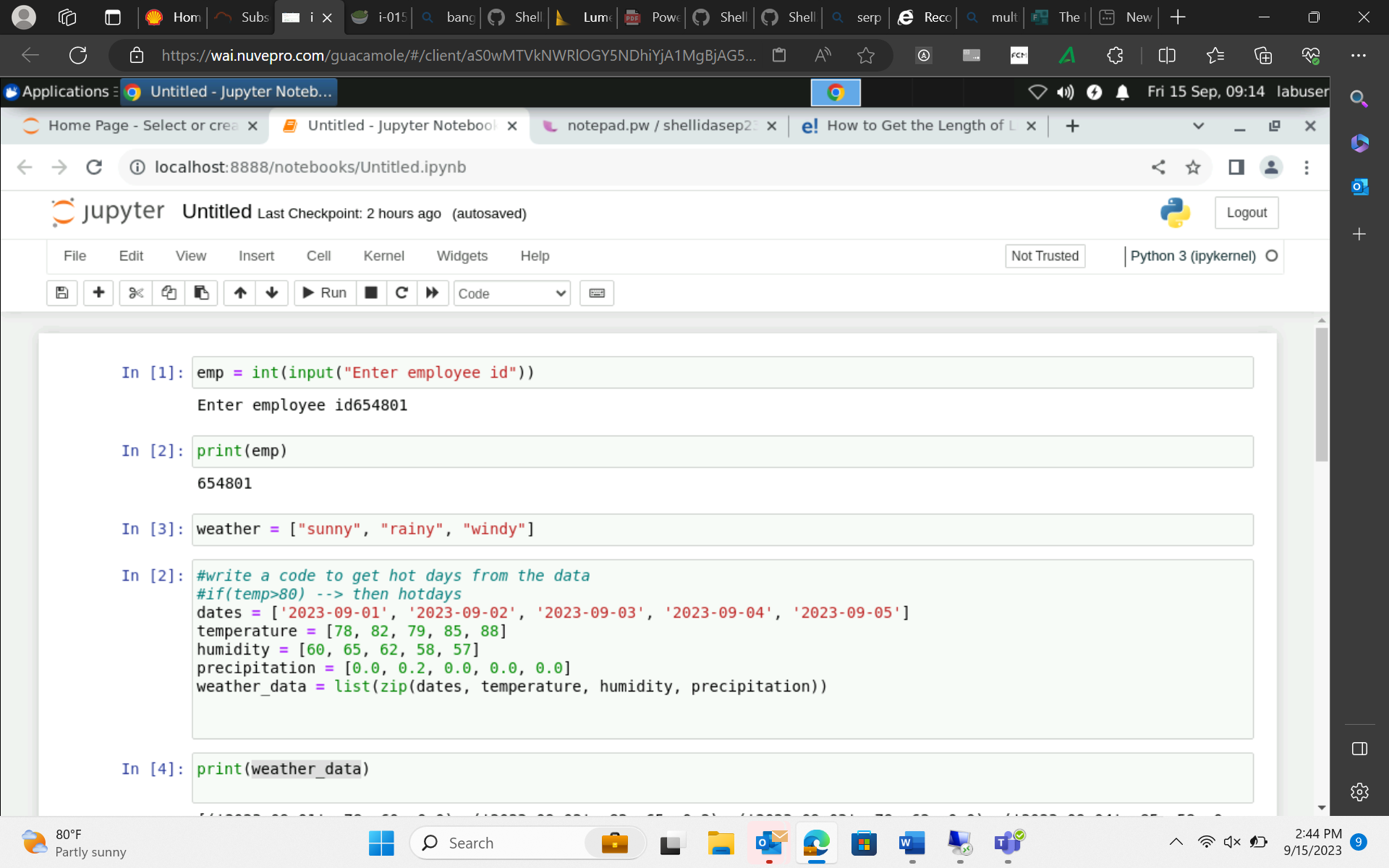
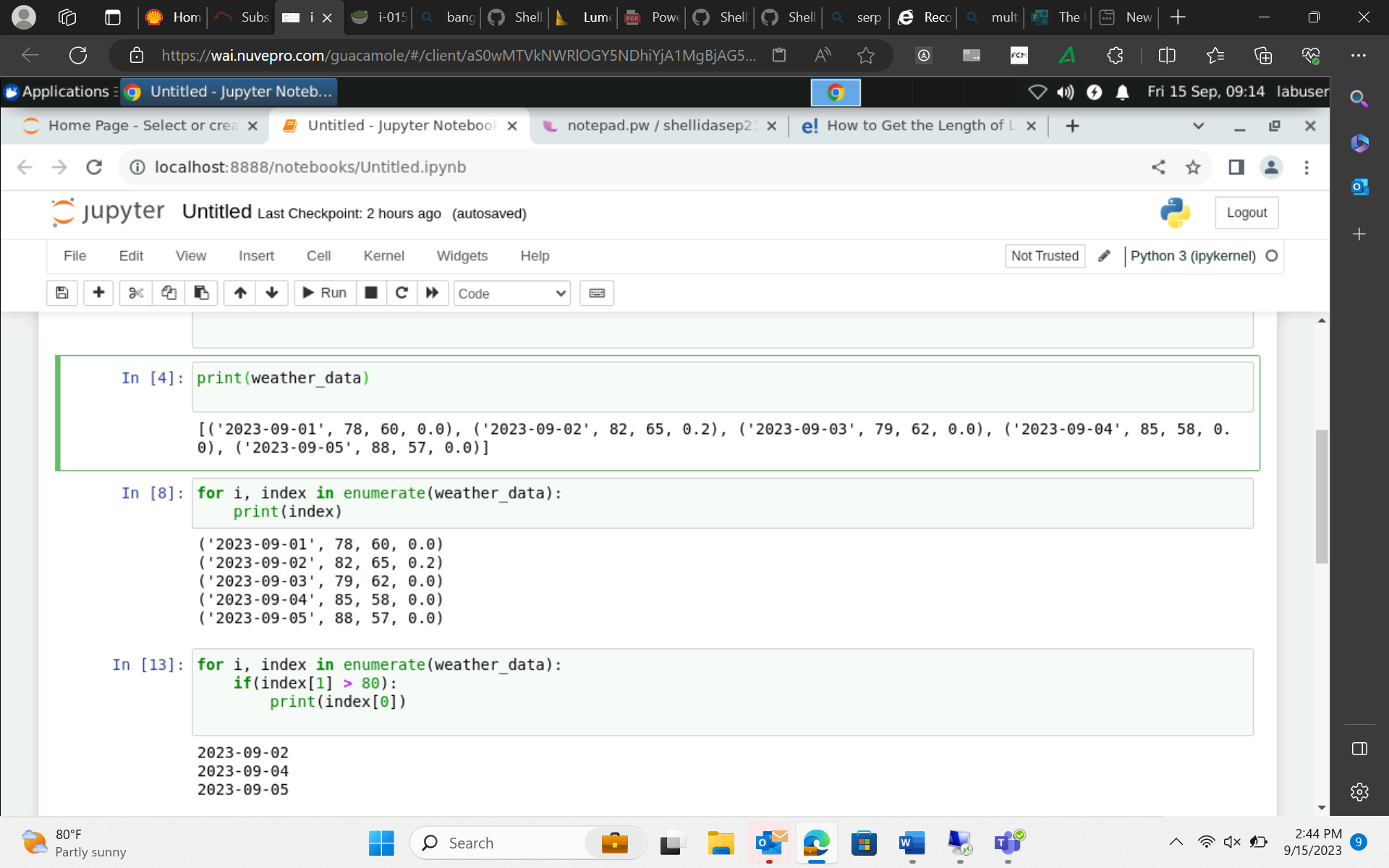
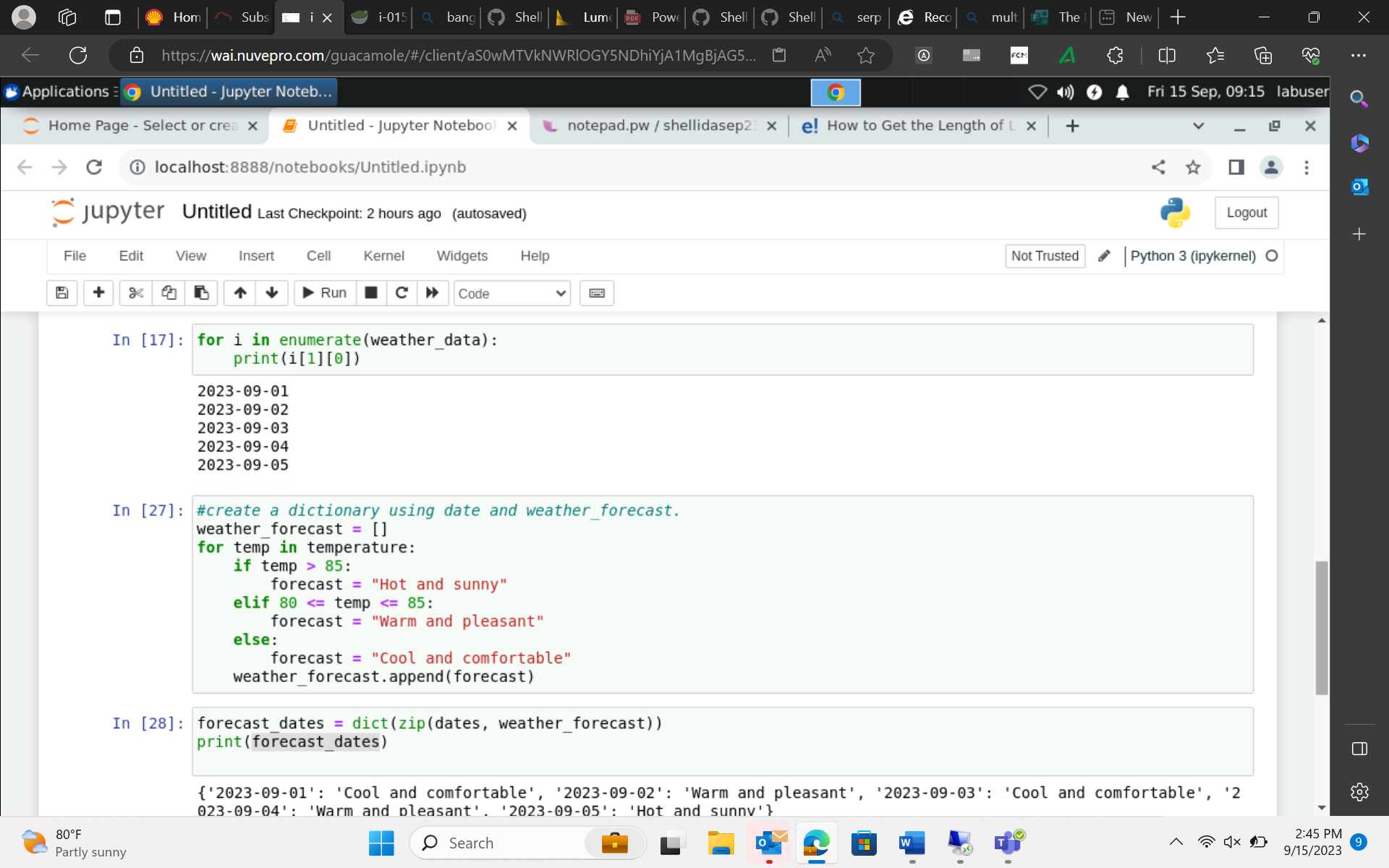
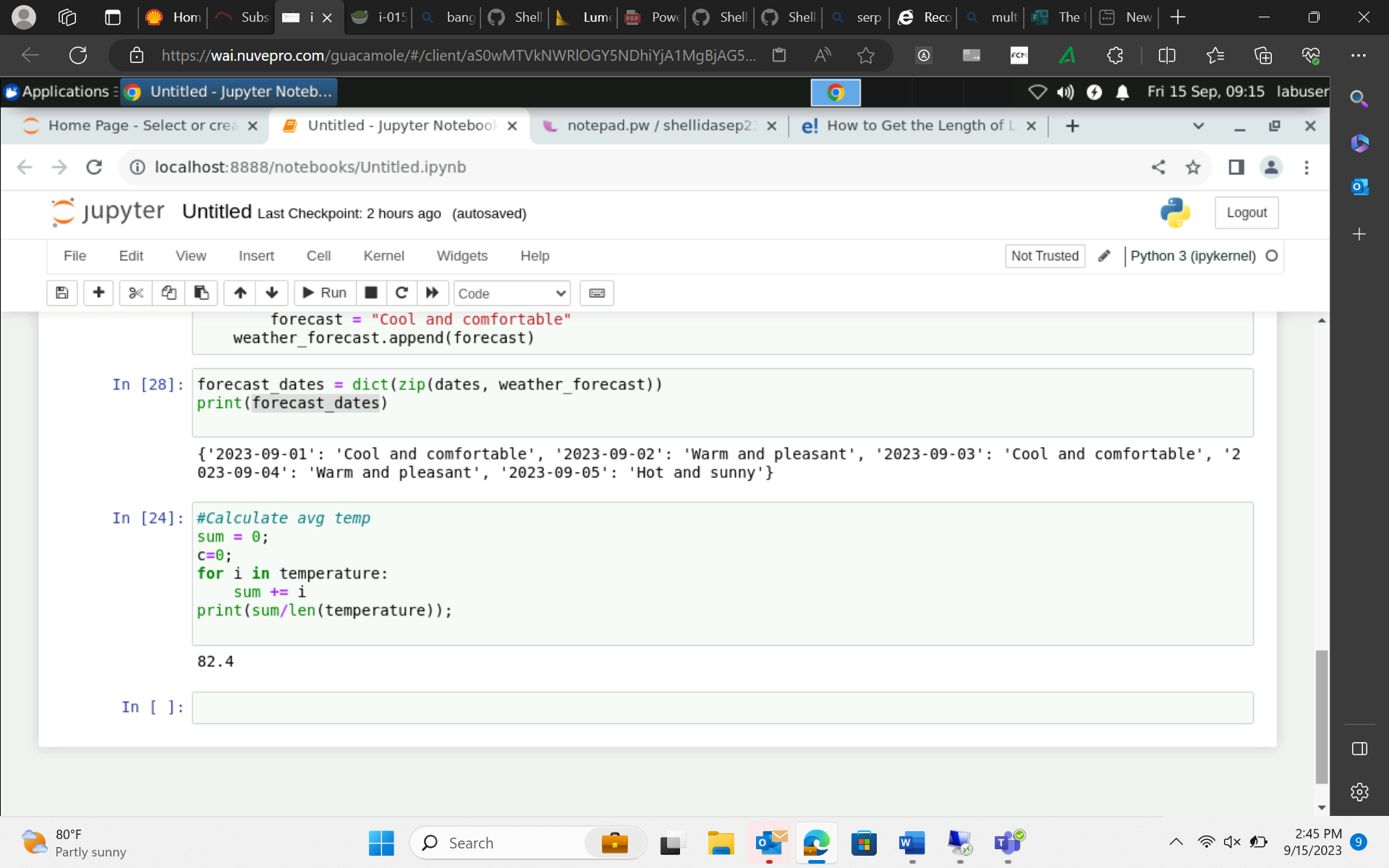
Python







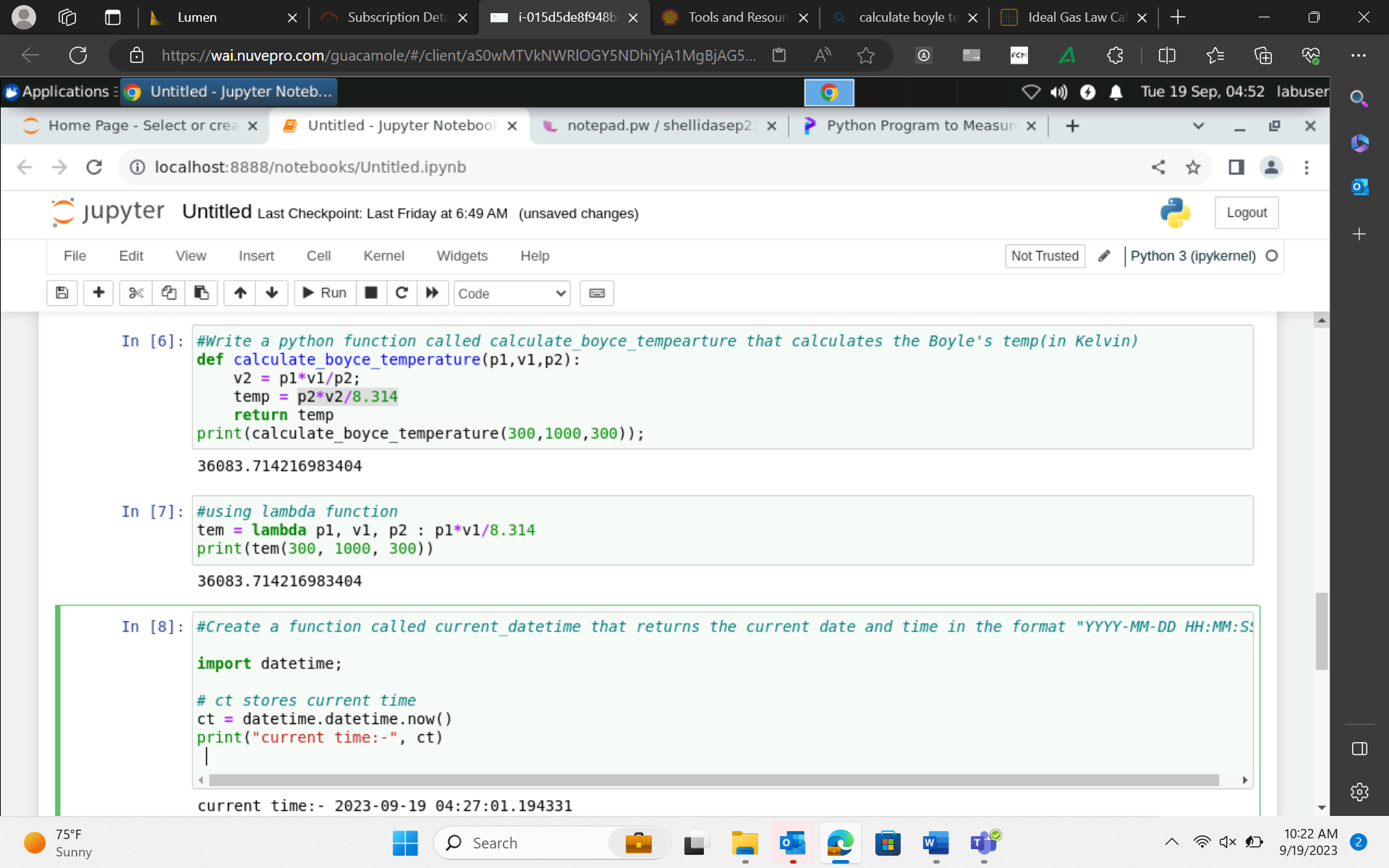


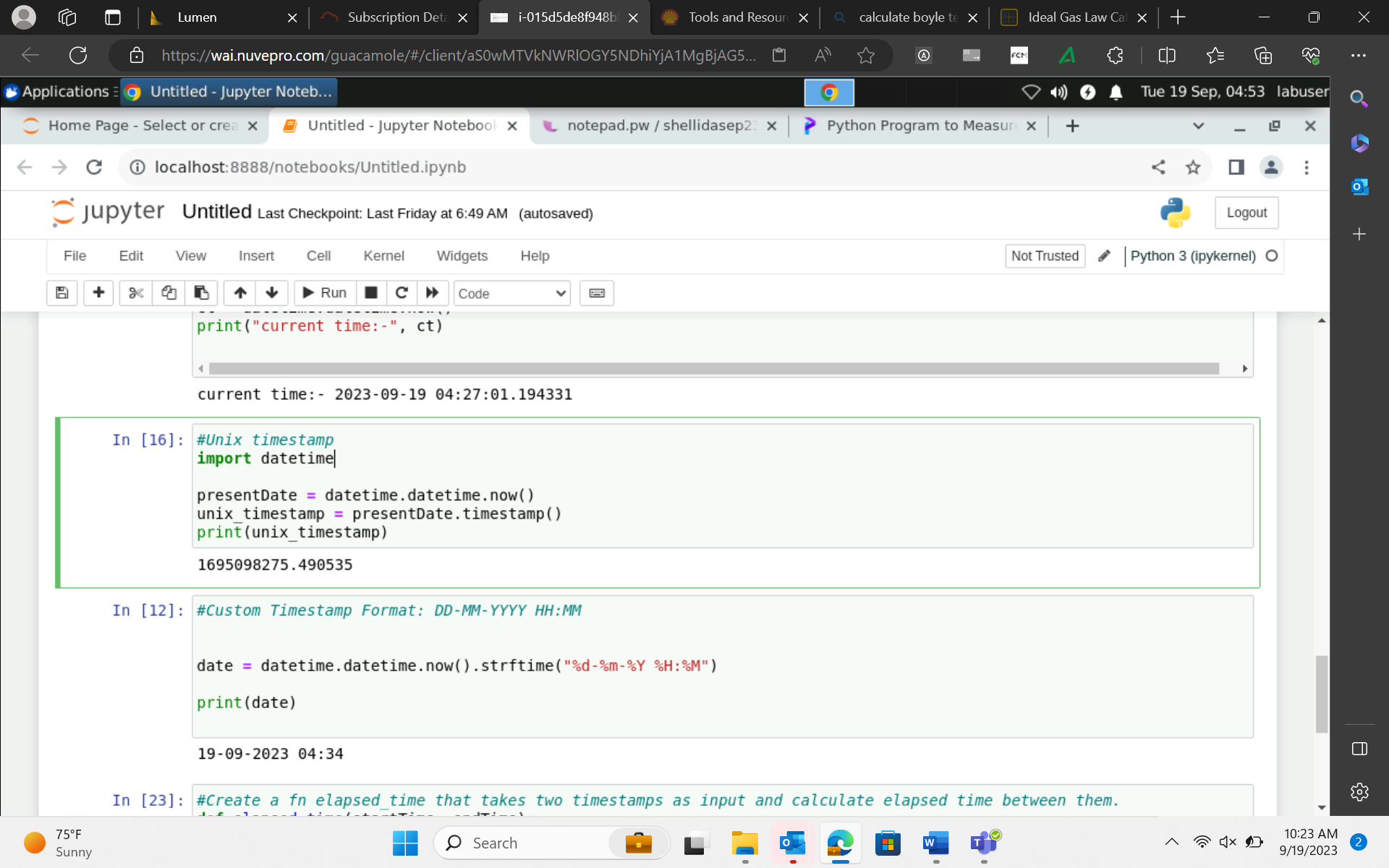
Data engineering: streaming data

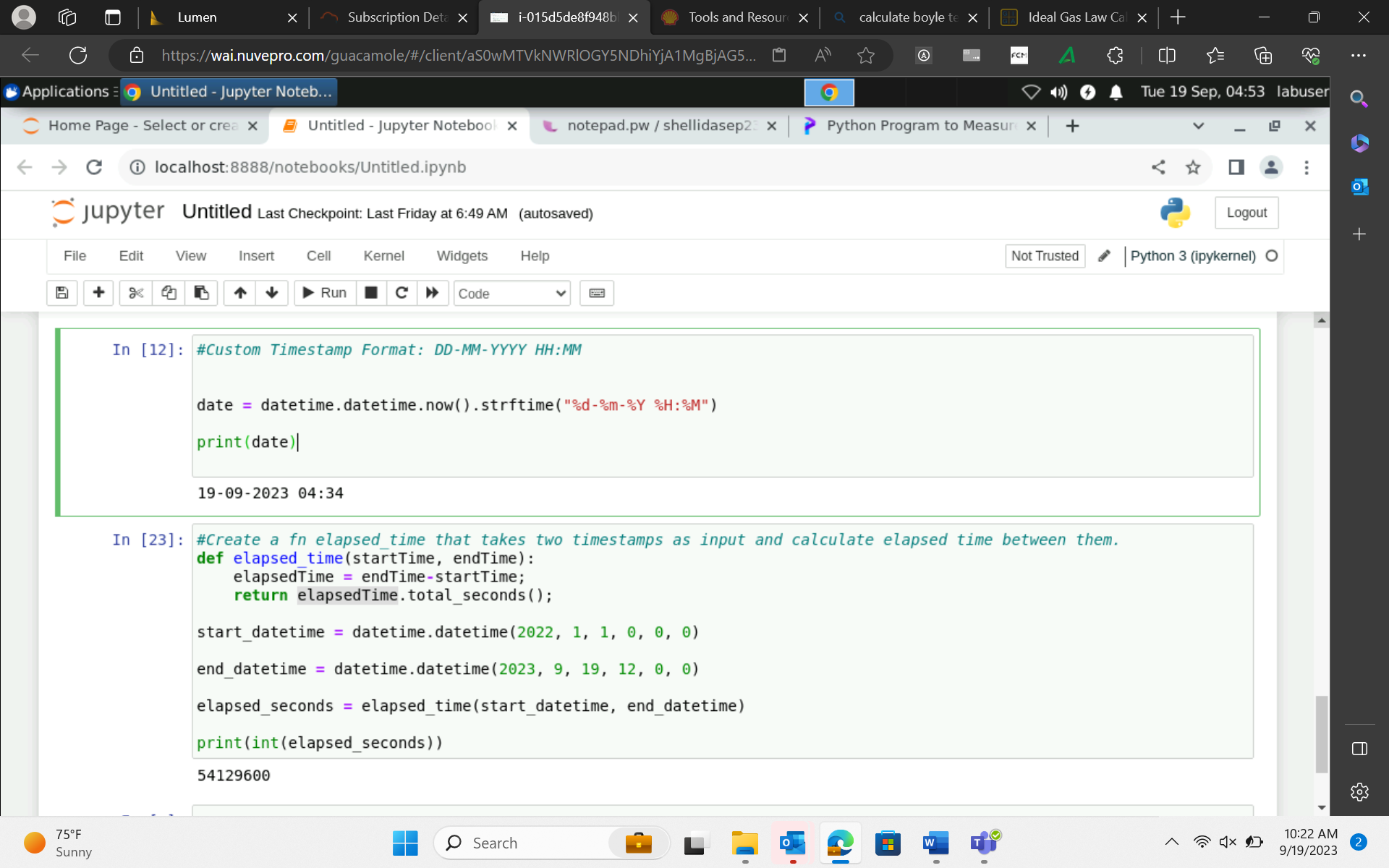
Event Time

Ingestion Time

Process time







Arguments :

Positional Arguments -> def fun(a,b,c) 🡪 same position 🡪 order is very important

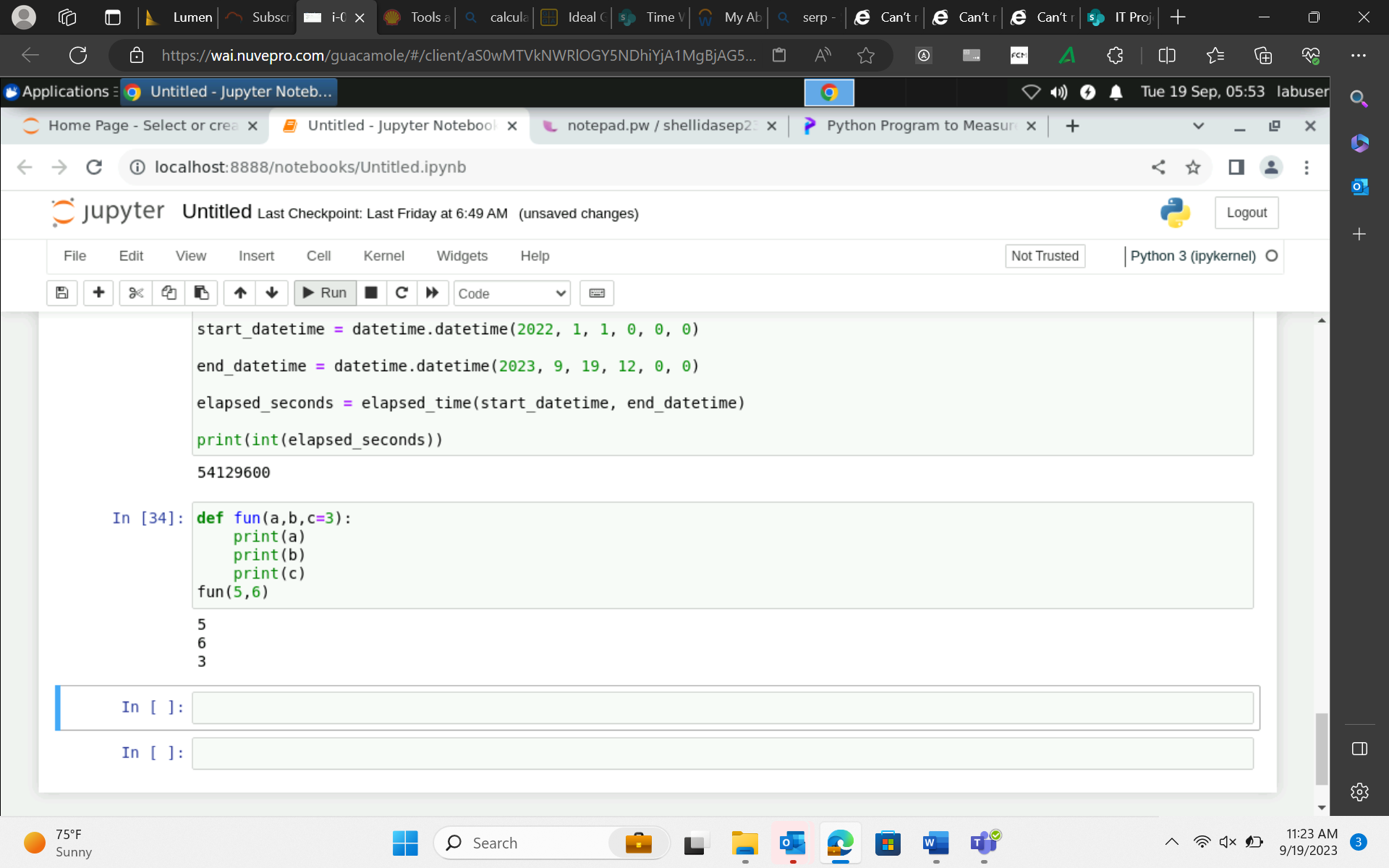
Fun(1,2,3)

Keyword Arguments 🡪 def fun(a=1, b=2, c=3)

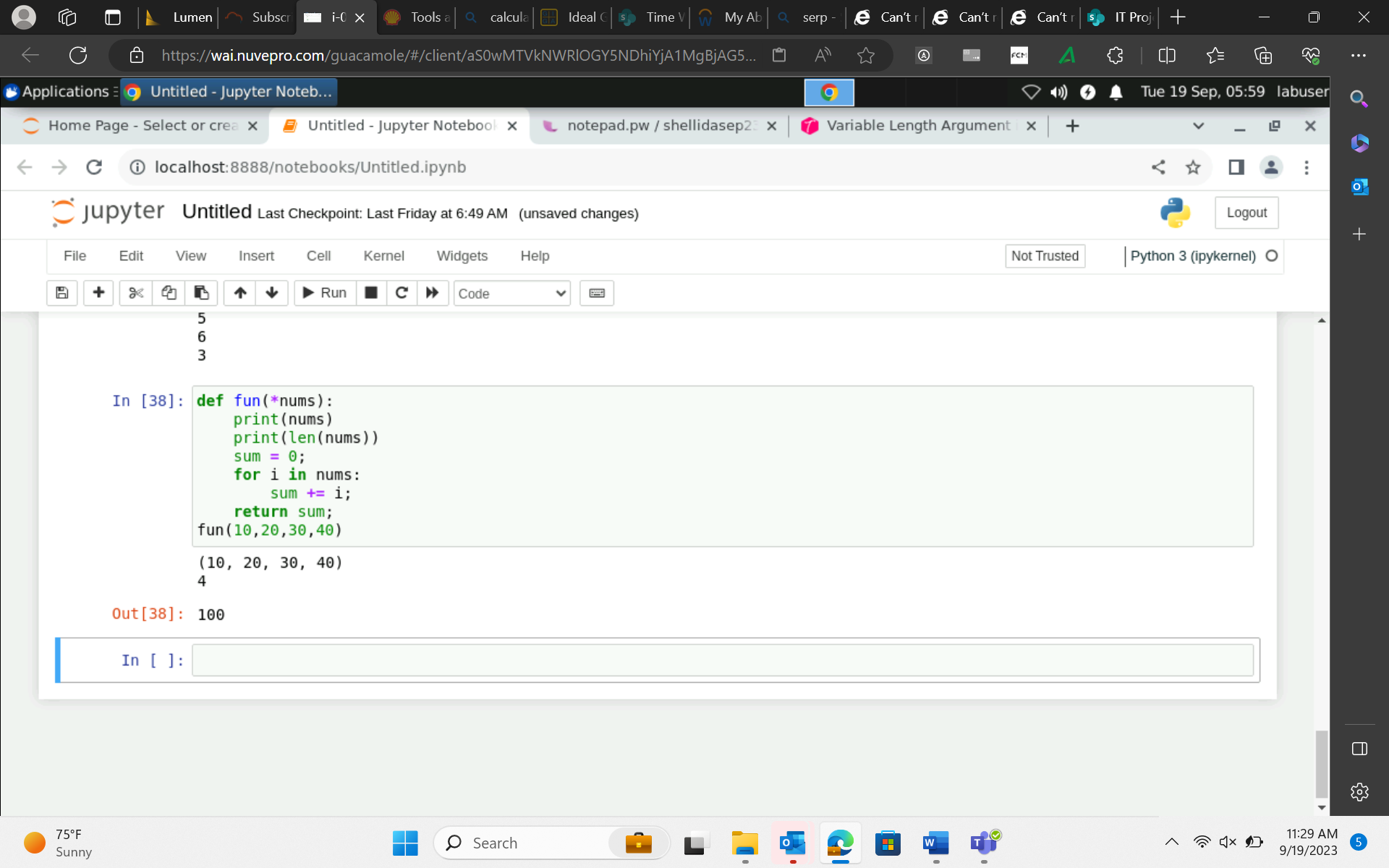
Default arguments 🡪 if value of third parameter not provided then replace with the default value

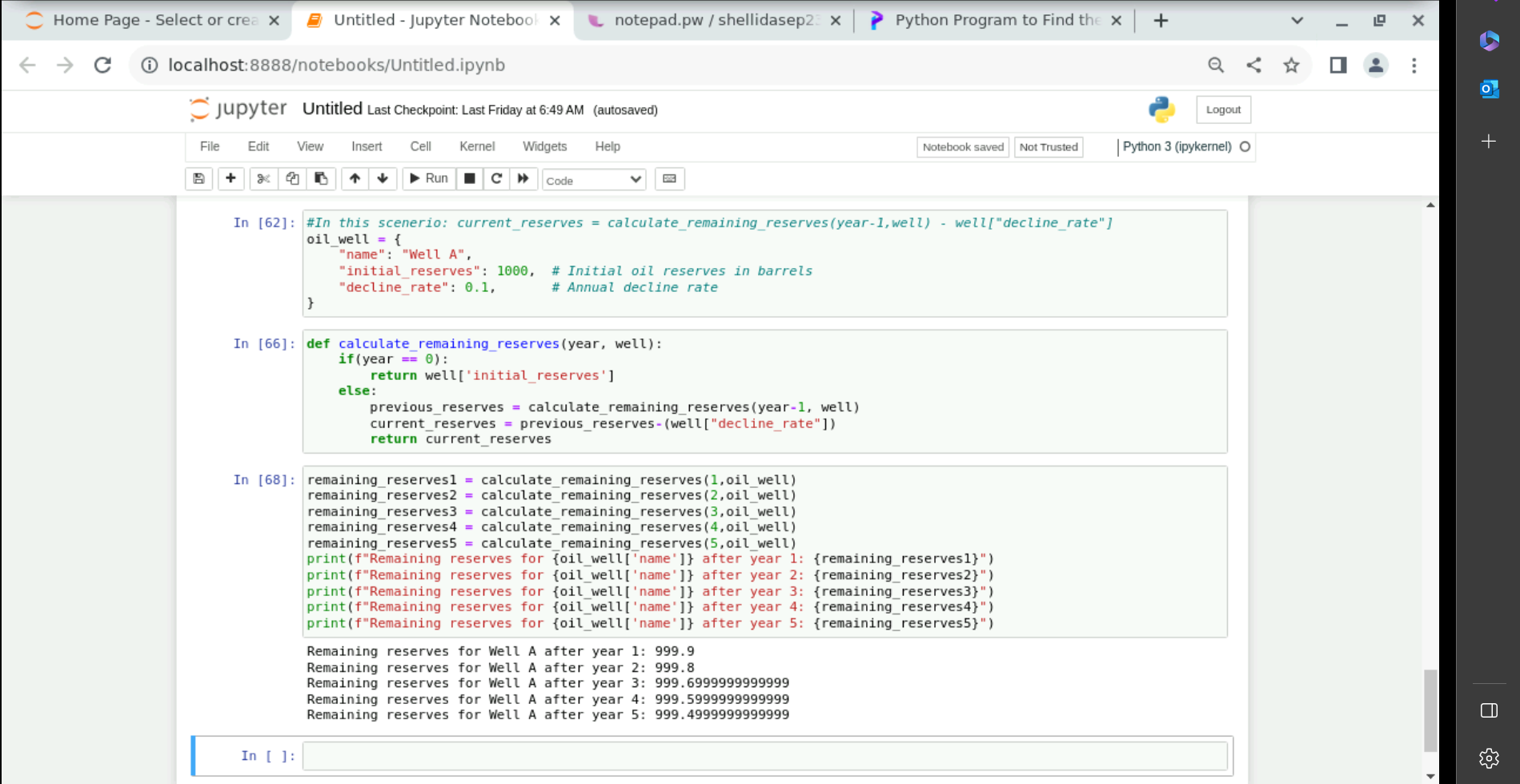
Def fun(a=5,b,c)

Fun(a,6,7) or fun(b,c)

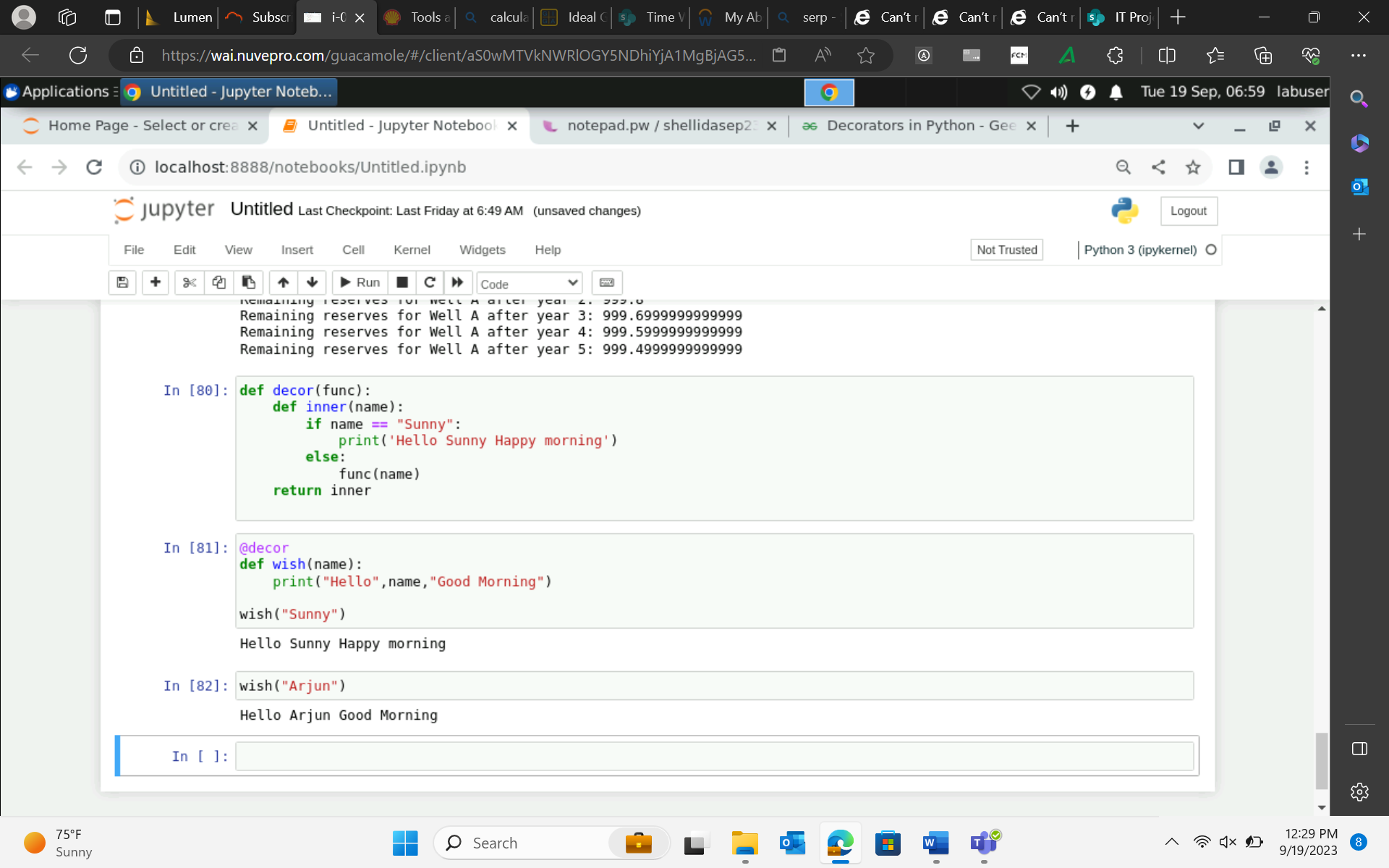


Variable length arguments





Decorator function python



Scenario: You are building a Power BI report for a sales team, and you need to calculate the total sales amount for each product. Should you use a calculated column or a measure for this task, and why?

Measure is used – performing aggregations

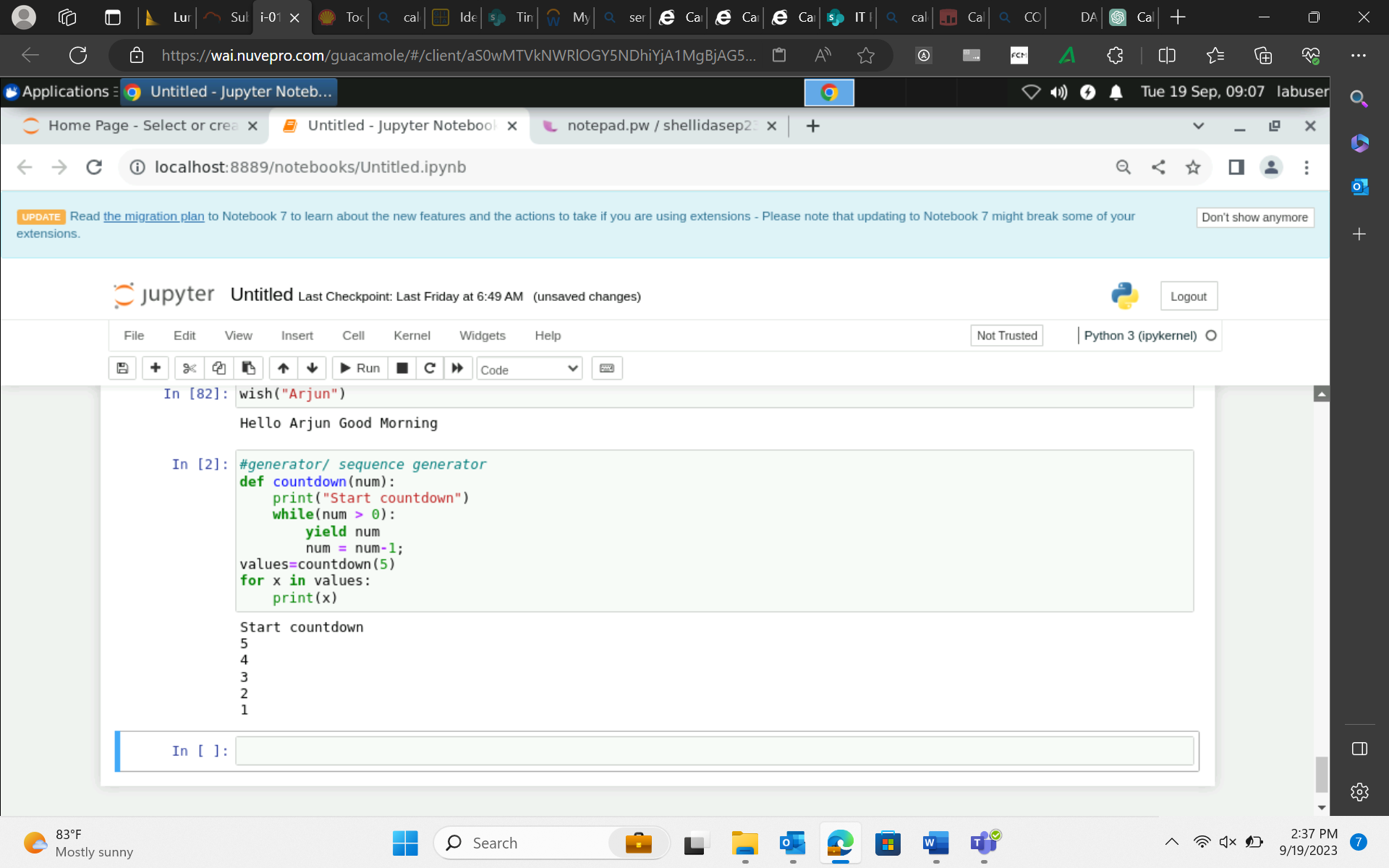
Scenario: You have a dataset containing a "Sales" column and a "Discount" column. You want to calculate the discounted sales amount for each row. Should you use a calculated column or a measure for this calculation, and why?

Calculated column – row level

" You have a dataset containing information about sales transactions, including a ""Sales"" table and a ""Products"" table. You want to count the number of products with a price greater than $50.

Which DAX function would you use to count the number of products with a price greater than $50 in this scenario?"

Sequence generator



class Route:

def \_\_init\_\_(self, route\_id, route\_name, capacity, cost\_per\_km):

self.route\_id = route\_id

self.route\_name = route\_name

self.capacity = capacity

self.cost\_per\_km = cost\_per\_km

self.capacity\_available = capacity

class Shipment:

def \_\_init\_\_(self, shipment\_id, product\_type, quantity, distance\_km):

self.shipment\_id = shipment\_id

self.product\_type = product\_type

self.quantity = quantity

self.distance\_km = distance\_km

self.cost = 0 # Placeholder for calculated cost

class DistributionCenter:

def \_\_init\_\_(self, center\_id, center\_name, location):

self.center\_id = center\_id

self.center\_name = center\_name

self.location = location

class LogisticsManager:

def \_\_init\_\_(self, routes, shipments, distribution\_centers):

self.routes = [Route(\*\*r) for r in routes]

self.shipments = [Shipment(\*\*s) for s in shipments]

self.distribution\_centers = [DistributionCenter(\*\*dc) for dc in distribution\_centers]

def calculate\_transport\_cost(self, route, shipment):

cost = route.cost\_per\_km \* shipment.distance\_km

return cost

def check\_route\_availability(self, route, shipment):

if route.capacity\_available >= shipment.quantity:

return True

else:

return False

def optimize\_shipments(self):

for shipment in self.shipments:

for route in self.routes:

if self.check\_route\_availability(route, shipment):

cost = self.calculate\_transport\_cost(route, shipment)

if not shipment.cost or cost < shipment.cost:

shipment.cost = cost

shipment.selected\_route = route

def display\_shipment\_schedule(self):

for shipment in self.shipments:

print(f"Shipment ID {shipment.shipment\_id}:")

print(f" Product Type: {shipment.product\_type}")

print(f" Quantity: {shipment.quantity}")

print(f" Distance (km): {shipment.distance\_km}")

print(f" Selected Route: {shipment.selected\_route.route\_name}")

print(f" Transportation Cost: {shipment.cost:.2f}")

print("=" \* 40)

# Sample data

routes\_data = [

{"route\_id": 1, "route\_name": "Route A", "capacity": 100, "cost\_per\_km": 2.5},

{"route\_id": 2, "route\_name": "Route B", "capacity": 150, "cost\_per\_km": 3.0},

{"route\_id": 3, "route\_name": "Route C", "capacity": 200, "cost\_per\_km": 2.0},

]

shipments\_data = [

{"shipment\_id": 101, "product\_type": "Crude Oil", "quantity": 50, "distance\_km": 200},

{"shipment\_id": 102, "product\_type": "Natural Gas", "quantity": 1000, "distance\_km": 300},

{"shipment\_id": 103, "product\_type": "Crude Oil", "quantity": 80, "distance\_km": 150},

{"shipment\_id": 104, "product\_type": "Liquefied Natural Gas (LNG)", "quantity": 500, "distance\_km": 250},

]

centers\_data = [

{"center\_id": 1, "center\_name": "Center X", "location": "City A"},

{"center\_id": 2, "center\_name": "Center Y", "location": "City B"},

{"center\_id": 3, "center\_name": "Center Z", "location": "City C"},

]

# Create and manage logistics

logistics = LogisticsManager(routes\_data, shipments\_data, centers\_data)

logistics.optimize\_shipments()

logistics.display\_shipment\_schedule()

# Sample data for transportation routes

routes = [

{"route\_id": 1, "route\_name": "Route A", "capacity": 100, "cost\_per\_km": 2.5},

{"route\_id": 2, "route\_name": "Route B", "capacity": 150, "cost\_per\_km": 3.0},

{"route\_id": 3, "route\_name": "Route C", "capacity": 200, "cost\_per\_km": 2.0},

]

# Sample data for product shipments

shipments = [

{"shipment\_id": 101, "product\_type": "Crude Oil", "quantity": 50, "distance\_km": 200},

{"shipment\_id": 102, "product\_type": "Natural Gas", "quantity": 1000, "distance\_km": 300},

{"shipment\_id": 103, "product\_type": "Crude Oil", "quantity": 80, "distance\_km": 150},

{"shipment\_id": 104, "product\_type": "Liquefied Natural Gas (LNG)", "quantity": 500, "distance\_km": 250},

]

# Sample data for distribution centers

distribution\_centers = [

{"center\_id": 1, "center\_name": "Center X", "location": "City A"},

{"center\_id": 2, "center\_name": "Center Y", "location": "City B"},

{"center\_id": 3, "center\_name": "Center Z", "location": "City C"},

]

# Sample data for available routes and their capacities

available\_routes = [

{"route\_id": 1, "capacity\_available": 80},

{"route\_id": 2, "capacity\_available": 120},

{"route\_id": 3, "capacity\_available": 180},

]

# Sample data for logging shipment activities

shipment\_logs = []

# Sample data for product scheduling (for recursive function)

product\_quantities = {

"Crude Oil": 200,

"Natural Gas": 800,

"Liquefied Natural Gas (LNG)": 300,

}

pd.read\_csv(filepath\_or\_buffer, sep=’ ,’ , header=’infer’, index\_col=None, usecols=None, engine=None, skiprows=None, nrows=None)

Parameters:

filepath\_or\_buffer: Location of the csv file. It accepts any string path or URL of the file.

sep: It stands for separator, default is ‘, ‘.

header: It accepts int, a list of int, row numbers to use as the column names, and the start of the data. If no names are passed, i.e., header=None, then, it will display the first column as 0, the second as 1, and so on.

usecols: Retrieves only selected columns from the CSV file.

nrows: Number of rows to be displayed from the dataset.

index\_col: If None, there are no index numbers displayed along with records.

skiprows: Skips passed rows in the new data frame.

Moviedf.head()