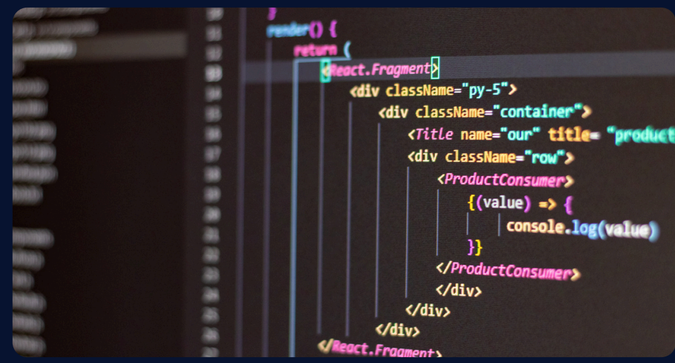


Recursion Recursion

Presented By

Kalaivanan R - 24MX108

Karthikeyan R - 24MX111

Nitesh Raja - 24MX211

What is a Recursion ?

Recursion is a programming technique where a function calls itself directly or indirectly to solve smaller instances of the same problem.

Key features of Recursion

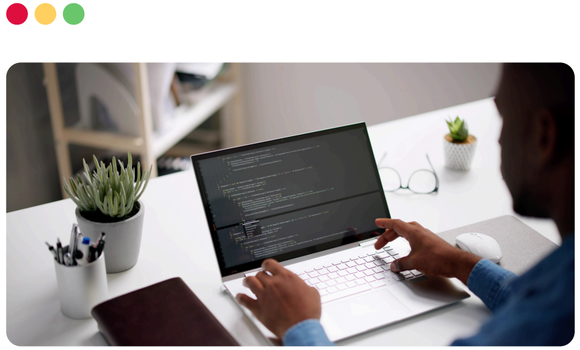
Recursive Case:

- The part where the function calls itself with smaller inputs.
- Breaks down the problem into smaller sub-problems.

Base Case:

- A condition that stops the recursion.
- Without it, the recursion would go on infinitely and result in a stack overflow.
- Example: In calculating the factorial of a number n, the base case is n=0 or n=1.

Why Recursion ?



Simplifies Complex Problems: Problems like tree traversal, backtracking, or divide-and-conquer algorithms are easier to implement with recursion.

Natural Fit for Repetitive Sub-problems : Tasks that involve breaking into smaller parts, such as solving the Fibonacci sequence or Towers of Hanoi.

Real Life Analogies

1. Russian Dolls:Each doll contains a smaller doll, and the process continues until the smallest doll is reached.
2. Mirrors Facing Each Other:A reflection creates smaller and smaller images, similar to how recursion works.

Types of Recusrion

- Direct Recursion
- Indirect Recursion
- Tail Recursion
- Non-Tail Recursion

Direct Recursion

A function directly calls itself.

```
int factorial(int n) {
    if (n == 0) return 1; // Base case
    return n * factorial(n - 1); // Recursive
case
}
```

Indirect Recursion

- A function calls another function, which in turn calls the first function.
- This creates a chain of calls between functions.

Example Code

```
void odd();
void even();
int n = 1;
void odd() {
  if (n <= 10) {
    printf("%d ", n+1);
    n++;
    even();
  }
  return ;
}
```

```
void even() {
  if (n <= 10) {
    printf("%d ", n-1);
    n++;
    odd();
  }
  return;
}
int main() {
  odd();
}
```

Tail Recursion

- The recursive call is the last operation in the function.
- No additional computation happens after the recursive call.

Example Code

```
void fun (int n){
  if (n == 0) {
    return;
  } else {
    printf("%d ", n);
  }
  return fun(n - 1);
}
```

```
int main() {
  fun(3);
  return 0;
}
```

Non-Tail Recursion

- The recursive call is not the last operation in the function.
- Further computations happen after the recursive call.

Example Code

```
int fun (int n) {
  if (n == 1) {
    return 0;
  } else {
    return 1 +
fun(n/2);
  }
}
```

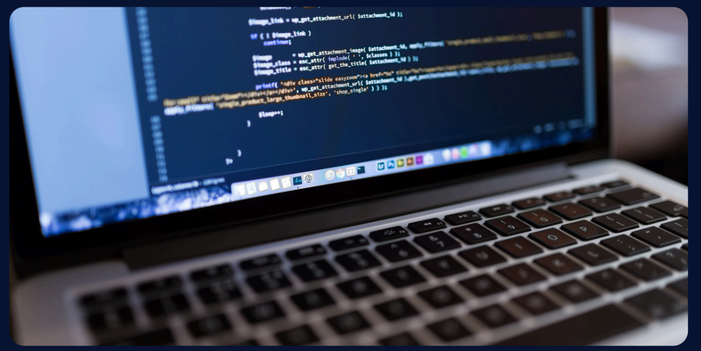
```
int main() {
  printf("%d",fun(8));
  return 0;
}
```

Recursion vs Iteration

Scenario	Preferred Approach	Reason
Problems with a natural recursive structure (e.g., tree traversal, divide-and-conquer algorithms).	**Recursion**	Simplifies the implementation, matches the problem structure.
Problems requiring repetitive computations without hierarchy (e.g., summing a list).	**Iteration**	Efficient in terms of memory and processing time.
Problems with a clear base case and smaller sub-problems (e.g., factorial, Fibonacci).	**Recursion**	Easier to express and understand in recursive terms.
Simple loops without recursive dependencies (e.g., printing numbers from 1 to 100).	**Iteration**	Avoids unnecessary overhead of function calls.

Conclusion

- Key Takeaways
- Recursion is a powerful concept where a function calls itself to solve problems by breaking them into smaller sub-problems.
- It relies on two main components:
  - Base Case: Stops the recursion.
  - Recursive Case: Continues the process with smaller inputs.
- Recursion simplifies problems with a natural hierarchical or divide-and-conquer structure (e.g., tree traversal, Fibonacci, sorting algorithms).



Thank You