Function

1. Introduction to Functions, Parameters, and Arguments

Functions help organize code and avoid repetition by grouping instructions that can be reused. Parameters are placeholders in functions, while arguments are actual values passed to these parameters when calling the function.

```
function say_hello_to($someone) {
  echo "Hello Mr $someone<br>";
}
say_hello_to("Osama"); // Output: Hello Mr Osama
```

2. Using Multiple Parameters

Functions can accept multiple parameters, allowing you to pass several values into the function.

```
function say_hello_to($one, $two) {
  echo "Hello $one $two<br>";
}
say_hello_to("Osama", "Ahmed"); // Output: Hello Osama Ahmed
```

3. Conditional Logic in Functions

Functions can contain logic to execute different code based on the value of the arguments.

```
function deep_freezer($item) {
  if ($item === "Water") {
    echo "Make Ice <br>";
```

```
  } elseif ($item === "Coca Cola") {
    echo "Make It Cold <br>";
  } else {
    echo "Unknown Item";
  }
}
deep_freezer("Water"); // Output: Make Ice
```

4. Return Values and Early Exit with `return`

The `return` statement exits the function immediately, returning a specified value. After `return`, any code is not executed.

```
function get_number($num_one, $num_two) {
  return $num_one + $num_two;
  echo "This will not run"; // Unreachable code
}
echo get_number(2, 1); // Output: 3
```

5. Default Parameter Values

Set default values for parameters to make them optional. If a parameter is not provided, it defaults to the specified value.

```
function get_data($country = "Private Country", $name = "Priv
ate") {
  return "Country: $country, Name: $name";
}
echo get_data(); // Output: Country: Private Country, Name: P
rivate
```

6. Variable-Length Argument Lists

PHP allows functions to accept a variable number of arguments using `...$nums`. This collects arguments into an array.

```
function calculate(...$nums) {
  $result = array_sum($nums);
  echo $result;
}
calculate(10, 20, 30); // Output: 60
```

7. Unpacking Arguments Using `...`

Using `...` lets you unpack arrays into separate arguments when calling a function.

```
$skills = ["HTML", "CSS"];
function get_data($name, ...$skills) {
  echo "Hello $name<br>";
}
get_data("Osama", ...$skills);
```

8. Variable Functions and `function_exists()`

In PHP, you can use a variable as a function name, and check if a function exists with `function_exists()`.

```
function hello() { return "Hello"; }
$func_name = "hello";

if (function_exists($func_name)) {
```

```
  echo $func_name(); // Output: Hello
}
```

8. **Passing by Reference**: Modify variables directly by passing them by reference using `&`.

```
function add_five(&$num) { $num += 5; }
$n = 10;
add_five($n); // $n becomes 15
```

9. Passing Arguments by Reference

By default, arguments are passed by value, meaning changes inside the function don't affect the outside variable. Use `&` to pass by reference, which allows the function to modify the original variable.

```
function add_five(&$number) {
  $number += 5;
}
$n = 10;
add_five($n);
echo $n; // Output: 15
```

10. Return Type Declarations

PHP allows you to specify the return type of a function with a `: type` declaration. This is especially useful to ensure consistent output.

```
function calculate($n1, $n2): int {
  return $n1 + $n2;
```

```
}
echo calculate(10.5, 9.5); // Output: 19 (as an integer)
```

11. Anonymous Functions (Closures)

Anonymous functions don't have a name and are often used for specific, short-lived tasks. They can inherit variables from the parent scope using `use`.

```
$msg = "Hi";
$say_hi = function($name) use ($msg) {
  return "$msg $name";
};
echo $say_hi("Osama"); // Output: Hi Osama
```

12. Arrow Functions

Arrow functions are shorthand for anonymous functions. They automatically inherit variables from the parent scope and don't require `use`.

```
$msg = "Hello";
$say_hello = fn($name) => "$msg $name";
echo $say_hello("Osama"); // Output: Hello Osama
```

13. Argument Skipping and Named Arguments

Starting from PHP 8, named arguments allow you to pass arguments out of order by specifying the parameter name, skipping others as needed.

```
function get_data($name = "Unknown", $age = "Unknown", $count
ry = "Unknown") {
  return "$name, Age: $age, Country: $country";
}
```

```
echo get_data(age: 30, country: "Egypt"); // Output: Unknown,
Age: 30, Country: Egypt
```