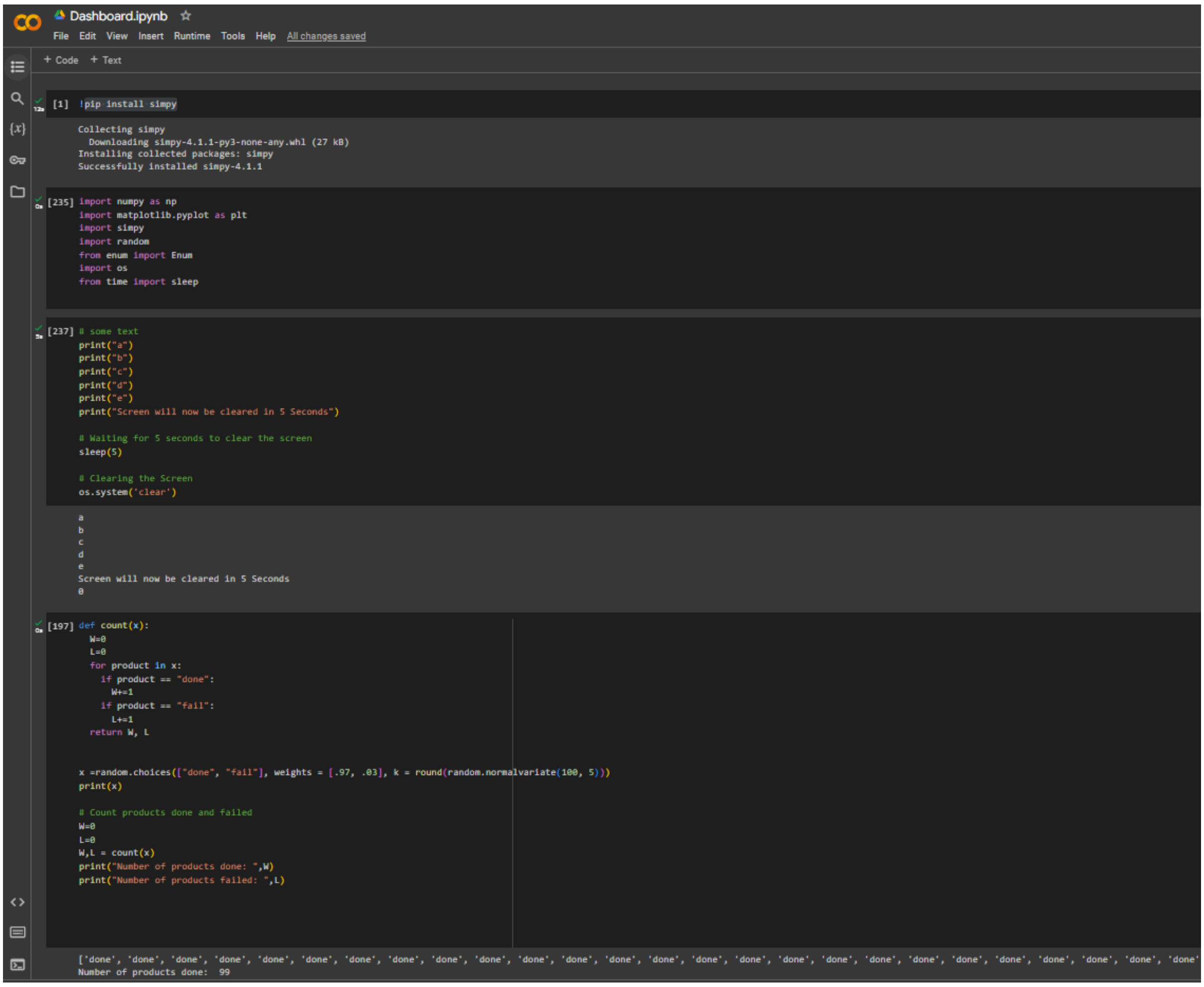
Souareba Sylla
Carlos Fernando Villaseñor Rábago

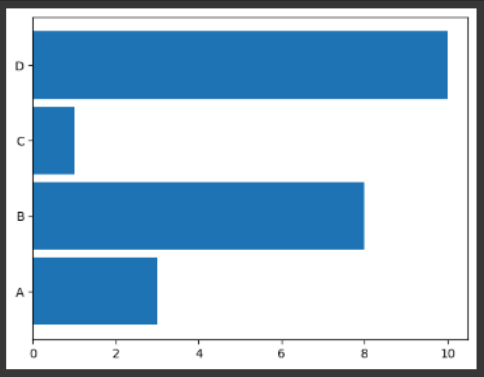
Dashboard Document

We decided to use Google Colab the exact day the teacher taught us how to use it. With this we practiced and experimented multiple ways to print out a graph with Matplotlib, other numpy functions to pull out random variables and other things in python that would help us develop this project.



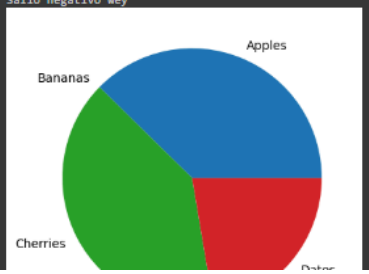
Souareba Sylla
Carlos Fernando Villaseñor Rábago

```
[3]  # For testing stuff

     x = np.array(["A", "B", "C", "D"])
     y = np.array([3, 8, 1, 10])

     plt.barh(x, y, height = 0.9)
     plt.show()
```




```
[190]
     def spawn(x):
       r= round(random.normalvariate(x, x/2))
       if not r < 0:
         return r
       else:
         print("Salio negativo wey")
         r = 0
         return r



     y = np.array([spawn(15), spawn(30), spawn(45), spawn(10)])
     mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

     plt.pie(y, labels = mylabels)
     plt.show()
```

```
Salio negativo wey
```



Souareba Sylla
Carlos Fernando Villaseñor Rábago

[5]
```python
#Class exercise

class Glider(object):

    def __init__(self, id: int, env: simpy.Environment, boost: simpy.Resource) -> None:
        self.env = env
        self.fly_distance = 0.0
        self.id = id
        self.boost = boost
        self.action = self.env.process(self.glide())


    def glide(self) -> None:
        while True:
            try:
                print('I am gliding (Glider %d) starting at %.2f' % (self.id, self.env.now))
                gliding_time = abs(random.normalvariate(4,0))
                print("Glide (Glider %d): %.2f" % (self.id, gliding_time))
                yield self.env.timeout(gliding_time)
                if gliding_time >= 3:
                    print("Should Boost (Glider %d)" % self.id)
                    with self.boost.request() as req:
                        yes = yield req | self.env.timeout(7)
                        if yes:
                            print("Boost (Glider %d)" % self.id)
                            gliding_time *= 1.2
                        else:
                            print("No Boost (Glider %d)" % self.id)

                # print("Glide_a: %.2f" % gliding_time)
                self.fly_distance += gliding_time

                # print('Wind is gone at %.2f' % self.env.now)
                no_wind_time = abs(random.normalvariate(0,0))
                print("No wind (Glider %d): %.2f" % (self.id, no_wind_time))
                yield self.env.timeout(no_wind_time)

            except simpy.Interrupt:
                print("Glider %d down at %.2f" % (self.id, self.env.now))
                return
```

[6]
```python
#Class exercise

def netShooter(env: simpy.Environment, glider: Glider):

    shoot_time = abs(random.normalvariate(15,0))

    print('Shooting in %.2f' % shoot_time)

    yield env.timeout(shoot_time)

    glider.action.interrupt()
```

[7]
```python
#Class exercise

env = simpy.Environment()

booster = simpy.Resource(env, 1)
glider1 = Glider(1, env, booster)
```

Souareba Sylla
Carlos Fernando Villaseñor Rábago

```
[7] print('1: I flew for %.2f miles' % (glider1.fly_distance*10.0))
```

```
I am gliding (Glider 1) starting at 0.00
Glide (Glider 1): 4.00
Shooting in 15.00
Should Boost (Glider 1)
Boost (Glider 1)
No wind (Glider 1): 0.00
I am gliding (Glider 1) starting at 4.00
Glide (Glider 1): 4.00
Should Boost (Glider 1)
Boost (Glider 1)
No wind (Glider 1): 0.00
I am gliding (Glider 1) starting at 8.00
Glide (Glider 1): 4.00
Should Boost (Glider 1)
Boost (Glider 1)
No wind (Glider 1): 0.00
I am gliding (Glider 1) starting at 12.00
Glide (Glider 1): 4.00
Glider 1 down at 15.00
1: I flew for 144.00 miles
```

Actual Project

```
[8] class Day(object):
        def __init__(self, dayNumber, expenses) -> None:
            self.dayNumber = dayNumber
            self.expenses = expenses
            self.income = 0
            self.net_income = 0
            self.cash_balance = 0

        def calculate_income(self):
            return random.randint(incomeMin, incomeMax)

        def simulate_day(self):
            self.income = self.calculate_income()
            self.net_income = self.income - self.expenses
            self.cash_balance += self.net_income

        def display_status(self):
            print(f"Day {self.dayNumber}:")
            print(f"Income: {self.income}")
            print(f"Expenses: {self.expenses}")
            print(f"Net Income: {self.net_income}")
            print(f"Cash Balance: {self.cash_balance}")
            print()

        def calculateTotal(self,IncTot,ExpTot,NetIncTot,CashBalTot):
          IncTot += self.income
          ExpTot += self.expenses
          NetIncTot += self.net_income
          CashBalTot += self.cash_balance
          return IncTot,ExpTot,NetIncTot,CashBalTot



    class Factory(object):
        def __init__(self, starting_cash, expenses, incomeMin, incomeMax, ndays) -> None:
            self.starting_cash = starting_cash
            self.expenses = expenses
            self.income_min = incomeMin
            self.income_max = incomeMax
            self.ndays = ndays
            self.days = []
```

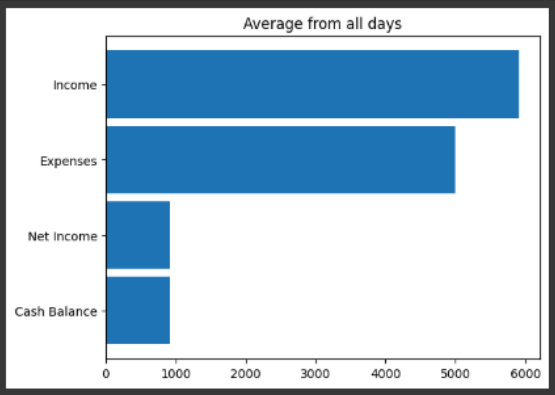
Souareba Sylla
Carlos Fernando Villaseñor Rábago

```python
        print(f"Cash Balance: {self.cash_balance}")
        print()

    def calculateTotal(self,IncTot,ExpTot,NetIncTot,CashBalTot):
      IncTot += self.income
      ExpTot += self.expenses
      NetIncTot += self.net_income
      CashBalTot += self.cash_balance
      return IncTot,ExpTot,NetIncTot,CashBalTot



class Factory(object):
    def __init__(self, starting_cash, expenses, incomeMin, incomeMax, ndays) -> None:
        self.starting_cash = starting_cash
        self.expenses = expenses
        self.income_min = incomeMin
        self.income_max = incomeMax
        self.ndays = ndays
        self.days = []

    def simulate_business(self):
        cash = self.starting_cash
        for dayNumber in range(1, self.ndays + 1):
            day = Day(dayNumber, self.expenses)
            day.simulate_day()
            self.days.append(day)

    def display_status(self):
        for day in self.days:
            day.display_status()

    def calculateTotal(self,IncTot,ExpTot,NetIncTot,CashBalTot):
        for day in self.days:
            IncTot,ExpTot,NetIncTot,CashBalTot = day.calculateTotal(IncTot,ExpTot,NetIncTot,CashBalTot)
        return IncTot,ExpTot,NetIncTot,CashBalTot




# Initial variables
starting_cash = 10000
expenses = 5000
incomeMin = 4000
incomeMax = 8000
ndays = 3
IncTot = 0
ExpTot = 0
NetIncTot = 0
CashBalTot = 0
array= []

# Run the simulation
simulation = Factory(starting_cash, expenses, incomeMin, incomeMax, ndays)
simulation.simulate_business()
simulation.display_status()

IncTot,ExpTot,NetIncTot,CashBalTot = simulation.calculateTotal(IncTot,ExpTot,NetIncTot,CashBalTot)

print()
print(IncTot,ExpTot,NetIncTot,CashBalTot)
```

```python
def average(x):
  return x/ndays

x = np.array(["Cash Balance", "Net Income", "Expenses","Income"])
```

Souareba Sylla
Carlos Fernando Villaseñor Rábago

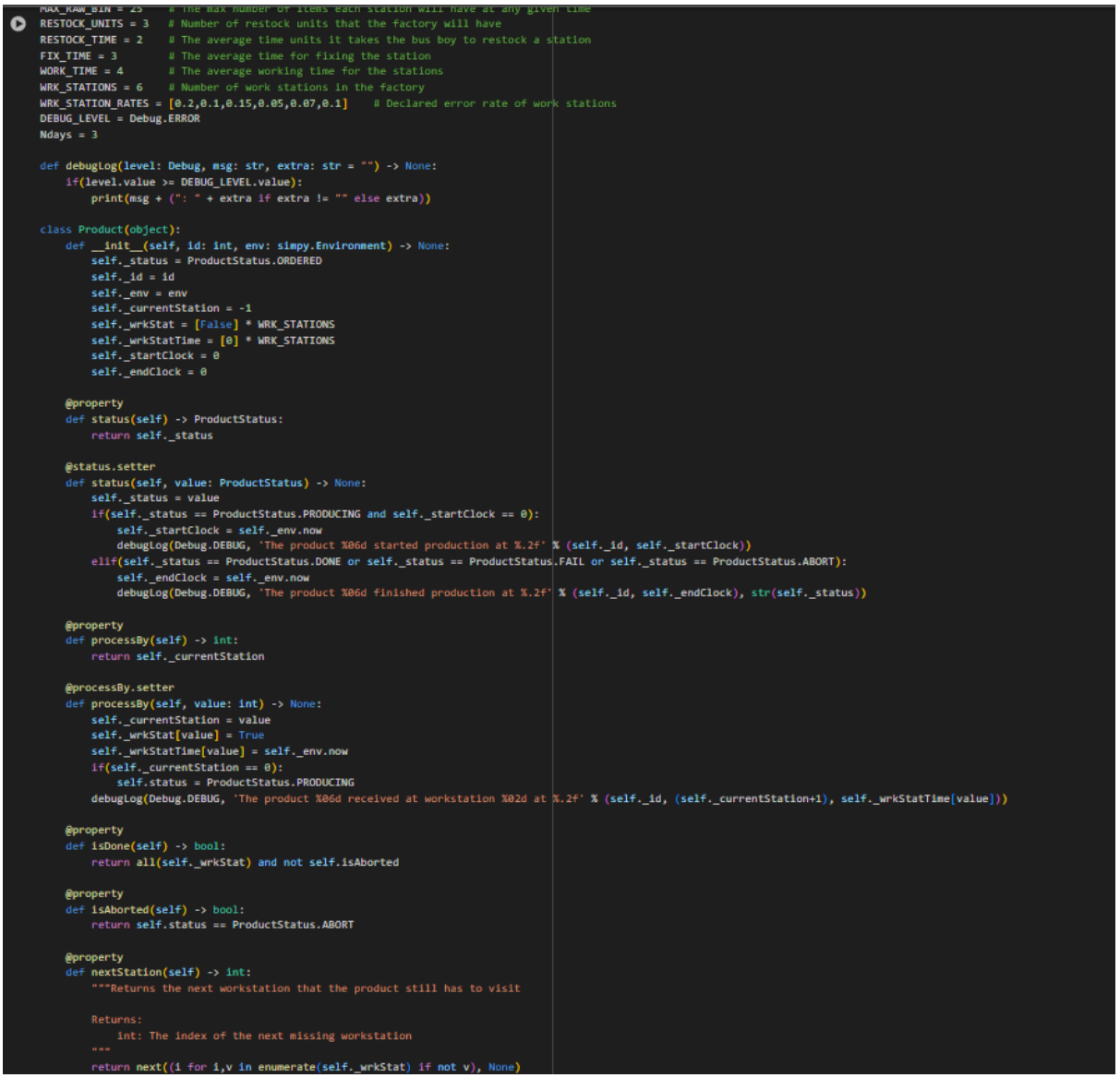
```
def average(x):
[9]   return x/ndays

    x = np.array(["Cash Balance", "Net Income", "Expenses","Income"])
    y = np.array([average(CashBalTot),average(NetIncTot),average(ExpTot),average(IncTot)])

    plt.barh(x, y, height = 0.9)
    plt.title("Average from all days")
    plt.show()
```



Project

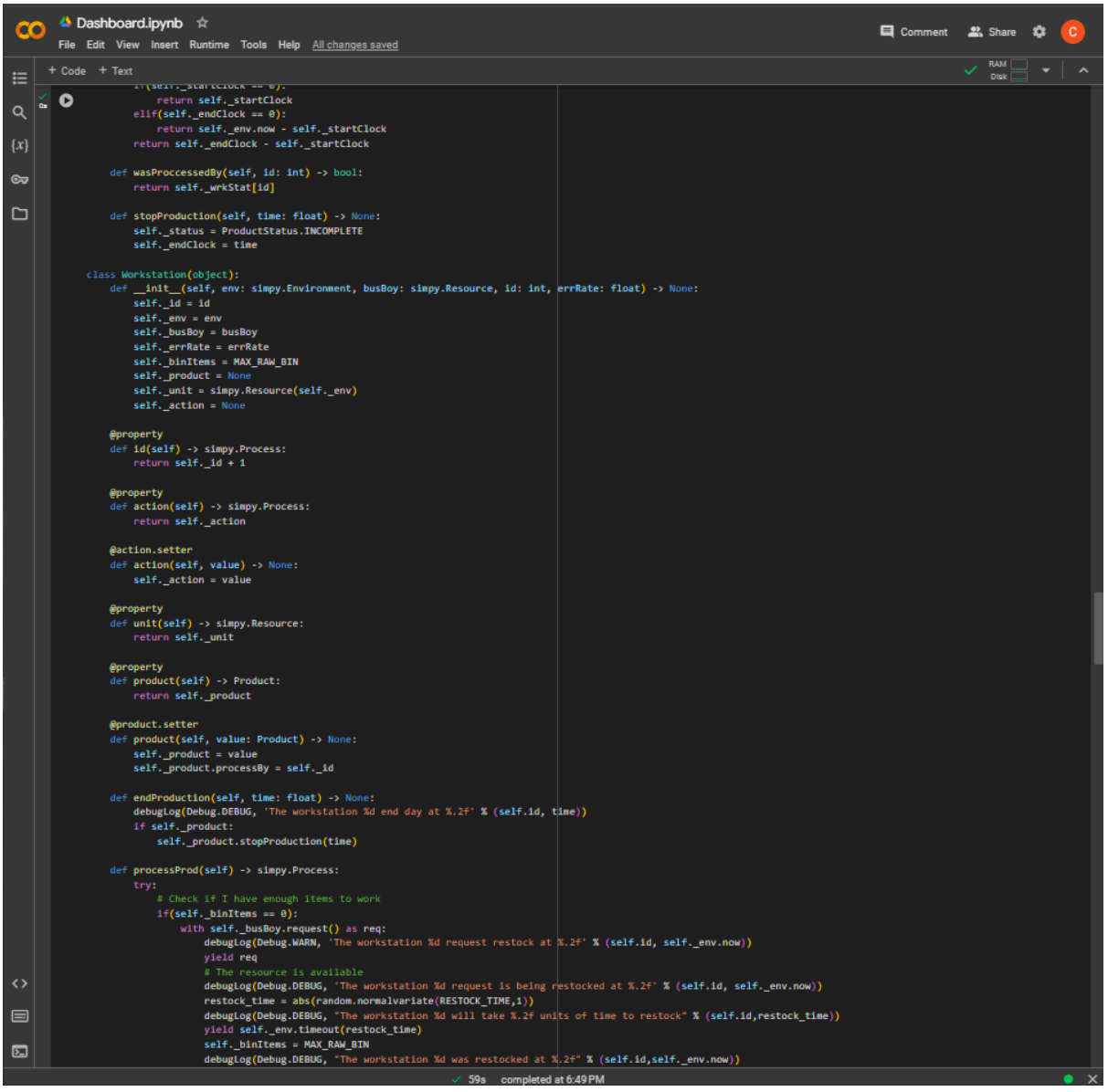
```
[10] class Debug(Enum):
        DEBUG = 0
        INFO = 1
        WARN = 2
        ERROR = 3
        FATAL = 4

    class WrkStationStatus(Enum):
        START = 1
        IDLE = 2
        PRODUCING = 3
        RESTOCK = 4
        DOWN = 5
        STOP = 6

        def __str__(self):
            return str(self.name)

    class ProductStatus(Enum):
        ORDERED = 1
        PRODUCING = 2
        DONE = 3
        FAIL = 4
        ABORT = 5
        INCOMPLETE = 6

        def __str__(self):
            return str(self.name)
```

Souareba Sylla
Carlos Fernando Villaseñor Rábago

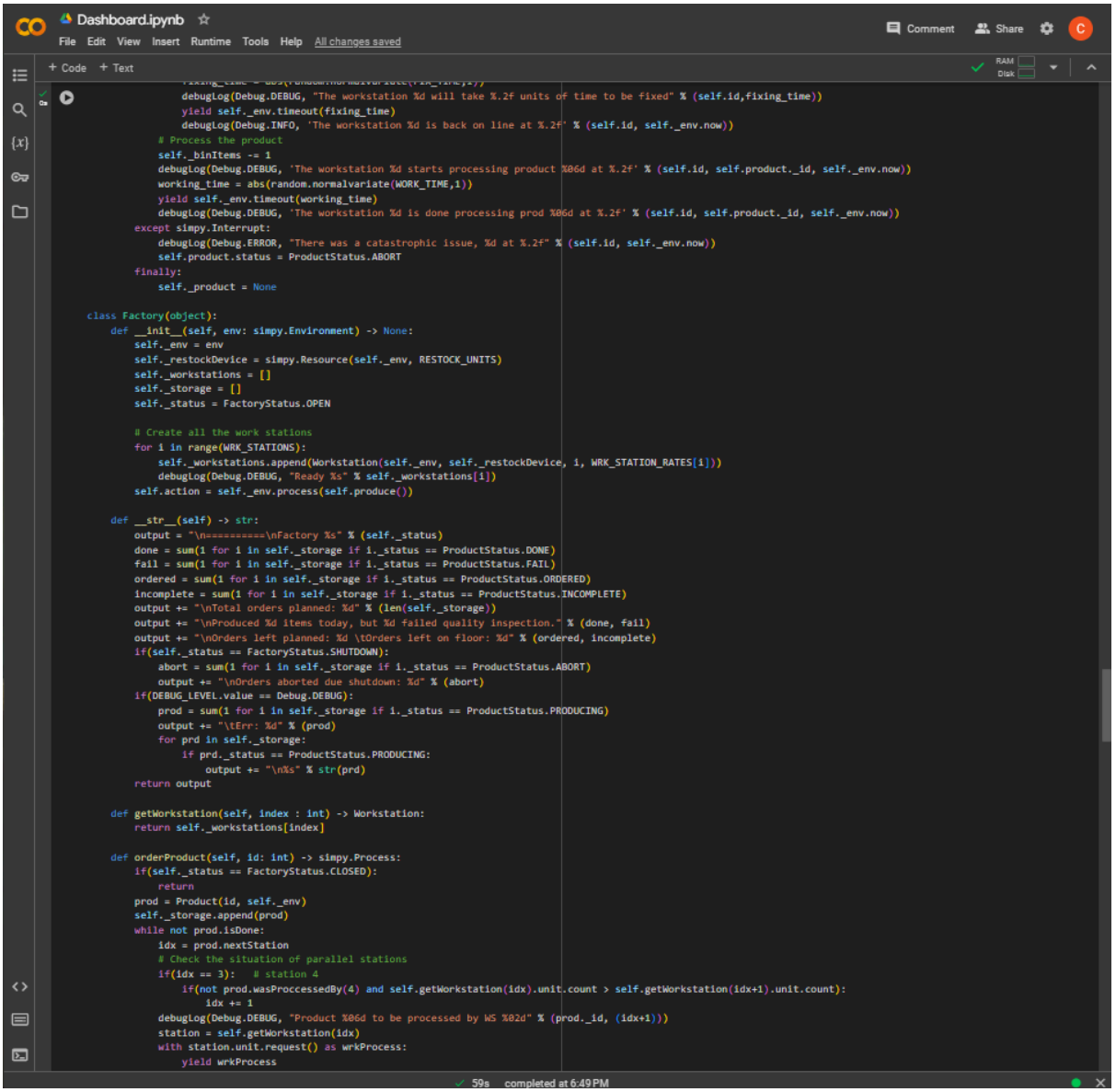


```
MAX_RAW_BIN = 25     # The max number of items each station will have at any given time
RESTOCK_UNITS = 3    # Number of restock units that the factory will have
RESTOCK_TIME = 2     # The average time units it takes the bus boy to restock a station
FIX_TIME = 3         # The average time for fixing the station
WORK_TIME = 4        # The average working time for the stations
WRK_STATIONS = 6     # Number of work stations in the factory
WRK_STATION_RATES = [0.2,0.1,0.15,0.05,0.07,0.1]    # Declared error rate of work stations
DEBUG_LEVEL = Debug.ERROR
Ndays = 3

def debugLog(level: Debug, msg: str, extra: str = "") -> None:
    if(level.value >= DEBUG_LEVEL.value):
        print(msg + (": " + extra if extra != "" else extra))

class Product(object):
    def __init__(self, id: int, env: simpy.Environment) -> None:
        self._status = ProductStatus.ORDERED
        self._id = id
        self._env = env
        self._currentStation = -1
        self._wrkStat = [False] * WRK_STATIONS
        self._wrkStatTime = [0] * WRK_STATIONS
        self._startClock = 0
        self._endClock = 0

    @property
    def status(self) -> ProductStatus:
        return self._status

    @status.setter
    def status(self, value: ProductStatus) -> None:
        self._status = value
        if(self._status == ProductStatus.PRODUCING and self._startClock == 0):
            self._startClock = self._env.now
            debugLog(Debug.DEBUG, 'The product %06d started production at %.2f' % (self._id, self._startClock))
        elif(self._status == ProductStatus.DONE or self._status == ProductStatus.FAIL or self._status == ProductStatus.ABORT):
            self._endClock = self._env.now
            debugLog(Debug.DEBUG, 'The product %06d finished production at %.2f' % (self._id, self._endClock), str(self._status))

    @property
    def processBy(self) -> int:
        return self._currentStation

    @processBy.setter
    def processBy(self, value: int) -> None:
        self._currentStation = value
        self._wrkStat[value] = True
        self._wrkStatTime[value] = self._env.now
        if(self._currentStation == 0):
            self.status = ProductStatus.PRODUCING
        debugLog(Debug.DEBUG, 'The product %06d received at workstation %02d at %.2f' % (self._id, (self._currentStation+1), self._wrkStatTime[value]))

    @property
    def isDone(self) -> bool:
        return all(self._wrkStat) and not self.isAborted

    @property
    def isAborted(self) -> bool:
        return self.status == ProductStatus.ABORT

    @property
    def nextStation(self) -> int:
        """Returns the next workstation that the product still has to visit

        Returns:
            int: The index of the next missing workstation
        """
        return next((i for i,v in enumerate(self._wrkStat) if not v), None)
```

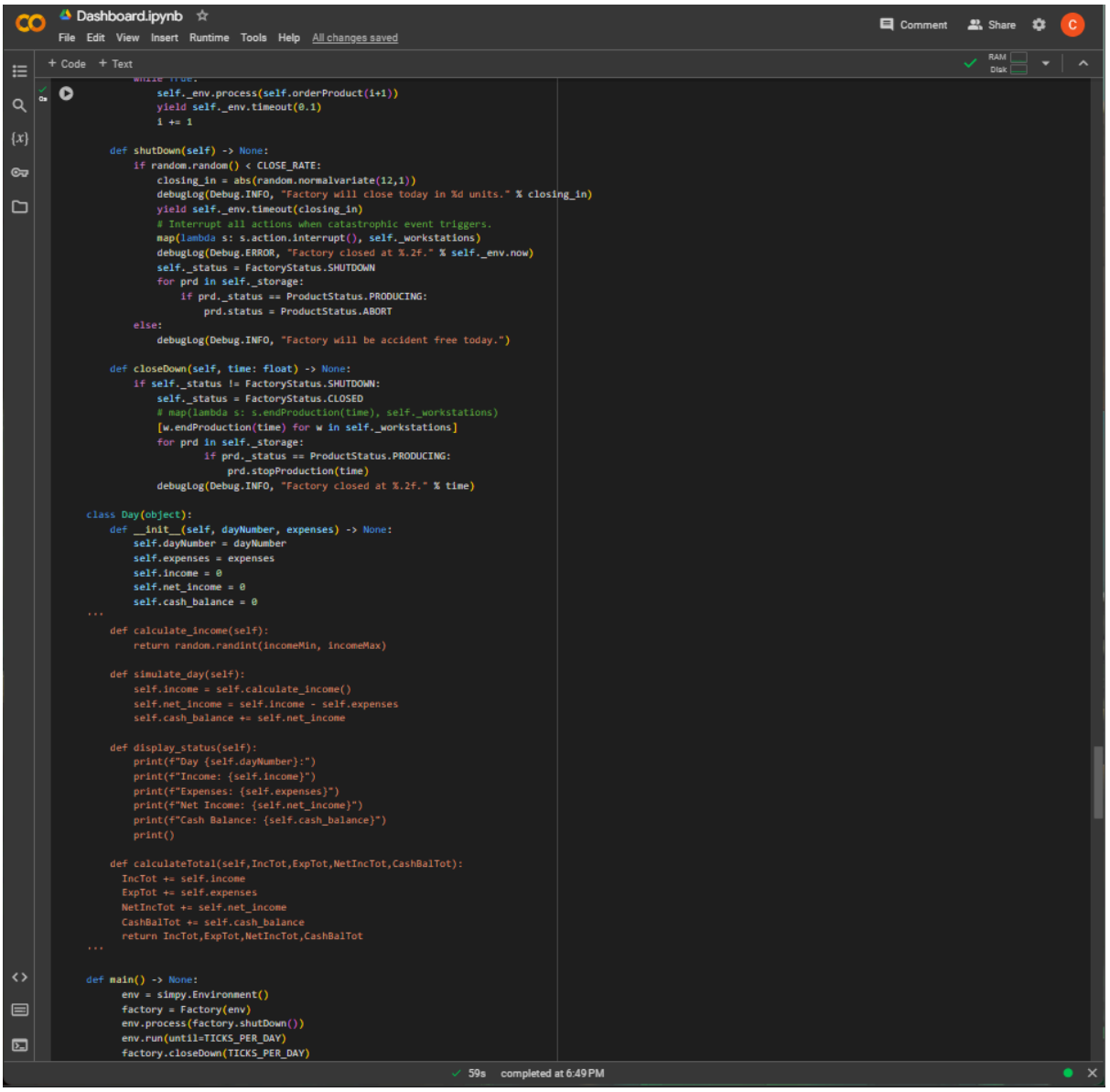
Souareba Sylla
Carlos Fernando Villaseñor Rábago



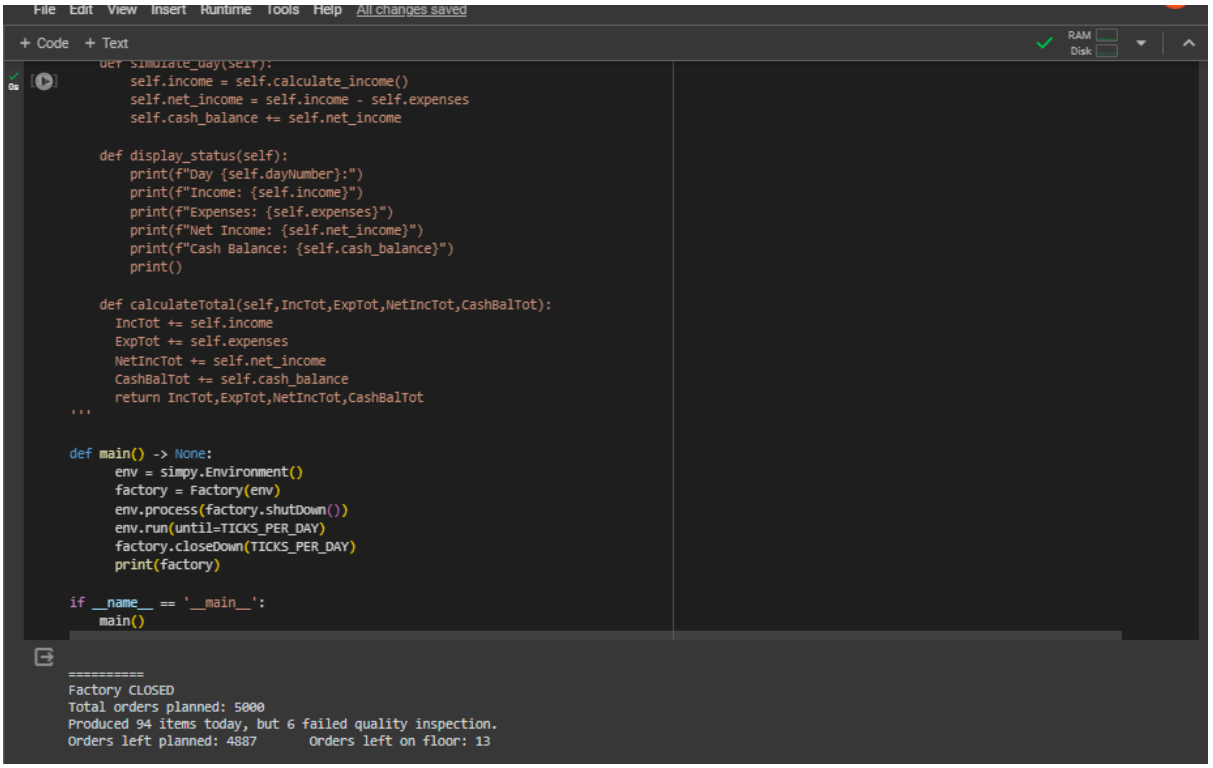
Souareba Sylla
Carlos Fernando Villaseñor Rábago



Souareba Sylla
Carlos Fernando Villaseñor Rábago

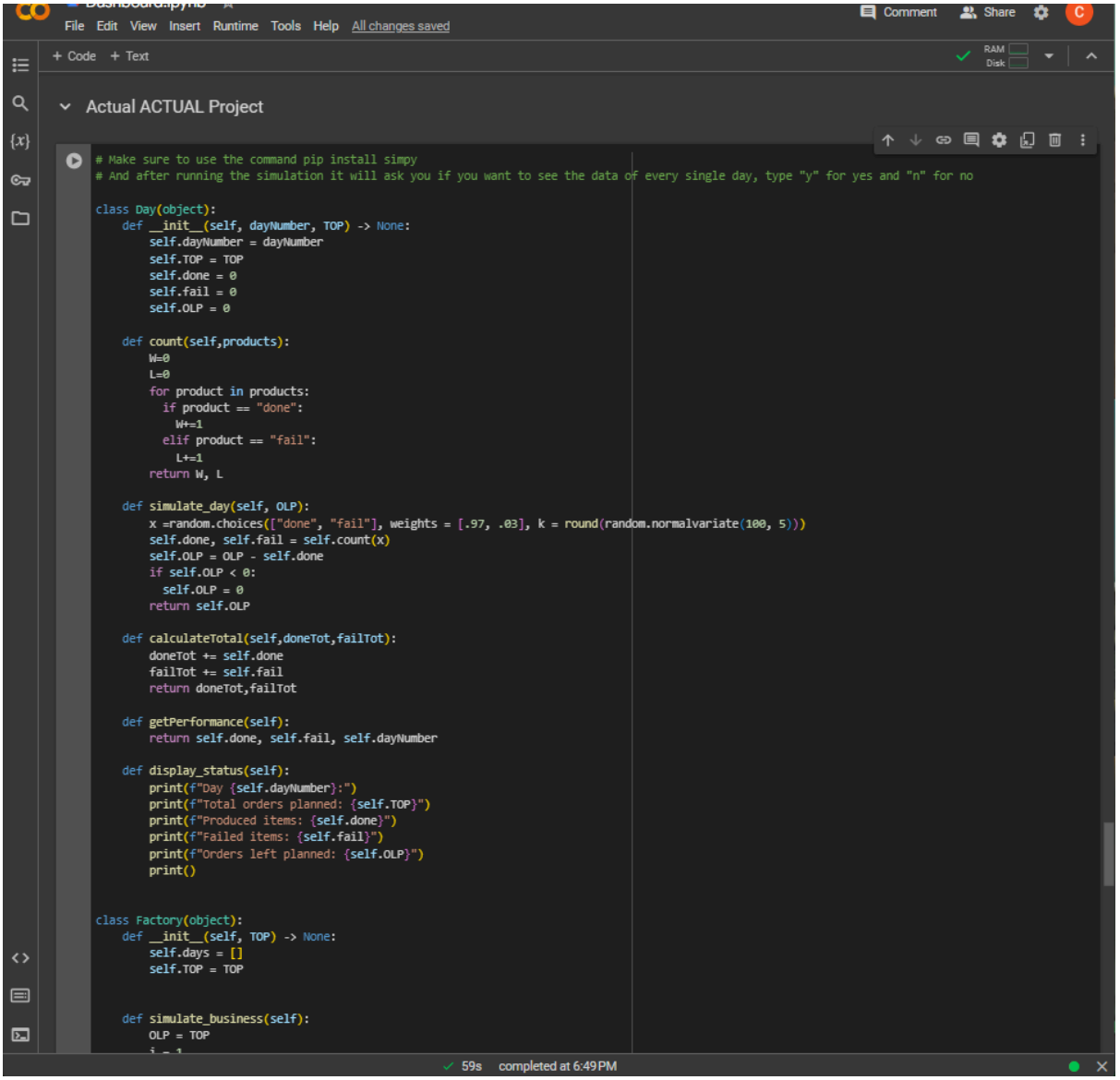


Souareba Sylla
Carlos Fernando Villaseñor Rábago

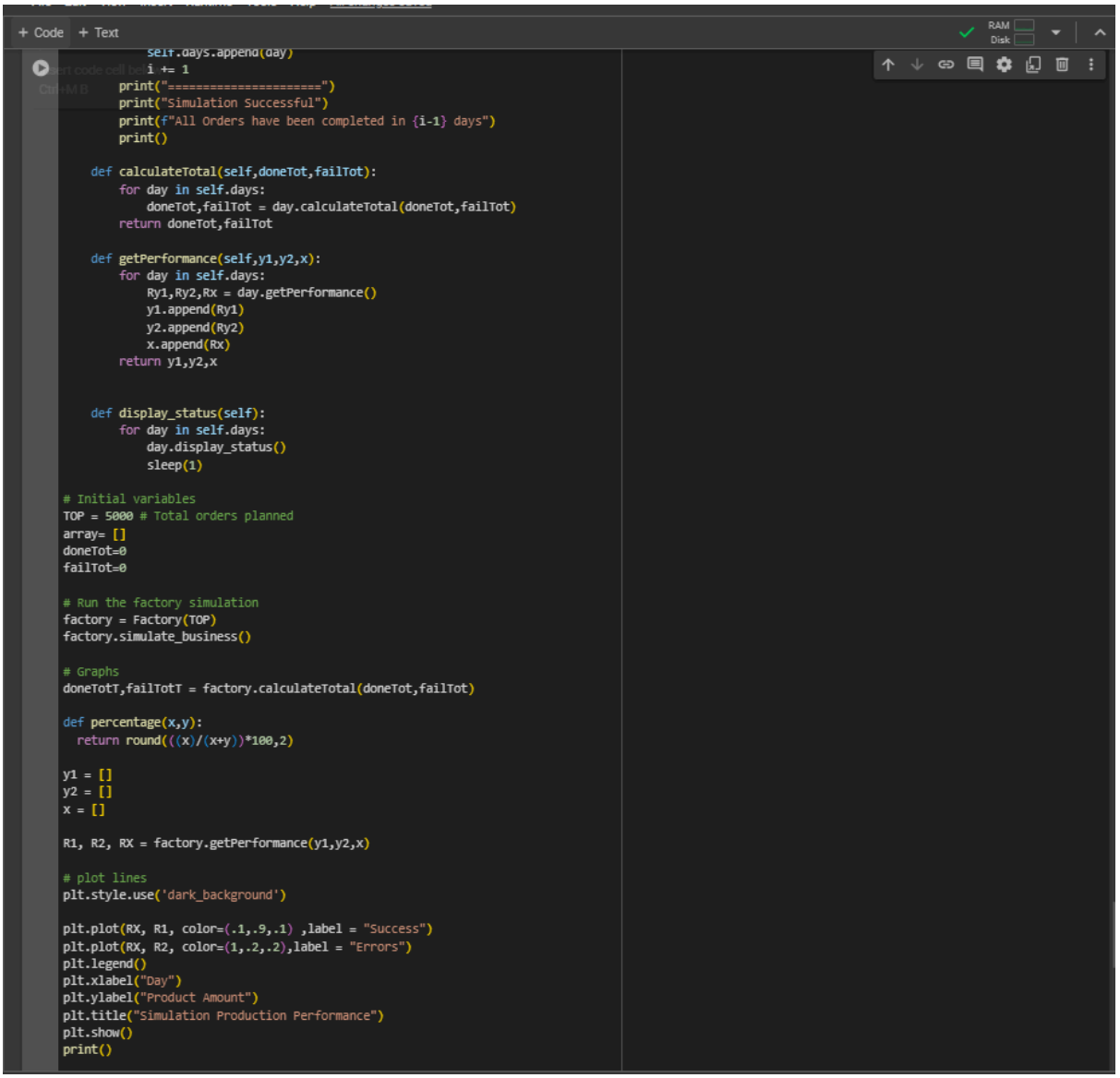


Souareba Sylla
Carlos Fernando Villaseñor Rábago

And at the very end we developed the ACTUAL code that we were planning on delivering



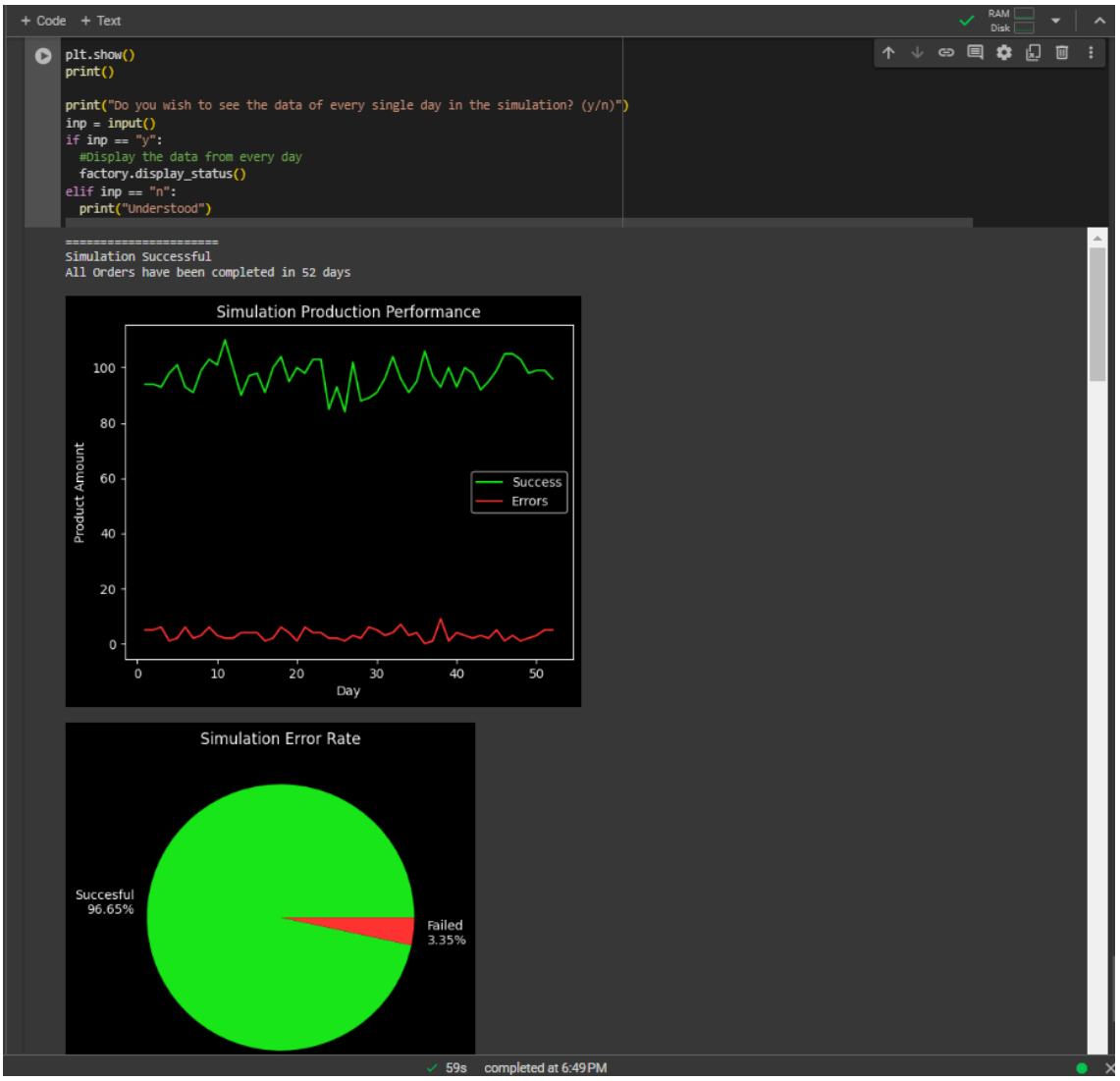
Souareba Sylla
Carlos Fernando Villaseñor Rábago



```
            self.days.append(day)
            i += 1
        print("====================")
        print("Simulation Successful")
        print(f"All Orders have been completed in {i-1} days")
        print()

    def calculateTotal(self,doneTot,failTot):
        for day in self.days:
            doneTot,failTot = day.calculateTotal(doneTot,failTot)
        return doneTot,failTot

    def getPerformance(self,y1,y2,x):
        for day in self.days:
            Ry1,Ry2,Rx = day.getPerformance()
            y1.append(Ry1)
            y2.append(Ry2)
            x.append(Rx)
        return y1,y2,x


    def display_status(self):
        for day in self.days:
            day.display_status()
            sleep(1)

# Initial variables
TOP = 5000 # Total orders planned
array= []
doneTot=0
failTot=0

# Run the factory simulation
factory = Factory(TOP)
factory.simulate_business()

# Graphs
doneTotT,failTotT = factory.calculateTotal(doneTot,failTot)

def percentage(x,y):
  return round(((x)/(x+y))*100,2)

y1 = []
y2 = []
x = []

R1, R2, RX = factory.getPerformance(y1,y2,x)

# plot lines
plt.style.use('dark_background')

plt.plot(RX, R1, color=(.1,.9,.1) ,label = "Success")
plt.plot(RX, R2, color=(1,.2,.2),label = "Errors")
plt.legend()
plt.xlabel("Day")
plt.ylabel("Product Amount")
plt.title("Simulation Production Performance")
plt.show()
print()
```

And finally in the design aspect we decided that dark mode was the nicest one to look at. And for the graphs we thought that a timeline that showed how the performance was doing day by day from both the success and the fail data, and a pie chart graph that showed how big the success was compared to the failed attempts was the two best options to visualize the data. And at the end to put a question whether the user wants to see the data from ALL the days from the simulation so this way the users that don't like seeing an excess of information wouldn’t be bothered.

Souareba Sylla
Carlos Fernando Villaseñor Rábago




```
Do you wish to see the data of every single day in the simulation? (y/n)
y
Day 1:
Total orders planned: 5000
Produced items: 94
Failed items: 5
Orders left planned: 4906

Day 2:
Total orders planned: 5000
Produced items: 94
Failed items: 5
Orders left planned: 4812

Day 3:
Total orders planned: 5000
Produced items: 93
Failed items: 6
Orders left planned: 4719

Day 4:
Total orders planned: 5000
Produced items: 98
Failed items: 1
Orders left planned: 4621

Day 5:
Total orders planned: 5000
Produced items: 101
Failed items: 2
Orders left planned: 4520
```