# Nim Game

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

1. If there are 5 stones in the heap, could you figure out a way to remove the stones such that you will always be the winner?

**Credits:**
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.

## Solution 1

> Theorem: The first one who got the number that is multiple of 4 (i.e. n % 4 == 0) will lost, otherwise he/she will win.

Proof:

1. the base case: when `n = 4`, as suggested by the hint from the problem, no matter which number that that first player, the second player would always be able to pick the remaining number.
2. For `1* 4 < n < 2 * 4, (n = 5, 6, 7)`, the first player can reduce the initial number into 4 accordingly, which will leave the death number 4 to the second player. i.e. The numbers 5, 6, 7 are winning numbers for any player who got it first.
3. Now to the beginning of the next cycle, `n = 8`, no matter which number that the first player picks, it would always leave the winning numbers (5, 6, 7) to the second player. Therefore, 8 % 4 == 0, again is a death number.
4. Following the second case, for numbers between (2*4 = 8) and (3*4=12), which are `9, 10, 11`, are winning numbers for the first player again, because the first player can always reduce the number into the death number 8.

Following the above theorem and proof, the solution could not be simpler:

```
public boolean canWinNim(int n) {
    return n % 4 != 0 ;
}
```

written by liaison original link here

## Solution 2

suppose there are x stones left for first player (A), he can take 1,2,3 stones away, so second player B will have three cases to deal with (x -1), (x-2), (x-3). after he pick the stones, there will be 9 cases left for A.

```
B (x-1) -> A: (x-2), (x-3), (x-4)
B (x-2) -> A: (x-3), (x-4), (x-5)
B (x-3) -> A: (x-4), (x-5), (x-6)
```

Now, if A can guarantee he win at either of three groups, then he can force B to into that one of the three states and A can end up in that particular group after B's move.

```
f(x) = (f(x-2)&&f(x-3)&&f(x-4)) || (f(x-3)&&f(x-4)&&f(x-5)) || (f(x-4)&&f(x-5)&&f
(x-6))
```

if we examine the equation a little closer, we can find f(x - 4) is a critical point, if f(x-4) is false, then f(x) will be always false.

we can also find out the initial conditions, f(1), f(2), f(3) will be true (A always win), and f(4) will be false. so based on previous equation and initial conditions f(5) = f(6) = f(7) = true, f(8) = false; obviously, f(1), f(2), f(3) can make all f(4n + 1), f(4n + 2), f(4n + 3) to be true, only f(4n) will be false then. so here we go our one line solution:

return (n % 4 != 0);

written by kennethliaoke original link here

## Solution 3

```
class Solution(object):
    def canWinNim(self, n):
        """
        :type n: int
        :rtype: bool
        """
        # strategy: the one with 4 remaining must loose
        # A, B players
        # if n == 4k, then at each round B can make A+B both take 4,
        # eventually leave 4 to A, A lose
        # if n == 4k + i (i <= 3), then A can always take i first and B will
        # finanly lose as he faces above scenario like A

        return bool(n%4!=0)
```

written by pennlio original link here