Name: ____

Directions: MAKE SURE TO COPY YOUR ANSWERS TO A SEPARATE SHEET FOR SENDING ME AN ELECTRONIC COPY LATER.

1. (10) Consider the example in Sec. 4.6.1, Nonblocking Sockets. This problem will concern what would happen if we had forgotten to include line 34 in the server code.

Treat this code as a very simple "game." Suppose that: there are two clients, run by Person A and Person B; Person A starts playing at least several seconds before Person B; the players will only type chatacters when invited to do so by the prompt; Person A is planning to type 'a', then 'b', then quit the game, while Person B is planning to type 'x', 'y' and 'z', then quit.

Give the final value of **v**, output by the server.

2. (30) The function **findtwins(x)** below returns a Python list of all indices **i** for which **x[i] = x[i+1]** (excluding the last element of **x**, for which there is no right-hand neighbor). It is assumed that none of the (original) elements of **x** has the value **None**.

Example:

```
>>> x = [12, 5, 13, 13, 3, 4, 5, 5]
>>> findtwins(x)
[2, 6]
>>> findtwins([1,12,5,13,13,13,8,8,12])
[3, 4, 6]
```

Fill in the blanks:

```
def findtwins(x):
   nx = len(x)
   x.append(None)
   def compxii1(i):
      if x[i] == x[i+1]: return blank (a)
      else: return -1
   indcs = map(compxii1,blank (b) ))
   # indcs now consists of the found
   # indices and -1s
   del x[nx]
   return filter(blank (c) ,indcs)
```

3. (60) Below is SimPy code that simulates the operation of a disk drive.

Recall how such a system works: Data is stored in concentric rings called *tracks*. Each track is divided into *sectors*, and each read/write operation is done on a sector. Time needed to fulfill a disk access request consists first of a *seek*, in which the read/write head is moved to the desired track, then a *rotational delay* during which the desired sector rotates around to the head, and finally a *data transfer time* to process the sector. In the simple model here, we assume the latter is negligible. The time to go from the innermost to outermost track is **endtoendsk**, and the time for a full rotation is **onerot**.

We model the track number as a continuous variable ranging from 0 (innermost track) to 1 (outermost track), and the track number requested by a job is modeled as a random number between 0 and 1. Similar statements hold for the sector number.

Fill in the blanks:

```
import sys
import math
from SimPy.Simulation import *
from random import Random,expovariate,random

class gb:  # globals
   rnd = Random(12345)
   ddrproc = None
   arrvproc = None

class ddr(Process):
   def __init__(self,endtoendsk,onerot):
      Process.__init__(self)
      self.endtoendsk = endtoendsk
      self.onerot = onerot
      self.currtrack = 0.5
      # state is used for code logic and debugging
      self.state = 'resting' # not processing a request
      self.nrequestdone = 0
      self.queue = []
   def Run(self):
      onerot = self.onerot
      endtoendsk = self.endtoendsk
      while True:
         if self.queue == []:
            self.state = 'resting'
            blank (a)
         self.state = 'seeking'
         job =  blank (b)
         yield hold,self, blank (c)
         self.currtrack = job
         sector = gb.rnd.random()
         currangle = math.fmod(now(),onerot) / onerot
         tmp = sector - currangle
         if tmp > 0: rotdelay = tmp * self.onerot
         else: rotdelay = (1-currangle+sector)*self.onerot
         self.state = 'waiting rotation'
         yield hold,self,rotdelay
         blank (d)

class arrivals(Process):
   def __init__(self,arrvrate):
      Process.__init__(self)
      self.arrvrate = arrvrate
   def Run(self):
      while True:
         yield hold,self,gb.rnd.expovariate(self.arrvrate)
         track = gb.rnd.random()
         gb.ddrproc.queue.append(track)
         if gb.ddrproc.state == 'resting':
            blank (f)

def main():
   initialize()
   ees = float(sys.argv[1])
   oner = float(sys.argv[2])
   d = ddr(ees,oner)
   gb.ddrproc = d
   activate(d,d.Run())
   arrvrate = float(sys.argv[3])
   a = arrivals(arrvrate)
   gb.arrvproc = a
   activate(a,a.Run())
   maxsimtime = float(sys.argv[4])
   simulate(until=maxsimtime)
   print 'throughput:',d.nrequestdone/maxsimtime
```

Solutions:

1. Answer will depend somewhat on order of players' "moves." But any answer beginning with 'ax' and ending with 'z' is OK.

2.

```
# goes through the list x and returns all indices i for which
# x[i] = x[i+1]; elements in x are assumed to not be None

def findtwins(x):
   nx = len(x)
   x.append(None)
   def compxii1(i):
      if x[i] == x[i+1]: return i
      else: return -1
   indcs = map(compxii1,range(nx))
   del x[nx]
   return filter(lambda u:u >= 0,indcs)

x = [12, 5, 13, 13, 3, 4, 5, 5]
findtwins(x)
```

3.

```
# Disk.py

# usage

#   python Disk.py endtoendsk onerot arrvrate maxsimtime

import sys
import math
from SimPy.Simulation import *
from random import Random,expovariate,random

class gb:  # globals
   rnd = Random(12345)
   ddrproc = None
   arrvproc = None

# each instance of the ddr class will simulate one disk drive (but we
# will have only one here)

# current track is modeled as continuous value between 0 and 1

# endtoendsk is time to go from innermost to outermost track; onerot
# is time for one rotation; data transfer time assumed negligible here

class ddr(Process):
   def __init__(self,endtoendsk,onerot):
      Process.__init__(self)
      self.endtoendsk = endtoendsk
      self.onerot = onerot
      self.currtrack = 0.5
      # state is used for code logic and debugging
      self.state = 'resting'
      self.nrequestdone = 0
      self.queue = []
   def Run(self):
      onerot = self.onerot
      endtoendsk = self.endtoendsk
      while True:
         if self.queue == []:
            self.state = 'resting'
            yield passivate,self
         self.state = 'seeking'
         job = self.queue.pop(0)
         yield hold,self,abs(job-self.currtrack)*endtoendsk
         self.currtrack = job
         sector = gb.rnd.random()
         currangle = math.fmod(now(),onerot) / onerot
         tmp = sector - currangle
         if tmp > 0: rotdelay = tmp * self.onerot
         else: rotdelay = (1-currangle+sector)*self.onerot
         self.state = 'waiting rotation'
         yield hold,self,rotdelay
         self.nrequestdone += 1

class arrivals(Process):
   def __init__(self,arrvrate):
      Process.__init__(self)
      self.arrvrate = arrvrate
   def Run(self):
      while True:
         yield hold,self,gb.rnd.expovariate(self.arrvrate)
         track = gb.rnd.random()
         gb.ddrproc.queue.append(track)
         if gb.ddrproc.state == 'resting':
            reactivate(gb.ddrproc)

def main():
   initialize()
   ees = float(sys.argv[1])
   oner = float(sys.argv[2])
   d = ddr(ees,oner)
   gb.ddrproc = d
   activate(d,d.Run())
   arrvrate = float(sys.argv[3])
   a = arrivals(arrvrate)
   gb.arrvproc = a
   activate(a,a.Run())
   maxsimtime = float(sys.argv[4])
   simulate(until=maxsimtime)
   print 'throughput:',d.nrequestdone/maxsimtime
```