

**You have a function `rand7()` that generates a random integer from 1 to 7. Use it to write a function `rand5()` that generates a random integer from 1 to 5.**

`rand7()` returns each integer with equal probability. `rand5()` must also return each integer with equal probability.

## Gotchas

Your first thought might be to simply take the result of `rand7()` and take a modulus:

```
def rand5():  
    return rand7() % 5 + 1
```

Python ▼

**However, this won't give an equal probability for each possible result.** We can write out each possible result from `rand7()` (each of which is equally probable, per the problem statement) and see that some results for `rand5()` are more likely because they are caused by more results from `rand7()`:

<code>rand7()</code>	<code>rand5()</code>
1	2
2	3
3	4
4	5
5	1
6	2
7	3

So we see that there are two ways to get 2 and 3, but only one way to get 1, 4, or 5. This makes 2 and 3 twice as likely as the others.

The answer takes worst-case infinite time. However, we can get away with worst-case  $O(1)$  space. **Does your answer have a non-constant space cost?** If you're using recursion (and your language doesn't have tail-call optimization,

*Some* compilers and interpreters will do what's called "**tail call optimization**" (TCO), where it can optimize *some* recursive functions to avoid building up a tall call stack. Python and Java decidedly do not use TCO. Some Ruby implementations do, but most don't. Scheme is one of the few languages that guarantee TCO in all implementations. In general, best not to assume your compiler/interpreter will do this work for you.

), you're potentially incurring a worst-case infinite space cost in the call stack.

The **call stack** is what a program uses to keep track of what function it's currently running and what to do with that function's return value.

Whenever you call a function, a new **frame** gets pushed onto the call stack, which is popped off when the function returns. As functions call other functions, the stack gets taller. In recursive functions, the stack can get as tall as the number of times the function calls itself. This can cause a problem: the stack has a limited amount of space, and if it gets too big you can get a **stack overflow** error.

But replacing your recursion with a loop avoids this.

## Breakdown

`rand5()` must return each integer with equal probability, but we don't need to make any guarantees about its runtime...

In fact, the solution has a small possibility of *never* returning...

## Solution

We simply "re-roll" whenever we get a number greater than 5.

Python ▼

```
def rand5():  
    result = 7 # arbitrarily large  
    while result > 5:  
        result = rand7()  
    return result
```

So each integer 1,2,3,4, or 5 has a probability  $\frac{1}{7}$  of appearing at each roll.

## Complexity

Worst-case  $O(\infty)$  time (we might keep re-rolling forever) and  $O(1)$  space.

Note that if we weren't worried about the potential space cost (nor the potential stack overflow) of recursion, we could use an arguably-more-readable recursive approach with  $O(\infty)$  space cost:

Python ▼

```
def rand5():  
    roll = rand7()  
    return roll if roll <= 5 else rand5()
```

## Bonus

This kind of math is generally outside the scope of a coding interview, but: if you know a bit of number theory you can *prove* that there exists no solution which is guaranteed to terminate.

Hint: it follows from the fundamental theorem of arithmetic.

## What We Learned

Making sure each possible result has *the same probability* is a big part of what makes this one tricky.

If you're ever struggling with the math to figure something like that out, don't be afraid to *just count*. As in, write out all the possible results from `rand7()`, and label each one with its final outcome for `rand5()`. Then count up how many ways there are to make each final outcome. If the counts aren't the same, the function isn't uniformly random.

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.