1381. Design a Stack With Increment Operation

Design a stack which supports the following operations.

Implement the `CustomStack` class:

- `CustomStack(int maxSize)` Initializes the object with `maxSize` which is the maximum number of elements in the stack or do nothing if the stack reached the `maxSize`.
- `void push(int x)` Adds `x` to the top of the stack if the stack hasn't reached the `maxSize`.
- `int pop()` Pops and returns the top of stack or **-1** if the stack is empty.
- `void inc(int k, int val)` Increments the bottom `k` elements of the stack by `val`. If there are less than `k` elements in the stack, just increment all the elements in the stack.

Example 1:

```
Input
["CustomStack","push","push","pop","push","push","push","increment","increme
nt","pop","pop","pop","pop"]
[[3],[1],[2],[],[2],[3],[4],[5,100],[2,100],[],[],[],[]]
Output
[null,null,null,2,null,null,null,null,null,103,202,201,-1]
Explanation
CustomStack customStack = new CustomStack(3); // Stack is Empty []
customStack.push(1);                          // stack becomes [1]
customStack.push(2);                          // stack becomes [1, 2]
customStack.pop();                            // return 2 --> Return top of
the stack 2, stack becomes [1]
customStack.push(2);                          // stack becomes [1, 2]
customStack.push(3);                          // stack becomes [1, 2, 3]
customStack.push(4);                          // stack still [1, 2, 3],
Don't add another elements as size is 4
customStack.increment(5, 100);                // stack becomes [101, 102,
103]
customStack.increment(2, 100);                // stack becomes [201, 202,
103]
customStack.pop();                            // return 103 --> Return top
of the stack 103, stack becomes [201, 202]
customStack.pop();                            // return 202 --> Return top
of the stack 102, stack becomes [201]
customStack.pop();                            // return 201 --> Return top
of the stack 101, stack becomes []
```

```
customStack.pop();                            // return -1 --> Stack is
empty return -1.
```

Constraints:

- `1 <= maxSize <= 1000`
- `1 <= x <= 1000`
- `1 <= k <= 1000`
- `0 <= val <= 100`
- At most `1000` calls will be made to each method of `increment`, `push` and `pop` each separately.

```
def __init__(self, maxSize: int):
        self.stack = []
        self.size = maxSize


    def push(self, x: int) -> None:
        if len(self.stack)<self.size:
            self.stack.append(x)


    def pop(self) -> int:
        if not len(self.stack):
            return -1
        else:
            return self.stack.pop()


    def increment(self, k: int, val: int) -> None:
        for i in range(min(k,len(self.stack))):
            self.stack[i] = self.stack[i]+val
```