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. Consider the variable **_e.events**.

(a) (5) Suppose in executing a SimPy program, the first three invocations of **yield hold** had hold time (i.e. the third operand for **yield**) of 0.52, 0.18 and 0.98, all executed at time 0.0. Show the current value of **_e.events**, making sure to use the correct Python syntax.

(b) (10) Suppose SimPy were to be changed to use a calendar queue instead of is current approach, which essentially involves a linear linked list. State the numbers of all lines in the function **_e._post()** in **Simulation.py** which would be need to be changed. The structure **_e.events** will not change.

2. Consider the program **MachRep4.py** in our PLN unit on advanced SimPy.

(a) (10) For each of the class variables **UpRate** and **K** in **MachineClass**, state the effect of increasing the value of that variable (while holding the other constant) on the long-run proportion of up time. In each case, answer either (i) increase, (ii) decrease or (iii) could either increase or decrease.

(b) (10) There is an error in **main()**. Identify it.

(c) (5) Suppose we were to have another error in **main()**, in the form of inclusion of a **yield** statement, say inserted after the call to **initialize()**. Give a complete list of the numbers of the lines which would execute.

3. (10) Consider the program **Bus5.py** in our PLN unit on analysis of simulation output. Suppose that in addition to estimating E(W) and P(W > 6.2), we wish to also estimate P(W > 10 | W > 6.2). Show what code to add in order to compute and print out this estimate and a margin of error for it. **Be very clear as to where your new code is to be inserted.**

4. Write two functions for random number generation, both using the inverse transformation method. Assume they will be added to the Python **random** module.

(a) (10) A function **rand05(self)**, for generating variates from the density $0.5t^{-0.5}$ on (0,1).

(b) (10) A function **randpois(self,mu)**, for generating variates from a Poisson distribution with mean μ.

5. (10) Consider the program **QoS.py** in our PLN unit on advanced SimPy. Suppose a line

```
G.Mon = Monitor()
```

is added to **main()**, with a corresponding line in **G**. Add code which uses the monitor to collect data so that we will later be able to form a confidence interval for the mean number of video packets in the queue for the channel, via the regenerative method. **Be very clear as to where your new code is to be inserted.**

6. (10) Write a function **regen(self)** to be added to the **Monitor** class, for use in computing regenerative confidence intervals. It will return a tuple **(cntr,rad)**, consisting of the center and radius of the confidence interval. It is assumed that the function **observe()** is called at regeneration points.

7. (10) Suppose we are simulating an M/M/1 queue. This means there is a single server, and that service and interarrival times have the distributions U(0,1) and U(c-0.5,c+0.5), respectively. The question at hand is whether to scan the event list from the front or the back. Give precise conditions on c under which it is better to start from the back. (If you cannot give precise conditions, then give vague ones.) Hint: You may find our formula $Var(R) = E(R^2) - [E(R)[^2$ useful.

Solutions:

1.a

```
{0.52:[iterator address], 0.18:[iterator address], {0.98:[iterator address]}
```

1.b The data structure **_e.timestamps** will no longer be a simple linked list. Thus line 200 must be changed. No other changes are necessary.

2.a If **UpRate** is increased, the machine goes down faster, hence decreased up time. If **K** is increased, the repairperson is summoned earlier, thus increased up time.

2.b In line 73, 2 should be **MachineClass.R**.

2.c Having a **yield** in **main()** makes the latter a generator. Thus calling it returns an iterator rather than resulting in the function executing. The only line to execute would be

```
if __name__ == '__main__':  main()
```

3.a After line 21, insert

```
gt70 = 0
```

Replace line 26 by

```
if wait > 6.2:
   gt62 += 1
   if wait > 7:
      gt70 += 1
```

At the end of **main()**, insert

```
p7062 = gt70/float(gt62)
print 'the estimated value of P(W > 7.0 | W > 6.2) is', p7062
me = 1.96*sqrt(p7062*(1-p7062)/gt62)
print 'its margin of error is', me
```

Note in particular that one divides by **gt62** instead of by **nreps**.

4.a Let H denote the corresponding cdf, which by integration we see is $t^{0.5}$ on (0,1). Then its inverse is $G(u) = u^2$ on (0,1). Thus our function is

```
def rand05(self):
   u = self.uniform(0,1)
   return u*u
```

4.b Use the material in the section titled "The Inverse Transformation Method" in the section titled "Generating Random Numbers from Discrete Distributions" in our PLN on random number generators.

```
from math import exp

def randpois(self,mu):
   lamb = 1.0/mu
   u = self.uniform(0,1)
   q = exp(-lamb)
   k = 0
   while True:
      if u <= q: return k
      k += 1
      q *= lamb/k
```

5. We cannot use **PerSmp** here, as it is entirely unsuited for the regenerative method.

First we must choose a set of regeneration points. The simplest set would consist of the times at which the queue length becomes zero. Doing the same thing for a queue length of one would not work, due to the nonexponential nature of the data packet lengths, though it would work if we imposed the additional condition that there is currently no data packet in the system. Then, during each regeneration cycle we would have code to keep track of the time-integrated queue length, **TimeInt**. Up hitting a regeneration point, we would call

```
G.Mon.observe(TimeInt)
```

6. The monitor will consist of a list of two-element lists $[a, b]$, where **a** is the simulated time at which **observe()** is called and **b** is the argument in that call. Then

- the ith value of **b** is C_i
- the difference between the ith and (i-1)st values of **a** is D_i
- n is **len(self)**

Thus the code is

```
from math import sqrt

def regen(self):
   csum = 0
   csum2 = 0
   dsum = 0
   dsum2 = 0
   cdsum = 0
   n = len(self)
   for i in range(n):
      si = self[i]
      ci = si[1]  # C_i
      csum += ci
      csum2 += ci**2
      # D_i
      if i == 0:  di = si[0]
      else:  di = si[0] - self[i-1][0]
      dsum += di
      dsum2 += di**2
      cdsum += ci * di
   cbar = csum/n
   dbar = dsum/n
   gammahat = cbar/dbar
   sx = csum2/n - cbar**2
   sx += gammahat**2*(dsum2/n - dbar**2)
   sx -= 2*gammahat*(cdsum/n - cbar*dbar)
   return (gammahat, 1.96*sqrt(sx/n)/dbar)
```

7. Since there will be as many services as arrivals, the distribution of hold time H is a mixture of two uniforms with range 1, one with mean 1 and the other with mean c, with 0.5 weighting on each. So,
For the service times S, we have

$$E(S^2) = Var(S) + [E(S)]^2 = \frac{1}{12} + 0.5^2 = \frac{1}{3} \tag{1}$$

Similarly, for the interarrival times I, we have $E(I^2) = \frac{1}{12} + c^2$.

$$\begin{aligned}
Var(H) &= E(H^2) - [E(H)]^2 && (2)\\
&= 0.5\left(\frac{1}{3}\right) + 0.5(\frac{1}{12} + c^2) - [0.5(0.5) + 0.5(c)]^2 && (3)\\
&= \frac{1}{6} + \frac{1}{24} + 0.5c^2 - 0.25(0.5 + c)^2 && (4)\\
&= \frac{10}{48} - \frac{1}{16} - 0.25c + 0.25c^2 && (5)\\
&= \frac{7}{48} - \frac{1}{4} \cdot c + \frac{1}{4} \cdot c^2 && (6)
\end{aligned}$$

Thus the cutoff point is found by setting this to 1 and solving for c. We get

$$c = \frac{12 + \sqrt{2112}}{24} \approx 2.42 \tag{7}$$

In other words, start at the back if $c > 2.42$.

Some people got partial credit for giving reasoned, intuitive explanations as to why if c is “very large,” we should start at the back.