Loops in Programming

Loops or Iteration Statements in Programming are helpful when we need a specific task in repetition. They're essential as they reduce hours of work to seconds. In this article, we will explore the basics of loops, with the different types and best practices.

Types of Loops in Programming:
In programming, loops are categorized into two main types based on the control mechanism: entry-controlled loops and exit-controlled loops.

1. Entry-Controlled loops:
In Entry controlled loops the test condition is checked before entering the main body of the loop. For Loop and While Loop are Entry-controlled loops.

2. Exit-Controlled loops:
In Exit controlled loops the test condition is evaluated at the end of the loop body. The loop body will execute at least once, irrespective of whether the condition is true or false. Do-while Loop is an example of Exit Controlled loop.

Functions in Programming

Functions in Programming is a block of code that encapsulates a specific task or related group of tasks. Functions are defined by a name, may have parameters and may return a value. The main idea behind functions is to take a large program, break it into smaller, more manageable pieces (or functions), each of which accomplishes a specific task.

1. Functions Declaration and Definition
2. Calling a Functions in Programming
3. Parameters and Return Values
4. Built-in Functions vs. User-Defined Functions
5. Recursion in Functions

Recursion in Functions

```
<?php
function factorial($n) {
    // Base case: If n is 0, the factorial is 1
    if ($n === 0) {
        return 1;
    } else {
        // Recursive call: n! = n * (n-1)!
        return $n * factorial($n - 1);
    }
}

// Example usage:
$number = 5;
$result = factorial($number);
echo "The factorial of " . $number . " is: " . $result; // Output: The factorial of 5 is: 120
?>
```

Explanation:

factorial($n) Function:

Takes an integer n as input.
Base Case: if ($n === 0): If n is 0, it returns 1 (because 0! = 1). This stops the recursion.
Recursive Call: else { return $n * factorial($n - 1); }: If n is not 0, it returns n multiplied by the factorial of n - 1. This is where the function calls itself.
How it Works (Example with factorial(5)):

factorial(5) calls 5 * factorial(4)
factorial(4) calls 4 * factorial(3)
factorial(3) calls 3 * factorial(2)
factorial(2) calls 2 * factorial(1)
factorial(1) calls 1 * factorial(0)
factorial(0) returns 1 (base case)

The results are then multiplied back up the chain: 1 * 1 * 2 * 3 * 4 * 5 = 120