# PHP - Strict Typing

PHP is widely regarded as a weakly typed language. In PHP, you need not declare the type of a variable before assigning it any value. The PHP parser tries to cast the variables into compatible type as far as possible.

For example, 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 the following example –

</> Open Compiler

```
<?php
   function addition($x, $y) {
      echo "First number: $x Second number: $y Addition: " . $x+$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, then you will encounter an error.

</> Open Compiler

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

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

It will produce the following **output** –

```
PHP Fatal error:  Uncaught TypeError: Unsupported operand
types: string + int in hello.php:5
```

## Type Hints

Type-hinting is supported from PHP 5.6 version onwards. It means you can explicitly state the expected type of a variable declared in your code. PHP allows you to type-hint function arguments, return values, and class properties. With this, it is possible to write more robust code.

Let us incorporate type-hinting in the addition function in the above program –

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

Note that by merely using the data types in the variable declarations doesn't prevent the unmatched type exception raised, as PHP is a dynamically typed language. In other words, $x="10" and $y=20 will still result in the addition as 30, whereas $x="Hello" makes the parser raise the error.

## Example

</> Open Compiler

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

   $x=10;
   $y=20;
```

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

    $x="10";
    $y=20;
    addition($x, $y);

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

It will produce the following **output** –

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

First number: 10
Second number: 20
Addition: 30

First number: Hello
Second number: 20
PHP Fatal error:  Uncaught TypeError: Unsupported operand
types: string + int in hello.php:5
```



## strict_types

PHP can be made to impose stricter rules for type conversion, so that "10" is not implicitly converted to 10. This can be enforced by setting **strict_types** directive to 1 in a declare() statement.

The declare() statement must be the first statement in the PHP code, just after the "<? php" tag.

## Example

Take a look at the following example –

</> Open Compiler

```
<?php
   declare (strict_types=1);
   function addition(int $x, int $y) {
      echo "First number: $x Second number: $y Addition: " . $x+$y;
   }

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

It will produce the following **output** –

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

Now, if **$x** is set to "10", the implicit conversion won't take place, resulting in the following **error** –

```
PHP Fatal error:  Uncaught TypeError: addition(): Argument #1
($x) must be of type int, string given
```

From PHP 7 onwards, type-hinting support has been extended for function returns to prevent unexpected return values. You can type-hint the return values by adding the intended type after the parameter list prefixed with a colon (:) symbol.

## Example

Let us add a type hint to the return value of the division() function below.

</> Open Compiler

```
<?php
   declare (strict_types=1);
   function division(int $x, int $y) : int {
      return $x/$y;
   }

   $x=10;
```

```
    $y=20;
    $result = division($x, $y);
    echo "First number: $x Second number: $y Addition: " . $result;
?>
```

Because the function returns 0.5, which is not of **int** type (that is, the type hint used for the return value of the function), the following **error** is displayed –

```
Fatal error: Uncaught TypeError: division(): Return value must be
of type int, float returned in hello.php:5
```