```

#php