PHP – Traits

In PHP, a class can inherit only from one parent class, multiple inheritance is not defined in PHP. Traits in PHP have been introduced to overcome this limitation. You can define one or more method in a trait, which can be reused freely in various independent classes.

Syntax

The "trait" keyword is used as per the following syntax –

```
trait mytrait {
   function method1() {
      /*function body*/
   }

   function method2() {
      /*function body*/
   }
}
```

To be able to call the methods in a trait, it needs to made available to another class with use keyword.

Example

A Trait is similar to a class, but only intended to group functionality in a fine-grained and consistent way. It is not possible to instantiate a Trait on its own.

Open Compiler

```
<?php
   trait mytrait {
      public function hello() {
         echo "Hello World from " . __TRAIT__ . "";
      }
   }
   class myclass {
      use mytrait;
   }
   $obj = new myclass();
   $obj->hello();
?>
```

It will produce the following **output** –

```
Hello World from mytrait
```

Explore our latest online courses and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Example

A trait can be used in more than one classes. The following example has a mytrait with avg() function int it. It is used inside a marks class. The percent() method internally calls the avg() function from the trait.

Take a look at the following example –

Open Compiler

```
<?php
   trait mytrait {
      function avg($x, $y) {
         return ($x+$y)/2;
      }
   }
   class marks {
      use mytrait;
      private int $m1, $m2;
      function __construct($x, $y) {
         $this->m1 = $x;
         $this->m2 = $y;
      }
      function percent():float {
         return $this->avg($this->m1, $this->m2);
      }
   }
   $obj = new marks(50, 60);
   echo "percentage: " . $obj->percent();
?>
```

It will produce the following **output** –

```
percentage: 55
```

Using Multiple Traits

A class can use more than one traits. Here we have two traits with one function each performing addition and multiplication of two numbers. Both are used inside a third class.

</> Open Compiler

```
<?php
   trait addition {
      function add($x, $y) {
         return $x+$y;
      }
   }

   trait multiplication {
      function multiply($x, $y) {
         return $x*$y;
      }
   }

   class numbers {
      use addition, multiplication;
      private int $m1, $m2;
      function __construct($x, $y) {
         $this->m1 = $x;
         $this->m2 = $y;
      }
      function calculate():array {
         $arr = [$this->add($this->m1, $this->m2), $this->multiply($this->m1, $thi
         return $arr;
      }
   }

   $obj = new numbers(50, 60);
```

```
    $res = $obj->calculate();
    echo "Addition: " . $res[0] . PHP_EOL;
    echo "Multiplication: " . $res[1] . PHP_EOL;
?>
```

It will produce the following **output** −

```
Addition: 110
Multiplication: 3000
```

Overriding Trait Function

When a class uses a certain trait, its function are available to it just as a child class inherits the parent methods. The trait function may also be overridden.

Open Compiler

```
<?php
   trait mytrait {
      public function sayHello() {
         echo 'Hello World!';
      }
   }

   class myclass {
      use mytrait;
      public function sayHello() {
         echo 'Hello PHP!';
      }
   }

   $o = new myclass();
   $o->sayHello();
?>
```

It will produce the following **output** −

```
Hello PHP!
```

The "insteadof" Keyword

Sometimes, more two traits might have same name of the function. Hence, using them in a class creates ambiguous situation. PHP provides insteadof keyword to tell the parser function from which trait you intend to use.

</> Open Compiler

```
<?php
   trait mytrait {
      public function sayHello() {
         echo 'Hello World!';
      }
   }

   trait newtrait {
      public function sayHello() {
         echo 'Hello PHP!';
      }
   }

   class myclass {
      use mytrait, newtrait{
         newtrait::sayHello insteadof mytrait;
      }
   }

   $o = new myclass();
   $o->sayHello();
?>
```

It will produce the following **output** –

```
Hello PHP!
```

Aliasing a Trait Function

If you want to be able to call functions from both traits even if they have function with same name, a workaround is to specify an alias name to one of them.

Example

In the following example, we will call sayHello() from mytrait as hello() −

Open Compiler

```php
<?php
   trait mytrait {
      public function sayHello() {
         echo 'Hello World!' . PHP_EOL;
      }
   }

   trait newtrait {
      public function sayHello() {
         echo 'Hello PHP!' . PHP_EOL;
      }
   }

   class myclass {
      use mytrait, newtrait{
         mytrait::sayHello as hello;
         newtrait::sayHello insteadof mytrait;
      }
   }

   $o = new myclass();
   $o->hello();
   $o->sayHello();
?>
```

It will produce the following **output** −

```
Hello World!
Hello PHP!
```