This is a Python programming assignment. You will edit your ongoing copy of `qc.py` and submit it for grading. The grader will `import qc` and then run their own testing code against it. Probably the grader will also inspect your code.

A. Write a function according to the following specification.

```
def continuedFraction(n, m, x0):
    '''x0 is a float in [0, 1). Tries probing depths j = 0, 1, 2, ... until
    the resulting rational approximation x0 ~ c / d satisfies either d >= m or
    |x0 - c / d| <= 1 / 2^(n + 1). Returns a pair (c, d) with gcd(c, d) = 1.'''
```

If you don't know how to get started, then consider my approach: I first wrote a recursive helper function `fraction(x0, j)` that returns a pair `(c, d)` representing c/d in lowest terms. Then I wrote `continuedFraction` to call `fraction` in a loop.

B. Improve your `shorTest` function. It should still take `n` and `m` as arguments, randomly pick an appropriate k, and make the f function and the corresponding F gate. But now, instead of merely running the core subroutine once, it should execute the entire period-finding algorithm: nested loops, finding d, d', $\text{lcm}(d, d')$, and all that. It should also compute the period p in a second way: by brute force. It should print both versions of p, so that the user can judge whether the algorithm is working correctly.

By the way, here are my favorite combinations of n and m.

- For $n = 5$, the largest m that can be handled is $m = 5$.
- For $n = 6$, the largest m is $m = 8$, but $m = 7$ is more interesting.
- For $n = 7$, the largest m is $m = 11$. My computer takes about a minute to find the period. I also have an optimized version of `shor` called `shorEfficient`, such that `shorTest` takes about a second. You are not required to write `shorEfficient`, but you might want to contemplate how you would. Hint: `shorEfficient` takes as input f instead of F.
- For $n = 8$, the largest m is $m = 15$. After a few minutes in the core subroutine, my operating system kills Python for using too much RAM. But you might have more RAM than I do.

Shameful disclaimer: Occasionally, my Shor p is a multiple of my brute-force p. I'm not sure why. I've seen it happen only in the `shorEfficient` version of `shorTest`. It might be because our numbers are so small that some of the approximations are off. Or maybe there is a bug in my code or an error deep in the math. Anyway, e-mail me know if you experience the same phenomenon: `shorTest` almost always finds the true p, but occasionally finds a multiple.