CS 3430: Python & Perl
Assignment 2: Mining Carmichael Numbers & Investigating their Relationship with Fermat's Little Theorem

Vladimir Kulyukin
Department of Computer Science
Utah State University

0. Learning Objectives

1. Procedural Programming in Py & PL
2. Defining Iterative Py Functions & PL Subroutines with FOR Loops
3. Boolean values in Py & PL
4. IF-THEN-ELSE Control Structures in Py & PL

1. Carmichael Numbers

The Carmichael numbers are named after Robert Daniel Charmichael, (https://en.wikipedia.org/wiki/Robert_Daniel_Carmichael) an American mathematician, who discovered them. Carmichael numbers play a prominent role in cryptography and digital coinage, so it is worth your time to learn about them.

The Carmichael numbers can be computed by the criterion proved by Alwin Reinhold Korselt (https://en.wikipedia.org/wiki/Alwin_Korselt), a German mathematician. This criterion, which is now known as the Korselt Criterion, is as follows.

A positive composite integer *n* is a Carmichael number if and only if *n* is square-free and for all prime divisors *p* of *n*, it is true and (*p*-1) divides (*n*-1), i.e., (*p*-1) | (*n*-1).

Let us dissect the Korselt Criterion one definition at a time. A number is composite if it is not prime. To define square-free integers, we need to define perfect squares first. An integer is a perfect square if it is the square of another integer. For example, 1 is a perfect square, because 1 = 1x1; 4 is a perfect square, because 4 = 2x2; 9 is a perfect square, because 9 = 3x3, etc. An integer is square-free if it is divisible by no perfect square other than 1. Some examples of square-free integers are 1, 2, 3, 5, 6, 7, 10, 11, etc. For instance, 6 is square-free, because it is divisible by 1 but is not divisible by 4. 10 is square-free, because it is divisible by 1 but inot divisible by 4 or 9. On the other hand, 18 is not square-free, because it is divisible by 9, which is a perfect square.

You can use the following PL subroutine to test if an integer is square-free. Remember that in PL, 0 is false and anything not equal to 0 is true. The Py function is very similar, except that Py has True and False as genuine Boolean types. Also, remember to put the **uses POSIX** pragma to use the PL subroutine ceil that computes the ceiling of the number. When you implement this function in Py, remember to **import math** so that you can use **math.ceil** for the same purposes.

```
use strict;
use warnings;
use POSIX;

sub is_square_free {
   my $n = $_[0];
   if ($n == 1 || $n == -1) {
          return 1;
   }
   for(my $d = 2; $d <= ceil(sqrt($n)) + 1; $d++) {
          if ( $n % ($d*$d) == 0 ) {
             return 0;
          }
   }
   return 1;
}
```

2. Mining Carmichael Numbers

We will mine the Carmichael Numbers in three stages.

Stage 1: Write the Py function and PL subroutine **is_carmichael(n)** that returns true if **n** is a Carmichael number. Use your implementation to mine and print in the command window or the Py shell the first 34 Carmichael numbers. State the list explicitly in the first comment in your Py or PL source code that you will submit.

Stage 2: Apply Fermat's test from for primality you implemented in HW 1 to each of the Carmichael numbers you mined in stage 1. State in the second comment in your Py or PL source which of the 34 numbers passed Fermat's test

Part 3: Write a standard **is_prime(n**) in Py and PL that tests if **n** is prime by checking if it has any divisor other than 1 and itself. Apply this test to each of the 34 Carmichael numbers. Which ones pass this test? State your observation in the third comment either in your Py or PL source code.

When you are testing your code, here are the first three Carmichael numbers: 561, 1105, 1729.

4. What to Submit

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

Happy Hacking! One last quick note for the road. The Carmichael numbers are pretty difficult to compute in that some, especially those larger than 10,000, require quite a few CPU cycles to find. So, write your PL and Py programs and leave them running over night, take a hike in the mountains, or visit with a friend, etc. Hopefully, by the time you get back to your computer, there will be a beautiful list of 34 Carmichael numbers nicely printed in your command window.