Exercise 1.28. One variant of the Fermat test that cannot be fooled is called the *Miller-Rabin test* (Miller 1976; Rabin 1980). This starts from an alternate form of Fermat's Little Theorem, which states that if n is a prime number and a is any positive integer less than n, then a raised to the $(n-1)$st power is congruent to 1 modulo n. To test the primality of a number n by the Miller-Rabin test, we pick a random number $a < n$ and raise a to the $(n-1)$st power modulo n using the `expmod` procedure. However, whenever we perform the squaring step in `expmod`, we check to see if we have discovered a "nontrivial square root of 1 modulo n," that is, a number not equal to 1 or $n-1$ whose square is equal to 1 modulo n. It is possible to prove that if such a nontrivial square root of 1 exists, then n is not prime. It is also possible to prove that if n is an odd number that is not prime, then, for at least half the numbers $a < n$, computing a^{n-1} in this way will reveal a nontrivial square root of 1 modulo n. (This is why the Miller-Rabin test cannot be fooled.) Modify the `expmod` procedure to signal if it discovers a nontrivial square root of 1, and use this to implement the Miller-Rabin test with a procedure analogous to `fermat-test`. Check your procedure by testing various known primes and non-primes. Hint: One convenient way to make `expmod` signal is to have it return 0.

Answer.

```
(define (expmod base exp m)
  (cond ((= exp 0) 1)
        ((even? exp)
         (if (nontri-sqrt? base m)
             0
             (remainder (square (expmod base (/ exp 2) m))
                        m)))
        (else
          (remainder (* base (expmod base (- exp 1) m))
                     m))))

(define (nontri-sqrt? a m)
  (define neq
    (lambda (x y) (not (= x y))))
  (and (and (neq a 1)
            (neq a (- m 1)))
       (= (remainder (square a)
                     m)
          1)))

(define (miller-rabin-test n)
  (define (try-it a)
    (= (expmod a (- n 1) n) 1))
  (try-it (+ (random (- n 1)) 1)))

(define (fast-prime? n times)
  (cond ((= times 0) true)
        ((miller-rabin-test n) (fast-prime? n (- times 1)))
        (else false)))
```