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. (20) Our PLNs use the network timeout example several times. Consider the one that makes use of interrupts. Here is a portion of that code:

```
29    activate(G.TO,G.TO.Run())
30    # wait for ACK, but could be timeout
31    yield hold,self,G.Rnd.expovariate(1.0)
32    if self.interrupted():
33       self.NTimeOuts += 1
34    else: self.cancel(G.TO)
```

Show the line in **Simulation.py** (give the line number and the actual Python statement) that is executed just before line 32 is executed.

2. (20) Consider the end of a SimPy simulation. Assuming it was not triggered by some event such as a call to **stopSimulation()**, the end will occur when some event has a scheduled time past the specified simulation time limit. Add a single line of code to **Simulation.py** that prints out the scheduled time for that event.

3. (20) Write a Python function (full, working code) with heading

```
def triangle(self):
```

that generates random variables for the density which has value 2 - 2t on (0,1), 0 elsewhere. The function is to be a method within the **random** class. Use any method that works.

4. Consider the pseudocode in our PLN that generates random variables from a normal distribution with mean 0 and variance 1.

(a) (10) Implement the pseudocode as a Python generator. The function is to be a method within the **random** class. (You'll need the Python functions **log()**, **sqrt()**, **sin()** and **cos()** from the **math** module.)

(b) (10) Using the generator you write in (a), write code that finds the approximate value of the probability that a N(0,1) random variable is greater than 0.5.

5. Consider a single-server queue with buffer space for just one job. (So, at any given time there were be either 0, 1 or 2 jobs in the system.) Let A be interarrival time and S be service time. We wish to find a valid set of regeneration points.

(a) (10) For each of the candidate sets of regeneration points below, state which random variables, among A and S, must be exponentially distributed for the set of points to be valid. For instance, the answer "A no, S yes" would mean that we need S to be exponentially distributed but A can have any distribution. The answer "impossible" means that even with both A and S exponentially distributed, the candidate set of points is still not valid.

- (i) The times at which an arriving job encounters an empty system.
- (ii) The times at which an arriving job encounters a full system, i.e. finds there is a job in service and another in the buffer (and immediately leaves).
- (iii) The times at which a job completes and the buffer is empty.

(b) (10) Now suppose there is *reneging* in the system. A job waiting in the buffer will leave the system without service if it has waited R amount of time, where R is a random variable. Consider the possible set of regeneration points consisting of the times at which jobs renege. As in (a), answer which of A, S and R must have an exponential distribution in order for this set of points to be valid.

Solutions:

1. Line 340,

```
resultTuple = nextEvent._nextpoint.next()
```

2. Print **_t** after the **while** loop in **simulate()**.

3. The inverse transformation method is too unwieldly here, so use the acceptance/rejection method:

```
def triangle(self):
   while True:
     u1 = self.uniform(0,1)
     u2 = self.uniform(0,1)
     if (u2 <= (2-2*u1)/2: return u1
```

4.a The key point is to implement the static nature of **n** and **Y**. This is easily resolved by using the fact that local variables retain their values between calls in a generator function. The most elegant solution, though, dispenses with **n** entirely:

```
def n01(self):
   while True:
     u1 = self.uniform(0,1)
     u2 = self.uniform(0,1)
     R = math.sqrt(-2*math.log(u1))
     T = 2*math.pi*u2
     X = R*math.cos(T)
     Y = R*math.sin(T)
     yield X
     yield Y
```

4.b Generate, say, 10000 variates from your function in (a), and compute the proportion of those values that are greater than 0.5.

5.a.i A no, S no. This is basically the example in the PLN.

5.a.ii A no, S yes. Since a job has just arrived, time starts over from the point of view of the arrival process. However, the amount of time the job being served has been in service will affect how much service time remains unless S has an exponential distribution.

5.a.iii A yes, S no. We don't need exponential service times, for the same reason we didn't need exponential interarrival times in (ii). But when the service completed, a certain amount of time had passed since the last arrival, so unless we have exponential interarrival times, these points won't be valid regeneration points.

5.b A yes, S yes, R no. (Note that only one set of regeneration points is being considered here, not three.) The factors are similar to those above.