CS415 - Modelling and simulations
Assignment 2 - Report

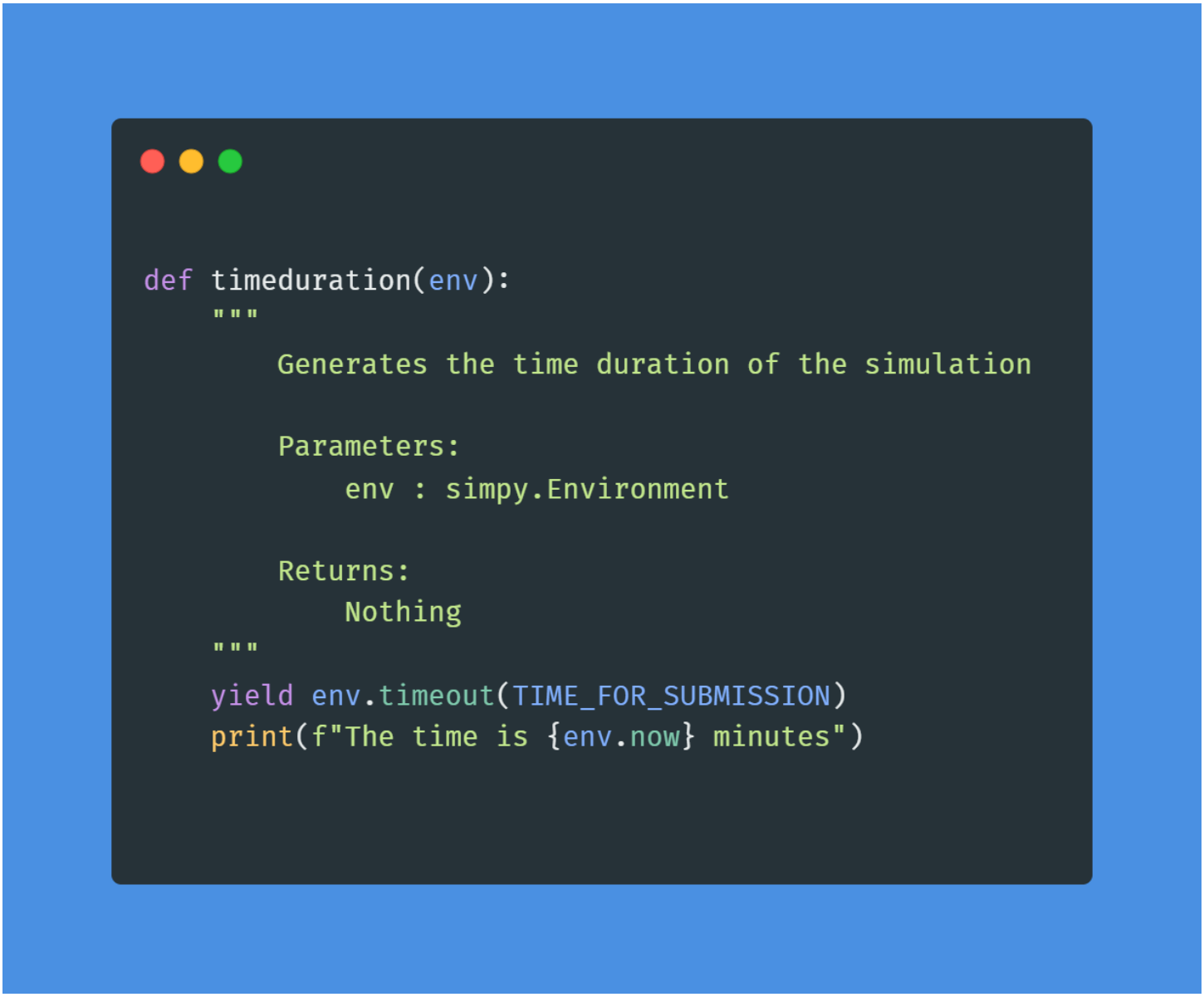
Feb 23, 2022

ADARSH ANAND
CSE
2003101

Q1) and Q2)
For the first part - Run the simulation once to show a detailed activity log showing all instances when power went/came back, what each student was doing and how much work was remaining.

```
'''Defining the constants'''

# Unit of time = minutes

NUMBER_OF_STUDENTS = 10  # Number of students in the simulation
TIME_FOR_SUBMISSION = 7 * 60  # Time for submission in minutes for each student

SLEEP_PROBABILITY = 0.4  # Probability of a student sleeping

# Number of students who finished assignment
NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT = 0
NUMBER_OF_POWER_CUTS = 0  # Number of power cuts
TIME_TAKEN_TO_SUBMIT = []  # List of time taken to submit for each student
COUNTER_STUDENT_ENVIRONMENT = 0
```

```python
def Power_Toggle(env):
    """
        Simulates the power cut event

        env : simpy.Environment
    """
    global POWER_DOWN_EVENT  # Event that is triggered when power is cut
    global POWER_UP_EVENT  # Event that is triggered when power is back on
    global NUMBER_OF_POWER_CUTS

    while True:  # Infinite loop
        print(f"🌕 Power is on at {env.now} minutes")
        yield env.timeout(random.randint(60, 2 * 60))
        POWER_DOWN_EVENT.succeed()  # Power is down now
        POWER_UP_EVENT = simpy.Event(env)
        NUMBER_OF_POWER_CUTS += 1

        print(f"🌑 Power is off at {env.now} minutes")
        # power down for 10-15 minutes
        yield env.timeout(random.randint(10, 15))
        POWER_UP_EVENT.succeed()  # Power is back on now
        POWER_DOWN_EVENT = simpy.Event(env)
```



```
def timeduration(env):
    """
        Generates the time duration of the simulation

        Parameters:
            env : simpy.Environment

        Returns:
            Nothing
    """
    yield env.timeout(TIME_FOR_SUBMISSION)
    print(f"The time is {env.now} minutes")
```

Time duration

```python
class Student:
    """
        Class to model the behavior of each student in the simulation

        Parameters:
            self : simpy.Environment
            env : simpy.Environment
            id : int
            SLEEP_PROBABILITY : float

        Returns :
            simpy.Process
    """

    def __init__(self, env, id, SLEEP_PROBABILITY):
        """ Initialize the student class """

        # work reqd for each student in minutes (4-5 hours)
        self.REQD_WORK = random.randint(4 * 60, 5 * 60)
        self.env = env  # environment
        self.id = id  # student id (unique)
        self.behavior_process = env.process(
            self.behavior())  # student behavior
        self.SLEEP_PROBABILITY = SLEEP_PROBABILITY  # probability of falling asleep
```

Main class

```
    def behavior(self):
        """ Modelling a Student behavior """

        # counter for the number of students that finished the assignment on time
        global NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT
        global NUMBER_OF_POWER_CUTS
        global TIME_TAKEN_TO_SUBMIT
        global COUNTER_STUDENT_ENVIRONMENT

        print(
            f"TIME={self.env.now} - 🙂 Student #{self.id} started working.Work left=
{self.REQD_WORK} minutes"
        )

        while True:
            START_TIME = env.now  # start time of the student

            # Student completes the work
            timeout_event = env.timeout(self.REQD_WORK)
            # wait for completing assignment event or power down event
            ret = yield timeout_event | POWER_DOWN_EVENT

            if timeout_event in ret:  # Student completes the work
                COUNTER_STUDENT_ENVIRONMENT += 1
                NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT += 1
                print(
                    f"TIME={self.env.now} - 😴 Student #{self.id} finished working.")

                TIME_TAKEN_TO_SUBMIT.append(
                    self.env.now)  # Time taken to submit

                if COUNTER_STUDENT_ENVIRONMENT == NUMBER_OF_STUDENTS:
                    # If all students finished the assignment on time
                    ALL_FINISHED.succeed()

                return
```

work Finished

```
    else:  # Power cut event
                # deduct the time taken by the student to complete the work
                self.REQD_WORK -= (self.env.now - START_TIME)

                print(
                    f"TIME={self.env.now} - 😞 Student #{self.id} interrupted by power
cut. Task remaining={self.REQD_WORK} minutes"
                )  # print the time when the student is interrupted

                yield POWER_DOWN_EVENT  # wait for power to come back

                if random.uniform(0, 1) <= self.SLEEP_PROBABILITY:
                    # if the student falls asleep
                    COUNTER_STUDENT_ENVIRONMENT += 1
                    # wait for 5 minutes for power to come back
                    yield env.timeout(5)

                    print(
                        f"TIME={self.env.now} - 😴 Student #{self.id} slept. Assignment
pending. Work left was={self.REQD_WORK} minutes"
                    )

                    if COUNTER_STUDENT_ENVIRONMENT == NUMBER_OF_STUDENTS:
                        ALL_FINISHED.succeed()  # If all students finished the assignment
on time

                    return

                else:
                    # if the student wakes up ans resumes working
                    yield POWER_UP_EVENT
                    print(
                        f"TIME={self.env.now} - 😃 Student #{self.id} resumes working.
Work left to do={self.REQD_WORK} minutes"
                    )
```

work remaining/Power interupt

```
''' Main simpy function code '''
env = simpy.Environment()  # create the environment

# Event that is triggered when power is cut
POWER_DOWN_EVENT = simpy.Event(env)
# Event that is triggered when power is back on
POWER_UP_EVENT = simpy.Event(env)

Power_Toggle_process = env.process(Power_Toggle(env))  # power toggle process
Time_Duration_process = env.process(timeduration(env))  # time duration process
Student_process = []  # list of student processes

for id in range(NUMBER_OF_STUDENTS):
    Student_obj = Student(env, id, SLEEP_PROBABILITY)
    Student_process.append(Student_obj.behavior_process)

# Event that is triggered when all students finished the assignment
ALL_FINISHED = simpy.Event(env)
TIME_OVER = env.process(timeduration(env))  # time duration process

# run the simulation until all students finished the assignment or time duration is over
env.run(until=ALL_FINISHED | TIME_OVER)
NUMBER_OF_STUDENTS_SLEPT = NUMBER_OF_STUDENTS - \
    NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT
```

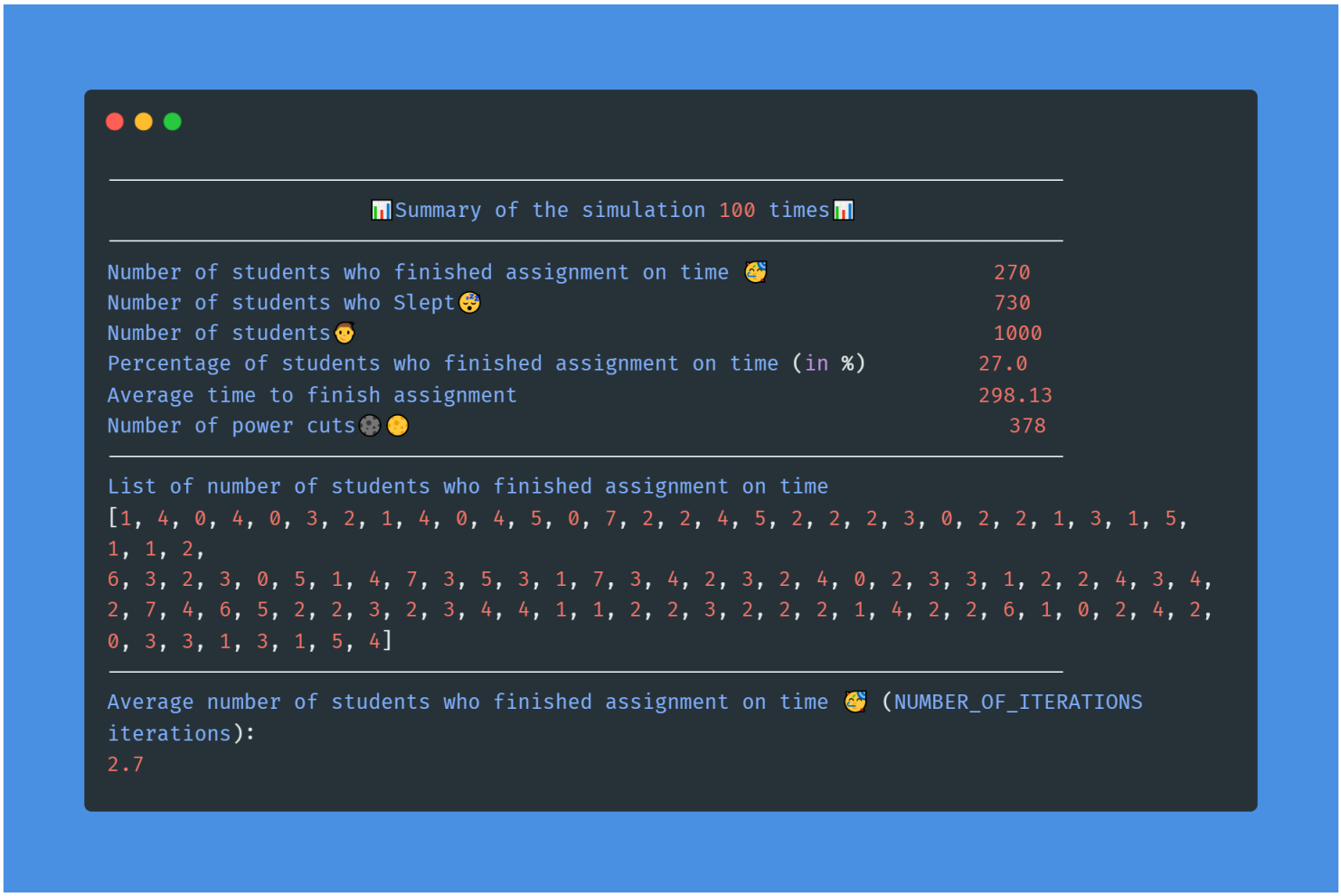
```
'''Summary of the simulation'''

dict = {
    "Number of students who finished assignment on time 😴":
NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT,
    "Number of students who Slept😴": NUMBER_OF_STUDENTS_SLEPT,
    "Number of students🧑": NUMBER_OF_STUDENTS,
    "Percentage of students who finished assignment on time (in %)":
(NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT / NUMBER_OF_STUDENTS)*100,
    "Average time to finish assignment": Average(TIME_TAKEN_TO_SUBMIT),
    "Number of power cuts🌑🌕": NUMBER_OF_POWER_CUTS,
}

print('————————————————————————————————————————————————————————————————————————————————')
print('{:<20} {:<20} {:<20}'.format(" ", "📊Summary of the simulation📊", " "))
print('————————————————————————————————————————————————————————————————————————————————')

for key, value in dict.items():
    print(f"{key:<{70}}{value}")

print('————————————————————————————————————————————————————————————————————————————————')
```

```
————————————————————————————————————————————————————————————————————————————————
                    📊Summary of the simulation📊
————————————————————————————————————————————————————————————————————————————————
Number of students who finished assignment on time 😴                3
Number of students who Slept😴                                       7
Number of students🧑                                                 10
Percentage of students who finished assignment on time (in %)        30.0
Average time to finish assignment                                    297.3333333333333
Number of power cuts🌑🌕                                              2
————————————————————————————————————————————————————————————————————————————————
```

Q3) Using 100 independent simulation runs, find out how many out of the ten students on average end up completing the assignment in time. Let us denote this quantity as N.

z

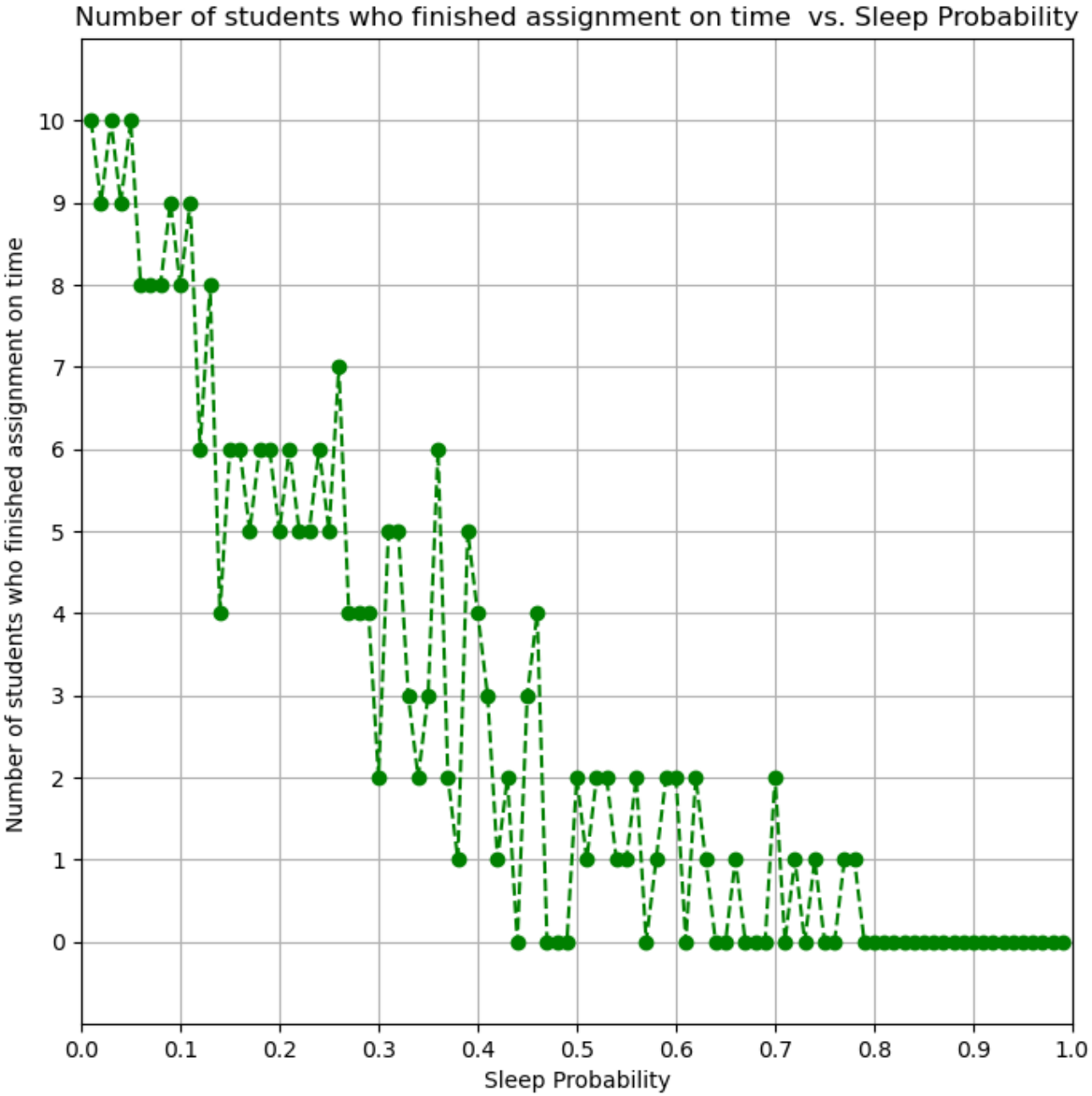
```
dict = {
    "Number of students who finished assignment on time 😴": sum(N),
    "Number of students who Slept🥱": NUMBER_OF_ITERATIONS*10 - sum(N),
    "Number of students👨": NUMBER_OF_STUDENTS*NUMBER_OF_ITERATIONS,
    "Percentage of students who finished assignment on time (in %)": (sum(N) /
(NUMBER_OF_STUDENTS*NUMBER_OF_ITERATIONS))*100,
    "Average time to finish assignment": Average(TIME_TAKEN_TO_SUBMIT),
    "Number of power cuts🛞🌕": NUMBER_OF_POWER_CUTS,
}

print('————————————————————————————————————————————————————————————————————————————————')
print('{:<20} {:<20} {:<20}'.format(
    " ", "📊Summary of the simulation 100 times📊", " "))
print('————————————————————————————————————————————————————————————————————————————————')

for key, value in dict.items():
    print(f"{key:<{70}}{value}")

print('————————————————————————————————————————————————————————————————————————————————')

print(f"List of number of students who finished assignment on time")
print(N)
print('————————————————————————————————————————————————————————————————————————————————')
print("Average number of students who finished assignment on time 😴
(NUMBER_OF_ITERATIONS iterations):", Average(N))
print('————————————————————————————————————————————————————————————————————————————————')
```


📊Summary of the simulation 100 times📊
Number of students who finished assignment on time 😴 270
Number of students who Slept🥱 730
Number of students👨 1000
Percentage of students who finished assignment on time (in %) 27.0
Average time to finish assignment 298.13
Number of power cuts🌑🌕 378
List of number of students who finished assignment on time
[1, 4, 0, 4, 0, 3, 2, 1, 4, 0, 4, 5, 0, 7, 2, 2, 4, 5, 2, 2, 2, 3, 0, 2, 2, 1, 3, 1, 5,
1, 1, 2,
6, 3, 2, 3, 0, 5, 1, 4, 7, 3, 5, 3, 1, 7, 3, 4, 2, 3, 2, 4, 0, 2, 3, 3, 1, 2, 2, 4, 3, 4,
2, 7, 4, 6, 5, 2, 2, 3, 2, 3, 4, 4, 1, 1, 2, 2, 3, 2, 2, 2, 1, 4, 2, 2, 6, 1, 0, 2, 4, 2,
0, 3, 3, 1, 3, 1, 5, 4]
Average number of students who finished assignment on time 😴 (NUMBER_OF_ITERATIONS
iterations):
2.7

Q4) Obtain a plot of how N varies with p as p ranges from 0 to 1.

```
''' Main simpy function code '''
Student_process = []  # list of student processes

N = []  # list of number of students who finished assignment
RANGE_OF_PROBABILITY = []  # list of range of probability of sleeping
SLEEP_PROBABILITY = 0.00  # Probability of a student sleeping

for p in range(99):
    SLEEP_PROBABILITY+=0.01
    NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT = 0

    env = simpy.Environment()  # create the environment

    # Event that is triggered when power is cut
    POWER_DOWN_EVENT = simpy.Event(env)
    # Event that is triggered when power is back on
    POWER_UP_EVENT = simpy.Event(env)

    Power_Toggle_process = env.process(
        Power_Toggle(env))  # power toggle process
    timeduration_process = env.process(
        timeduration(env))  # timeduration process

    for id in range(NUMBER_OF_STUDENTS):
        Student_obj = Student(env, id, SLEEP_PROBABILITY)
        Student_process.append(Student_obj.behavior_process)

    # Event that is triggered when all students finished the assignment
    ALL_FINISHED = simpy.Event(env)
    TIME_OVER = env.process(timeduration(env))  # time duration process

    # run the simulation until all students finished the assignment or time duration is
over
    env.run(until=ALL_FINISHED | TIME_OVER)
    NUMBER_OF_STUDENTS_SLEPT = NUMBER_OF_STUDENTS - \
        NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT

    N.append(NUMBER_OF_STUDENTS_FINISHED_ASSIGNMENT)
    RANGE_OF_PROBABILITY.append(SLEEP_PROBABILITY)

    print('\n')
```

```
''' Plotting the graph '''

plt.figure(figsize=(8, 8))
plt.plot(RANGE_OF_PROBABILITY, N,marker='o', linestyle='--', color='green')
plt.title("Number of students who finished assignment on time  vs. Sleep Probability")
plt.xlabel("Sleep Probability")
plt.ylabel("Number of students who finished assignment on time ")
plt.xlim(0, 1)
plt.xticks(np.arange(0, 1.1, 0.1))
plt.yticks(range(0, 11, 1))
plt.ylim(-1, NUMBER_OF_STUDENTS+1)
plt.grid()
plt.show()
```


Number of students who finished assignment on time vs. Sleep Probability
Number of students who finished assignment on time
Sleep Probability
0
1
2
3
4
5
6
7
8
9
10
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

Q5) In the average case, for what value of p would you say that it is worth while to take a coffee break at every instance of power failure? To be specific, find out if N(with coffee breaks) is always better than N(without coffee breaks), whether this depends on the value of p, and for what value of p would you say that one approach is better than theother.

```
''' Coffee break '''
                if(random.uniform(0,1)⩽self.SLEEP_PROBABILITY):
                    print(f"TIME={self.env.now} - 🥱 Student #{self.id} is having coffee.Task
remaining={self.REQD_WORK} minutes")

                    yield env.timeout(30)

                    print(f"TIME={self.env.now} - Student #{self.id} finished having coffee.Task
remaining={self.REQD_WORK} minutes")
```

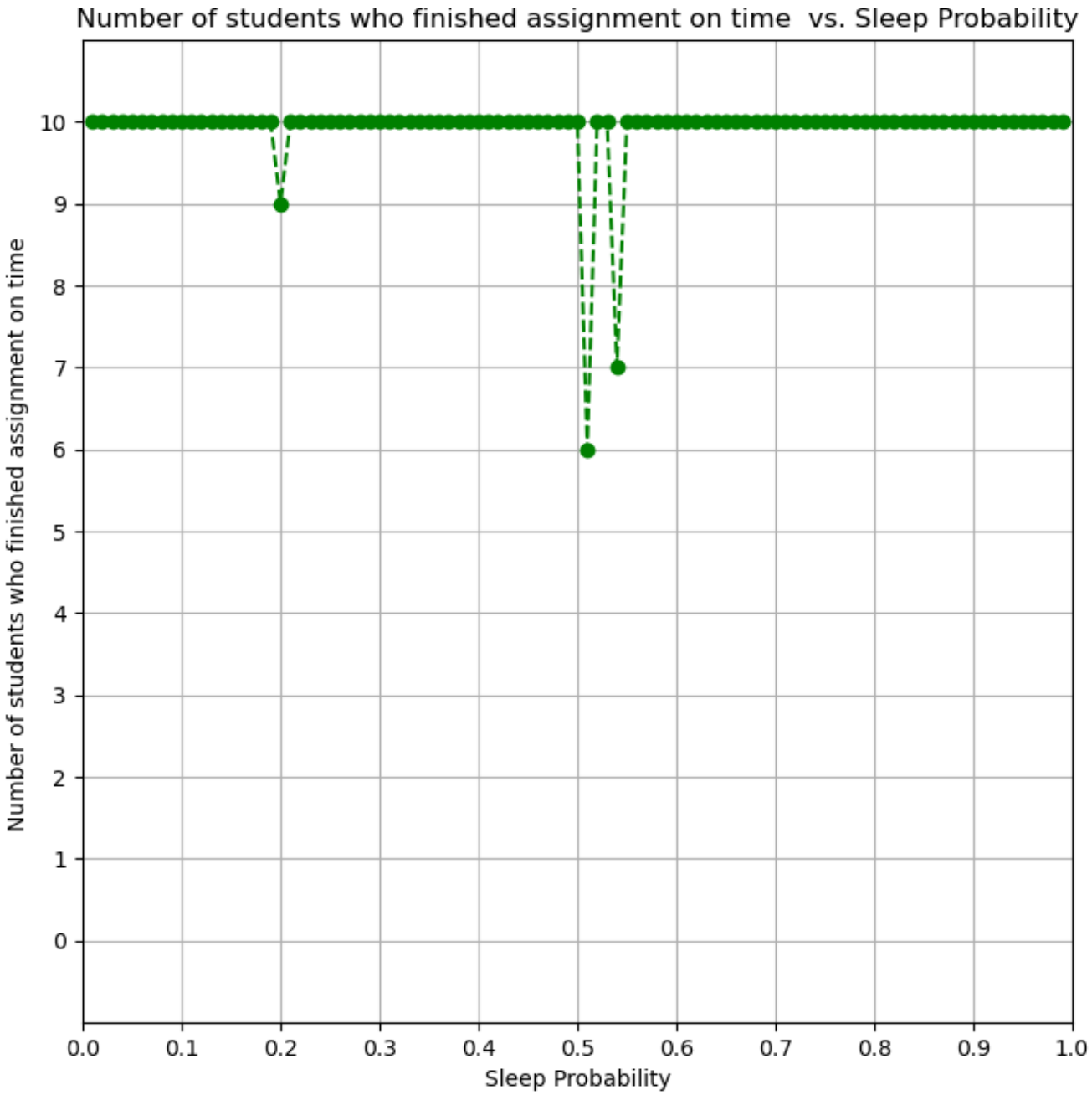


Fig - with coffee effect almost all people finish assignment

For all values of p taking into account, a coffee break is always better. i.e. p>0