We take advantage of the fact that the recursive function `move_stack` is guaranteed to move `n` disks from `start` to `end` while obeying the rules of Towers of Hanoi. The only thing that remains is to make sure that we have set up the playing board to make that possible.

Since we move a disk to end rod, we run the risk of `move_stack` doing an improper move (big disk on top of small disk). But since we're moving the biggest disk possible, nothing in the `n-1` disks above that is bigger. Therefore, even though we do not explicitly state the Towers of Hanoi constraints, we can still carry out the correct steps.

Video walkthrough: https://youtu.be/VwynGQiCTFM (https://youtu.be/VwynGQiCTFM)

Extra questions

> Extra questions are not worth extra credit and are entirely optional. They are designed to challenge you to think creatively!

Q6: Anonymous factorial

The recursive factorial function can be written as a single expression by using a conditional expression (http://docs.python.org/py3k/reference/expressions.html#conditional-expressions).

```
>>> fact = lambda n: 1 if n == 1 else mul(n, fact(sub(n, 1)))
>>> fact(5)
120
```

However, this implementation relies on the fact (no pun intended) that `fact` has a name, to which we refer in the body of `fact`. To write a recursive function, we have always given it a name using a `def` or assignment statement so that we can refer to the function within its own body. In this question, your job is to define fact recursively without giving it a name!

Write an expression that computes `n` factorial using only call expressions, conditional expressions, and lambda expressions (no assignment or def statements). *Note in particular that you are not allowed to use `make_anonymous_factorial` in your return expression.* The `sub` and `mul` functions from the `operator` module are the only built-in functions required to solve this problem:

```
from operator import sub, mul

def make_anonymous_factorial():
    """Return the value of an expression that computes factorial.

    >>> make_anonymous_factorial()(5)
    120
    >>> from construct_check import check
    >>> check(HW_SOURCE_FILE, 'make_anonymous_factorial', ['Assign', 'AugAssign', 'FunctionDef
    True
    """
    return (lambda f: lambda k: f(f, k))(lambda f, k: k if k == 1 else mul(k, f(f, sub(k, 1))))
    # Alternate solution:
    #   return (lambda f: f(f))(lambda f: lambda x: 1 if x == 0 else x * f(f)(x - 1))
```

Use Ok to test your code: