

Souareba Sylla  
Carlos Fernando Villaseñor Rábago

# Dashboard Document

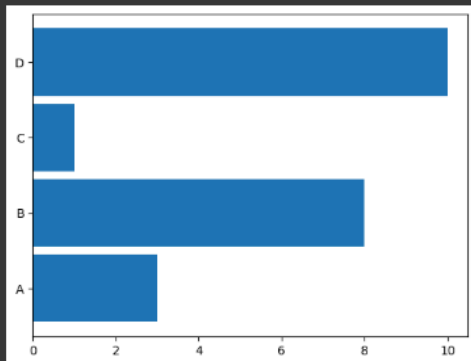
We decided to use Google Colab the exact day the teacher taught us how to use it. With this we practiced and experimented multiple ways to print out a graph with Matplotlib, other numpy functions to pull out random variables and other things in python that would help us develop this project.

[illegible]

```
[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()
```



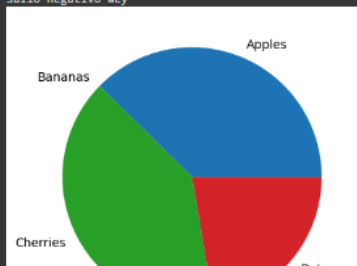
```
[190]
```

```
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] #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] #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] #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("Day {self.dayNumber}:")  
        print("Income: {self.income}")  
        print("Expenses: {self.expenses}")  
        print("Net Income: {self.net_income}")  
        print("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

```
+ Code + Text
[8] 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)

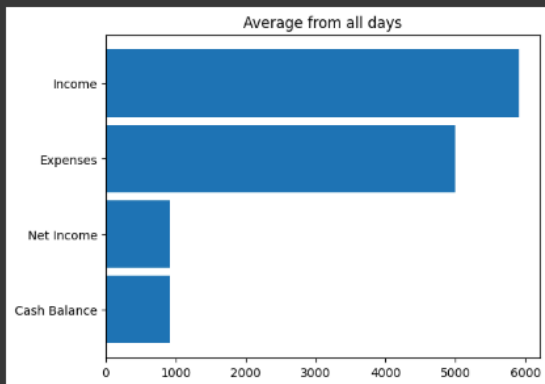
print()
print(IncTot, ExpTot, NetIncTot, CashBalTot)

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

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

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_BLN = 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]))

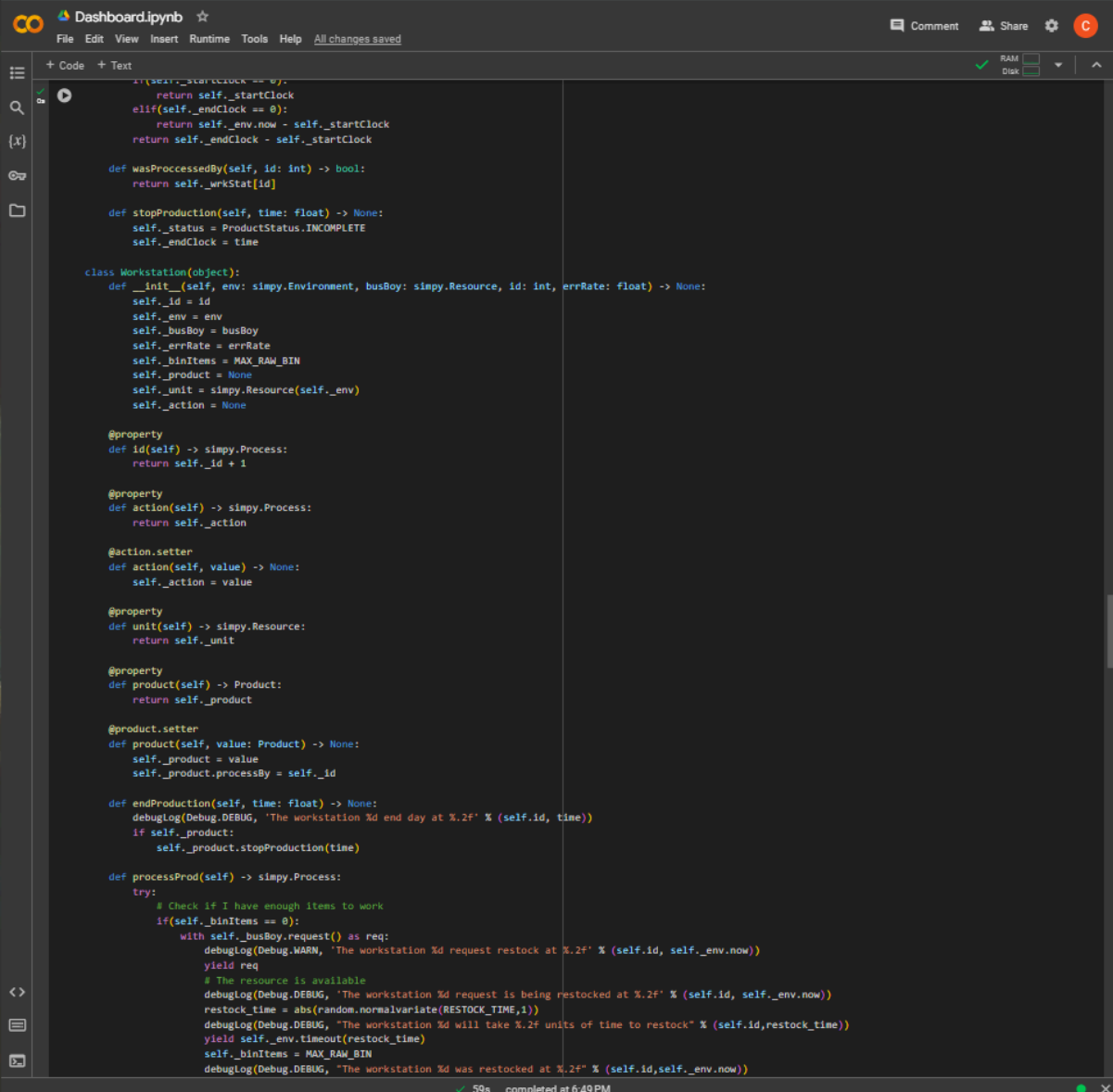
    @property
    def isDone(self) -> bool:
        return all(self._wrkStat) and not self.isAborted

    @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



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes the Jupyter logo, the filename 'Dashboard.ipynb', and standard menu items (File, Edit, View, Insert, Runtime, Tools, Help). On the right, there are icons for Comment, Share, and a settings gear, along with a red circle containing the letter 'C'. Below the top bar, a toolbar shows '+ Code' and '+ Text' tabs, a green checkmark, and RAM/Disk usage indicators. The main area contains Python code for a simulation model. The code defines a 'Workstation' class with various attributes and methods, including a 'processProd' method that uses a 'with' statement to request a resource from a 'busBoy'.

```
11(self._startClock = 0):
12    return self._startClock
13    elif(self._endClock == 0):
14        return self._env.now - self._startClock
15    return self._endClock - self._startClock

def wasProcessedBy(self, id: int) -> bool:
    return self._wrkStat[id]

def stopProduction(self, time: float) -> None:
    self._status = ProductStatus.INCOMPLETE
    self._endClock = time

class Workstation(object):
    def __init__(self, env: simpy.Environment, busBoy: simpy.Resource, id: int, errRate: float) -> None:
        self._id = id
        self._env = env
        self._busBoy = busBoy
        self._errRate = errRate
        self._binItems = MAX_RAM_BIN
        self._product = None
        self._unit = simpy.Resource(self._env)
        self._action = None

    @property
    def id(self) -> simpy.Process:
        return self._id + 1

    @property
    def action(self) -> simpy.Process:
        return self._action

    @action.setter
    def action(self, value) -> None:
        self._action = value

    @property
    def unit(self) -> simpy.Resource:
        return self._unit

    @property
    def product(self) -> Product:
        return self._product

    @product.setter
    def product(self, value: Product) -> None:
        self._product = value
        self._product.processBy = self._id

def endProduction(self, time: float) -> None:
    debugLog(Debug.DEBUG, 'The workstation %d end day at %.2f' % (self.id, time))
    if self._product:
        self._product.stopProduction(time)

def processProd(self) -> simpy.Process:
    try:
        # Check if I have enough items to work
        if(self._binItems == 0):
            with self._busBoy.request() as req:
                debugLog(Debug.WARN, 'The workstation %d request restock at %.2f' % (self.id, self._env.now))
                yield req
                # The resource is available
                debugLog(Debug.DEBUG, 'The workstation %d request is being restocked at %.2f' % (self.id, self._env.now))
                restock_time = abs(random.normalvariate(RESTOCK_TIME,1))
                debugLog(Debug.DEBUG, 'The workstation %d will take %.2f units of time to restock' % (self.id, restock_time))
                yield self._env.timeout(restock_time)
                self._binItems = MAX_RAM_BIN
                debugLog(Debug.DEBUG, 'The workstation %d was restocked at %.2f' % (self.id, self._env.now))
```

59s completed at 6:49 PM



Souareba Sylla  
Carlos Fernando Villaseñor Rábago

```
Dashboard.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share Settings

+ Code + Text
RAM Disk

fixing_time = abs(random.normalvariate(0,fixing_time))
debugLog(Debug.DEBUG, "The workstation %d will take %.2f units of time to be fixed" % (self.id,fixing_time))
yield self._env.timeout(fixing_time)
debugLog(Debug.INFO, "The workstation %d is back on line at %.2f" % (self.id, self._env.now))

# Process the product
self._binItems -= 1
debugLog(Debug.DEBUG, "The workstation %d starts processing product %06d at %.2f" % (self.id, self.product_id, self._env.now))
working_time = abs(random.normalvariate(WORK_TIME,1))
yield self._env.timeout(working_time)
debugLog(Debug.DEBUG, "The workstation %d is done processing prod %06d at %.2f" % (self.id, self.product_id, self._env.now))

except simpy.Interrupt:
    debugLog(Debug.ERROR, "There was a catastrophic issue, %d at %.2f" % (self.id, self._env.now))
    self.product.status = ProductStatus.ABORT
finally:
    self._product = None

class Factory(object):
    def __init__(self, env: simpy.Environment) -> None:
        self._env = env
        self._restockDevice = simpy.Resource(self._env, RESTOCK_UNITS)
        self._workstations = []
        self._storage = []
        self._status = FactoryStatus.OPEN

    # Create all the work stations
    for i in range(WRK_STATIONS):
        self._workstations.append(Workstation(self._env, self._restockDevice, 1, WRK_STATION_RATES[i]))
        debugLog(Debug.DEBUG, "Ready %s" % self._workstations[i])
    self.action = self._env.process(self.produce())

    def __str__(self) -> str:
        output = "\n=====Factory %s" % (self._status)
        done = sum(1 for i in self._storage if i._status == ProductStatus.DONE)
        fail = sum(1 for i in self._storage if i._status == ProductStatus.FAIL)
        ordered = sum(1 for i in self._storage if i._status == ProductStatus.ORDERED)
        incomplete = sum(1 for i in self._storage if i._status == ProductStatus.INCOMPLETE)
        output += "\nTotal orders planned: %d" % (len(self._storage))
        output += "\nProduced %d items today, but %d failed quality inspection." % (done, fail)
        output += "\nOrders left planned: %d \tOrders left on floor: %d" % (ordered, incomplete)
        if(self._status == FactoryStatus.SHUTDOWN):
            abort = sum(1 for i in self._storage if i._status == ProductStatus.ABORT)
            output += "\nOrders aborted due shutdown: %d" % (abort)
        if(DEBUG_LEVEL.value == Debug.DEBUG):
            prod = sum(1 for i in self._storage if i._status == ProductStatus.PRODUCING)
            output += "\tErr: %d" % (prod)
            for prd in self._storage:
                if prd._status == ProductStatus.PRODUCING:
                    output += "\n%s" % str(prd)
        return output

    def getWorkstation(self, index : int) -> Workstation:
        return self._workstations[index]

    def orderProduct(self, id: int) -> simpy.Process:
        if(self._status == FactoryStatus.CLOSED):
            return
        prod = Product(id, self._env)
        self._storage.append(prod)
        while not prod.isDone:
            idx = prod.nextStation
            # Check the situation of parallel stations
            if(idx == 3): # station 4
                if(not prod.wasProcessedBy(4) and self.getWorkstation(idx).unit.count > self.getWorkstation(idx+1).unit.count):
                    idx += 1
            debugLog(Debug.DEBUG, "Product %06d to be processed by WS %02d" % (prod_id, (idx+1)))
            station = self.getWorkstation(idx)
            with station.unit.request() as wrkProcess:
                yield wrkProcess

59s completed at 6:49 PM
```

Souareba Sylla  
Carlos Fernando Villaseñor Rábago

```
Dashboard.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share Settings

+ Code + Text
RAM Disk

while True:
    self._env.process(self.orderProduct(i+1))
    yield self._env.timeout(0.1)
    i += 1

def shutDown(self) -> None:
    if random.random() < CLOSE_RATE:
        closing_in = abs(random.normalvariate(12,1))
        debugLog(Debug.INFO, "Factory will close today in %d units." % closing_in)
        yield self._env.timeout(closing_in)
        # Interrupt all actions when catastrophic event triggers.
        map(lambda s: s.action.interrupt(), self._workstations)
        debugLog(Debug.ERROR, "Factory closed at %.2f." % self._env.now)
        self._status = FactoryStatus.SHUTDOWN
        for prd in self._storage:
            if prd._status == ProductStatus.PRODUCING:
                prd.status = ProductStatus.ABORT
    else:
        debugLog(Debug.INFO, "Factory will be accident free today.")

def closeDown(self, time: float) -> None:
    if self._status != FactoryStatus.SHUTDOWN:
        self._status = FactoryStatus.CLOSED
        # map(lambda s: s.endProduction(time), self._workstations)
        [w.endProduction(time) for w in self._workstations]
        for prd in self._storage:
            if prd._status == ProductStatus.PRODUCING:
                prd.stopProduction(time)
        debugLog(Debug.INFO, "Factory closed at %.2f." % time)

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

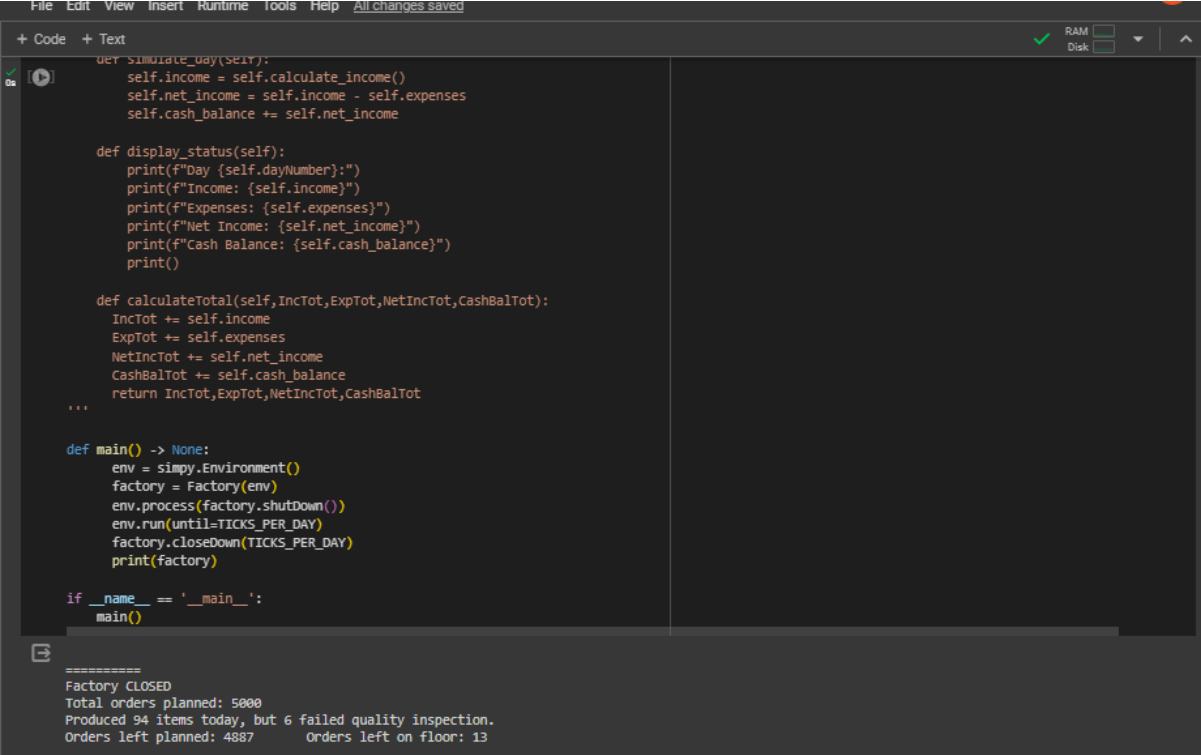
    def display_status(self):
        print(f"Day {self.dayNumber}:")
        print(f"Income: {self.income}")
        print(f"Expenses: {self.expenses}")
        print(f"Net Income: {self.net_income}")
        print(f"Cash Balance: {self.cash_balance}")
        print()

    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
    ...

def main() -> None:
    env = slippy.Environment()
    factory = Factory(env)
    env.process(factory.shutDown())
    env.run(until=TICKS_PER_DAY)
    factory.closeDown(TICKS_PER_DAY)
```

59s completed at 6:49 PM

Souareba Sylla  
Carlos Fernando Villaseñor Rábago



```
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
✓ RAM
Disk

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
    ...

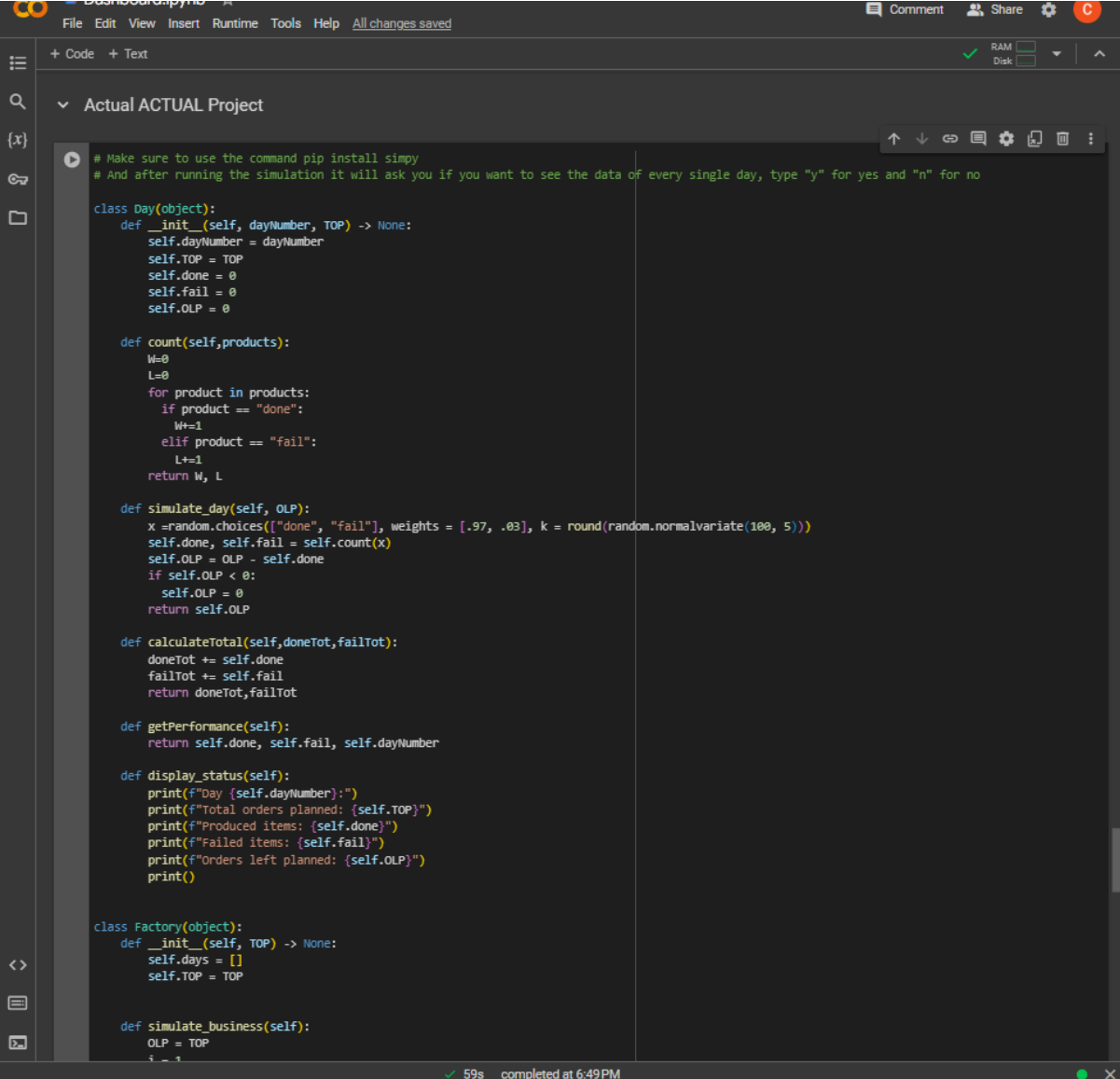
def main() -> None:
    env = simpy.Environment()
    factory = Factory(env)
    env.process(factory.shutdown())
    env.run(until=TICKS_PER_DAY)
    factory.closeDown(TICKS_PER_DAY)
    print(factory)

if __name__ == '__main__':
    main()

=====
Factory CLOSED
Total orders planned: 5000
Produced 94 items today, but 6 failed quality inspection.
Orders left planned: 4887      Orders left on floor: 13
```

Souareba Sylla  
Carlos Fernando Villaseñor Rábago

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



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar includes the Jupyter logo, a file explorer, and a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a status bar indicating 'All changes saved'. The main area displays Python code for a simulation project. The code defines two classes: 'Day' and 'Factory'. The 'Day' class has methods for initialization, counting products, simulating a day, calculating totals, getting performance, and displaying status. The 'Factory' class has methods for initialization and simulating business. The code is written in a clean, readable style with proper indentation and comments. The status bar at the bottom indicates '59s completed at 6:49 PM'.

```
# Make sure to use the command pip install simply
# And after running the simulation it will ask you if you want to see the data of every single day, type "y" for yes and "n" for no

class Day(object):
    def __init__(self, dayNumber, TOP) -> None:
        self.dayNumber = dayNumber
        self.TOP = TOP
        self.done = 0
        self.fail = 0
        self.OLP = 0

    def count(self, products):
        W=0
        L=0
        for product in products:
            if product == "done":
                W+=1
            elif product == "fail":
                L+=1
        return W, L

    def simulate_day(self, OLP):
        x = random.choices(["done", "fail"], weights = [.97, .03], k = round(random.normalvariate(100, 5)))
        self.done, self.fail = self.count(x)
        self.OLP = OLP - self.done
        if self.OLP < 0:
            self.OLP = 0
        return self.OLP

    def calculateTotal(self, doneTot, failTot):
        doneTot += self.done
        failTot += self.fail
        return doneTot, failTot

    def getPerformance(self):
        return self.done, self.fail, self.dayNumber

    def display_status(self):
        print(f"Day {self.dayNumber}:")
        print(f"Total orders planned: {self.TOP}")
        print(f"Produced items: {self.done}")
        print(f"Failed items: {self.fail}")
        print(f"Orders left planned: {self.OLP}")
        print()

class Factory(object):
    def __init__(self, TOP) -> None:
        self.days = []
        self.TOP = TOP

    def simulate_business(self):
        OLP = TOP
        i = 1
```

```
+ Code + Text
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.



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