Name: ____________________________

Directions: **Work only on this sheet** (on both sides, if needed). DO NOT turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing. When appropriate, SHOW YOUR WORK.

1. (15) Fill in the blanks in the following version of our prime number finding program:

```
import sys,math,threading,Queue

class primefind(threading.Thread):
   n = int(sys.argv[1])
   prime = (n+1) * [1]
   lim = int(math.sqrt(n)) + 1
   _______________________________________
   for i in range(2,lim+1):
      _______________________________________
   def __init__(self):
      _______________________________________
   def run(self):
      nk = 0
      while True:
         try:
            k = primefind.nexti.get(block=False)
         except:
            break
         nk += 1
         if primefind.prime[k]:
            r = primefind.n / k
            for i in range(2,r+1): primefind.prime[i*k] = 0
      print ’this thread handled’,nk,’values of k’

def main():
   mythreads = []
   for i in range(int(sys.argv[2])):
      pf = primefind()
      mythreads.append(pf)
      pf.start()
   for pf in mythreads: pf.join()
   print reduce(lambda x,y: x+y, primefind.prime)-2,’primes’

if __name__ == ’__main__’: main()
```

2. (10) Sec. 2.4 of our PLN on iterators and generators is titled, “Multiple Iterators from the Same Generator.” Give an example in which we actually had such a situation, i.e. we had code in which several iterators from the same generator function would be in existence at the same time.

3. This question involves our **thrd** class for nonpreemptive threads.

(a) (15) We could add an analog of the Queue class but there would not be much point to it. Why not?

(b) (15) Give the line number in pp.17-19 that will be executed immediately after line 33 on p.14.

(c) (15) Suppose we are debugging the code on pp.14-15 of our iterators/generators PLN (the one that illustrates the capabilities of **thrd**). Give the PDB command to set a conditional breakpoint at line 70, p.17 (beginning of **thrd.do_pause()**), breaking only if the function is triggered by **a()** in pp.14-15. (The command set listed in our intro PLN if you need it.)

4. (15) In this problem, you will write a generator function **il()** that interleaves two iterators of the same length. For example:

```
>>> a = range(3)
>>> b = [-2,6,8]
>>> d = il(a,b)
>>> d.next()
0
>>> d.next()
-2
>>> d.next()
1
>>> d.next()
6
```

We’ll use the function **itertools.izip()**, which works like **zip()**, but inputs two iterators and outputs a third one. Fill in the blanks:

```
from itertools import *

def il(i,j):
   k = izip(i,j)
   for _______________________________________:
      ______________________________________________
      ______________________________________________
```

5. (15) Solve the mystery! We ran the Fibonacci number iterator from our PLN on iterators and generators, and though it ran correctly for a while, it got stuck on the number 8!

```
>>> from fib import *
>>> f = fibnum()
>>> f.next()
1
>>> f.next()
1
>>> f.next()
2
>>> f.next()
3
>>> f.next()
5
>>> f.next()
8
>>> # here we executed a one-line Python statement (not shown)
>>> f.next()
8
>>> f.next()
8
```

What was that one-line Python mystery statement?

Solutions:

1.

```
nexti = Queue.Queue()
nexti.put(i)
threading.Thread.__init__(self)
```

2. Two examples were from discussions section, concerning tree traversal and partitions of n.

3.a The main utility of Queue is that it automatically handles locks for us. But we don’t need locks in the case of **thrd**.

3.b 157

3.c

```
b 70, yv[0] == ’1’
```

4.

```
(u,v) in k
yield u
yield v
```

5.

```
f.next = lambda : 8
```