CA4011 – Simulation Assignment Patient Queue Simulator

*Name: Glen Devlin*

*Student no. 12524303*

*Email:* [*glen.devlin4@mail.dcu.ie*](mailto:glen.devlin4@mail.dcu.ie)

# DCU School of Computing  Assignment Submission

|  |  |
| --- | --- |
| **Student Name(s):** | **Glen Devlin** |
| **Student Number(s):** | **12524303** |
| **Programme:** | **BSc in Computer Applications** |
| **Project Title:** | **Simulation Assignment** |
| **Module code:** | **CA4011** |
| **Lecturer:** | **Liam Tuohey** |
| **Project Due Date:** | **15/3/19** |

|  |
| --- |
| **Declaration**  **I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.**  **I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.**  **This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at**[**http://www.library.dcu.ie/citing&refguide08.pdf**](https://websvc.dcu.ie/%22http:/www.library.dcu.ie/citing&refguide08.pdf/%22)**and/or recommended in the assignment guidelines.**  **I understand that I may be required to discuss with the module lecturer/s the contents of this submission.**  **I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.**  **Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

*Element A1*



*The source code for this program was made in python and is available in the appendix.*

Program Description

Single program for both multiplicative method and language random number generator. When using language random number generator, the parameters are the desired number of integers (N) and the upper range of integers (r). The lower range is always 0. When using the multiplicative congruential method, the parameters are the seed, a, b and mod.

The number of replications is also taken from user input. The program amasses a series of Chi-squared stats and out puts how many are in the good range 2√r.

Each test consisted of 100 repetitions.

Multiplicative Congruential Method

* seed = 1, a = 13, b = 0, mod = 31
  + 0% in good range, chi-squared stats were all in between 1 and 0
* seed = 0, a = 4, b = 1, mod = 9
  + 0% in good range
* seed = 11, a = 9, b = 5, mod = 12
  + 0% in good range

Uniform Random Number Generator

* N = 1000, r = 10.
  + 87% of chi-squared stats were in good range
* N = 1000, r 20
  + 84% of chi-squared stats were in good range
* N = 1000, r = 50
  + 86% of chi-squared stats were in good range

Conclusion

From these results it seems that the obvious conclusion is that the built-in language random generators perform far better to the multiplicative congruential method. The congruential method seems to repeat the same numbers the same amount of time while missing other numbers completely.

Element A2

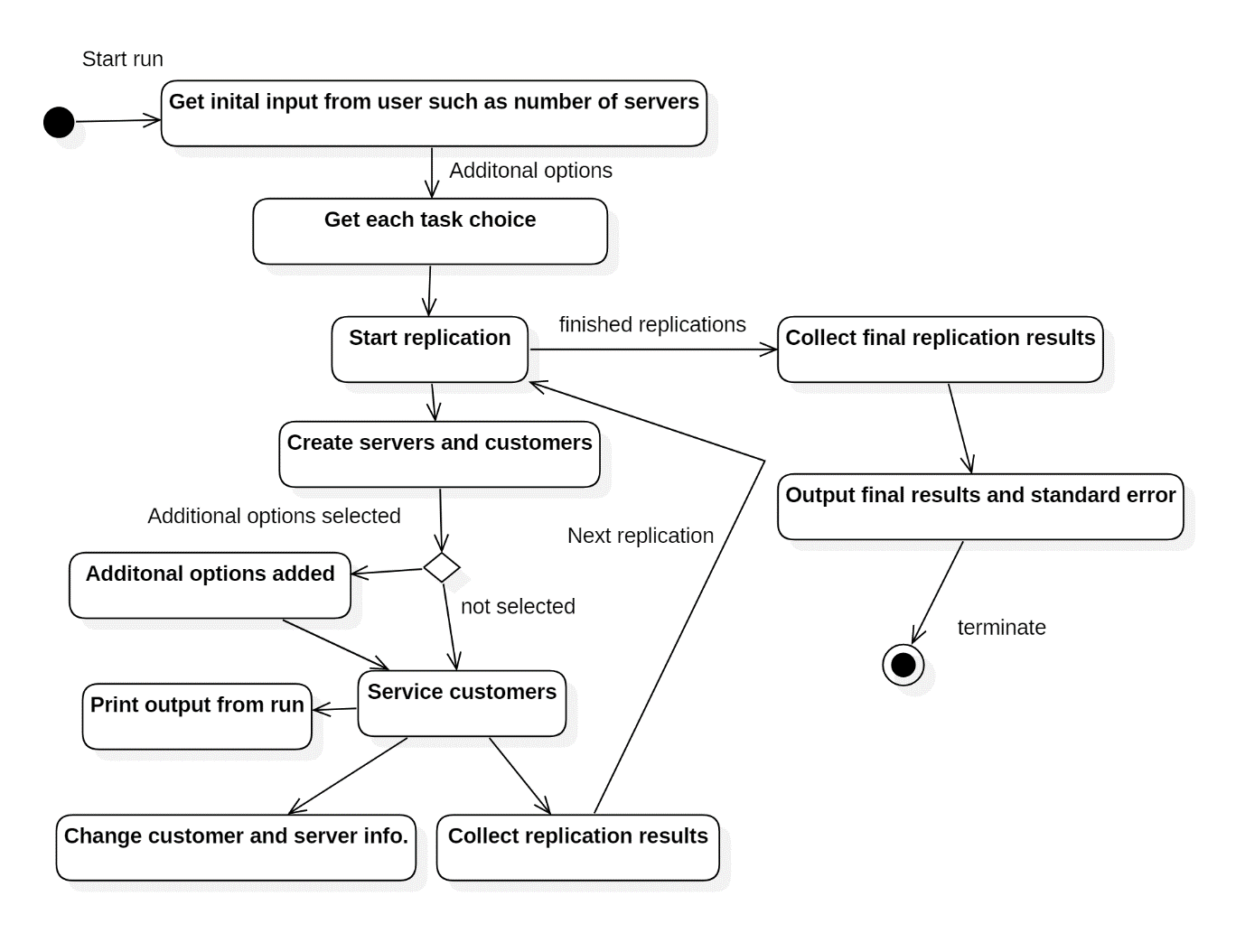
*The source code for this program was made in python and is available in the appendix.*

Program Description

The model for the simulation of the patient queueing system was made using python. The program takes various inputs from the user such as the number of servers, rate a server takes to see a patient and the arrival rate per hour. The program also takes a start time and end time. These are two point in which all initial appointments are scheduled. Appointments can still happen later than the end time, but it is because patients had to queue for a long time.

The program generates a list of customers based on the amount off arrival times created. Initially arrival times are distributed evenly with a bit of deviation. If the user chooses to use a Poisson distribution the times the patient times are changed to random times between the start and end point. There will still however be the same number.

Additional choices such as the ones for each task, like giving the patients gender can also be selected. Either together or by themselves. The program makes extensive use of the python time and timedelta classes. Output is printed at the end of every cycle and the final data from all replications is presented at the end of the program.



Note: For all of the below results the results are given as minutes unless otherwise specified. The standard deviation on times was 5 minutes and the start time and end time that were consistently used were 9.05 and 17.30 respectively. Every test was run with 100 replications.

*2.1 Initial Simulation*

(a) Deviated Regular Distribution

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 2 | Standard Error |
| Avg. time in system (W) | 15.2541 | 0.0097 | 34.6279 | 0.1231 | 152.001 | 0.2372 |
| Avg. time in Queue (Wq) | 0.2541 | 0.0097 | 19.6279 | 0.1231 | 122.001 | 0.2372 |
| Max time in system | 18.87 | 0.0895 | 55.87 | 0.2073 | 279.81 | 0.04503 |
| Max time in queue | 3.87 | 0.0895 | 40.87 | 0.2073 | 249.81 | 0.4503 |
| Avg. no. in system (L) | 1.5074 | 00.014 | 4.5304 | 0.0163 | 9.9258 | 0.0155 |
| Avg. no. in queue (Lq) | 0.0251 | 0.001 | 2.568 | 0.162 | 7.9668 | 0.0155 |
| Proportion of idle server time | 4 hours, 25 minutes | 0.8105 | 18 minutes | 0.4891 | 30 minutes | 0 |
| Last customer leaves | 17.42 |  | 18.22 |  | 22.04 |  |

(b) Poisson Distributed Times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 2 | Standard Error |
| Avg. time in system (W) | 22.115 | 0.3239 | 50.127 | 0.9755 | 164.4082 | 1.6549 |
| Avg. time in Queue (Wq) | 7.1155 | 0.3239 | 35.127 | 0.9755 | 134.4082 | 1.6549 |
| Max time in system | 44.88 | 1.1266 | 92.86 | 1.7621 | 303.52 | 1.7051 |
| Max time in queue | 29.88 | 1.1266 | 77.86 | 1.7621 | 273.52 | 1.7051 |
| Avg. no. in system (L) | 2.2061 | 0.327 | 6.4137 | 0.124 | 10.6758 | 0.109 |
| Avg. no. in queue (Lq) | 0.709 | 0.0323 | 4.4934 | 0.1239 | 8.7279 | 0.1087 |
| Proportion of idle server time | 4 hrs, 16 mins | 3.5237 | 44 min, 14 sec | 3.062 | 39 mins, 8 sec | 1.7385 |
| Last customer leaves | 17.44 |  | 18.41 |  | 22.19 |  |

Overall the Poisson distribution led to mostly higher times. The average time in the system was higher and the average time in the queue was far higher for the Poisson distributed times. The maximum time the patient spent in the system and queue was also higher. This is all understandable as patient arrival times could all clump together leading to a bottleneck situation.

The average amount of patients in the system and queue was only slightly higher for the Poisson times. This may be because it is the same number of patients over the same number of minutes in a day. The proportion of the time both servers were idle was stable as either way there was the same number of patients with the same amount of time spent on them. The time the last patient left was also quite consistent.

Queue Theory checks

λ = 6, µ = 4

W = 1.5, average time in system was 22 minutes which was close to a quarter of an hour

Wq = 0.37,

L = -3

Lq = 0.0617, close to my estimate of .709

P0 = 4.5, if 1 is equal to one hour, then I did have servers being idle for close to 4 and a half hours.

*2.2 Patients have Genders*

(a) Deviated Regular Distribution

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 23.1994 | 0.362 | 56.9581 | 1.1612 | 53.8818 | 1.138 |
| Avg. time in Queue (Wq) | 8.1994 | 0.362 | 41.9581 | 1.1612 | 33.8818 | 115.75 |
| Max time in system | 47.06 | 1.072 | 130.62 | 3.446 | 115.75 | 3.152 |
| Max time in queue | 32.06 | 1.072 | 115.62 | 3.446 | 95.75 | 3.152 |
| Avg. no. in system (L) | 2.2513 | 0.0336 | 6.8558 | 0.1191 | 4.6727 | 0.0789 |
| Avg. no. in queue (Lq) | 0.7952 | 0.0344 | 5.04 | 0.1231 | 2.9288 | 0.0844 |
| Proportion of idle server time | 4 hrs, 41 min | 2.113 | 7 hrs, 56 min | 58.8635 | 2 hrs, 2 min | 8.504 |
| Last customer leaves | 17.48 |  | 18.48 |  | 18.35 |  |

Having servers see specific genders gave patients a slightly higher time in system and queueing time. The max time in system and queue however rose quite a bit. This may be because there are now two chances for two different queues to bottleneck. For this same reason there is a higher number of patients in the system and in the queue at any given time.

(b) Poisson Distributed Times

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 30.6782 | 0.6148 | 69.5677 | 1.3045 | 66.6949 | 1.2401 |
| Avg. time in Queue (Wq) | 15.6782 | 0.6148 | 54.5677 | 1.3045 | 46.6949 | 1.2401 |
| Max time in system | 74.11 | 2.1044 | 163.3 | 3.9101 | 151.85 | 3.3178 |
| Max time in queue | 59.11 | 2.1044 | 148.3 | 3.9101 | 131.85 | 3.3178 |
| Avg. no. in system (L) | 2.9632 | 0.0572 | 8.2816 | 0.1353 | 5.7699 | 0.1062 |
| Avg. no. in queue (Lq) | 1.5139 | 0.0576 | 6.4933 | 0.1353 | 4.045 | 0.1051 |
| Proportion of idle server time | 4 hrs, 42 min | 4.7996 | 6 hrs, 24 mins | 51.9805 | 4 hrs, 35 min | 39.4263 |
| Last customer leaves | 17.58 |  | 19.08 |  | 18.50 |  |

Average time in system was higher, as was average time in queue. The max time in the system and the max time in the queue were much higher. Again, this is from bottlenecks happening. The number of patients in the system and queue was slightly higher but still quite stable. However, the time the servers were idle was much higher here. The time the last patient left was a little later.

*2.3 Servers Take Breaks*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 18.2253 | 0.1648 | 69.073 | 0.4416 | 56.5876 | 0.3762 |
| Avg. time in Queue (Wq) | 3.2253 | 0.1648 | 53.073 | 0.4416 | 36.5876 | 0.3762 |
| Max time in system | 34.79 | 0.6669 | 113.24 | 0.5523 | 87.97 | 0.3775 |
| Max time in queue | 19.79 | 0.6669 | 98.24 | 0.5523 | 67.97 | 0.3775 |
| Avg. no. in system (L) | 1.8028 | 0.0164 | 8.0716 | 0.0462 | 4.945 | 0.0303 |
| Avg. no. in queue (Lq) | 0.3191 | 0.0163 | 6.292 | 0.477 | 3.1964 | 0.315 |
| Proportion of idle server time | 4 hrs, 24 min | 0.8463 | 2 hrs, 13 min | 1.2249 | 2 hrs, 24 min | 1.0704 |
| Last customer leaves | 17.40 |  | 19.19 |  | 18.47 |  |

The time a patient spends in the system and in the queue increases by bit when there is only 6 patients and hour but by a lot when there are 8 patients an hour. The max time a patient spends in the system and the max time a patient spends in a queue is over doubled. The time that the servers spent idle was quite a bit higher across the results. This may be from the breaks or the time waiting around after breaks for a new patient to arrive. When there are 8 patients arriving every hour, the time the last patient left increased by nearly an hour in comparison to task 1.

*2.4 Patients are new and regular*

(a) New and regular patients

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 17.3924 | 0.026 | 51.019 | 0.1386 | 45.0902 | 0.0963 |
| Avg. time in Queue (Wq) | 0.9218 | 0.026 | 34.9916 | 0.1386 | 23.1294 | 0.0963 |
| Max time in system | 31.38 | 0.1523 | 96.6 | 0.3143 | 84.08 | 0.3589 |
| Max time in queue | 9.5 | 0.2533 | 77.11 | 0.3107 | 48.69 | 0.1376 |
| Avg. no. in system (L) | 1.7115 | 0.0028 | 6.2606 | 0.018 | 4.0561 | 0.0093 |
| Avg. no. in queue (Lq) | 0.0907 | 0.0912 | 4.2939 | 0.0176 | 2.0885 | 0.009 |
| Proportion of idle server time | 4 hrs, 24 min | 0.9288 | 1 hrs, 32 min | 0.6595 | 1 hr, 47 min | 0.5747 |
| Last customer leaves | 17.39 |  | 18.59 |  | 18.29 |  |

For this task the proportion of patients that were new was 20%.

The results are very similar to part 3. This is surprising as some patients would been in service for a longer time.

(b) All new patient at front­

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 33.001 | 0.1267 | 129.34 | 0.208 | 112.1018 | 0.2349 |
| Avg. time in Queue (Wq) | 15.0598 | 0.1267 | 111.4633 | 0.208 | 88.1802 | 0.2349 |
| Max time in system | 73.23 | 0.3287 | 159.55 | 0.2556 | 126.66 | 0.2727 |
| Max time in queue | 52.58 | 0.3599 | 144.55 | 0.2556 | 106.65 | 0.2724 |
| Avg. no. in system (L) | 3.2635 | 0.0144 | 14.2837 | 0.023 | 9.2064 | 0.0193 |
| Avg. no. in queue (Lq) | 1.4896 | 0.0133 | 12.3095 | 0.023 | 7.2419 | 0.0193 |
| Proportion of idle server time | 4 hrs, 24 min | 0.9397 | 3 hrs, 45 min | 0.0281 | 3 hrs, 40 min | 0 |
| Last customer leaves | 17.40 |  | 20.04 |  | 19.24 |  |

Compared to the first part of this task the average time in the system is much higher here. This may be from the first few patients causing a queue which led to a knock-on effect through the day. Both the average time in the system and average time in the queue are higher, but the max time in system and queue takes huge leaps. This is from a huge queue forming early in the day. Server idle time sees an increase as does the last time a customer leaves.

*2.5 Experienced servers*

(a) 1 experienced server

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 7.7624 | 0.0195 | 11.6389 | 0.0391 | 14.5733 | 0.0456 |
| Avg. time in Queue (Wq) | 0.7624 | 0.0195 | 4.6389 | 0.0391 | 4.5733 | 0.0456 |
| Max time in system | 12.95 | 0.1019 | 16.98 | 0.0141 | 19.96 | 0.0243 |
| Max time in queue | 5.95 | 0.1019 | 9.98 | 0.0141 | 9.96 | 0.0243 |
| Avg. no. in system (L) | 0.7773 | 0.0022 | 1.6441 | 0.0059 | 1.4419 | 0.0047 |
| Avg. no. in queue (Lq) | 0.0764 | 0.0022 | 0.6554 | 0.0057 | 0.4525 | 0.0046 |
| Proportion of idle server time | 2 hrs, 31 min | 0.4054 | 4 min, 49 sec | 0.3229 | 4 min, 30 sec | 0.3003 |
| Last customer leaves | 17.33 |  | 17.41 |  | 17.39 |  |

(b) 3 servers, 1 is experienced

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | λ = 6, µ = 4 | Standard Error | λ = 8, µ = 4 | Standard Error | λ = 6, µ = 3 | Standard Error |
| Avg. time in system (W) | 8.7396 | 0.0298 | 10.0082 | 0.0217 | 13.4549 | 0.0287 |
| Avg. time in Queue (Wq) | 0 | 0 | 0.0022 | 0.0008 | 0 | 0 |
| Max time in system | 15 | 0 | 15.1 | 0.0414 | 20 | 0 |
| Max time in queue | 0 | 0 | 0.13 | 0.0442 | 0 | 0 |
| Avg. no. in system (L) | 0.874 | 0.003 | 1.4208 | 0.0033 | 14 hrs, 13 min | 2.4086 |
| Avg. no. in queue (Lq) | 0 | 0 | 0.0003 | 0.0001 | 1.3344 | 0.034 |
| Proportion of idle server time | 18 hrs, 1 min |  | 13 hrs, 28 min | 2.0903 | 0 | 0 |
| Last customer leaves | 17.34 |  | 17 hrs, 38 min |  | 17.37 |  |

For this task server 1’s servicing time was set to half of that of the other servers. For example, if it took server 2 and 3 20 minutes to see a customer, then it took server 1 only 10 minutes.

Compared to each part of this task to each other, the queueing time was non-existent when there were 3 servers on. This is because server 1 could see almost all patients by themselves. Server 3 was in fact rarely, if ever used. This had the knock-on effect of drastically increasing server idle times as server 3 spent most of the day doing nothing.

This may be the only test that resulted in better results that task 1. Customer time in the system extremely reduced as server 1 could do the vast amount of the work in half the time.

Conclusion

Overall, the results seem to be as expected. Standard error seems fairly stable across all results and only takes a select few jumps up to higher number. Results such as average time in system increase in areas where you would expect them too. In areas where queues build up. The task that showed the most promising results was task 5. Having an experienced server made a huge impact on seeing patients.

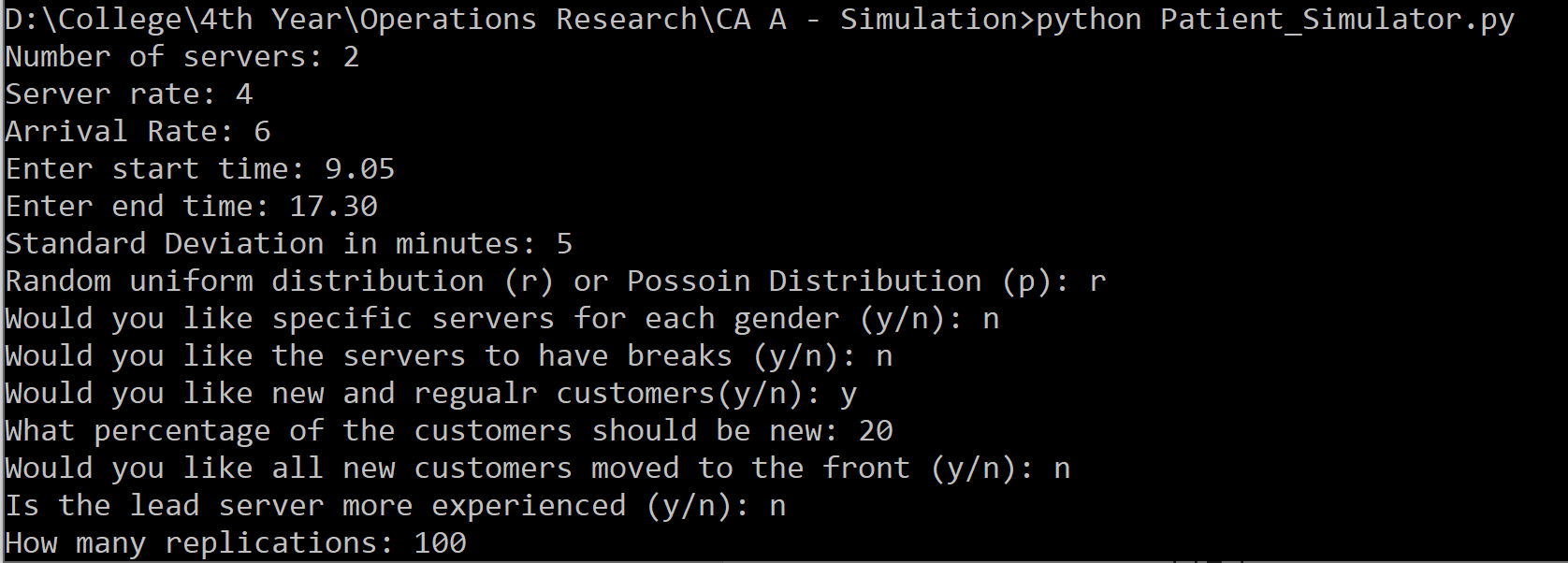
Tasks 2, 3, and 4 understandably increased the time patients spent in the system due to the extra constraints that servers had to work around.

Poisson distributed times also had the effect of making the servicing more efficient as they led to a lot of patients arriving at once and this led to the patients having to queue much longer and servers having to do most of their work in more concentrated pockets of the day.

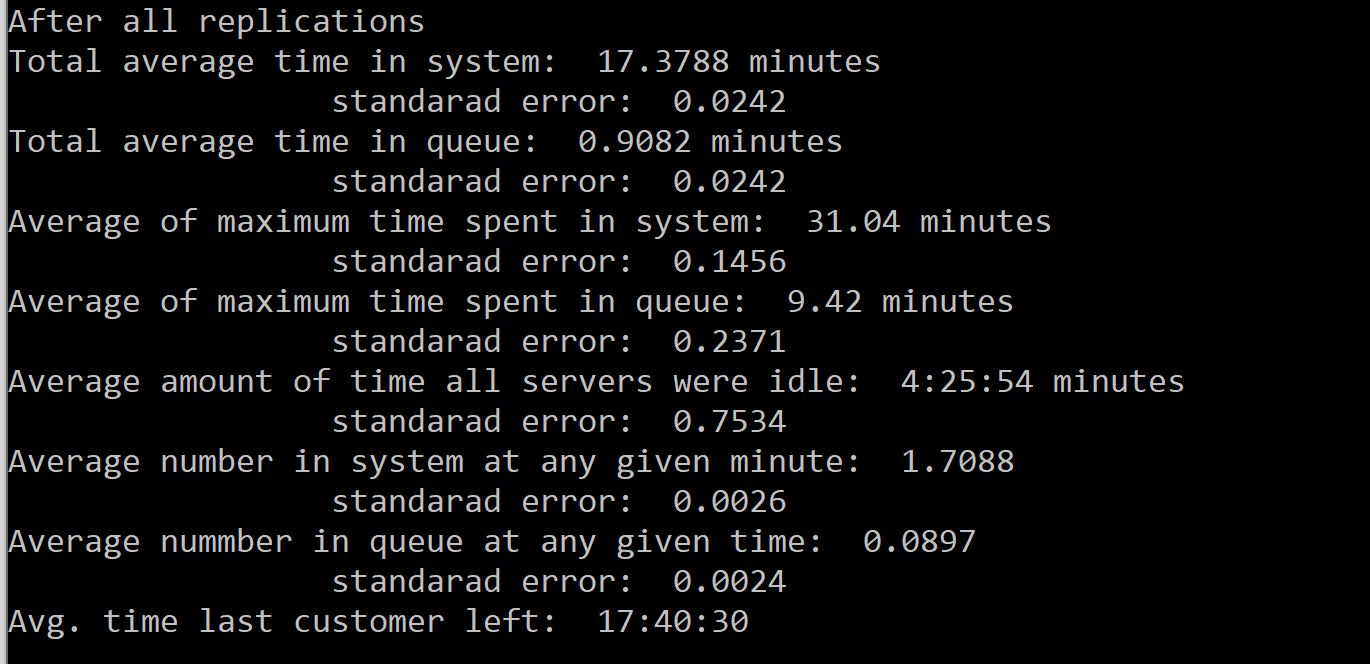
Appendix

Screen shots of A2 Program

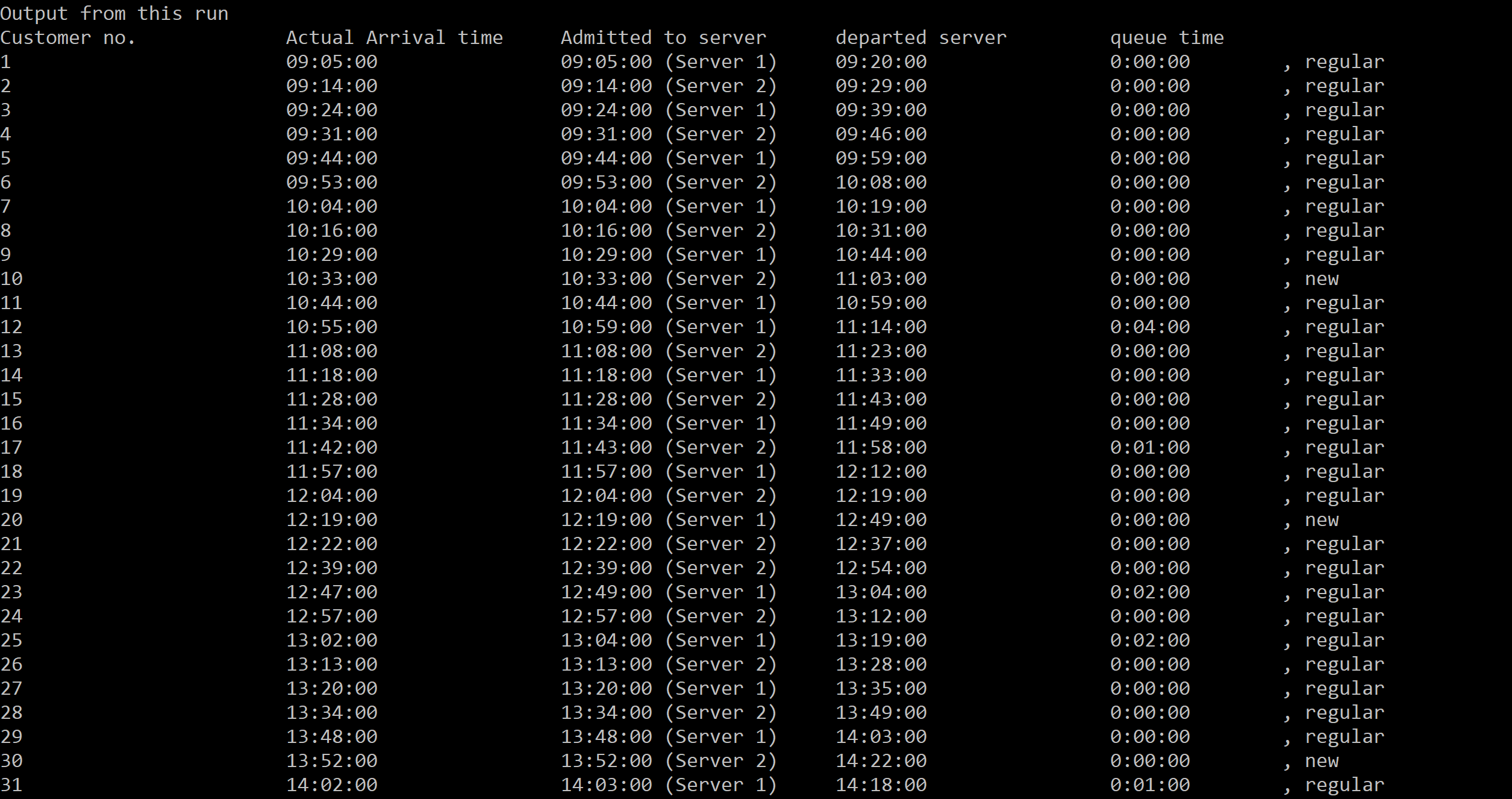
Input



Final output



One cycle output



Element A1 program

# Get 1000 random numbers and their frequencies

from scipy.stats import chisquare

import random

import math

def populate\_dictionary(num\_range):

num\_count = {}

i = 0

while i < num\_range:

num\_count[str(i)] = 0

i += 1

return num\_count

def chi\_sqaure\_test(num\_count):

num\_frequencies = list(num\_count.values())

chi\_sqaure\_stat = chisquare(num\_frequencies)

return chi\_sqaure\_stat

def language\_random(num\_integers, num\_range, num\_count):

num\_count = populate\_dictionary(num\_range)

for i in range(num\_integers):

rand\_num = random.randint(0, num\_range - 1)

if str(rand\_num) in num\_count:

num\_count[str(rand\_num)] += 1

else:

num\_count[str(rand\_num)] = 1

return num\_count

def mult\_congru(num\_count, num\_integers, seed, a, b, m):

while num\_integers > 0:

rand\_num = ((a \* seed) + b) % m

if str(rand\_num) in num\_count:

num\_count[str(rand\_num)] += 1

else:

num\_count[str(rand\_num)] = 1

seed = rand\_num

num\_integers -= 1

return num\_count

#Returns percentage of chi-squared stats that are in good range

def results\_check(list\_chisq\_stats, num\_range):

in\_good\_range = 2\*(math.sqrt(num\_range))#2SqrootR

count = 0

for chisq in list\_chisq\_stats:

if not ((num\_range - in\_good\_range) <= chisq and chisq <= (num\_range + in\_good\_range)):

count += 1

amount\_in\_range = len(list\_chisq\_stats) - count

percentage\_good\_range = (100 \* amount\_in\_range)/ len(list\_chisq\_stats)

print("percentage in good range", percentage\_good\_range)

good\_range = "Acceptable number range: " + str(num\_range - in\_good\_range) + " - " + str(num\_range + in\_good\_range)

return good\_range

def main():

print("Which random number generator would you like to use?")

generator\_decision = input("Language(l) or Muliplicative Congruential Method(m): ")

repeitions = int(input("How many repeitions: "))

list\_chisq\_stats = []

num\_count = {}#Dict of random numbers and their occurences

num\_integers = int(input("Enter number of integers: "))

if generator\_decision == 'l':

num\_range = int(input("Enter Upper Range, (lower range = 0): "))

num\_count = language\_random(num\_integers, num\_range, num\_count)

elif generator\_decision == 'm':

seed = int(input("Seed(u0): "))#u0

a = int(input("a: "))

b = int(input("b: "))

m = int(input("m: "))

num\_count = mult\_congru(num\_count, num\_integers, seed, a, b, m)

while repeitions > 0:

num\_count.clear()

if generator\_decision == 'l':

num\_count = language\_random(num\_integers, num\_range, num\_count)

elif generator\_decision == 'm':

num\_count = mult\_congru(num\_count, num\_integers, seed, a, b, m)

print("Frequencies of numbers: ", num\_count)

chisq\_stat = chi\_sqaure\_test(num\_count)

print("Chi-Sqaure stat: ", chisq\_stat[0])

list\_chisq\_stats.append(chisq\_stat[0])

repeitions -= 1

print()

if generator\_decision == 'l':

print(list\_chisq\_stats)

print(results\_check(list\_chisq\_stats, num\_range))

elif generator\_decision == 'm':

print("List of chi squared stats: ", list\_chisq\_stats)

print(results\_check(list\_chisq\_stats, m-1))

if \_\_name\_\_ == '\_\_main\_\_':

main()

Element A2 Program

#Simulation of hospital queue

from datetime import time

from datetime import timedelta

import datetime

import random

import math

#Server Class

class Server:

def \_\_init\_\_(self, number, rate):

self.number = number

self.rate = rate

self.service\_time = int(60/rate)

self.is\_free = True

self.next\_free = 0

self.time\_idle = 0

self.server\_gender = "unknown"

self.server\_break\_times = []

self.is\_server\_on\_break = False

self.experience = "normal"

def get\_server\_number(self):

return self.number

def get\_is\_free(self):

return self.is\_free

def get\_service\_time(self):

return self.service\_time

def get\_next\_free(self):

return self.next\_free

def get\_time\_idle(self):

return self.time\_idle

def get\_gender(self):

return self.server\_gender

def get\_server\_break\_times(self):

return self.server\_break\_times

def get\_server\_on\_break(self):

return self.is\_server\_on\_break

def get\_server\_experience(self):

return self.experience

def set\_next\_free(self, next\_free):

self.next\_free = next\_free

def set\_is\_free(self, free\_status):

self.is\_free = free\_status

def set\_time\_idle(self, time\_idle):

self.time\_idle = time\_idle

def set\_gender(self, server\_gender):

self.server\_gender = server\_gender

def subtract\_from\_time\_idle(self, time\_to\_subtract):

time\_to\_subtract\_seconds = timedelta(seconds = time\_to\_subtract \* 60)

self.time\_idle = self.time\_idle - time\_to\_subtract\_seconds

def set\_server\_break\_times(self, server\_break\_times):

self.server\_break\_times = server\_break\_times

def set\_server\_on\_break(self, break\_status):

self.is\_server\_on\_break = break\_status

def set\_experience(self, experience):

self.experience = experience

#Customer Class

class Customer:

def \_\_init\_\_(self, number, arrival\_time):

self.number = number

self.arrival\_time = arrival\_time

self.has\_been\_serviced = False

self.time\_admitted = 0

self.time\_in\_queue = 0

self.departure\_time = 0

self.which\_sever = 0

self.c\_gender = "unknown"

self.visiting\_status = "regular"

def \_\_str\_\_(self):

return "Customer %s Arrival Time %s" % (self.number, self.arrival\_time)

def get\_customer\_number(self):

return self.number

def get\_arrival\_time(self):

return self.arrival\_time

def get\_customer\_number(self):

return self.number

def get\_has\_been\_serviced(self):

return self.has\_been\_serviced

def get\_which\_server(self):

return self.which\_sever

def get\_departure\_time(self):

return self.departure\_time

def get\_time\_admitted(self):

return self.time\_admitted

def get\_time\_in\_queue(self):

return self.time\_in\_queue

def get\_customer\_gender(self):

return self.c\_gender

def get\_visiting\_status(self):

return self.visiting\_status

def set\_arrival\_time(self, arrival\_time):

self.arrival\_time = arrival\_time

def set\_has\_been\_serviced(self, service\_status):

self.has\_been\_serviced = service\_status

def set\_departure\_time(self, departure\_time):

self.departure\_time = departure\_time

def set\_which\_server(self, which\_sever):

self.which\_sever = which\_sever

def set\_time\_admitted(self, time\_admitted):

self.time\_admitted = time\_admitted

def set\_queue\_time(self, time\_in\_queue):

self.time\_in\_queue = time\_in\_queue

def set\_customer\_gender(self, c\_gender):

self.c\_gender = c\_gender

def set\_visiting\_status(self, visiting\_status):

self.visiting\_status = visiting\_status

#Parse time to create a time object

def parse\_time(time\_string):

time\_input = time\_string.split(".")

input\_hour = int(time\_input[0])

input\_minute = int(time\_input[1])

time\_object = time(input\_hour, input\_minute)

return time\_object

#Add/take away minutes from time

def add\_minutes\_to\_time(time\_object, minutes):

time\_hour = time\_object.hour

time\_minute = time\_object.minute + minutes

if time\_minute >= 60:

time\_minute = time\_minute - 60

time\_hour += 1

elif time\_minute < 0:

time\_minute = 60 + (time\_minute)

time\_hour -= 1

if time\_hour == 24:

time\_hour = 0

new\_time = time(time\_hour, time\_minute)

return new\_time

#Generate a uniformally distributed list of times

#Starting with start times

def generate\_arrival\_times(start\_time, last\_time, arrival\_rate):

list\_of\_times = []

arrival\_interval = int(60/arrival\_rate)

list\_of\_times.append(start\_time)

next\_time = start\_time

while next\_time <= last\_time:

next\_time = add\_minutes\_to\_time(next\_time, arrival\_interval)

if next\_time <= last\_time:

list\_of\_times.append(next\_time)

return list\_of\_times

#Add standard deviation to times

def deviate\_times(list\_of\_times, stand\_dev):

deviated\_list\_of\_times = []

for arr\_time in list\_of\_times:

minute\_variation = random.randint(stand\_dev \* -1, stand\_dev)

deviated\_time = add\_minutes\_to\_time(arr\_time, minute\_variation)

deviated\_list\_of\_times.append(deviated\_time)

return deviated\_list\_of\_times

#Later time should be second

def difference\_between\_times(first\_time, second\_time):

first\_delta = timedelta(hours = first\_time.hour, minutes = first\_time.minute, seconds = first\_time.second)

second\_delta = timedelta(hours = second\_time.hour, minutes = second\_time.minute, seconds = second\_time.second)

time\_diff = second\_delta - first\_delta

return time\_diff

def possoin\_dristributed\_times(customers, start\_time, end\_time):

print("possoin\_times")

poisson\_times = []

num\_of\_customers = len(customers)

p\_count = 0

while p\_count < num\_of\_customers:

rand\_hour = random.randint(start\_time.hour, end\_time.hour)

rand\_minute = random.randint(0, 59)

rand\_time = time(rand\_hour, rand\_minute)

if start\_time <= rand\_time <= end\_time:

poisson\_times.append(rand\_time)

p\_count += 1

poisson\_times.sort()

i = 0

for p\_time in poisson\_times:

customers[i].set\_arrival\_time(p\_time)

i += 1

return customers

def service\_customers(servers, customers, start\_time, last\_time):

current\_time = customers[0].get\_arrival\_time()

longest\_server\_time = 0

for serv in servers:

if serv.get\_service\_time() > longest\_server\_time:

longest\_server\_time = serv.get\_service\_time()

latest\_time = add\_minutes\_to\_time(last\_time, longest\_server\_time)

serviced\_customers = []

print()

print("Customer arriving and being served.")

all\_customers\_served = False

while all\_customers\_served == False:

#while current\_time <= latest\_time:

for cust in customers:

#Customer Arrived

if cust.get\_arrival\_time() == current\_time:

print(current\_time, "Customer", cust.get\_customer\_number(), "arrived")

#Customer Departed

elif cust.get\_departure\_time() == current\_time:

#Set server free

for serv in servers:

if cust.get\_which\_server() == serv.get\_server\_number():

if serv.get\_server\_on\_break() == False:

serv.set\_is\_free(True)

#Is it a servers break time

for serv in servers:

if current\_time in serv.get\_server\_break\_times():

if serv.get\_is\_free():

end\_of\_break = add\_minutes\_to\_time(current\_time, 30)

serv.set\_next\_free(end\_of\_break)

serv.set\_server\_on\_break(True)

else:

end\_of\_break = add\_minutes\_to\_time(serv.get\_next\_free(), 30)

serv.set\_next\_free(end\_of\_break)

serv.set\_server\_on\_break(True)

#Is a break over

if current\_time == serv.get\_next\_free() and serv.get\_server\_on\_break() == True:

serv.set\_is\_free(True)

#Is server free for unseen customers

for serv in servers:

if serv.get\_is\_free():

#If a server is free, see a customer that has arrived

for cust in customers:

if (cust.get\_arrival\_time() <= current\_time and

cust.get\_has\_been\_serviced() == False and cust.get\_customer\_gender() == serv.get\_gender()):

#Announce Action

#Set server info

serv.set\_is\_free(False)

server\_next\_free = add\_minutes\_to\_time(current\_time, serv.get\_service\_time())

if cust.get\_visiting\_status() == "new":

server\_next\_free = add\_minutes\_to\_time(current\_time, serv.get\_service\_time()\*2)

if serv.get\_server\_experience() == "experienced":

server\_next\_free = add\_minutes\_to\_time(current\_time, int(serv.get\_service\_time()/2))

serv.set\_next\_free(server\_next\_free)

#Set customer info

cust.set\_has\_been\_serviced(True)

customer\_departure\_time = add\_minutes\_to\_time(current\_time, serv.get\_service\_time())

if cust.get\_visiting\_status() == "new":

customer\_departure\_time = add\_minutes\_to\_time(current\_time, serv.get\_service\_time()\*2)

if serv.get\_server\_experience() == "experienced":

customer\_departure\_time = add\_minutes\_to\_time(current\_time, int(serv.get\_service\_time()/2))

cust.set\_departure\_time(customer\_departure\_time)

cust.set\_which\_server(serv.get\_server\_number())

cust.set\_time\_admitted(current\_time)

customer\_queueing\_time = difference\_between\_times(cust.get\_arrival\_time(), cust.get\_time\_admitted())

cust.set\_queue\_time(customer\_queueing\_time)

break

cust\_count = 0

for cust in customers:

if cust.get\_has\_been\_serviced() == True:

cust\_count += 1

if cust\_count == len(customers):

all\_customers\_served = True

current\_time = add\_minutes\_to\_time(current\_time, 1)

return customers

def print\_output(customers, gender\_choice, new\_regular\_choice):

print()

print("Output from this run")

print("Customer no. \t\t Actual Arrival time \t Admitted to server \t departed server \t queue time")

for cust in customers:

server\_string = "(" + str(cust.get\_which\_server()) + ")"

additonal\_string = "\t"

if gender\_choice == 'y':

additonal\_string += ", " + str(cust.get\_customer\_gender())

if new\_regular\_choice == 'y':

additonal\_string += ", " + str(cust.get\_visiting\_status())

print(cust.get\_customer\_number(), '\t\t\t', cust.get\_arrival\_time(), '\t\t', cust.get\_time\_admitted(), server\_string, '\t', cust.get\_departure\_time(), '\t\t', cust.get\_time\_in\_queue(), additonal\_string)

def performence\_metrics(servers, customers):

print()

print("Final Performence Metrics (from this cycle)")

# Avg time a customer is in system

total\_time\_in\_system = 0

total\_time\_in\_queue = 0

# Maximum time a customer spends in service and in queue

max\_time\_in\_system = 0

max\_time\_in\_queue = 0

for cust in customers:

time\_in\_system = difference\_between\_times(cust.get\_arrival\_time(), cust.get\_departure\_time())

#Convert time to minutes, timedeltas store time in seconds

time\_in\_system = time\_in\_system.seconds / 60

total\_time\_in\_system += time\_in\_system

time\_in\_queue = cust.get\_time\_in\_queue().seconds / 60

total\_time\_in\_queue += time\_in\_queue

if time\_in\_system > max\_time\_in\_system:

max\_time\_in\_system = time\_in\_system

if time\_in\_queue > max\_time\_in\_queue:

max\_time\_in\_queue= time\_in\_queue

avg\_time\_in\_system = total\_time\_in\_system/len(customers)

avg\_time\_in\_queue = total\_time\_in\_queue/len(customers)

#Time server spent idle

first\_time = customers[0].get\_arrival\_time()

second\_time = customers[len(customers)-1].get\_departure\_time()

total\_time\_in\_day = difference\_between\_times(first\_time, second\_time)

time\_all\_servers\_idle = timedelta()

for serv in servers:

serv.set\_time\_idle(total\_time\_in\_day)

for cust in customers:

if cust.get\_which\_server() == serv.get\_server\_number():

if serv.get\_server\_experience() == "experienced":

serv.subtract\_from\_time\_idle(int(serv.get\_service\_time()/2))

else:

serv.subtract\_from\_time\_idle(serv.get\_service\_time())

#time\_server\_spent\_idle = difference\_between\_times(total\_time\_servicing, total\_time\_in\_day)

#serv.set\_time\_idle(time\_server\_spent\_idle)

print("Amount of time server ", serv.get\_server\_number(), "was idle:", serv.get\_time\_idle())

time\_all\_servers\_idle += serv.get\_time\_idle()

#Avg number in queue, system

current\_time = first\_time

minute\_count = 0

in\_system\_times = []

in\_queue\_times = []

currently\_in\_system = 0

currently\_in\_queue = 0

while current\_time <= second\_time:

for cust in customers:

if current\_time == cust.get\_arrival\_time():

currently\_in\_system += 1

currently\_in\_queue += 1

if current\_time == cust.get\_time\_admitted():

currently\_in\_queue -= 1

if current\_time == cust.get\_departure\_time():

currently\_in\_system -= 1

in\_system\_times.append(currently\_in\_system)

in\_queue\_times.append(currently\_in\_queue)

current\_time = add\_minutes\_to\_time(current\_time, 1)

minute\_count += 1

print(minute\_count)

avg\_num\_in\_system = sum(in\_system\_times)/minute\_count

avg\_num\_in\_queue = sum(in\_queue\_times)/minute\_count

#Print Metrics

print("Avg. time of a customer in system (W):", avg\_time\_in\_system)

print("Avg. time of a customer in queue (Wq):", avg\_time\_in\_queue)

print("Max time a customer spends in system: ", max\_time\_in\_system)

print("Max time a customer spends in queue: ", max\_time\_in\_queue)

print("Total time all servers were idle: ", time\_all\_servers\_idle)

print("Avg. in system at any given minute: ", avg\_num\_in\_system)

print("Avg. in queue at any given minute: ", avg\_num\_in\_queue)

rep\_performence\_metrics = []

rep\_performence\_metrics.append(avg\_time\_in\_system)

rep\_performence\_metrics.append(avg\_time\_in\_queue)

rep\_performence\_metrics.append(max\_time\_in\_system)

rep\_performence\_metrics.append(max\_time\_in\_queue)

rep\_performence\_metrics.append(time\_all\_servers\_idle)

rep\_performence\_metrics.append(avg\_num\_in\_system)

rep\_performence\_metrics.append(avg\_num\_in\_queue)

#time last customer leaves

rep\_performence\_metrics.append(customers[len(customers)-1].get\_departure\_time())

return rep\_performence\_metrics

def calculate\_standard\_error(list\_of\_num):

mean\_of\_list = sum(list\_of\_num)/float(len(list\_of\_num))

deviations\_from\_mean = []

for num in list\_of\_num:

deviation = mean\_of\_list - num

deviation = deviation \* deviation#Square number to get rid of negatives

deviations\_from\_mean.append(deviation)

total\_squared\_deviations = sum(deviations\_from\_mean)

total\_squared\_deviations = total\_squared\_deviations/(len(list\_of\_num)-1)

sd = math.sqrt(total\_squared\_deviations)

se = sd/(math.sqrt(len(list\_of\_num)))

return se

def calculate\_standard\_error\_timedelta(list\_of\_timedelta):

td\_list = []

for t in list\_of\_timedelta:

td\_list.append(t.seconds)

mean\_of\_list = sum(td\_list)/float(len(td\_list))

deviations\_from\_mean = []

for num in td\_list:

deviation = mean\_of\_list - num

deviation = deviation \* deviation#Square number to get rid of negatives

deviations\_from\_mean.append(deviation)

total\_squared\_deviations = sum(deviations\_from\_mean)

total\_squared\_deviations = total\_squared\_deviations/(len(td\_list)-1)

sd = math.sqrt(total\_squared\_deviations)

se = sd/(math.sqrt(len(td\_list)))

return se/60

def main():

#Default values

# num\_servers = 2

# server\_rate = 4

# arrival\_rate = 6

# stand\_dev = 5

# scheduling\_choice = 'r'

# time\_string = "9.05"

# last\_time\_string = "17.30"

# replications = 100

#User Input

num\_servers = int(input("Number of servers: "))

server\_rate = int(input("Server rate: "))# Rate servers take per hour

arrival\_rate = int(input("Arrival Rate: "))

#first customer enters service

time\_string = input("Enter start time: ")

start\_time = parse\_time(time\_string)

#last time a customer can enter service

last\_time\_string = input("Enter end time: ")

last\_time = parse\_time(last\_time\_string)

stand\_dev = int(input("Standard Deviation in minutes: "))#Standard Deviation in minutes

scheduling\_choice = input("Random uniform distribution (r) or Possoin Distribution (p): ")

gender\_choice = input("Would you like specific servers for each gender (y/n): ")

break\_choice = input("Would you like the servers to have breaks (y/n): ")

new\_regular\_choice = input("Would you like new and regualr customers(y/n): ")

if new\_regular\_choice == 'y':

percent\_new = int(input("What percentage of the customers should be new: "))

new\_at\_front = input("Would you like all new customers moved to the front (y/n): ")

experienced\_server\_choice = input("Is the lead server more experienced (y/n): ")

#Replicate from here:

replications = int(input("How many replications: "))

total\_avg\_time\_in\_system = []

total\_avg\_time\_in\_queue = []

total\_max\_time\_in\_system = []

total\_max\_time\_in\_queue = []

total\_time\_servers\_idle = []

total\_avg\_num\_in\_system = []

total\_avg\_num\_in\_queue = []

times\_last\_customer\_leaves = []

rep\_count = 0

while rep\_count < replications:

#Generated list of arrival times

list\_of\_times = generate\_arrival\_times(start\_time, last\_time, arrival\_rate)

list\_of\_times = deviate\_times(list\_of\_times, stand\_dev)

list\_of\_times.sort()

# Create a list of servers

servers = []

server\_count = 1

while server\_count <= num\_servers:

server\_name = "Server " + str(server\_count)

server = Server(server\_name, server\_rate)

servers.append(server)

server\_count += 1

#Create a list of customers

customers = []

customer\_count = 1

for arr\_time in list\_of\_times:

new\_customer = Customer(customer\_count, arr\_time)

customers.append(new\_customer)

customer\_count += 1

#Poisson Distributed times

if scheduling\_choice == 'p':

customers = possoin\_dristributed\_times(customers, start\_time, last\_time)

#Give customers genders

if gender\_choice == 'y':

genders = ["male", "female"]

for cust in customers:

cust.set\_customer\_gender(random.choice(genders))

next\_gender = 0

for serv in servers:

if next\_gender == 0:

serv.set\_gender(genders[0])

next\_gender = 1

elif next\_gender == 1:

serv.set\_gender(genders[1])

next\_gnder = 0

#Give customers new or regular status

if new\_regular\_choice == 'y':

amount\_to\_change = int((len(customers)\*percent\_new)/100)

if new\_at\_front == 'y':

i = 0

while i < amount\_to\_change:

customers[i].set\_visiting\_status("new")

i += 1

else:

print("change change\_index", int(amount\_to\_change))

i = 1

while i <= len(customers):

if i % int(amount\_to\_change) == 0:

customers[i-1].set\_visiting\_status("new")

i += 1

[print(cust.get\_visiting\_status()) for cust in customers]

#Make lead server more experienced

if experienced\_server\_choice == 'y':

servers[0].set\_experience("experienced")

[print(serv.get\_server\_experience()) for serv in servers]

#Give serverrs break times

set\_of\_break\_times =[[parse\_time("10.45"), parse\_time("14.45")], [parse\_time("11.15"), parse\_time("15.15")]]

next\_break\_set = 0

if break\_choice == 'y':

for serv in servers:

if next\_break\_set == 0:

serv.set\_server\_break\_times(set\_of\_break\_times[0])

next\_break\_set = 1

elif next\_break\_set == 1:

serv.set\_server\_break\_times(set\_of\_break\_times[1])

next\_break\_set == 0

print("Service customers throughout day")

customers = service\_customers(servers, customers, start\_time, last\_time)

#output

print\_output(customers, gender\_choice, new\_regular\_choice)

rep\_performence\_metrics = performence\_metrics(servers, customers)

#Collect results from this cyclle

total\_avg\_time\_in\_system.append(rep\_performence\_metrics[0])

total\_avg\_time\_in\_queue.append(rep\_performence\_metrics[1])

total\_max\_time\_in\_system.append(rep\_performence\_metrics[2])

total\_max\_time\_in\_queue.append(rep\_performence\_metrics[3])

total\_time\_servers\_idle.append(rep\_performence\_metrics[4])

total\_avg\_num\_in\_system.append(rep\_performence\_metrics[5])

total\_avg\_num\_in\_queue.append(rep\_performence\_metrics[6])

times\_last\_customer\_leaves.append(rep\_performence\_metrics[7])

rep\_count += 1

# Final Replication Results

print()

print("After all replications")

print("Total average time in system: ", round(sum(total\_avg\_time\_in\_system)/replications, 4), "minutes")

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_avg\_time\_in\_system), 4))

print("Total average time in queue: ", round(sum(total\_avg\_time\_in\_queue)/replications, 4), "minutes")

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_avg\_time\_in\_queue), 4))

print("Average of maximum time spent in system: ", round(sum(total\_max\_time\_in\_system)/replications, 4), "minutes")

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_max\_time\_in\_system), 4))

print("Average of maximum time spent in queue: ", sum(total\_max\_time\_in\_queue)/replications, "minutes")

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_max\_time\_in\_queue), 4))

total\_idle\_time = 0

for idle\_time in total\_time\_servers\_idle:

total\_idle\_time += idle\_time.seconds

total\_idle\_time = total\_idle\_time/replications

total\_idle\_time\_delta = timedelta(seconds = int(total\_idle\_time))

print("Average amount of time all servers were idle: ", total\_idle\_time\_delta, "minutes")

print("\t\t standarad error: ", round(calculate\_standard\_error\_timedelta(total\_time\_servers\_idle), 4))

print("Average number in system at any given minute: ", round(sum(total\_avg\_num\_in\_system)/replications, 4))

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_avg\_num\_in\_system), 4))

print("Average nummber in queue at any given time: ", round(sum(total\_avg\_num\_in\_queue)/replications, 4))

print("\t\t standarad error: ", round(calculate\_standard\_error(total\_avg\_num\_in\_queue), 4))

total\_seconds = 0

for last\_time in times\_last\_customer\_leaves:

t\_delta = timedelta(hours = last\_time.hour, minutes = last\_time.minute ,seconds = last\_time.second)

total\_seconds += t\_delta.seconds

avg\_last\_time = timedelta(seconds = total\_seconds/len(times\_last\_customer\_leaves))

print("Avg. time last customer left: ", avg\_last\_time)

if \_\_name\_\_ == '\_\_main\_\_':

main()