

# PHP - Scalar Type Declarations

The feature of providing type hints has been in PHP since version 5. **Type hinting** refers to the practice of providing the data type of a parameter in the function definition. Before PHP 7, it was possible to use only the array, callable, and class for type hints in a function. PHP 7 onwards, you can also insert type hints for parameters of scalar data type such as int, string, bool, etc.

PHP is a dynamically (and weakly) typed language. Hence, you don't need to declare the type of the parameter when a function is defined, something which is necessary in a statically type language like C or Java.

A typical definition of function in PHP is as follows –

```
function addition($x, $y) {  
    echo "First number: $x Second number: $y Addition: " . $x+$y;  
}
```

Here, we assume that the parameters \$x and \$y are numeric. However, even if the values passed to the function aren't numeric, the PHP parser tries to cast the variables into compatible type as far as possible.

If one of the values passed is a string representation of a number, and the second is a numeric variable, PHP casts the string variable to numeric in order to perform the addition operation.

## Example

Take a look at this following example –

[Open Compiler](#)

```
<?php  
function addition($x, $y) {  
    echo "First number: " . $x;  
    echo "\nSecond number: " . $y;  
    echo "\nAddition: " . $x+$y;  
}  
  
$x="10";  
$y=20;
```

```
    addition($x, $y);  
    ?>
```

It will produce the following **output** –

```
First number: 10  
Second number: 20  
Addition: 30
```

However, if `$x` in the above example is a string that doesn't hold a valid numeric representation, an error is encountered.

</>

Open Compiler

```
<?php  
function addition($x, $y) {  
    echo "First number: " . $x;  
    echo "\nSecond number: " . $y;  
    echo "\nAddition: " . $x+$y;  
}  
  
$x="Hello";  
$y=20;  
addition($x, $y);  
?>
```

Run this code and see how it shows an **error**.

## Scalar Type Declarations in PHP 7

A new feature introduced with PHP version 7 allows defining a function with parameters whose data type can be specified within the parenthesis.

PHP 7 has introduced the following Scalar type declarations –

- Int
- Float
- Bool
- String
- Interfaces

- Array
- Callable

Older versions of PHP allowed only the array, callable and class types to be used as type hints. Furthermore, in the older versions of PHP (PHP 5), the fatal error used to be a recoverable error while the new release (PHP 7) returns a throwable error.

Scalar type declaration is implemented in two modes –

- **Coercive Mode** – Coercive is the default mode and need not to be specified.
- **Strict Mode** – Strict mode has to be explicitly hinted.

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

## Coercive Mode

The addition() function defined in the earlier example can now be re-written by incorporating the type declarations as follows –

```
function addition(int $x, int $y) {  
    echo "First number: $x Second number: $y Addition: " . $x+$y;  
}
```

Note that the parser still casts the incompatible types i.e., string to an int if the string contains an integer as earlier.

## Example

Take a look at this following example –

[Open Compiler](#)

```
<?php  
function addition(int $x, int $y) {  
    echo "First number: " . $x;  
    echo "\nSecond number: " . $y;  
    echo "\nAddition: " . $x+$y;  
}
```

```
$x="10";  
$y=20;  
echo addition($x, $y);  
?>
```

It will produce the following **output** –

```
First number: 10  
Second number: 20  
Addition: 30
```

Obviously, this is because PHP is a weakly typed language, as PHP tries to coerce a variable of string type to an integer. PHP 7 has introduced a strict mode feature that addresses this issue.

## Strict Mode

To counter the weak type checking of PHP, a strict mode has been introduced. This mode is enabled with a **declare statement** –

```
declare (strict_types=1);
```

You should put this statement at the top of the PHP script (usually just below the PHP tag). This means that the strictness of typing for scalars is configured on a per-file basis.

In the weak mode, the strict\_types flag is 0. Setting it to 1 forces the PHP parser to check the compatibility of the parameters and values passed. Add this statement in the above code and check the result. It will show the following error message –

```
Fatal error: Uncaught TypeError: addition():  
Argument #1 ($x) must be of type int, string given,  
called in add.php on line 12 and defined in add.php:4  
  
Stack trace:  
#0 add.php(12): addition('10', 20)  
#1 {main}  
    thrown in add.php on line 4
```

## Example

Here is another example of scalar type declaration in the function definition. The strict mode when enabled raises fatal error if the incompatible types are passed as parameters.



Open Compiler

&lt;?php

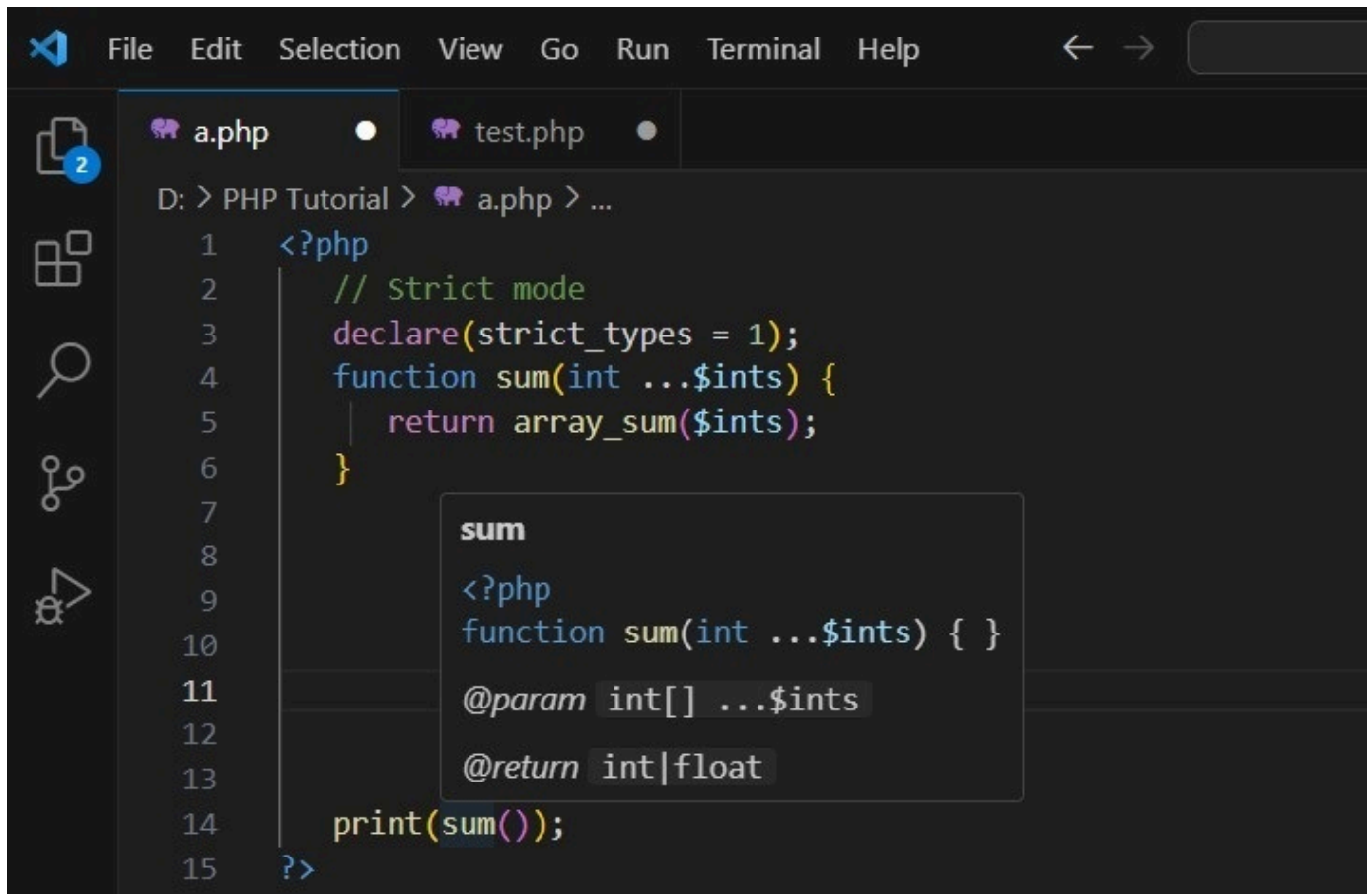
```
// Strict mode
// declare(strict_types = 1);
function sum(int ...$ints) {
    return array_sum($ints);
}

print(sum(2, '3', 4.1));
?>
```

Uncomment the **declare** statement at the top of this code and run it. Now it will produce an **error** –

```
Fatal error: Uncaught TypeError:
sum(): Argument #2 must be of type int, string given,
called in add.php on line 9 and defined in add.php:4
Stack trace:
#0 add.php(9): sum(2, '3', 4.1)
#1 {main}
  thrown in add.php on line 4
```

The type-hinting feature is mostly used by IDEs to prompt the user about the expected types of the parameters used in the function declaration. The following screenshot shows the VS Code editor popping up the function prototype as you type.



The screenshot shows a code editor with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar has icons for Explorer, Search, and Run and Debug. The main editor area shows a file named 'a.php' with the following code:

```
1 <?php
2     // Strict mode
3     declare(strict_types = 1);
4     function sum(int ...$ints) {
5         return array_sum($ints);
6     }
7
8
9
10
11
12
13
14     print(sum());
15 ?>
```

A tooltip is displayed over the function signature on line 4, showing the function name 'sum', the opening PHP tag '<?php', the function definition 'function sum(int ...\$ints) { }', and the parameter and return type declarations '@param int[] ...\$ints' and '@return int|float'.