Top-Down Approach with Memoization

Whenever we solve a subproblem, we cache its result so that we don't end up solving it repeatedly if it's called multiple times. This technique of storing the results of the previously solved subproblems is called Memoization.

In memoization, we solve the bigger problem by recursively finding the solution to the subproblems. It is a top-down approach.

In [23]:
```
#Implementation of Fibonacci Series using Memoization
""" The Fibonacci Series is a series of numbers in which each number
    is the sum of the preceding two numbers.
    By definition, the first two numbers are 0 and 1.

    Implement with the following steps:
    - Declare function with parameters: Number N and Dictionary Memo.
    - If n equals 1, return 0
    - If n equals 2, return 1
    - If current element is not in memo, add to memoy by recursive call for previous f

def fibMemo(n, memo):
  if n == 1:
    return 0
  if n == 2:
    return 1
  if not n in memo:
      memo[n] = fibMemo(n-1, memo) + fibMemo(n-2, memo)
  return memo[n]

tempDict = {}
fibMemo(6, tempDict)

print("Fibonacci Series - Memoization")
print("0")
print("1")
for element in tempDict.values():
    print(element)


#Implementation of Factorial of N numbers using Memoization
def factMemo(n, memo={}):
  if n == 0 or n == 1:
    return 1
  if n in memo:
    return memo[n]
  answer = n * factMemo(n-1, memo)
  memo[n] = answer
  return answer


print("\nFactorial - Memoization")
result = factMemo(6)
print(result)
```

```
Fibonacci Series - Memoization
0
1
1
2
3
5

Factorial - Memoization
720
```

Bottom-Up Approach with Tabulation

Tabulation is the opposite of the top-down approach and does not involve recursion. In this approach, we solve the problem “bottom-up”. This means that the subproblems are solved first and are then combined to form the solution to the original problem.

This is achieved by filling up a table. Based on the results of the table, the solution to the original problem is computed.

In [20]:
```python
#Implementation of Fibonacci Series using Tabulation
""" Fibonacci Series can be implemented using Tabulation using the following steps:
    - Declare the function and take the number whose Fibonacci Series is to be printed
    - Initialize the list and input the values 0 and 1 in it.
    - Iterate over the range of 2 to n+1.
    - Append the list with the sum of the previous two values of the list.
    - Return the list as output. """

def fibTab(n):
    tb = [0, 1]
    for i in range(2, n+1):
      tb.append(tb[i-1] + tb[i-2])
    return tb

print("Fibonacci Series - Tabulation")
print(fibTab(6))

#Implementation of Factorial of N numbers using Tabulation
def factTab(n):
  if n < 0:
    return None #no negative numbers
  elif n == 0 or n == 1:
    return 1

  tb = [0] * (n+1)
  tb[0] = tb[1] = 1

  for i in range(2, n+1):
    tb[i] = i * tb[i-1]

  return tb[n]

print("\nFactorial - Tabulation")
print(factTab(6))
```

```
Fibonacci Series - Tabulation
[0, 1, 1, 2, 3, 5, 8]

Factorial - Tabulation
720
```