PHP FUNCTIONS: DEFINITION AND CALL

Functions are reusable code blocks that only execute when called. They allow the code to be divided into smaller parts that are easier to understand and reuse.

```
function myFunc()
{
echo 'Hello World';
}
```

Once defined, a function can be called (invoked) from anywhere on the page by typing its name followed by a set of parenthesis

```
myFunc(); // "Hello World"
```

PHP FUNCTIONS: PARAMETERS

The parentheses that follow the function name are used to pass arguments to the function. To do this, the corresponding parameters must first be specified in the function definition in the form of a comma-separated list of variables. The parameters can then be used in the function.

```
function myFunc($x, $y)
{
echo $x . $y;
}
```

With the parameters specified, the function can be called with the same number of arguments

```
myFunc('Hello', ' World'); // "Hello World"
```

PHP FUNCTIONS: RETURN STATEMENT

Return is a jump statement that causes the function to end its execution and return to the location where it was called from.

```
function myFunc()
{
return; // exit function
echo 'Hi'; // never executes
}
```

It can optionally be given a value to return, in which case it makes the function call evaluate to that value.

```
function myFunc()
{
// Exit function and return value
return 'Hello';
}
echo myFunc(); // "Hello"
```

A function without a return value automatically returns null.

PHP FUNCTIONS: CREATE A FUNCTION

```
function [&] name(parameters)
{
   // instructions

    return value; // optional: only if it returns a value
}
```

PHP FUNCTIONS: EXAMPLES

```
function showconcat($cad1, $cad2)
{
    echo $cad1.$cad2;
}

function getconcat($cad1, $cad2)
{
    $cad3=$cad1.$cad2;
    return $cad3;
}
```

To call:

```
showconcat('hello ','world.');
```

Or… it is the same

```
$cad=getconcat('hello ','world.');
echo $cad;
```

PHP FUNCTIONS: DEFAULT PARAMETERS

It is possible to specify default values for parameters by assigning them a value inside the parameter list.

```
function myFunc($x, $y = ' Earth')
{
echo $x . $y;
}
myFunc('Hello'); // "Hello Earth"
```

PHP FUNCTIONS: VARIABLE PARAMETER LISTS

A function cannot be called with fewer arguments than is specified in its declaration, but it may be called with more arguments. This allows for the passing of a variable number of arguments, which can then be accessed using three built-in functions: func_get_arg(), func_num_args(), func_get_args()

func_get_arg Returns an item from the argument list

```
function myArgs()
{

$x = func_get_arg(0);
$y = func_get_arg(1);
$z = func_get_arg(2);
echo $x . $y . $z;

}

myArgs('Fee', 'Fi', 'Fo'); // "FeeFiFo"
```

This works with 3 parameters maximum

PHP FUNCTIONS: VARIABLE PARAMETER LISTS

Example: This works without parameters limit (no maximum and no minimum parameters)

func_get_args Returns an array containing all those arguments
func_num_args Returns the number of arguments passed to the function

```
function fsum()
{
    if(func_num_args()==0) // no parameters
    {
        return false;
    }
    else
    {
        $tot=0;
        for($i=0;$i<func_num_args();$i++)
        {
            $tot=$tot+func_get_arg($i);
        }
        return $tot;
    }
}
```

Call the function with different number of values…. echo fsum(4,5,6);

PHP FUNCTIONS: VARIABLE PARAMETER LISTS

Example: Same function page 7 but without parameters limit (no maximum and no minimum parameters)

```
function myArgs2()
{
$num = func_num_args();
$args = func_get_args();
for ($i = 0; $i < $num; $i++)
echo $args[$i];
}
myArgs2('Fee', 'Fi', 'Fo'); // "FeeFiFo"
```

PHP FUNCTIONS: SCOPE AND LIFETIME

By default, any variable used inside a function is limited to this local scope.
Once the scope of the function ends, the local variable is destroyed

```
$x = 'Hello'; // global variable
function myFunc()
{
$y = ' World'; // local variable
}
```

We can access global variables, and modify them, if we declare them with global inside the function

```
$x = 'Hello'; // global $x
function myFunc()
{
global $x; // use global $x
$x .= ' World'; // change global $x
}
myFunc();
echo $x; // "Hello World"
```

PHP FUNCTIONS: SCOPE AND LIFETIME

An alternative way to access variables from the global scope is by using the predefined $GLOBALS array.

The variable is referenced by its name, specified as a string without the dollar sign.

```
function myFunc()
{
$GLOBALS['x'] .= ' World'; // change global $x
}
```

PHP FUNCTIONS: PASSING ARGUMENTS BY VALUE

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

```
function showconcat($cad1, $cad2)
{
    echo $cad1.$cad2;
}
```

PHP FUNCTIONS: PASSING ARGUMENTS BY REFERENCE

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

```
function addstring(&$cad1, $cad2)
{
    $cad1 = $cad1.$cad2;
}

//call
$cad1='hello ';
$cad2=' world';
addstring($cad1,$cad2);
echo $cad1; //hello world. The variable $cad1 has been modified
```

PHP FUNCTIONS: ARGUMENT TYPE DECLARATIONS

To allow for functions that are more robust, PHP 5 began to introduce argument type declarations, permitting the type of a function parameter to be specified.

```
function myPrint(array $a)
{
foreach ($a as $v) { echo $v; }
}
myPrint( array(1,2,3) ); // "123"
myPrint('Test'); // error!!!
```

PHP FUNCTIONS: RETURN TYPE DECLARATIONS

Support for return type declarations was added in PHP 7 as a way to prevent unintended return values.

```
function f(): array {
return [];
}
```

PHP FUNCTIONS: VARIABLE FUNCTIONS

WE CAN HAVE THE NAME OF A FUNCTION IN A VARIABLE, AND EXECUTE THE FUNCTION BY CALLING IT WITH THE NAME OF THE VARIABLE

Instead of doing this:

```
if($operation==1)
{ add();}
else
 { subs();}
```

Do this:

```
function add ()
{  echo "Add"; }

function subs ()
{    echo "Subs";}

$operation = 1;

if($operation==1)
 { $function="add"; }
else
{ $function="subs"; }

$function();
```

INCLUDE STATEMENT

This statement takes all the text in the specified file and includes it in the script, as if the code had been copied to that location

We can use include to have a library of functions, and include it every time in our pages.

```
include "functions.php";
…
// our web page
...
```

In addition to include, there are three other language constructs available for importing the content of one file into another: require, include_once and require_once.

REQUIRE STATEMENT

The require construct includes and evaluates the specified file. It is identical to include, except in how it handles failure. When a file import fails, require halts the script with an error; whereas include only issues a warning.

INCLUDE_ONCE STATEMENT

The include_once statement behaves like include, except that if the specified file has already been included, it is not included again.

```
include_once 'myfile.php'; // include only once
```

REQUIRE_ONCE STATEMENT

The require_once statement works like require, but it does not import a file if it has already been imported.

```
require_once 'myfile.php'; // require only once
```

WHEN USE EACH STATEMENT

We can think of using include when the file to be inserted is not decisive regarding the operation of our program and require when the file is necessary for the correct operation of our program.

Finally, the variants with _once should be used when our program has considerable dimensions and it may be the case that the inclusion of the file occurs several times.