## Question - 1
**Ways to Sum**

An automated packaging system is responsible for packing boxes. A box is certified to hold a certain weight. Given an integer *total*, calculate the number of possible ways to achieve *total* as a sum of the weights of items weighing integer weights from *1* to *k*, inclusive.

**Example**

*total = 8*

*k = 2*

To reach a weight of 8, there are 5 different ways that items with weights between 1 and 2 can be combined:

- [1, 1, 1, 1, 1, 1, 1, 1]
- [1, 1, 1, 1, 1, 1, 2]
- [1, 1, 1, 1, 2, 2]
- [1, 1, 2, 2, 2]
- [2, 2, 2, 2]

**Function Description**

Complete the function *ways* in the editor below.

ways has the following parameter(s):

*int total:* the value to sum to

*int k:* the maximum of the range of integers to consider when summing to *total*

**Returns**

*int:* the number of ways to sum to the *total*; the number might be very large, so return the integer modulo *1000000007* ($10^9+7$)

**Constraints**

- $1 \le total \le 1000$
- $1 \le k \le 100$

**▼ Input Format For Custom Testing**

The first line contains an integer, *total*, that denotes the target sum.

The second line contains an integer, *k*, that denotes the maximum value in the range of integers to be considered, i.e, from *1* to *k*.

**▼ Sample Case 0**

**Sample Input For Custom Testing**

```
STDIN       Function
-----       --------
5       →   total = 5
3       →   k = 3
```

**Sample Output**

Help

```
5
```

**Explanation**

The sum required is *5*. *k = 3* so the integers that can be considered to reach the sum are *[ 1, 2, 3 ]*.

The *5* ways to reach the target sum are:

1. *1 + 1 + 1 + 1 + 1 = 5*
2. *1 + 1 + 1 + 2 = 5*
3. *1 + 2 + 2 = 5*
4. *1 + 1 + 3 = 5*
5. *2 + 3 = 5*

*5 modulo 1000000007 = 5*

**▼ Sample Case 1**

**Sample Input For Custom Testing**

```
STDIN    Function
-----    --------
4     →  total = 4
2     →  k = 2
```

**Sample Output**

```
3
```

**Explanation**

The sum required is *4*, and the range of integers is *[ 1, 2 ]*

There are *3* ways to reach the target sum:

1. *1 + 1 + 1 + 1 = 4*
2. *1 + 1 + 2 = 4*
3. *2 + 2 = 4*

*3 modulo 1000000007 = 3.*

## Question - 2

**String Patterns**

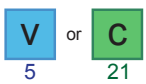
Given the length of a word (*wordLen*) and the maximum number of consecutive vowels that it can contain (*maxVowels*), determine how many unique words can be generated. Words will consist of English alphabetic letters a through z only. Vowels are *v*: {*a, e, i, o, u*}; consonants are *c*: the remaining *21* letters. In the explanations, *v* and *c* represent vowels and consonants.

*wordLen = 1*

*maxVowels = 1*

Patterns: *{v, c}*

That means there are *26* possibilities, one for each letter in the alphabet.

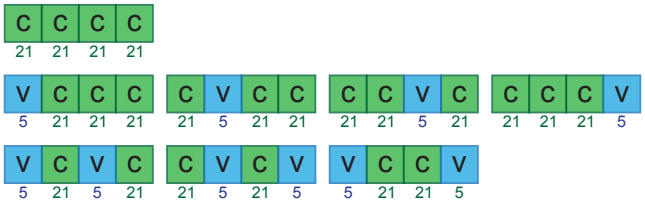



*wordLen = 4*

*maxVowels = 1*

Patterns: *{cccc, vccc, cvcc, ccvc, cccv, vcvc, cvcv, vccv}*

There are 412,776 possibilities -- see below:




(21 * 21 * 21 * 21) = 194481

(5 * 21 * 21 * 21) + (21 * 5 * 21 * 21) + (21 * 21 * 5 * 21) + (21 * 21 * 21 * 5) = 4 * 46305 = 185220

(5 * 21 * 5 * 21) + (21 * 5 * 21 * 5) + (5 * 21 * 21 * 5) = 3 * 11025 = 33075

194481 + 185220 + 33075 = 412776 possible solutions.

*wordLen = 4*

*maxVowels = 2*

In this case, all of the combinations from the previous example are still valid.

- There are 5 additional patterns to consider, three with 2 vowels (vvcc, cvvc, ccvv) and 2 with 3 vowels (vvcv and vcvv).
- Their counts are 3 * (5 * 5 * 21 * 21) = 3 * 11025 = 33075 and 2 * (5 * 5 * 5 * 21) = 2 * 2625 = 5250.
- The total number of combinations then is 412776 + 33075 + 5250 = 451101.

The result may be a very large number, so return the answer modulo ($10^9$+7).

**Note:** While the answers will be within the limit of a 32 bit integer, interim values may exceed that limit. Within the function, you may need to use a 64 bit integer type to store them.

**Function Description**

Complete the function *calculateWays* in the editor below.

calculateWays has the following parameter(s):

*int wordLen:* the length of a word

*int maxVowels:* the maximum number of consecutive vowels allowed in a word

**Returns**

*int:* the number of well-formed strings that can be created, modulo *1000000007* ($10^9$+7)

**Constraints**

- *1 ≤ wordLen ≤ 2500*
- *0 ≤ maxVowels ≤ n*

**▼ Input Format Format for Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *wordLen*, the length of the words to create.

The next line contains an integer *maxVowels*, maximum number of consecutive vowels allowed.

### ▼ Sample Case 0

**Sample Input 0**

```
STDIN     Function
-----     --------
2     →   wordLen = 2
1     →   maxVowels = 1
```

**Sample Output 0**

```
651
```

**Explanation 0**

Words take the forms *{vc, cv, cc}*. There is a vowel in the first position, the second position or no position. The total number of unique words is *(5 * 21) + (21 * 5) + (21 * 21) = 651* and *651 modulo 1000000007 = 651*.

### ▼ Sample Case 1

**Sample Input 1**

```
STDIN     Function
-----     --------
2     →   wordLen = 2
2     →   maxVowels = 2
```

**Sample Output 1**

```
676
```

**Explanation 1**

Since the words are *2* characters, and there can be *2* consecutive vowels, each position can contain any character. Words take the forms *{vv, vc, cv, cc}*. The total number of unique words is *(26 * 26) = 676* and *676 modulo 1000000007 = 676*.

### ▼ Sample Case 2

**Sample Input 2**

```
STDIN     Function
-----     --------
2     →   wordLen = 2
0     →   maxVowels = 0
```

**Sample Output 2**

```
441
```

**Explanation 2**

No vowels are allowed in a word, therefore the words are in the form *{cc}*. The total number of unique words is *(21 * 21) = 441* and *441 modulo 1000000007 = 441*.