```
! pip install simpy
```

```
Collecting simpy
  Downloading simpy-4.0.1-py2.py3-none-any.whl (29 kB)
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

```
import simpy
import random
import math


RANDOM_SEED = 978

CUSTOMER_COUNT = 1000

INTERARRIVAL_MEAN = 14.3
INTERARRIVAL_RATE = 1.0 / INTERARRIVAL_MEAN

m = 7.2
v = 2.7
phi = math.sqrt(v + m ** 2)
SERVICE_FRONTDESK_MEAN = math.log(m ** 2 / phi)
SERVICE_FRONTDESK_STD = math.sqrt(math.log(phi ** 2 / m ** 2))

SERVICE_EXPERT_MEAN = 10.2
SERVICE_EXPERT_RATE = 1.0 / SERVICE_EXPERT_MEAN

RENEGING_MEAN = 60.0
RENEGING_RATE = 1.0 / RENEGING_MEAN

BREAK_MEAN = 60.0
BREAK_RATE = 1.0 / BREAK_MEAN

BREAK_TIME = 3


class Customer(object):
    def __init__(self, name, env, frontdesk, expert):
        self.name = name
        self.env = env
        self.frontdesk = frontdesk
        self.expert = expert
        self.arrival = self.env.now
        self.action = env.process(self.call())

    def call(self):
        #print('%s initiated a call at %g' % (self.name, self.env.now))

        # a call is initiated and registered as a request to the frontdesk operator
        with self.frontdesk.request() as req:
```

```
            with self.frontdesk.request() as req:
                yield req
                #print('%s is assigned to the frontdesk at %g' % (self.name, self.env.now))
                # add the waiting time of that customer to waiting_times
                self.waiting_time_frontdesk = self.env.now - self.arrival
                # call is served
                yield self.env.process(self.serve_frontdesk())
                #print('%s is done with the frontdesk at %g' % (self.name, self.env.now))
                self.frontdesk_exited = self.env.now

            # call is registered as a request to the expert operator
            with self.expert.request() as req:
                reneg_time = random.expovariate(RENEGING_RATE)
                # wait for expert or leave the system
                results = yield req | self.env.timeout(reneg_time)
                # customer waited less than reneg_time
                if req in results:
                    #print('%s is assigned to the expert at %g' % (self.name, self.env.now))
                    # add the waiting time of that customer to waiting_times
                    self.waiting_time_expert = self.env.now - self.frontdesk_exited
                    # call is served
                    yield self.env.process(self.serve_expert())
                    #print('%s is done with the expert at %g' % (self.name, self.env.now))
                else:
                    # customer reneged
                    #print('%s is reneged at %g' % (self.name, self.env.now))
                    self.waiting_time_expert = reneg_time
                    self.service_time_expert = 0

            customers.append(self)
            # last customer sets end time
            if len(customers) == CUSTOMER_COUNT:
                global end_time
                end_time = self.env.now

        def serve_frontdesk(self):
            self.service_time_frontdesk = random.lognormvariate(SERVICE_FRONTDESK_MEAN, SERVICE_F
            yield self.env.timeout(self.service_time_frontdesk)

        def serve_expert(self):
            self.service_time_expert = random.expovariate(SERVICE_EXPERT_RATE)
            yield self.env.timeout(self.service_time_expert)


    def customer_generator(env, frontdesk, expert):
        i = 1
        # while end time has not been set by last customer yet
        while end_time == 0:
            yield env.timeout(random.expovariate(INTERARRIVAL_RATE))
            Customer('Customer %s' % (i), env, frontdesk, expert)
            i += 1
```

```
def break_generator(env, expert):
    # while end time has not been set by last customer yet
    while end_time == 0:
        yield env.timeout(random.expovariate(BREAK_RATE))
        #print('Expert wants break at %g' % (env.now))
        with expert.request() as req:
            yield req
            #print('Expert gives break at %g' % (env.now))
            yield env.timeout(BREAK_TIME)
            #print('Expert exits break at %g' % (env.now))
            global break_counter
            break_counter += BREAK_TIME


customers = []
end_time = 0
break_counter = 0


random.seed(RANDOM_SEED)
env = simpy.Environment()
frontdesk = simpy.Resource(env, capacity = 1)
expert = simpy.Resource(env, capacity = 1)
env.process(customer_generator(env, frontdesk, expert))
env.process(break_generator(env, expert))
env.run()
total_service_time_frontdesk = 0
total_service_time_expert = 0
total_waiting_time_expert = 0
total_waiting_time = 0
max_ratio = 0
for i in range(CUSTOMER_COUNT):
    c = customers[i]
    total_service_time_frontdesk += c.service_time_frontdesk
    total_service_time_expert += c.service_time_expert
    total_waiting_time_expert += c.waiting_time_expert
    total_waiting_time += c.waiting_time_frontdesk + c.waiting_time_expert
    max_ratio = max([max_ratio, ((c.waiting_time_frontdesk + c.waiting_time_expert) / (c.wait
print('Utilization of frontdesk: %g' % (total_service_time_frontdesk / end_time))
print('Utilization of expert (including breaks): %g' % (total_service_time_expert / end_time)
print('Utilization of expert (excluding breaks): %g' % (total_service_time_expert / (end_time
print('Average Total Waiting Time: %g' % (total_waiting_time / CUSTOMER_COUNT))
print('Maximum Total Waiting Time to Total System Time Ratio: %g' % max_ratio)
print('Average number of people waiting to be served by expert: %g' % (total_waiting_time_exp
```

```
Utilization of frontdesk: 0.510645
Utilization of expert (including breaks): 0.624112
Utilization of expert (excluding breaks): 0.650412
Average Total Waiting Time: 11.1983
Maximum Total Waiting Time to Total System Time Ratio: 0.917661
Average number of people waiting to be served by expert: 0.523443
```

```
CUSTOMER_COUNT = 5000
customers = []
end_time = 0
break_counter = 0
random.seed(RANDOM_SEED)
env = simpy.Environment()
frontdesk = simpy.Resource(env, capacity = 1)
expert = simpy.Resource(env, capacity = 1)
env.process(customer_generator(env, frontdesk, expert))
env.process(break_generator(env, expert))
env.run()
total_service_time_frontdesk = 0
total_service_time_expert = 0
total_waiting_time_expert = 0
total_waiting_time = 0
max_ratio = 0
for i in range(CUSTOMER_COUNT):
    c = customers[i]
    total_service_time_frontdesk += c.service_time_frontdesk
    total_service_time_expert += c.service_time_expert
    total_waiting_time_expert += c.waiting_time_expert
    total_waiting_time += c.waiting_time_frontdesk + c.waiting_time_expert
    max_ratio = max([max_ratio, ((c.waiting_time_frontdesk + c.waiting_time_expert) / (c.wait
print('Utilization of frontdesk: %g' % (total_service_time_frontdesk / end_time))
print('Utilization of expert (including breaks): %g' % (total_service_time_expert / end_time)
print('Utilization of expert (excluding breaks): %g' % (total_service_time_expert / (end_time
print('Average Total Waiting Time: %g' % (total_waiting_time / CUSTOMER_COUNT))
print('Maximum Total Waiting Time to Total System Time Ratio: %g' % max_ratio)
print('Average number of people waiting to be served by expert: %g' % (total_waiting_time_exp
```

```
Utilization of frontdesk: 0.497092
Utilization of expert (including breaks): 0.609585
Utilization of expert (excluding breaks): 0.636116
Average Total Waiting Time: 11.7349
Maximum Total Waiting Time to Total System Time Ratio: 0.952532
Average number of people waiting to be served by expert: 0.564514
```