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.

1. Here is a list of values taken on by some discrete random variable X, along with their probabilities: $\{(1, \frac{1}{2}), (4, \frac{1}{4}), (5, \frac{1}{4})\}$.

(a) (15) This list is called the _____ of X.

(b) (15) Suppose this random variable were so important that the Python organization decided to add to the class **random** a function **randx()** that would generate random variates like X, i.e. having the same values and probabilities. The function's definition would begin as **def randx(self):** Write the rest of the function.

2. Consider our example program **MachRep1.py**.

(a) (15) Give a complete list of line numbers from this file whose statements could be the next one to execute after the statement on line 57.

(b) (20) Suppose we wish to print out the percentage of time during which both machines are up. Referring to line numbers, show code to add to achieve this.

3. (15) State the numerical values printed out below. **YOU MUST PROVIDE AN EXPLANATION.**

```
import random
r = random.Random(98765)
sumy = 0
sumy2 = 0
for rep in range(10000):
   x = 0
   y = 0
   while True:
      y += 1
      u = r.uniform(0.0,1.0)
      if u < 0.5: x += 1
      if x == 3: break
   sumy += y
   sumy2 += y*y
m1 = sumy/10000.0
m2 = sumy2/10000.0
print m1, m2-m1*m1
```

4. (20) The SimPy program below models a single machine. Up time and repair time are exponentially distributed with means 1.0 and 0.5, respectively. There is a continuing supply of jobs waiting to use the machine, i.e. when one job finishes, the next begins. When a job is interrupted by a breakdown, it resumes "where it left off" upon repair, with whatever time remaining that it had before. Fill in the two gaps (though not necessarily using the same number of lines). Use **cancel()** somewhere. (**NInts** is not used, but put in the correct code to maintain it anyway.)

```
 1        from SimPy.Simulation import *
 2        import SimPy.Simulation
 3        from random import Random,expovariate
 4
 5        import sys
 6
 7        class G:  # globals
 8           CurrentJob = None
 9           Rnd = Random(12345)
10           M = None  # our one machine
11
12        class Machine(Process):
13           def __init__(self):
14              Process.__init__(self)
15           def Run(self):
16              while 1:
17                 UpTime = G.Rnd.expovariate(Machine.UpRate)
18                 yield hold,self,UpTime
19                 CJ = G.CurrentJob
```

```
20                # start gap 1
21
22
23
24
25
26
27                # end gap 1
28
29        class Job(Process):
30           ServiceRate = None
31           NDone = 0
32           TotWait = 0.0
33           def __init__(self,ID,TimeLeft,NInts,OrigStart,LatestStart):
34              Process.__init__(self)
35              self.ID = ID
36              self.TimeLeft = TimeLeft  # work left for this job
37              self.NInts = NInts  # number of interruptions so far
38              # time this job originally started
39              self.OrigStart = OrigStart
40              # time the latest work period began for this job
41              self.LatestStart = LatestStart
42           def Run(self):
43              yield hold,self,self.TimeLeft
44              # job done
45              Job.NDone += 1
46              Job.TotWait += now() - self.OrigStart
47              # start the next job
48              # start gap 2
49
50              # end gap 2
51
52        def main():
53           Job.ServiceRate = float(sys.argv[1])
54           Machine.UpRate = float(sys.argv[2])
55           Machine.RepairRate = float(sys.argv[3])
56           initialize()
57           SrvTm = G.Rnd.expovariate(Job.ServiceRate)
58           G.CurrentJob = Job(0,SrvTm,0,0.0,0.0)
59           activate(G.CurrentJob,G.CurrentJob.Run(),delay=0.0)
60           G.M = Machine()
61           activate(G.M,G.M.Run(),delay=0.0)
62           MaxSimtime = float(sys.argv[4])
63           simulate(until=MaxSimtime)
64           print 'mean wait:', Job.TotWait/Job.NDone
65
66        if __name__ == '__main__': main()
```

Solutions:

1.a distribution

1.b

```
def randx(self):
   u = self.uniform(0,1)
   if u < 0.5: return 1
   elif u < 0.75: return 4
   else: return 5
```

2. First there are the lines at which a thread resumes after a **yield hold**: 59, 55 (or 53). Then there is line 52, which will be executed for the second thread after the first one does its first **yield hold**. Finally, there is line 87.

3. This simulates tossing a coin until we get three heads. The number of tosses needed has a negative binomial distribution with k = 3 and p = 0.5. The mean and variance, using the formulas in our notes, are thus $3/0.5 = 6$ and $3 \cdot \frac{0.5}{0.5^2}$.

4. This is worked out in our later PLN, at `http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf`.