About Lab 3

Lab 3 has three parts:

- A probabilistic estimate of pi (monte.py)
- A program that inputs n, prints the first n primes, and counts the number of twin primes in that list. (primes.py)
- Two graphical programs where you draw bricks to form an ell or a pyramid.

Program monte.py

You get a number n from the user that specifies a number of "iterations". For each iteration you get a random x and a random y, each a float between -1 and 1. Let hits be the number of iterations where x*x+y*y<= 1. Then 4*hits/n is an estimate of pi. The prelab goes into why; the idea is that hits/n is an estimate of the ratio of the area of the circle x*x+y*y<= 1 to the area of the square -1<=x,y<=1 and this ratio is exactly pi/4.

Program primes.py

In this program you get a number n from the user and print the first n primes; we have done that much in class, though here you want to print them on one line. A nice solution would be to print them in a table, with a preset number of columns (maybe 10 or 15). What is new about this program is that you also need to count the number of *twin primes*; prime numbers that differ by 2, such as 3 and 5, 5 and 7, 11 and 13. There is an unproven conjecture in Mathematics that there are infinitely many twin prime pairs.

If we had an isPrime(x) function it would be easy to find twin primes: just ask

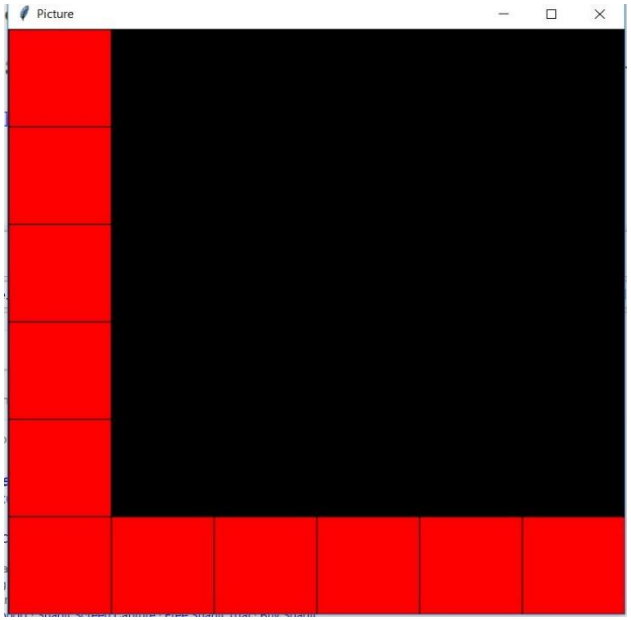
```
if isPrime(x) and isPrime(x+2):
```

That isn't how we want you to do this. You need to have a loop testing whether numbers are prime; we want you to modify that loop to remember if the previous odd number was prime; if it was and the current odd number is prime, you have found a pair of twin primes.

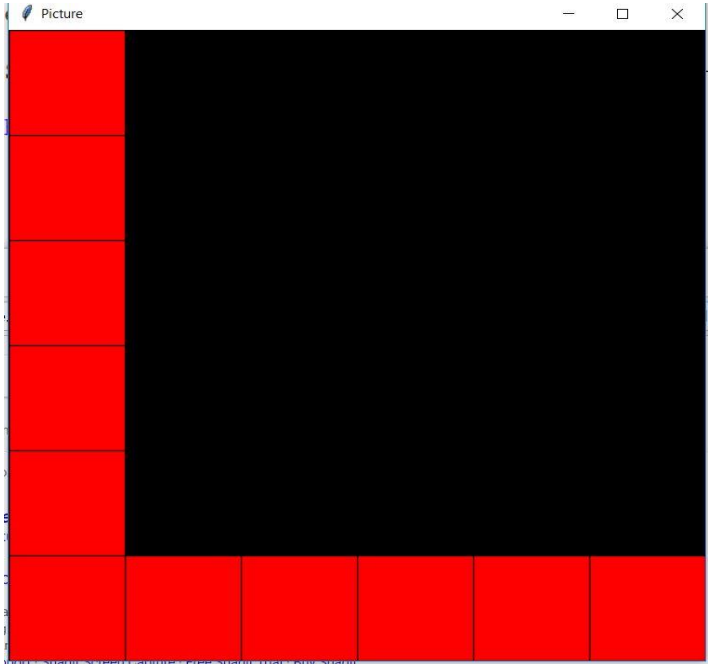
It helps to get the only even prime number, 2, out of the loop. If the user only asks for one prime number, just print 2, say there are no twin primes, and quit. For any larger number of primes you can print 2, start your prime count at 1, and only consider odd numbers. Keep a variable that says whether the previous odd number was prime. If you test an odd number and find it is prime and this previousWasPrime variable is True, you know you have found another twin prime pair. The program asks you to count twin primes, not print them out.

The Graphical Programs

There are two similar graphical programs to write. The first one draws an L-shaped figure out of 100x100 squares:



Remember that do draw a square we need its upper left corner. For the vertical column these are (0,0), (1,100), (0,200) and so forth



I'll leave it to you to figure out the coordinates for the horizontal row.

The last program draws a pyramid.

You build the pyramid row by row, starting at the bottom. Each row has one fewer brick than the row below it, so the height of the pyramid in bricks is the same as the number of bricks on the bottom row.

Your program should ask the user for two numbers:

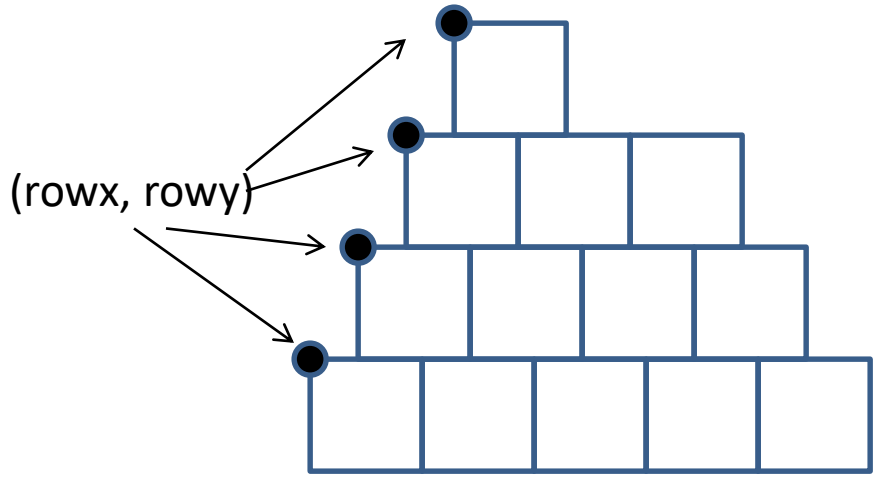
The number of bricks on the bottom row (or the height of the pyramid in bricks)
We'll call this n.

The width of the square canvas in pixels.
We'll call this width.

Then size=width//n is the size of one brick.

Your program should then open a canvas that is width x width.

Consider the following picture. (rowx, rowy) are the coordinates of the upper left corner of the leftmost box in each row.



If we knew rowx and rowy and the number of boxes in a row we could easily draw the row:

This loop draws one row:

```
x = rowx
y = rowy
for box in range(numInRow):
        canvas.drawSquareFill(x, y, size)
        x = x+size
```

After drawing one row, we need to update rowx, rowy and numInRow for the row above it. This is not hard; just look at the picture and remember that y coordinates *decrease* as you go up.

Note that the initial values of our variables for the bottom row are:

rowx = 0

rowy = width-size

numInRow = n