PHP - Function Parameters

A function in PHP may be defined to accept one or more parameters. Function parameters are a comma-separated list of expressions inside the parenthesis in front of the function name while defining a function. A parameter may be of any scalar type (number, string or Boolean), an array, an object, or even another function.

```
function foo($arg_1, $arg_2, $arg_n) {
   statements;
   return $retval;
}
```

The arguments act as the variables to be processed inside the function body. Hence, they follow the same naming conventions as any normal variable, i.e., they should start with "$" and can contain alphabets, digits and underscore.

Note – There is no restriction on how many parameters can be defined.

When a parameterized function needs to be called, you have to make sure that the same number of values as in the number of arguments in function's definition, are passed to it.

```
foo(val1, val2, val_n);
```

A function defined with parameters can produce a result that changes dynamically depending on the passed values.

Example

The following code contains the definition of addition() function with two parameters, and displays the addition of the two. The run-time output depends on the two values passed to the function.

</> Open Compiler

```
<?php
   function addition($first, $second) {
      $result = $first+$second;
      echo "First number: $first \n";
      echo "Second number: $second \n";
      echo "Addition: $result";
   }
```

```
   addition(10, 20);

   $x=100;
   $y=200;
   addition($x, $y);
?>
```

It will produce the following **output** –

```
First number: 10
Second number: 20
Addition: 30
First number: 100
Second number: 200
Addition: 300
```

Formal and Actual Arguments

Sometimes the term **argument** is used for **parameter**. Actually, the two terms have a certain difference.

- A parameter refers to the variable used in function's definition, whereas an argument refers to the value passed to the function while calling.
- An argument may be a literal, a variable or an expression
- The parameters in a function definition are also often called as **formal arguments**, and what is passed is called **actual arguments**.
- The names of formal arguments and actual arguments need not be same. The value of the actual argument is assigned to the corresponding formal argument, from left to right order.
- The number of formal arguments defined in the function and the number of actual arguments passed should be same.

Example

PHP raises an **ArgumentCountError** when the number of actual arguments is less than formal arguments. However, the additional actual arguments are ignored if they are more than the formal arguments.

</> Open Compiler

```php
<?php
   function addition($first, $second) {
      $result = $first+$second;
      echo "First number: $first \n";
      echo "Second number: $second \n";
      echo "Addition: $result \n";
   }

   # Actual arguments more than formal arguments
   addition(10, 20, 30);

   # Actual arguments fewer than formal arguments
   $x=10;
   $y=20;
   addition($x);
?>
```

It will produce the following **output** –

```
First number: 10
Second number: 20
Addition: 30
PHP Fatal error:  Uncaught ArgumentCountError: Too few arguments
to function addition(), 1 passed in /home/cg/root/20048/main.php
on line 16 and exactly 2 expected in /home/cg/root/20048/main.php:2
```


Arguments Type Mismatch

PHP is a dynamically typed language, hence it doesn't enforce type checking when copying the value of an actual argument with a formal argument. However, if any statement inside the function body tries to perform an operation specific to a particular data type which doesn't support it, PHP raises an exception.

In the addition() function above, it is assumed that numeric arguments are passed. PHP doesn't have any objection if string arguments are passed, but the statement performing the addition encounters exception because the "+" operation is not defined for string type.

Example

Take a look at the following example –

```
<?php
   function addition($first, $second) {
      $result = $first+$second;
      echo "First number: $first \n";
      echo "Second number: $second \n";
      echo "Addition: $result";
   }

   # Actual arguments are strings
   $x="Hello";
   $y="World";
   addition($x, $y);
?>
```

It will produce the following **output** –

```
PHP Fatal error:  Uncaught TypeError: Unsupported operand types: string + string in hell
```

However, PHP is a weakly typed language. It attempts to cast the variables into compatible type as far as possible. Hence, if one of the values passed is a string representation of a number and the second is a numeric variable, then PHP casts the string variable to numeric in order to perform the addition operation.

Example

Take a look at the following example –

```
<?php
   function addition($first, $second) {
      $result = $first+$second;
      echo "First number: $first \n";
      echo "Second number: $second \n";
```

```
      echo "Addition: $result";
   }

   # Actual arguments are strings
   $x="10";
   $y=20;
   addition($x, $y);
?>
```

It will produce the following **output** –

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