nCr

Example 1:

```
Input: n = 3, r = 2
Output: 3
Explaination: <sup>3</sup>C<sub>2</sub> = 3.
```

Example 2:

```
Input: n = 2, r = 4
Output: 0
Explaination: r is greater than n.
```

Your Task:
You do not need to take input or print anything. Your task is to complete the function **nCr()** which takes n and r as input parameters and returns ${}^{n}C_{r}$ modulo $10^{9}+7$..

Expected Time Complexity: O(n*r)
Expected Auxiliary Space: O(r)

Constraints:
$1 \le n \le 1000$
$1 \le r \le 800$

The value of ${}^{n}C_{r}\%p$ is generally needed for large values of n when ${}^{n}C_{r}$ cannot fit in a variable, and causes overflow. So computing ${}^{n}C_{r}$ and then using modular operator is not a good idea as there will be overflow even for slightly larger values of n and r.

The idea is to compute ${}^{n}C_{r}$ using below formula

```
C(n, r) = C(n-1, r-1) + C(n-1, r)
  C(n, 0) = C(n, n) = 1
```

Extension of above formula for modular arithmetic:
We can use distributive property of modulor operator to find nCr % p using above formula.

```
C(n, r)%p = [ C(n-1, r-1)%p + C(n-1, r)%p ] % p
  C(n, 0) = C(n, n) = 1
```

So, let's take an example of 4C3. Pascal's triangle for 4C3 is as:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Rows in Pascal's triangle decides the value of given nCr. So, try to fill the entries of current row using the previous row values (nCj = (n-1)Cj + (n-1)C(j-1)). Try to think of doing this using 1D array.

```
class Solution:
    def nCr(self, n, r):
        # code here
        C = [[0 for x in range(r+1)] for x in range(n+1)]

    # Calculate value of Binomial
    # Coefficient in bottom up manner
        for i in range(n+1):
            for j in range(0,min(i, r)+1):
                # Base Cases
                if j == 0 or j == i:
                    C[i][j] = 1

                # Calculate value using
                # previously stored values
                else:
                    C[i][j] = (C[i-1][j-1]%(10**9+7) + C[i-1][j]%
(10**9+7))%(10**9+7)

        return C[n][r]
```