CS 3430: Python & Perl
Assignment 1: Using Fermat's Little Theorem to Compute Primes

Vladimir Kulyukin
Department of Computer Science
Utah State University

0. Learning Objectives

1. Py Functions
2. PL Subroutines
3. Procedural Programming in Py & PL
4. Recursive Definitions
5. Fermat's Little Theorem

1. Fermat's Little Theorem

Fermat's Little Theorem is a beautiful result from number theory that states that if ***n*** is a prime number and ***a*** is a positive number less than ***n***, then ***a*** raised to the power of ***n*** is congruent to ***a*** modulo ***n***. Recall that two numbers are congruent modulo ***n*** if they have the same remainder when divided by ***n***.

To implement the test for primality that uses Fermat's Little Theorem, we need a procedure that computes the exponential of a number modulo ***n***. We can use the following procedure, expressed in pseudocode, that computes **b^e modulo m**.

```
expmod(b, e, m) {
  if (e == 0) {
        return 1;
  }
  else if ( is_even(e) ) {
        expm = expmod(b, e/2, m);
        return remainder(expm*expm, m);
  }
  else {
        return remainder(b * expmod(b, e-1, m), m);
  }
}
```

Thus, **expmod(2, 3, 2) = 2^3 modulo 2 = 0; expmod(2, 3, 3) = 2^3 modulo 3 = 2**; **expmod(2, 3, 5) = 2^3 modulo 5 = 3**, etc.

2. Fermat's Test

The Fermat test of whether a given number ***n*** is prime is done by choosing a number ***a*** from **[2, *n*-1]** and checking if the remainder of the ***n***-th power of ***a*** is equal to ***a*** itself. In other words:

```
fermat_test(n) {
  a = random number from [2, n-1];
 return expmod(a, n, n) == a;
}
```

Now we can define a procedure **is_fermat_prime** that uses **fermat_test** a specific number of times to test if a number is prime. Obviously, the more we perform **fermat_test**, the more confidence we have that the number is prime. Hence, the second argument is specifies the number of times **fermat_test** is executed. Here is the pseudocode.

```
is_fermat_prime(n, num_times) {
  if ( num_times == 0 ) {
     return True;
  }
 else if ( fermat_test(n) ) {
     return is_fermat_prime(n, n_times-1);
 }
 else {
    return False;
 }
}
```

3. Using Fermat's Test to Compute Sums of Primes

Implement a function, both in Python & Perl, called **sum_of_fermat_primes(n)** that uses **is_fermat_prime** test to compute the sum of primes less than or equal to n. Thus, **sum_of_fermat primes(1000)** computes the sum of the primes in the range **[0, 1000]**.

4. What to Submit

Save your Py solution in **sum_of_fermat_primes.py** and your PL solution in **sum_of_fermat_primes.pl** and submit both files in Canvas.

5. Hints & Suggestions

In Py, you can use the function **random.randint(a, b)** to generate random numbers in **[a, b]**. Place the import random at the beginning of your program to import the package where this function is defined.

In PL, you can implement the same functionality with the native **rand()** function as follows:

```
sub randint {
        my ($a, $b) = @_;
        return int($a + rand($b-$a+1));
}
```

5.1 Python Blueprint

You can use the following function stubs to structure your Py solution to the problem.

```
#!/usr/bin/python

import random

def is_even(n): return n % 2 == 0

def square(n): return n**2

def expmod(b, e, m):
    ## your code

def fermat_test(n):
     ## your code

def is_fermat_prime(n, ntimes):
    ## your code

## This is the test you can use to test your code
def sum_of_fermat_primes(n):
    sum = 0
    for i in xrange(n+1):
        if is_fermat_prime(i, 70):
            sum += i
    return sum

print sum_of_fermat_primes(10)
print sum_of_fermat_primes(20)
print sum_of_fermat_primes(560)
print sum_of_fermat_primes(570)
```

5.2 Perl Blueprint

You can use the following subroutine stubs to structure your PL solution to the problem.

```
#!/usr/bin/perl
use strict;
use warnings;

sub randint {
   my ($a, $b) = @_;
   return int($a + rand($b-$a+1));
}

sub is_even { return $_[0] % 2 == 0; }

sub square { return $_[0]**2; }

sub expmod {
   my ($b, $e, $m) = @_;
   ## your code is here
}

sub fermat_test {
   my $n = $_[0];
   ## your code is here
}

sub is_fermat_prime {
   my ($n, $times) = @_;
   ## your code is here
}

## you can use this subroutine to test your code
sub sum_of_fermat_primes {
   my $n = $_[0];
   my $sum = 0;
   for(my $i = 2; $i <= $n; $i++) {
          if ( is_fermat_prime($i, 70) ) {
             $sum += $i;
          }
   }
   return $sum;
}

print 'my sum   =', sum_of_fermat_primes(10), "\n";
print 'my sum   =', sum_of_fermat_primes(20), "\n";
print 'my sum   =', sum_of_fermat_primes(560), "\n";
print 'my sum   =', sum_of_fermat_primes(570), "\n";
```