This is project work due at the end of the term.

A. In qUtilities.py, implement the following function.

```
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.

I haven't bothered to write a test. If you want to write a test (not a bad idea), the approximations for $1/\pi$ are $1/3, 7/22, 106/333, \ldots$. The approximations for a rational number of the form $1/q$ should just be $1/q, 1/q, 1/q, \ldots$. The approximations for 0 should just be $0/1, 0/1, 0/1, \ldots$.

B. In qAlgorithms.py, improve your `shorTest` function. It shouldn't print b. Instead it should execute the entire period-finding algorithm outlined in periodsFactors.pdf: nested loops, continued fractions, finding d, d', $\operatorname{lcm}(d, d')$, and all that. It should also compute the period p in the brute-force way, by simply raising k to higher and higher powers until 1 is obtained. (In an actual period-finding problem, this brute-force approach is too costly. That's the point of Shor's algorithm. But we're doing only small instances of the problem, so brute force is doable.) It should pass the test if the two versions of p agree, and it should fail the test (and print useful information) if not.