Tutorial Link https://codequotient.com/tutorials/Recursion - Stack Overflow/5a0148e7cbb2fe34b777504e

TUTORIAL

# Recursion - Stack Overflow

## Chapter

Recursion if not implemented in perfect manner may lead to infinite recursion. There can be many situations of non-ending recursion. Consider the following implementation of factorial function: -

Javascript

```
function fact_recursive(num){
    let result;
    if (num == 100)             // Base case for recursion is not good.
        return 1;
    else{
        result = num * fact_recursive(num-1);      // Call recursively with lesser number.
        return result;
    }
}

function main(){
    let number, fact1;
    number = 5;
    fact1 = fact_recursive(number);        // Call the Recursive version
    console.log(`Number=${number}`);
    console.log(`Recursive_Factorial=${fact1}`);
}

main()
```

C

```c
#include<stdio.h>

int fact_recursive(int num)
{
  if (num == 100)       // This base case is not good for recursion.
    return 1;     // fact() return 1 if argument is 100
  else
    return num * fact_recursive(num-1);
  /* Call recursively with lesser number,
  but never reache to 100 if called with a number lesser than 100. */
}

int main()
{
  int number, fact1;
  number = 5;
  fact1 = fact_recursive(number);        // Call the Recursive version
  printf("Number=%d\n", number);
  printf("Recursive_Factorial=%d\n", fact1);
  return 0;
}

```

Java

```java
class Main{
    static int fact_recursive(int num){
        int result;
        if (num == 100)         // This base case is not good for recursion.
            return 1;
        else{
            result = num * fact_recursive(num-1);    // Call recursively with lesser number.
            return result;
        }
    }

    public static void main(String[] args){
        int number, fact1;
        number = 5;
        fact1 = fact_recursive(number);         // Call the Recursive version
        System.out.println("Number=" + number);
        System.out.println("Recursive_Factorial=" + fact1);
    }
}
```

Python 3

```
def fact_recursive(num):
  if (num == 100):              # Base case for recursion is not good.
    return 1
  else:
    result = num * fact_recursive(num-1);      # Call recursively with lesser number.
    return result

if __name__ == '__main__':
    number = 5
    fact1 = fact_recursive(number)        # Call the Recursive version
    print("Number=",number)
    print("Recursive_Factorial=",str(fact1))
```

C++

```
#include<iostream>
using namespace std;

int fact_recursive(int num)
{
  if (num == 100)       // This base case is not good for recursion.
    return 1;
  else
    return num * fact_recursive(num-1);      // Call recursively with lesser number.
}
int main(){
  int number, fact1;
  int num1, num2;
  number = 5;
  fact1 = fact_recursive(number);        // Call the Recursive version
  cout<<"Number="<<number<<endl;
  cout<<"Recursive_Factorial="<<fact1<<endl;
}
```

The above program may not return the answer as the recursion will never end. The base case in this program will hit when number=100, whereas each time the function calls itself it will call with decremented value, so if user enters a values less than 100, then it will never reaches the base case, resulting in infinite recursion. So reaching to the base case is necessary in recursion.

Also the factorial can be written in another recursive manner as below, 0!=1 and, for all n > 0, n! = (n + 1)! / (n+1). For example,

```
Fact (5)        = fact(6) / 6
                = 720 / 6
                = 120
```

Although it will clearly calculates the factorial of a number, but it is not going to call the base case. So the recursion never ends. To find the factorial of 5, we need the factorial of 6, to find the factorial of 6, we need of 7 and so on, and we are moving away from 0 (the base case). So these problems must be tackled in recursion, otherwise recursion will not be a better idea.