PHP - Type Casting

The term "Type Casting" refers to conversion of one type of data to another. Since PHP is a weakly typed language, the parser coerces certain data types into others while performing certain operations. For example, a string having digits is converted to integer if it is one of the operands involved in the addition operation.

Implicit Type Casting

Here is an example of coercive or implicit type casting −

```
<?php
   $a = 10;
   $b = '20';
   $c = $a+$b;
   echo "c = " . $c;
?>
```

In this case, $b is a string variable, cast into an integer to enable addition. It will produce the following **output** −

```
c = 30
```

Let's take another example. Here, an integer variable $a is converted to a string so that it is concatenated with a string variable.

```
<?php
   $a = 10;
   $b = '20';
   $c = $a.$b;
   echo "c = " . $c;
?>
```

It will produce the following **output** −

```
c = 1020
```

In addition to such coercive type conversion, there are other ways to explicitly cast one type of data to other. You can use PHP's type casting operators or type casting functions for this purpose.

Type Casting Operators

To convert an expression of one type to another, you need to put the data type of the latter in parenthesis before the expression.

```
$var = (type)expr;
```

Some of the type casting operators in PHP are –

- (int) or (integer) casts to an integer
- (bool) or (boolean) casts to a boolean
- (float) or (double) or (real) casts to a float
- (string) casts to a string
- (array) casts to an array
- (object) casts to an object

Casting to Integer

You can easily convert a float value to an integer. Take a look at the following **example** –

```
<?php
   $a = 9.99;
   $b = (int)$a;
   var_dump($b);
?>
```

It will produce the following **output** –

```
int(9)
```

Note that the float value is not rounded to the nearest integer; instead it just returns the integer part.

String to Integer Conversion

The (int) operator also coverts a string to integer. The conversion is straightforward if the string consists of just digits.

```
<?php
   $a = "99";
   $b = (int)$a;
   var_dump($b);
?>
```

Here, you will get the following **output** –

```
int(99)
```

Even if the string contains a floating point number, the (int) operator returns just the integer part.

Now let's take another example to understand a special case. **If the string is alphanumeric, casting with (int) works differently**.

- If the string starts with digits followed by non-numeric characters, only the initial digits are considered.
- If the string starts with non-numeric characters and the digits are in the middle, the csting operator returns "0".

Take a look at the following **example** –

```
<?php
   $a = "10 Rs.";
   $b = (int)$a;
   var_dump($b);

   $a = "$100";
   $b = (int)$a;
   var_dump($b);
?>
```

It will produce the following **output** –

```
int(10)
int(0)
```

Casting to Float Type

You can use either the (float) or (double) casting operator to explicitly convert a variable or expression to a float.

```php
<?php
   $a = 100;
   $b = (double)$a;
   var_dump($b);
?>
```

It will produce the following **output** –

```
float(100)
```

A string containing any valid numeric representation may be cast to a float type by using a casting operator.

```php
<?php
   $a = "100";
   $b = (double)$a;
   var_dump($b);

   $a = "9.99";
   $b = (float)$a;
   var_dump($b);
?>
```

Here, you will get the following **output** –

```
float(100)
float(9.99)
```

The string gets converted to float even when it embeds a scientific notation of float. Take a look at the following example –

```php
<?php
   $a = "1.23E01";
   $b = (double)$a;
   var_dump($b);
   $a = "5.5E-5";
   $b = (float)$a;
   var_dump($b);
?>
```

It will produce the following **output** –

```
float(12.3)
float(5.5E-5)
```

All the non-numeric characters after the floating point numbers are ignored. Similarly, the string converts to "0" if it starts with any non-numeric character. See the following **example** –

```php
<?php
   $a = "295.95 only";
   $b = (double)$a;
   var_dump($b);

   $a = "$2.50";
   $b = (float)$a;
   var_dump($b);
?>
```

It will produce the following **output** –

```
float(295.95)
float(0)
```

Casting to String Type

By using a casting operator, any expression evaluating to a floating-point or integer may be cast to a string type. Few examples are given below –

```php
<?php
   $a = 100;
```

```
   $b = (string)$a;
   var_dump($b);

   $x = 55.50;
   $y = (string)$x;
   var_dump($y);
?>
```

You will get the following **output** –

```
string(3) "100"
string(4) "55.5"
```

Casting to Bool Type

Any non-zero number, either integer or float, is cast to true with (bool) operator. An expression evaluating to "0" returns false. A string is always cast to true.

Take a look at the following **example** –

```
<?php
   $a = 100;
   $b = (bool)$a;

   $x = 0;
   $y = (bool)$x;

   $m = "Hello";
   $n = (bool)$m;

   var_dump($b);
   var_dump($y);
   var_dump($n);
?>
```

It will produce the following **output** –

```
bool(true)
bool(false)
bool(true)
```

Type Casting Functions

PHP includes the following built-in functions for performing type casting –

- intval()
- floatval()
- strval()

Let's discuss these built-in functions in detail.

The intval() Function

This function gets the integer value of a variable.

```
intval(mixed $value, int $base = 10): int
```

The **$base** parameter is 10 by default, which means the value is converted to decimal number.

- If the value is a float, the intval() function returns an integer, discarding the fractional part.
- A string representation of a number returns a corresponding integer, discarding the fractional part, if any.
- If the value is a string with a valid octal number and the base is 8, the intval() function returns a corresponding octal number.

When the base is "0", the conversion of value takes place on the basis of character prefix.

- If the value starts with 0X or 0x, a hexadecimal number is returned.
- If the value starts with 0B or 0b, a binary number is returned
- If the value starts with 0, the function returns an octal number.

The intval() function returns 1 for true, 0 for false Boolean values.

Example

The following example shows how the intval() function works –

```php
<?php
   echo intval(42). PHP_EOL;
   echo intval(4.2). PHP_EOL;
   echo intval('42') . PHP_EOL;

   echo intval(042) . PHP_EOL;          # 0ctal number
   echo intval('042', 0) . PHP_EOL;     # 0ctal number
   echo intval('42', 8) . PHP_EOL;      # octal

   echo intval(0x1A) . PHP_EOL;         # Hexadecimal
   echo intval('0x1A', 16) . PHP_EOL;   # Hexadecimal
   echo intval('0x1A', 0) . PHP_EOL;    # Hexadecimal

   echo intval(false) . PHP_EOL;
   echo intval(true) . PHP_EOL;
?>
```

It will produce the following **output** –

```
42
4
42
34
34
34
26
26
26
0
1
```

The floatval() Function

The floatval() function gets the float value of an expression.

```php
floatval(mixed $value): float
```

The value may be any scalar variable. String with non-numeric characters returns "0". A string with a numeric representation or with the starting substring with a numeric representation returns the corresponding number. The following **example** shows how the floatval() function works –

```php
<?php
   echo floatval(42). PHP_EOL;
   echo floatval(4.2). PHP_EOL;
   echo floatval('42') . PHP_EOL;

   echo floatval('99.90 Rs') . PHP_EOL;
   echo floatval('$100.50') . PHP_EOL;
   echo floatval('ABC123!@#') . PHP_EOL;

   echo (true) . PHP_EOL; ;
   echo (false) . PHP_EOL;
?>
```

It will produce the following **output** –

```
42
4.2
42
99.9
0
0
1
```

The **doubleval()** function is an alias of floatval() function, and hence returns similar results.

The strval() Function

The strval() function gets the string value of a variable. This function performs no formatting on the returned value.

```php
strval(mixed $value): string
```

The value that is being converted to a string may be any scalar type, null, or an object that implements the __toString() method. Take a look at the following **example** –

```php
<?php
   echo strval(42). PHP_EOL;
   echo strval(4.2). PHP_EOL;
   echo strval(4.2E5) . PHP_EOL;
```

```
   echo strval(NULL) . PHP_EOL;

   echo (true) . PHP_EOL;
   echo (false) . PHP_EOL;
?>
```

It will produce the following **output** –

```
42
4.2
420000

1
```

The following **example** defines a class that implements the **_toString() method**.

```
<?php
   class myclass {
      public function __toString() {
         return __CLASS__;
      }
   }
   echo strval(new myclass);
?>
```

Here, you will get the following **output** –

```
myclass
```