☆ Jumping Jack

Jumping Jack is standing at the bottom of a flight of stairs at step number *0*, and each subsequent step up the staircase is numbered sequentially from *1* to infinity. Jack performs *n* consecutively numbered actions; for example, if *n = 3*, then Jack

or jump to step *j + i*.

1
2
3
4

Complete the *maxStep* function in the editor below. It has two parameters:

1. An integer, *n*, denoting the number of actions Jack must take.
2. An integer, *k*, denoting the step number Jack must not land on.

The function must return an integer denoting the *maximum* step number Jack can reach from step *0* if he performs exactly *n* actions and never jumps on step *k* (though he may jump *over* it).

Input Format

Locked stub code in the editor reads the following input from stdin and passes it to the function:

The first line contains an integer, *n*, denoting the number of actions Jack must take.

The second line contains an integer, *k*, denoting the step number Jack must not land on.

Constraints

- $1 \le n \le 2 \times 10^3$
- $1 \le k \le 4 \times 10^6$

Output Format

The function must return an integer denoting the *maximal* step number Jack can reach. This is printed to stdout by locked stub code in the editor.

Sample Input 0

```
2
2
```

Sample Output 0

```
3
```

Explanation 0

Jack performs the following sequence of $n = 2$ actions:

1. Jack jumps from step 0 to step $0 + 1 = 1$.
2. Jack jumps from step 1 to step $1 + 2 = 3$; observe that he avoided step $k = 2$ by jumping *over* it.

Sample Input 1

```
2
1
```

Sample Output 1

```
2
```

Explanation 1

Jack performs the following sequence of $n = 2$ actions:

1. Jack cannot jump onto step 1 (because $k = 1$ and he can only jump 1 step during his first action), so he stays on step 0.
2. Jack jumps from step 0 to step $0 + 2 = 2$.

Sample Input 2

```
3
3
```

Sample Output 2

```
5
```

Explanation 2

Jack must skip some jump, because performing one jump during each step will land him on step *k = 3* on the second jump. There are two ways for him to perform all *n = 3* actions:

- For the first action, jump *1* unit to step *0 + 1 = 1*. For the second action, remain at step *1*. For the third action, jump *3* units to step *1 + 3 = 4*. In other words, his sequence of actions is *0 → 1 → 1 → 4*.
- For the first action, remain at step *0*. For the second action, jump *2* units to step *0 + 2 = 2*. For the third action, jump *3* units to step *2 + 3 = 5*. In other words, his sequence of actions is *0 → 0 → 2 → 5*.

Because we want the maximal step number that Jack can reach by performing any sequence of possible actions, we return *5* as our answer.

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. ✖

Start tour

Original code   C ⌄   ⚙

```
#include ↔

/*
 * Complete the function below.
 */
int maxStep(int n, int k) {


}

```

```
int main() {↔}

```

Line: 10 Col: 1

☐ **Test against custom input**

Run Code

Submit code & Continue

(You can submit any number of times)

Download sample test cases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*

About Privacy Policy Terms of Service