Question 12.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a design of experiments approach would be appropriate.

From Work:

I described a similar situation to this one when I answered the conceptual question about regression. A lot of what I'm doing at work has to do with looking at retail portfolios. I'm a real estate consultant so the types of things that I'm doing include some analysis but the scope of most of the work rests outside of analysis, and has more to do with contracts, legal ramifications, negotiation, etc.. Anyway, a retailer has a large portfolio (100+ stores) and they want to figure out why their sales numbers ($/SF) across their portfolio vary from location to location, and which are the best locations/why. They almost certainly already know which ones perform the best when they come to me. Now, they could just run regression using certain factors or customer demographics (education levels, median income, etc..) and in many cases that would be sufficient enough to allow them to make educated choices about where to put the next set of stores or where to close some low performing stores and open some potentially higher performing stores.

However, they might decide that they want to use a design of experiments approach and run regression with that. Why would they do this? Well, it might be a bit of stretch to say that they would actually do this/get this detailed, but a design of experiments approach aims to introduce circumstances that will directly affect the output and in doing this the experimenter (the retailer/company in this case) aims to create a more detailed understanding or map of the solution (their portfolio). Independent, dependent, and control variables can be established, whereas the farthest you might go with regular linear regression would be to understand some factors that affect the output (sales as a response variable). Why is this important? Even if you could weight the variables in a regression, you might not understand which ones have relationships with each other and why. In the design of experiments approach, the retailer would come to understand which factors (control variables) need to be held constant or above a certain level, which factors (dependent variables) are important but connected to the performance of another variable, and which factors (independent variables) are important to the success of a location but not dependent on the presence of another factor. In other words, a design of experiments approach would create a deeper understanding of the portfolio and provide the retailer with more options as to how to pick new locations.

Question 12.2

To determine the value of 10 different yes/no features to the market value of a house (large yard, solar roof, etc.), a real estate agent plans to survey 50 potential buyers, showing a fictitious house with different combinations of features. To reduce the survey size, the agent wants to show just 16 fictitious houses. Use R's `FrF2` function (in the `FrF2` package) to find a fractional factorial design for this experiment: what set of features should each of the 16 fictitious houses have? Note: the output of `FrF2` is "1" (include) or "-1" (don't include) for each feature.

Ok, we want to come up with a fractional factorial design but first we need to learn a few things:

In a fractional factorial design (vs. a full factorial design), a fraction of the possible combinations are run. Why would we seek to do this? Well, when we wish to run a full factorial design, there are often so many combinations present that we can only evaluate using a couple of factors if we don't want to end up with thousands of test combinations. This particular case will lend itself to less combinations because the factors are only two-level factors. We could get into more complicated/multi-level factors but, thankfully, Professor Sokol hasn't tasked us with that. Phew! Also, 2-level design is the most common and widely used version of this process.

So, we want to use FrF2() to find a fractional factorial design for our 10-factor analysis using 16 fictional houses. So, we'll want to think about how many factors we'll need for 16 runs.

If we constructed a full factorial design, which I will run and show a portion of, we would see 1024 (2^{10}) houses or runs, but our fractional factorial design will use 16 and some number of the 10 factors. The best I can tell, there happen to be 50 different buyers for the interviews, but that number is sort of a red herring in terms of our design. This take on it may or may not be correct but it's how I have interpreted the language given in the prompt/question. Let's go over the formula for fractional factorial design before we go further.

Let's look at the formula for a fractional factorial design:
So, generally, the formula for fractional factorial design is l^{k-p}, where l is the number of levels, k is the number of factors and p is the fraction index. In this case, and because we are using 2-level factors for every factor, we would use 2^k to represent a full factorial design and 2^{k-p} to represent the fractional design.

We want to choose the optimal fraction:
What is the optimal fraction? It depends on which method of fractional factorial design we choose.
We could run using a few different methods. The main things that are important are that we achieve the maximum resolution for a given model, the minimum aberration, and that we use the correct number of runs for our given design. FrF2() will run using a resolution III design and seek to find the maximum resolution. It will also tell us which factors are aliasing or confounding with each other. In a resolution III design, the main effects are not aliased with any other main effects. They are, however, aliased with two factor interactions(2fis). Aliasing is an effect that causes different variables or their interactions to become indistinguishable. In practice aliasing is essentially saying that there is an A, a B, a C, and D=ABC. This particular example would be for a model that is 2^{4-1}. We gain efficiency by doing this and we lose some information as well. So, we know we have 16 runs. 1024/64 or $(2^{10})/(2^6)$ = 16 or (2^4).

This means that our model is a fractional design of 2^{10-6}.

Now, we run:

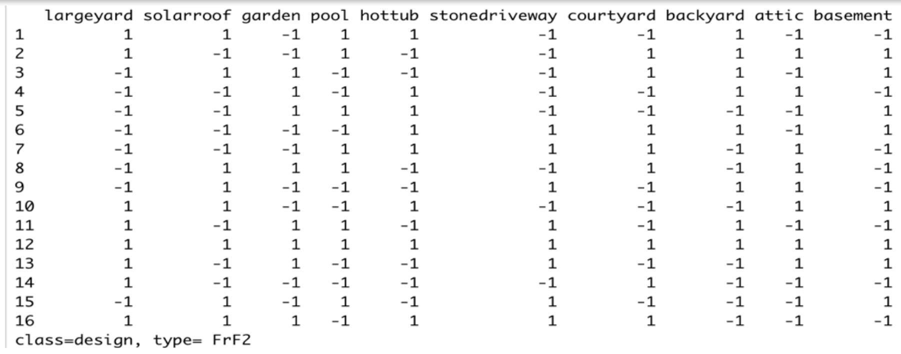
```
>FrF2(1024,10,factor.names=list(largeyard="",solarroof="",garden="",pool="",hottub="",stonedriveway
    ="",courtyard="",backyard="",attic="",basement=""))
```

Note a couple of things: We could just include the factor names and the function would still know there were 10 factors, and we could make the categories output yes or no instead of 1 and -1, which isn't necessary here but would be a nice touch given a real world application of this.

```
creating full factorial with 1024 runs ...

   largeyard solarroof garden pool hottub stonedriveway courtyard backyard attic basement
1         -1        -1     -1   -1     -1            -1        -1       -1    -1       -1
2          1        -1     -1   -1     -1            -1        -1       -1    -1       -1
3         -1         1     -1   -1     -1            -1        -1       -1    -1       -1
4          1         1     -1   -1     -1            -1        -1       -1    -1       -1
5         -1        -1      1   -1     -1            -1        -1       -1    -1       -1
6          1        -1      1   -1     -1            -1        -1       -1    -1       -1
7         -1         1      1   -1     -1            -1        -1       -1    -1       -1
8          1         1      1   -1     -1            -1        -1       -1    -1       -1
9         -1        -1     -1    1     -1            -1        -1       -1    -1       -1
10         1        -1     -1    1     -1            -1        -1       -1    -1       -1
11        -1         1     -1    1     -1            -1        -1       -1    -1       -1
12         1         1     -1    1     -1            -1        -1       -1    -1       -1
13        -1        -1      1    1     -1            -1        -1       -1    -1       -1
14         1        -1      1    1     -1            -1        -1       -1    -1       -1
15        -1         1      1    1     -1            -1        -1       -1    -1       -1
16         1         1      1    1     -1            -1        -1       -1    -1       -1
17        -1        -1     -1   -1      1            -1        -1       -1    -1       -1
18         1        -1     -1   -1      1            -1        -1       -1    -1       -1
19        -1         1     -1   -1      1            -1        -1       -1    -1       -1
20         1         1     -1   -1      1            -1        -1       -1    -1       -1
```

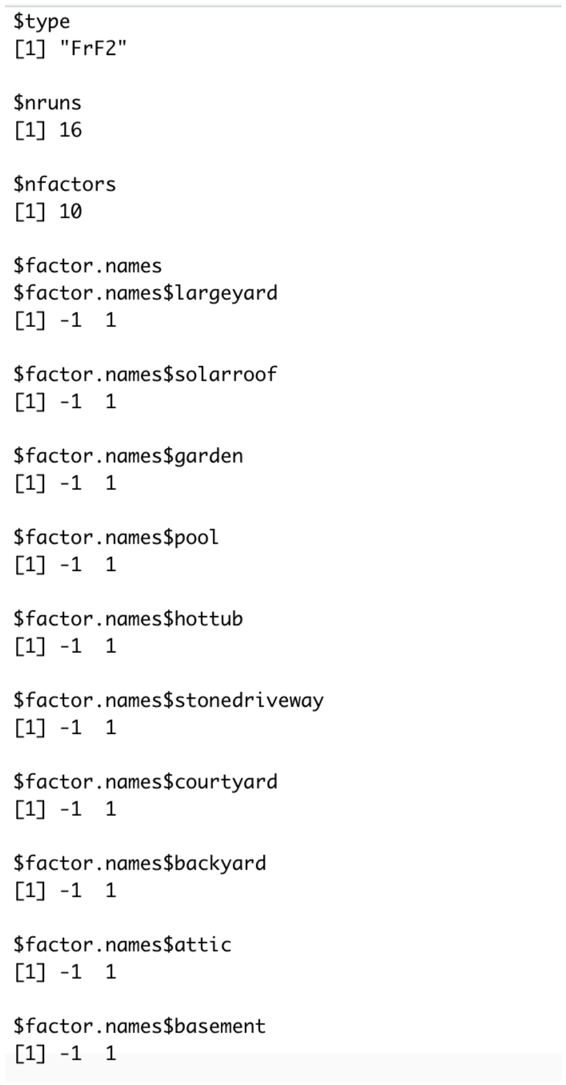
```
> ffactorial<- FrF2(16,10,factor.names=list(largeyard="",solarroof="",garden="",pool="",
hottub="",stonedriveway="",courtyard="",backyard="",attic="",basement=""))
> print(ffactorial)
```



```
   largeyard solarroof garden pool hottub stonedriveway courtyard backyard attic basement
1          1         1     -1    1      1            -1        -1        1    -1       -1
2          1        -1     -1    1     -1            -1         1        1     1        1
3         -1         1      1   -1     -1            -1         1        1    -1        1
4         -1        -1      1   -1      1            -1        -1        1     1       -1
5         -1        -1      1    1      1            -1        -1       -1    -1        1
6         -1        -1     -1   -1      1             1         1        1    -1        1
7         -1        -1     -1    1      1             1         1       -1     1       -1
8         -1         1      1    1     -1            -1         1       -1     1       -1
9         -1         1     -1   -1     -1             1        -1        1     1       -1
10         1         1     -1   -1      1            -1        -1       -1     1        1
11         1        -1      1    1     -1             1        -1        1    -1       -1
12         1         1      1    1      1             1         1        1     1        1
13         1        -1      1   -1     -1             1        -1       -1     1        1
14         1        -1     -1   -1     -1            -1         1       -1    -1       -1
15        -1         1     -1    1     -1             1        -1       -1    -1        1
16         1         1      1   -1      1             1         1       -1    -1       -1
class=design, type= FrF2
```

If we want to find out about our design, we can run one more line of code:
This allows us to see what is aliasing with what and why, along with how many 2fis there are. It also lists the generators (generating factors) which would be important if we were going to take this further but I think this is as far as the homework is asking us to go.

> design.info(ffactorial)



```
$type
[1] "FrF2"

$nruns
[1] 16

$nfactors
[1] 10

$factor.names
$factor.names$largeyard
[1] -1  1

$factor.names$solarroof
[1] -1  1

$factor.names$garden
[1] -1  1

$factor.names$pool
[1] -1  1

$factor.names$hottub
[1] -1  1

$factor.names$stonedriveway
[1] -1  1

$factor.names$courtyard
[1] -1  1

$factor.names$backyard
[1] -1  1

$factor.names$attic
[1] -1  1

$factor.names$basement
[1] -1  1
```

```
$catlg.name
[1] "catlg"

$catlg.entry
Design:  10-6.1
   16  runs,  10  factors,
   Resolution  III
   Generating columns:  3 5 6 9 14 15
   WLP (3plus):  8 18 16 8 8 ,  0  clear 2fis

$aliased
$aliased$legend
 [1] "A=largeyard"     "B=solarroof"     "C=garden"        "D=pool"          "E=hottub"        "F=stonedriveway"
 [7] "G=courtyard"     "H=backyard"      "J=attic"         "K=basement"

$aliased$main
 [1] "A=BE=CF=DH=JK" "B=AE=CG"       "C=AF=BG"       "D=AH=GJ"       "E=AB=FG"       "F=AC=EG"       "G=BC=DJ=EF=HK"
 [8] "H=AD=GK"       "J=AK=DG"       "K=AJ=GH"

$aliased$fi2
[1] "AG=BF=CE=DK=HJ" "BD=CJ=EH=FK"    "BH=CK=DE=FJ"    "BJ=CD=EK=FH"    "BK=CH=DF=EJ"


$FrF2.version
[1] "2.2-2"

$replications
[1] 1

$repeat.only
[1] FALSE

$randomize
[1] TRUE

$seed
NULL

$creator
FrF2(16, 10, factor.names = list(largeyard = "", solarroof = "",
    garden = "", pool = "", hottub = "", stonedriveway = "",
    courtyard = "", backyard = "", attic = "", basement = ""))
```

Citations

1. This presentation was helpful in learning about fractional factorial designs, https://www.slideshare.net/tabraham/fractional-factorial-designs, Accessed on 3/21/2021
2. This source was helpful in explaining fractional factorial design, https://www.stat.purdue.edu/~yuzhu/stat514s2006/Lecnot/fracfacts2006.pdf, Accessed 3/23/2021

See following page for 13.1

Question 13.1

For each of the following distributions, give an example of data that you would expect to follow this distribution (besides the examples already discussed in class).

a. Binomial
b. Geometric
c. Poisson
d. Exponential
e. Weibull

a. In a Binomial Distribution, there needs to be a fixed number of trials, each trial needs to be independent of the others, with only two outcomes; success or failure, and the probability of those outcomes has to be consistent from trial to trial. A couple of examples come to mind, namely casino or carnival games. Some casino games are just based on simple probability so they would make ideal examples. For example, a dice game. Another pretty good example is free throw shooting in Basketball. One could argue that previous makes or misses affect the next shot but assuming that they don't (because they actually don't physically), this data set captures a binomial distribution perfectly.

b. A Geometric Distribution honors some of the conditions set above. The thing being modeled needs to occur as a sequence of independent trials, have only two possible outcomes, and have the same probability of success for each trial. However, a geometric distribution will follow a geometric sequence. In a geometric sequence, we are looking at the number of failures (Y) it takes to achieve a success or alternatively, the number of failures up to and including the first success (X-1). A dice game is also a good example here. For example, how many rolls does it take to roll a 2? A really good example is throwing a dart a board and seeing how many times it will take to throw a bullseye.

c. A Poisson Distribution is useful in predicting when we know the number of occurrences of a given event over a time period. This could be the number of rentals that a car rental business will do on a given day throughout a season or annually. In terms of business, one could use a Poisson Distribution in order to forecast sales on a given day or during a given season/ during some holiday sale, given that the company has operated in that season/time period before.

d. An Exponential Distribution is continuous and usually shown as a sort of downward curve. It's notable that there's an important link between the Poisson Distribution and the Exponential Distribution in that failures occurring according to a Poisson Distribution will have a time t which occurs according to an Exponential Distribution. Notable examples are radioactive decay and models of financial default.

e. A Weibull Distribution is continuous and gives a failure rate that is proportional to the power of time. K is given as a tuning parameter and allows one to tune the distribution to decreasing over time (k<1), constant(k=1) or increasing over time(k>1). Also, if you set k =2 you get a Rayleigh distribution which I'm not going to detail here. A solid real life example of a Weibull Distribution is a reliability analysis; how long can a particular machine or model of machine be expected to perform at or above a certain level before it needs to be maintenance or, in some cases, replaced?

Question 13.2

In this problem you, can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with λ_1 = 5 per minute (i.e., mean interarrival rate μ_1 = 0.2 minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate μ_2 = 0.75 minutes. [Hint: model them as one block that has more than one resource.] After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).

Use the Arena software (PC users) or Python with SimPy (PC or Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use λ_1 = 50 to simulate a busier airport.]

Caveat:
As a caveat to this assignment, I don't work in Python. I have some (very little) basic Python skills and I'm going to rely on tutorials to try to get through this. The challenge that I can see up front is that a lot of the tutorials will use a random distribution and teach someone how to seed the distribution as an experimental control. I'm not sure, at this point, how to use a Poisson distribution or how to specify parameters, such as those stated above, within this model. Anyway, I'm going to give it my best attempt. For this assignment, I'll be working in Visual Code Studio, in a Jupyter Notebook.

Ok, so, first we want to visualize our approach:
So, passengers arrive according to a Poisson distribution to the first queue (check-in). We are given parameters here for both arrival and interarrival, λ_1 and μ_1. Then, they go through the first set of servers. We are given a parameter here as well for service time, μ_2. Next, they are assigned to a new set of queues for security screenings. We are given a parameter here too for time between .5 and 1 and told that time is uniformly distributed.

First, we import simpy:

```
import simpy
```

At this point, I'm not sure if there's an easier way to do this but I'm going to go ahead and generate 1000 samples using a Poisson distribution and a λ_1 of 50 (for a busy airport).

I've altered some of the coloring in my code so that you, the reader, can see it properly.

```
import numpy as np
samples_1 = np.random.poisson(50,1000)
print(samples_1)
```

```
52 50 43 36 44 56 39 38 53 53 51 40 51 44 53 56 39 44 42 48 52 49 52 59 67 50 32 55 43
 50 51 47 35 49 54 44 44 48 50 42 48 53 45 50 54 63 53 51 42 52 53 46 49 45 57 51 53 4
9 44 55 51 51 50 62 45 49 44 44 54 54 47 54 42 61 60 52 57 56 60 47 59 60 67 53 52 49
50 34 38 60 52 51 48 46 45 57 57 49 45 41 44 52 52 43 54 47 53 46 59 65 49 57 47 51 57
 41 48 62 53 70 45 59 53 54 63 54 50 38 50 53 47 49 43 56 43 56 47 59 51 56 55 58 47 4
5 51 47 59 50 48 52 45 51 53 52 64 54 45 52 41 43 57 45 53 63 48 60 55 56 48 52 35 53
```

```
51 60 49 46 52 49 60 55 48 60 59 60 59 58 43 60 50 44 45 48 47 42 53 47 57 50 54 46 58
 43 56 47 41 50 61 59 46 52 44 51 47 52 62 42 62 46 57 64 52 46 55 52 51 48 47 55 44 4
2 45 41 54 51 53 52 54 55 37 38 54 52 61 43 49 37 37 45 47 53 48 56 58 55 46 55 55 55
51 60 44 56 27 53 62 36 49 45 56 62 54 44 44 44 44 49 51 44 53 52 48 53 42 46 57 58 56
 55 51 54 55 68 54 38 44 45 45 42 59 60 41 52 56 42 55 51 45 46 41 57 46 43 62 47 47 4
6 54 61 55 47 42 56 46 46 51 44 51 46 53 35 61 33 54 61 38 44 44 47 49 51 52 53 38 64
61 52 52 56 49 58 54 58 59 50 43 65 52 54 56 52 58 50 45 51 50 35 59 43 41 51 59 53 46
 37 36 54 45 44 42 45 48 59 49 55 56 54 39 53 57 55 57 58 55 37 43 49 64 58 56 52 45 4
8 44 40 48 52 71 53 53 43 48 51 50 55 56 44 54 64 52 41 38 47 50 45 47 46 51 47 48 51
54 46 64 50 50 60 45 49 55 54 60 46 66 43 53 44 40 44 55 52 57 37 65 65 49 47 44 33 57
 47 59 41 52 44 40 49 43 49 46 56 49 50 50 50 43 45 63 51 41 43 43 58 55 51 49 46 40 3
1 61 44 53 50 50 56 54 51 49 54 46 55 56 46 45 43 48 54 43 58 58 66 42 54 55 49 46 56
46 56 66 50 58 53 51 55 40 47 48 49 54 51 51 48 43 52 52 59 42 48 43 55 47 46 55 45 44
 53 66 52 48 42 46 46 54 39 51 64 41 51 47 67 40 60 51 57 59 49 51 48 42 41 39 39 49 4
9 49 62 42 56 45 50 49 57 51 39 45 36 39 37 65 43 48 52 46 47 38 44 61 61 58 61 62 50
41 50 37 51 35 52 61 50 52 51 54 54 54 51 39 46 60 57 56 51 50 57 54 59 46 45 47 44 59
 60 41 52 46 42 46 58 40 58 57 53 37 44 59 34 52 49 51 45 47 57 51 35 60 47 49 45 36 4
1 49 43 60 48 53 36 50 51 50 45 52 45 53 46 51 52 51 48 41 59 57 50 44 55 48 58 56 60
47 54 49 51 54 43 43 40 58 55 44 49 48 54 64 64 47 42 54 36 68 40 52 56 49 37 57 41 55
 46 53 51 50 46 44 65 61 53 41 45 65 62 56 53 62 48 46 47 36 38 50 44 53 47 48 29 52 3
2 50 44 43 44 50 50 55 55 49 52 52 55 54 53 43 55 40 53 52 52 44 41 40 51 43 51 48 35
42 48 50 52 59 49 63 38 55 46 52 49 62 66 61 75 54 54 50 37 51 41 52 58 55 46 49 43 56
 52 50 57 44 49 50 37 38 45 48 52 58 49 48 47 52 45 39 49 66 39 45 61 62 53 40 71 51 4
1 54 39 38 40 64 45 53 59 56 46 52 49 62 43 50 43 43 44 48 47 40 46 50 52 52 50 56 46
55 51 45 50 51 39 49 48 40 54 54 42 51 63 46 54 52 45 49 49 57 38 56 49 54 51 52 48 51
 61 50 59 42 54 37 47 47 57 48 53 54 53 51 58 57 47 43 55 43 47 45 54 54 33 57 50 55 5
0 54 52 61 53 57 52 49 43 53 47 56 57 45 46 60 54 43 53 47 57 39 43 40 54 53 37 50 47
48 53 49 59 56 45 43 51 49 44 43 45 57 53 50 46 61 61 54 50 50 56 45 65 43 56 49 60 46
 40 62 46 44 35 44 58 50 42 50 51 58 46 47 57 32 49 44 47 50 51 51 54 49 40]
```
[-]

The other possible approach involves just importing simpy and random, and then setting time as a frequency:

```
import simpy
import random
```

Either way, I think the way to go from here is to build the first queue first, the second queue second, and then connect everything into some kind of loop or just one experiment if no loop(s) are needed.

Ok, I've done a bit of research and it seems like we need to build a generator function to simulate arrivals, and then a second generator function that tracks those arrivals(people) as they move through the airport. These will probably use a while loop. We might also be able to get away with building a simpler model in the second scenario above.

Yep, I've confirmed that we don't need to sample the Poisson Distribution if we are using interarrival time as one of our variables in the first loop. In this case, we can use random.expovariate for an exponential distribution as corresponding to the Poisson Distribution.

The following is a list of possible variables that we may or may not need:
Note that I've chosen the number of servers (4) and the number of scanners (5). These numbers will matter as we move through this process. **Please see the following page for the list.**

```
RANDOM_SEED = 50
NUM_SERVERS = 4
NUM_SCANNERS = 5
MEAN_SERVICETIME = 0.75
PERSONALSCANNER = [0.5,1]
INTERARRIVAL = 0.2
TIMEOFSIM = 500
```

We're going to go ahead and build the first loop:
This is our generator function and while loop used for generating arrivals. The question in my mind right now is how do we account for multiple servers? We may not end up using my variable but instead, some math. Also, we may decide to reference the original λ_1 of 5 and not 50 given that it may be simpler to construct on the first attempt. Actually, if we wanted to do this, we could just use 10/INTERARRIVAL as opposed to 1.

```
env = simpy.Environment()

def arrival_generator(env, INTERARRIVAL, MEAN_SERVICETIME, SERVER):
    c_id = 0

    while True:


    ARRIVALINSTANCE = arrival_generator(env, MEAN_SERVICETIME, SERVER, c_id)

    env.process(ARRIVALINSTANCE)

    t= random.expovariate(1.0/INTERARRIVAL)
# Our Poisson Distribution is created here when we divide 1.0/0.2, which gives us 5,
  which is our lamda.
    yield env.timeout(t)
# Ok, I may be able to specify number of servers here by dividing by 4 but I'm going
to hold off and do this when I set up the resources
    c_id +=1
```

Ok, we're a lot of the way there. We've created a loop with two generator functions (see below):

Note that, while I'm pretty proud of this loop especially given that I don't really know much Python, the assignment is not yet complete. We need to add a second resource to this loop and figure out how to add up the time properly. **See following page**

```
import simpy
import random

def arrival_generator(env, INTERARRIVAL, MEAN_SERVICETIME, SERVER):
    c_id = 0

    while True:

        ARRIVALINSTANCE = arrival_generator(env, MEAN_SERVICETIME, SERVER,          c_
id)

        env.process(ARRIVALINSTANCE)

        t= random.expovariate(1.0/INTERARRIVAL)
#Our Poisson Distribution is created here when we divide 1.0/0.2, which gives us 5, wh
ich was our lamda.
        yield env.timeout(t)
# Ok, I may be able to specify number of servers here by dividing by 4 but I won't. I'
ll do that in the resources section, I think.

        c_id +=1

def activity_generator(env, MEAN_SERVICETIME, SERVER, c_id):
    time_entered_queue_for_server = env.now
    print ("Customer", c_id, "entered the queue at",
           time_entered_queue_for_server, sep= " ")

    with SERVER.request() as req:

        yield req

        time_left_queue_for_server = env.now
        print ("Customer", c_id, "left queue at",time_left_queue_for_server,
     sep= " ")
        time_in_queue_for_server = (time_left_queue_for_server -
                                    time_entered_queue_for_server)
        print ("Customer", c_id, "queued for", time_in_queue_for_server,
   "minutes", sep= " ")

        sampled_server_time = random.expovariate(1.0/MEAN_SERVICETIME)


        yield env.timeout(sampled_server_time)

env = simpy.Environment()

SERVER = simpy.Resource(env, capacity=4)
```

```
INTERARRIVAL = 0.2
MEAN_SERVICETIME = 0.75

env.process(arrival_generator(env, INTERARRIVAL, MEAN_SERVICETIME, SERVER))

env.run(until=180)
```

Ok, I checked this loop for errors and ran it. It worked just fine but it basically shut down my Jupyter Notebook and took forever. I downloaded Spyder so I could go faster. Now, everything is off, all my permissions for Python, etc. Given that I don't usually code in Python, I have no idea how to fix it quickly and it's 9pm on Wednesday night/time to go to sleep. I've managed to get mostly 100's so far with two 90's so that's a shame. I didn't manage to get the output off of it before it went down either. The output of it was super cool with messages that tracked each individual customer as they came to the airport and how long it took them to get through the servers, as well as overall activity for 3 hours. Note to self, use a set up that can handle this next time. Oh well, live and learn!

Hypothetically, going forward, I would have added in the PERSONALSCANNER variable as a part of the loop and/or made some edit to the yield to get both of the calculated times to add before sampling them.

I imagine, given the size of the airport (very small), and the $\boldsymbol{\lambda}_1$ that we used (5), our customers would have been getting through the airport in minutes.

Also, note that some of my variables changed or were excluded during the building of the loop but that's not surprising given that I taught myself a lot of new things to build it.

Citations

1. This source was helpful in understanding the Python needed to run SimPy and how to approach SimPy as well, https://realpython.com/simpy-simulating-with-python/, Accessed on 3/24/21
2. This source gave a good idea of some in depth approaches to multiple customers, https://pythonhosted.org/SimPy/Tutorials/TheBank.html, Accessed on 3/24/21
3. One of the most helpful tutorials of all time; relied on this heavily in constructing my own examples, https://www.youtube.com/watch?v=jXDjrWKcu6w, Accessed on 3/24/21
4. This tutorial was good too, https://pythonhosted.org/SimPy/Tutorials/TheBank.html#several-service-counters, Accessed on 3/24/21