```
from single_server import single_basic, single_queueing, single_queueing_B
from multi_server import *
from multi_server_b import *
import sys


total = len(sys.argv)
if total ≠ 2:
        print 'Usage: python', str(sys.argv[0]), '<problem_number>'
        exit(0)

problem = str(sys.argv[1])

if problem ≡ '1':
        print 'Problem 1:'
        single_basic()

elif problem ≡ '2':
        print 'Problem 2:'
        my_rate = .2
        while my_rate ≤ .951:
                single_queueing(my_rate, 200000, 1)
                my_rate = my_rate + .15

elif problem ≡ '3':
        print 'Problem 3:'
        my_B = 10
        while my_B ≤ 50:
                single_queueing_B(.95, my_B, 200000, 1)
                my_B = my_B + 5

elif problem ≡ '4a':
        #1: 3(lambda) arrival, 3 servers rate mu
        my_lambda = .1
        while my_lambda ≤ .91:
                single_queueing_3_server(my_lambda, 10000, 200000, 1)
                my_lambda = my_lambda + .1

elif problem ≡ '4b':
        #2: 3(lambda) arrival, 1 server rate 3*mu
        my_lambda = .1
        while my_lambda ≤ .91:
                single_queueing(3*my_lambda, 200000, 3*1)
                my_lambda = my_lambda + .1

elif problem ≡ '4c':
        #3: 3 queue lambda arrival, 1 server rate 3*mu (priority)
        my_lambda = .1
        while my_lambda ≤ .91:
                three_queueing_single_server_priority(my_lambda, 10000, 200000, 3*1)
                my_lambda = my_lambda + .1

elif problem ≡ '4d':
        #4: 3 queue lambda arrival, 1 server rate 3*mu (Longest Queue First)
        my_lambda = .1
        while my_lambda ≤ .91:
                three_queueing_single_server_LQF(my_lambda, 10000, 200000, 3*1)
                my_lambda = my_lambda + .1

elif problem ≡ '5a':
        #1: 3(lambda) arrival, 3 servers rate mu
        my_B = 10
        while my_B ≤ 50:
                single_queueing_3_server_b(3*0.9, my_B, 200000, 1)
                my_B = my_B + 5

elif problem ≡ '5b':
        #2: 3(lambda) arrival, 1 server rate 3*mu
        my_B = 10
        while my_B ≤ 50:
                single_queueing_B(3*0.9, my_B, 200000, 3*1)
                my_B = my_B + 5

elif problem ≡ '5c':
        #3: 3 queue lambda arrival, 1 server rate 3*mu (priority)
        my_B = 10
        while my_B ≤ 50:
                three_queueing_single_server_priority_b(0.9, my_B, 200000, 3*1)
                my_B = my_B + 5

elif problem ≡ '5d':
        #4: 3 queue lambda arrival, 1 server rate 3*mu (Longest Queue First)
```



```
        my_B = 10
        while my_B ≤ 50:
                three_queueing_single_server_LQF_b(0.9, my_B, 200000, 3*1)
                my_B = my_B + 5

elif problem ≡ 'custom':
        #Use this space to construct custom queries for testing.
        three_queueing_single_server_LQF_b(0.9, 10, 200000, 3*1)

else:
        print 'Invalid Problem Specified. Exiting.'
        print 'Usage: python', str(sys.argv[0]), '<problem_number>'
```

```
import math
import random

Seed = 10
random.seed( Seed )

def expTime(rate):
     #Individual values in exponential distribution can be represented via:
     #x = ln(1-R)/(-lambda)
     #Can do this because of inversion method: this is arrived at via
     #integrating and inverting the exponential distribution:
     #F(y) = 1 - e^(-(lambda)y)
     #
     #Can use this function for both arrival rate and service rate. Mu = 1 for
     #entire project, so for service rate, we will always call expTime(1).
     return math.log(1.0 - random.uniform(1, 10000000)) / -rate

class Packet:
    #Packet Class. Returns arrival time, wait time, service time
    arrival = -1.0
    serviceStart = -1.0
    service = -1.0

    def getTotalTime(self):
        return (self.serviceStart+self.service)-self.arrival
```

```
from Queue import *
import matplotlib.pyplot as plt
import numpy as np

from sim_definitions import expTime, Packet, Seed

def single_basic():
        B = 5
        N = 200000
        rate = 0.5
        mu = 1
        total = 0

        list = []
        for x in xrange(1, N):
                total = total + 1
                list.append(expTime(rate));
        hist, bins = np.histogram(list, bins=200, normed=True)
        width = 0.7 * (bins[1] - bins[0])
        center = (bins[:-1] + bins[1:]) / 2
        plt.bar(center, hist, align='center', width=width)
        plt.show()
        print 'Total:', total
        print 'Average Arrival Time:', sum(list)/N
        print 'Seed:', Seed

def single_queueing(rate, N, mu):
        currentTime = 0
        totalLeft = N
        q = Queue()
        lost = 0.0
        minWait = 9999999
        maxWait = 0
        serverBusy = 0

        list = []
        while totalLeft > 0:

                oldTime = currentTime
                nextArrival = currentTime + expTime(rate)
                nextDeparture = currentTime + expTime(mu)
                currentTime = min(nextArrival, nextDeparture)

                if nextArrival < nextDeparture ∧ ¬(q.full()):
                        totalLeft = totalLeft - 1
                        packet = Packet()
                        packet.arrival = currentTime
                        q.put(packet)
                if nextArrival ≥ nextDeparture ∧ ¬(q.empty()):
                        item = q.get()
                        currentWait = currentTime-item.arrival
                        if currentWait < minWait:
                                minWait = currentWait
                        elif currentWait > maxWait:
                                maxWait = currentWait
                        list.append(currentTime-item.arrival)
                if ¬(q.empty()):
                        serverBusy = serverBusy + (currentTime - oldTime)

        print 'Parameters - Seed:', Seed, 'Rate:', rate, 'Mu:', mu, 'Number:', N
        print 'End Time of Simulation:', currentTime
        print 'Wait Times - Average:', sum(list)/N,
        print 'Server Utilization: ', serverBusy/currentTime
        print 'Percentage lost: ', (lost/(N+lost))*100, '%'
        print ''

def single_queueing_B(rate, B, N, mu):

        currentTime = 0
        totalLeft = 0
        lost = 0.0
        ##Queue size plus the service position, so B+1
        q = Queue(maxsize=B+1)
        serverBusy = 0
        minWait = 9999999
        maxWait = 0

        list = []
        while totalLeft < N:

                oldTime = currentTime
                nextArrival = currentTime + expTime(rate)
```



```
                if q.empty():
                        currentTime = nextArrival
                        nextDeparture = nextArrival + 1
                else:
                        nextDeparture = currentTime + expTime(mu)
                        currentTime = min(nextArrival, nextDeparture)

                if nextArrival < nextDeparture:
                        if q.full():
                                lost = lost + 1.0
                        else:
                                packet = Packet()
                                packet.arrival = currentTime
                                q.put(packet)

                elif nextArrival ≥ nextDeparture:
                        item = q.get()
                        list.append(currentTime-item.arrival)
                        currentWait = currentTime-item.arrival
                        if currentWait < minWait:
                                minWait = currentWait
                        elif currentWait > maxWait:
                                maxWait = currentWait
                        totalLeft = totalLeft + 1

                if ¬(q.empty()):
                        serverBusy = serverBusy + (currentTime - oldTime)

        print 'Parameters - Seed:', Seed, 'Rate:', rate, 'Mu:', mu, 'Number:', N
        print 'End Time of Simulation:', currentTime
        print 'Wait Times - Average (B=', B, ')', sum(list)/N
        print 'Server Utilization: ', serverBusy/currentTime
        print 'Percentage lost: ', (lost/(N+lost))*100, '%'
        print ''
```