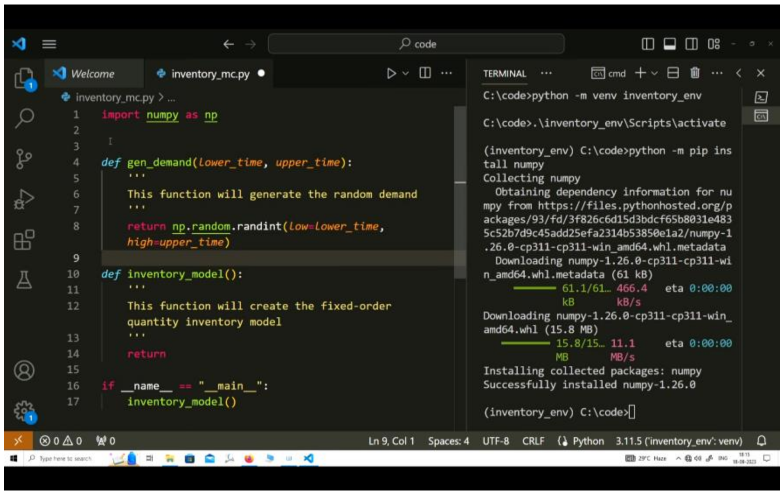
Advanced Business Decision Support Systems

Professor Deepu Philip
Department of Industrial Engineering and Management Engineering
Indian Institute of Technology, Kanpur
Professor Amandeep Singh
Imagineering Laboratory
Dr. Prabal Pratap Singh
Indian Institute of Technology, Kanpur

Lecture 36
DSS for Monte Carlo Inventory Simulation (Part 1 of 2)

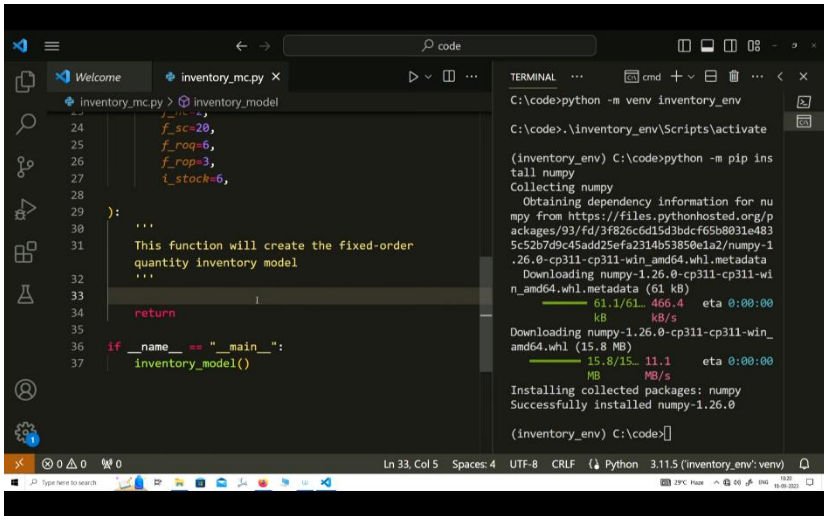
Hello everyone, I welcome you all to the lecture series on Advanced Business Decision Support Systems. I am Prabhul Pratap Singh from IIT Kanpur and we are discussing how to create different Decision Support Systems Using Python Programming Language. So, till now we have covered all our Basic Foundations for Creating a complete DSS system, we have also seen that How to Use Different Scientific Computing Libraries and we Developed a few Different Decision Support Systems as well. Now, in the last lecture, we have discussed How We Can Manage Inventory Systems Using Monte Carlo Simulations by Hand. Now, today we will start creating our Decision Support Systems based on these Inventory Systems and we will start by Creating a Complete Inventory Module Using the Python Programming Language.

So, let us start by opening a new directory, I named this as code and inside this we are opening our VS code using this open with code option.



```python
import numpy as np


def gen_demand(lower_time, upper_time):
    '''
    This function will generate the random demand
    '''
    return np.random.randint(low=lower_time, high=upper_time)

def inventory_model():
    '''
    This function will create the fixed-order quantity inventory model
    '''
    return

if __name__ == "__main__":
    inventory_model()
```

```
C:\code>python -m venv inventory_env

C:\code>.\inventory_env\Scripts\activate

(inventory_env) C:\code>python -m pip ins
tall numpy
Collecting numpy
  Obtaining dependency information for nu
mpy from https://files.pythonhosted.org/p
ackages/93/fd/3f826c6d15d3bdcf65b8031e483
5c52b7d9c45add25efa2314b53850e1a2/numpy-1
.26.0-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.26.0-cp311-cp311-wi
n_amd64.whl.metadata (61 kB)
     ━━━━━━━ 61.1/61… 466.4    eta 0:00:00
             kB       kB/s
Downloading numpy-1.26.0-cp311-cp311-win_
amd64.whl (15.8 MB)
   ━━━━━━━━ 15.8/15… 11.1      eta 0:00:00
            MB       MB/s
Installing collected packages: numpy
Successfully installed numpy-1.26.0

(inventory_env) C:\code>
```



```python
        f_sc=20,
        f_roq=6,
        f_rop=3,
        i_stock=6,

):
    '''
    This function will create the fixed-order
    quantity inventory model
    '''

    return

if __name__ == "__main__":
    inventory_model()
```

```
C:\code>python -m venv inventory_env

C:\code>.\inventory_env\Scripts\activate

(inventory_env) C:\code>python -m pip ins
tall numpy
Collecting numpy
  Obtaining dependency information for nu
mpy from https://files.pythonhosted.org/p
ackages/93/fd/3f826c6d15d3bdcf65b8031e483
5c52b7d9c45add25efa2314b53850e1a2/numpy-1
.26.0-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.26.0-cp311-cp311-wi
n_amd64.whl.metadata (61 kB)
     ━━━━━━━ 61.1/61… 466.4    eta 0:00:00
             kB       kB/s
Downloading numpy-1.26.0-cp311-cp311-win_
amd64.whl (15.8 MB)
     ━━━━━━━ 15.8/15… 11.1     eta 0:00:00
             MB       MB/s
Installing collected packages: numpy
Successfully installed numpy-1.26.0

(inventory_env) C:\code>
```

So, I have already opened it here and you can see that my directory is still empty and my terminal is active and I have no base environment. So, my first task is to create a new python environment using python -m venv inventory_env. So, now you can see here that as in the previous lectures this command has created a new environment here. Now, to activate this environment, we can write \inventory_env\scripts\activate.

Now, I have successfully activated my environment and I can push this panel to the right, so we can have more space. Now, let us create a new file and name it as inventory Monte Carlo.py. So, this is the main module which we are going to start creating. So, our first task is to create our various imports, so before importing we need to first install NumPy because we need different kinds of random numbers to generate for developing our Monte Carlo Simulation. So, we need this library named NumPy.

So, we know how to install this, that is we can write python m pip install NumPy. Now, you can see that it has successfully installed and we can also check this by moving to our environment folder and we can see that inside the lib folder this is the NumPy which was not earlier available. So, now we can start using this module, so we can write import NumPy as np. Now, we can write if_ name =, so here we are checking that if we are executing this file directly from the terminal, then this should run. So, we can write here pass, so whatever happens if we run this file from the terminal, then this function will run.

Now, let us start developing our function def inventory_model, this function will create the inventory model of the type 50 fixed-order quantity inventory model. So, let us just write a simple return statement and if we now call this function here, inventory model. So, until now, we have defined that we are importing this module as an alias called NumPy, then we have created this conditional statement which will check whether we are running this file from the terminal, and then if it is true, then it will call this function. What is this function? This function will hold the complete module for creating our fixed-order quantity model.

So, as we have discussed in our previous lecture that while running the simulation by hand we were generating demand and delivery time by hand, but now since we have our NumPy installed, we can use this library to generate the random demand.

So, we can write a different function for that and let us say the name of the function is gen demand. And, for this function we will do is, we can write the doc string saying this function will generate the random demand and let us write the random demand generation by using random dot. So, I am generating random integers and the low value is 1 unit 0, or instead of hard coding what we can do is, we can write here 0, and we can again write here the high value as some 4 like we discussed earlier, but instead of hard coding this we can also create our arguments here.

So, we can write here that the lower time is an argument for this function and upper higher time or upper time is another argument here and we can directly pass these arguments to this random function. So, we can write lower and upper time and high is upper time and we are not using any size argument here for this random because we only need one number each time we call this gen demand function.

Again, we also need to create a delivery time demand. So, we can write def gen delivery_time, and we can pass two arguments here as well. Which is, this was the delivery time and here this could be lower demand and this is upper demand, lower demand, upper demand. Now, we can use lower time because we are now generating the time and this is the upper_time.

Again, we need to provide the doc string here which will say this function will generate the delivery time of the order randomly, and we can generate this demand by saying np.random.z int and the low value is lower_time, high is upper_time. So, this way we have created two different functions which will generate the demand and time, each time we call that function.

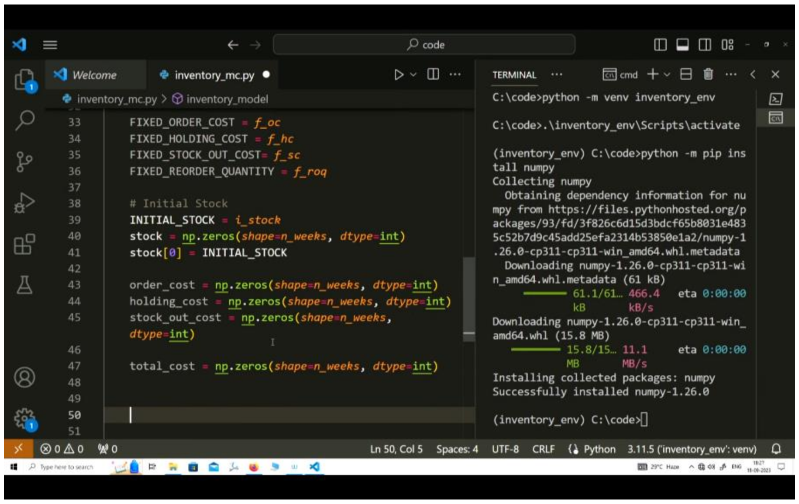
Now, let us start writing this function, so this function should have different types of arguments that should be passed by the user. So, what are those arguments? The first should be the number of weeks for which we are running this simulation. So, let us say the argument for that is 'n' week and the default value for this is 20.

So, I have already discussed that we can pass the default values directly during the function definition, like this. So, even if the user call this function without any arguments this 'n' week will have the value 20. The next thing that it should have is the lower quantity of lower demand and let us say the default value is 1.

Again, the upper demand should be 4. So, we can also capture these data points from the user but for this case since we have already discussed this in our previous lecture that these were the values, so we are running the same simulation by coding it in python. So, the lower time can have this argument 'l' order_time. So, this is, that the lower ordering time can have the value from 1 to upper order time which can have the value until 3. And, now define all the ranges for the demand and time, now we need to also tell our simulation model that what are the different types of fixed cost that this simulation model holds.

So, the fixed-ordering cost or OC is ordering cost should be 30 units, the fixed holding cost or carrying cost should be 2 units and we need to use the comma. Then, we should also mention the fixed stock out cost which is SC should be 20 units. So, these are all the three main costs for each week which will get aggregated to give us the final total cost of the week.

Since this is a fixed-order quantity model, we also need to tell what is the fixed quantity and the fixed-order quantity and the fixed quantity that is available initially. So, we can mention this also by creating a new argument named as fixed reorder quantity which should be 6 and the fixed reorder point which we discussed as 3 and the initial stock for starting the simulation is also 6,



```python
    FIXED_ORDER_COST = f_oc
    FIXED_HOLDING_COST = f_hc
    FIXED_STOCK_OUT_COST= f_sc
    FIXED_REORDER_QUANTITY = f_roq

    # Initial Stock
    INITIAL_STOCK = i_stock
    stock = np.zeros(shape=n_weeks, dtype=int)
    stock[0] = INITIAL_STOCK

    order_cost = np.zeros(shape=n_weeks, dtype=int)
    holding_cost = np.zeros(shape=n_weeks, dtype=int)
    stock_out_cost = np.zeros(shape=n_weeks,
    dtype=int)

    total_cost = np.zeros(shape=n_weeks, dtype=int)
```

```
C:\code>python -m venv inventory_env

C:\code>.\inventory_env\Scripts\activate

(inventory_env) C:\code>python -m pip ins
tall numpy
Collecting numpy
  Obtaining dependency information for nu
mpy from https://files.pythonhosted.org/p
ackages/93/fd/3f826c6d15d3bdcf65b8031e483
5c52b7d9c45add25efa2314b53850e1a2/numpy-1
.26.0-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.26.0-cp311-cp311-wi
n_amd64.whl.metadata (61 kB)
     ━━━━━━━ 61.1/61… 466.4    eta 0:00:00
             kB       kB/s
Downloading numpy-1.26.0-cp311-cp311-win_
amd64.whl (15.8 MB)
    ━━━━━━━━ 15.8/15… 11.1     eta 0:00:00
             MB       MB/s
Installing collected packages: numpy
Successfully installed numpy-1.26.0

(inventory_env) C:\code>
```

code
Welcome
inventory_mc.py
inventory_mc.py > inventory_model
TERMINAL
cmd
Ln 50, Col 5   Spaces: 4   UTF-8   CRLF   Python   3.11.5 ('inventory_env': venv)

So, these were the main arguments which we require and after that we can start coding our simulation model. So, to do that, we first need to capture these three arguments and I have already discussed that to create some constants that do not change for the complete run, we can define these constants using the upper case variable name. So, the first constant that remains unchanged for the whole program is the order cost. So, we can write fixed_order_cost equals and capture this under FOC. So, this is going to get the value from this FOC argument,

Similarly, we can write a Fixed Holding Cost which is FHC. Next is Fixed Stock-out Cost which is FSC and the last fixed thing is Fixed Reorder Quantity which is FROQ. So, we have captured all these into our different constant variables. Now, we need to initialize our stock. Encoding as well, we are trying to capture all the steps that we have already discussed by hand and we are just doing all these steps using the python language.

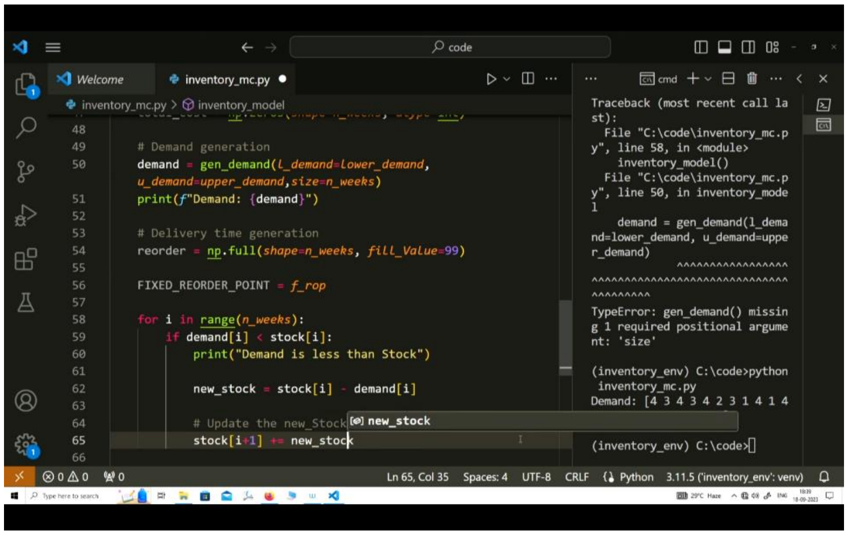
So, to initialize the stock, we need to create a new variable named as stock and we need to generate np.random.send int. So, we need to generate some random stock values, but what we can do is, since we do not know beforehand what are the different values that this will hold. So, we can just create a new data location in our python, so that it can hold 0 values.

And, once we get these stock values for each week, we will keep on filling those values into this variable. So, the initial stock can have the 0s by using the NumPy function, and we need to mention how many 0s we need to generate. So, since we have already defined that we are running the simulation for these number of weeks, so we need these at least 20 0s for this array, So, we can write the shape as 'n'_weeks. And, we must mention our data type of this variable as integer. So, we need to initialize our initial stock for this simulation. So, this is also a constant that will not change for the whole run. So, the initial stock is the 'i'_stock.

Now, our stock related thing is over and we must tell our code that the first stock value which is currently 0 should be equal to initial stock. Now, this stock initialization is over, we can save the file until now. And, the next thing is, to generate other arrays as well, like for order. So, we can write the order cost. So, for each week, we need a data variable that can hold the order cost for that week. So, we can again generate this using np.0s shape = n_weeks nd type = it.

The other cost is the holding cost. We can generate this again using 0s with the shape = n_weeks, type is it, The third thing is the stock out cost. And, finally the total cost, we will also capture this total cost by calculating this cost for each week. So, to do that, we will again initialize a blank array with 0s.

Now, we have created and initiated all the variables that we require for our model. Now, we will use this generate demand function which we have defined here.



```
    # Demand generation
    demand = gen_demand(l_demand=lower_demand,
    u_demand=upper_demand,size=n_weeks)
    print(f"Demand: {demand}")

    # Delivery time generation
    reorder = np.full(shape=n_weeks, fill_value=99)

    FIXED_REORDER_POINT = f_rop

    for i in range(n_weeks):
        if demand[i] < stock[i]:
            print("Demand is less than Stock")

            new_stock = stock[i] - demand[i]

            # Update the new_Stock
            stock[i+1] += new_stock
```

```
Traceback (most recent call la
st):
  File "C:\code\inventory_mc.p
y", line 58, in <module>
    inventory_model()
  File "C:\code\inventory_mc.p
y", line 50, in inventory_mode
l
    demand = gen_demand(l_dema
nd=lower_demand, u_demand=uppe
r_demand)
             ^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^
TypeError: gen_demand() missin
g 1 required positional argume
nt: 'size'

(inventory_env) C:\code>python
 inventory_mc.py
Demand: [4 3 4 3 4 2 3 1 4 1 4

(inventory_env) C:\code>
```

So, to do that, we can write here demand generation. And, how to do this is, let us say, we will call the function gen demand, and we call the lower demand = the lower demand. So, what is happening here is, this first lower demand is the argument for this function, this lower demand.

And, since this function is also getting the same argument with the same name, so we can write here lower demand. We can also change this, let us say, we can write here to remove this confusion, we can write here 'l' demand and upper demand can be 'u' demand. And, we need to keep this synchronized with the function call. So, we will make this similar. Now, what I was discussing is that now we can write here 'l' demand. For this function the argument is 'l' demand and since we are working in this inventory model function and this has this argument. So, we are using here l demand = lower demand. Similarly, u demand = upper demand.

Now, what will happen is, this function call will get its default lower demand as 1 and upper demand as 4. And, this function, when we are calling this generate demand function, it is transferring the lower demand value as 1 to this function and upper demand value as 4 to this function. So, it will return a value generated from the random module from the NumPy and we can capture this value in the demand variable here.

Now, this will generate only 1 demand but what we can do is, here we can generate a complete set of demands for all the weeks because we know that in every week we need a particular demand to get generated. So, to do that, we can actually use the size argument here and we can write here the size as the size + 1, and we need to also get this size argument here.

Now, y + 1 because we need to add one more because this is at the exclusive command. So, we need to add one more value each time. Similarly, we need a + 1 here. So, if we need the data from 1 to 4, then we will type 1 to 5, and it will generate until 4 only. Now, we have generated a series of values which is an NumPy array for the demand.

Now, moving further ahead let us also check whether this demand is getting generated or not. So, we can write a simple print statement here using the as strings. Let us try to run this. So, we have added one argument here, but we have not modified our function call. So, we can write here as the size as 'n'_weeks.

So, now we can see here that we are generating this demand. So, each number is a demand for a particular week. So, let us again move forward and try to add other features to this model.

So, the next thing which we need to generate randomly is the delivery time of the reorder. So, we can write delivery time generation and let us store this array into this reorder variable and can create. Now, the question is why I am using this other function. I have already discussed this function from the NumPy which is full, and what it will do is, it will create an array filled with the value 99 for all the data points available in this array. But why we are doing this and what is the significance of using 99, can we not use any other value.

So, the point is, this reorder we do not know and we need to track whether we are going to reorder in a particular week or not. So, with 99, when I was doing the simulation by hand, I was using a -wherever I was not reordering anything. So, this 99 holds a similar functionality here in our python code that if the initial value we initiated this array with 99 value.

If the 99 value does not get changed during the function call, then it means that the decision made by this inventory module says that the particular week does not have a reorder. So, this way we can create some arrays that can have a particular value which coder knows why it is creating, prefilling it with a particular value and how he or she is using this value to track a particular feature in your module,

So, I will show you how I will use this 99 value in the code that I will write further ahead. So, the next constant which we need to mention is fixed reorder point and this is captured from the FROP value which we are capturing from the user or as a default value as 3 units. So, anything if the stock goes before below less than or equal to 3 units, then we have touched our fixed reorder point.

Now, the initial phase is over, so what we can do here is, we can start iterating for all the weeks. So, there are 20 weeks. So, to iterate over all the weeks we need to use a loop statement. We have learnt different kinds of loop statements like 'for' or 'while', so we will be using 'for loop' here. To iterate over the weeks we can write- for i in range n_weeks and what is the first thing we need to check is whether the demand is less than or not with respect to the stock.

So, we can write our conditional statement as if demand is 'i', since demand is an array, so we can use this locational operator 'i' and check whether it is less than or not with respect to the stock, the 'i'th value of stock. So, we can write here for our information that the demand is less than stock. Now, if the demand is less than stock, then what is the next thing we need to do is, by hand we are updating our stock value. So, we need to fulfill the demand from the consumer, so the new stock will be stock_i, demand 'i'.

Now, for each week we know what the new stock value is, and we need to also update this, update the new stock value for the next week. So, this is the 'i'th week in which we are iterating right now, but since we have the new stock value we need to update the next stock that is available for the next week.

So, what we can do here is, we can update stock i + 1 that is the next week and we should update this with new stock. So, this + equal to sign is nothing but if we write equal to stock i + 1 + new stock. So, instead of writing this statement, here we are writing this stock i + 1 twice, but python provides this feature that if we remove this and just write + equal to, it means the same thing, so it is incrementing this new stock value to this actual i + 1 location of the stock array. But here is a problem that, let's say, in the 20th week we are running this command, so for the stock location of the 21st week stock should update the new stock value, but there is no 21st week we have defined. So, what will happen is, your code will stop and it will show you an error. So, to bypass that error, there is a feature in python known as try except block, so we can write this try block and what it will say to the python interpreter is to try this statement and if it does not work, then it should do this next block.

So, usually, this error is known as index error because we are trying to assess some index that is not available, so we can write if this except index error occurs, then we can write here something like print index out of bounds and we are updating stocks in advance.

So, these kinds of print commands will help you when you are running the code and try to debug the code. So, there is a different module named logging which you can use here instead of writing a print command. This is not the standard way of writing these debugging errors, but since this is a beginner introduction to how to write python, we are not using that different module of logging. So, we can write a simple print statement. It will work fine and we can also mention that we are inside the demand less than stock module.

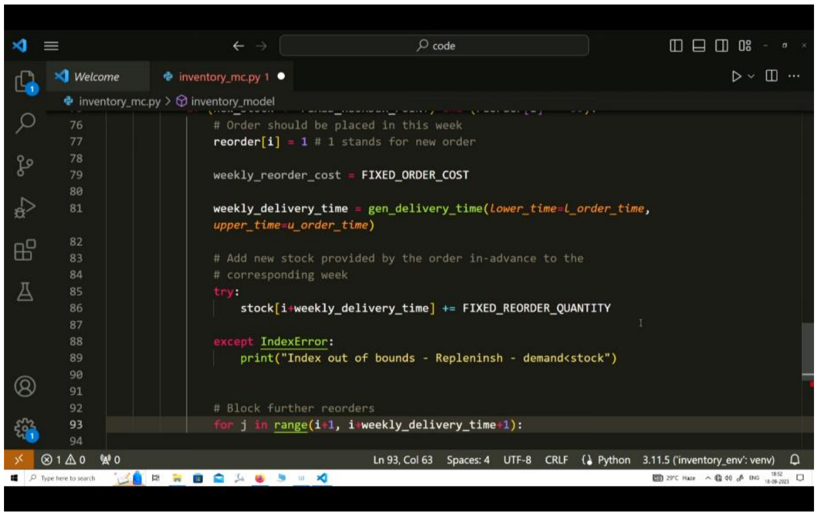
So, in this conditional statement this error is occurring, because this error will occur in other conditional statements as well that is why we are giving this much information to the debugger.

So, the next thing is, we can create a new variable named as weekly stock out cost and we can initiate it this with 0, so what we are doing here is since the demand is less than stock, so we have fulfilled the complete demand that is why our stock out cost will be 0 for this period. So, we can write here a comment like all demand is fulfilled. That is why this stock out cost is 0, otherwise we need to calculate this stock out cost as well.

Now, we need to check whether we need to reorder or not, so to do that we need to create a new conditional statement which will check whether we have reached the reorder point or not. So, we can write if new under store stock is less than equal to fixed reorder point and reorder 'i'. This is the same array which we generated to capture whether there is a reorder or not, so we are writing whether it = 99 or not. If this is true, if both of these things are true, then order should be placed in this week.

So, how to place an order? We need to update this reorder array for the 'I'th location as 1, so 1 stands for new order and we can write here where we are generating this reorder array, we can write here that 99 is a placeholder that will tell that there are no active orders in a particular week. So, now we have to reorder. So, to do a reorder, we will accumulate a cost and we can store that cost as a weekly reorder under store cost which should since this is fixed, so we can use fixed order cost.

Now, we know that we need to order. In how many weeks will we receive our order? To do that probabilistically, we will generate a new number for that week.


code
Welcome
inventory_mc.py 1
inventory_mc.py > inventory_model
Ln 93, Col 63 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('inventory_env': venv)

```
            # Order should be placed in this week
            reorder[i] = 1 # 1 stands for new order

            weekly_reorder_cost = FIXED_ORDER_COST

            weekly_delivery_time = gen_delivery_time(lower_time=l_order_time,
            upper_time=u_order_time)

            # Add new stock provided by the order in-advance to the
            # corresponding week
            try:
                stock[i+weekly_delivery_time] += FIXED_REORDER_QUANTITY

            except IndexError:
                print("Index out of bounds - Repleninsh - demand<stock")


            # Block further reorders
            for j in range(i+1, i+weekly_delivery_time+1):

```

So, we can use our function that we have defined here which was to generate delivery time and we can write here as gen_delivery time and what it requires is, the lower time as 'l' order time and upper time as 'u' order time. So, it will give us one value for this order delivery time. So, we can store this one value into a different variable named as weekly_delivery_time.

Now, we know in how many weeks we will receive this order. So, we can in advance, add this new stock to the particular week. So, to do that, we can provide a comment here that adds new stock provided by the order in advance to the corresponding week. So, since we are again using locational access of the array, so we need to provide the try and except statements to not get the error and what we need to run is, we need to update our stock for the i + 1th week and i + 1 will tell that in the next week we will get the order which we ordered in this here. But this gen delivery time is probabilistic, so this is the actual time which we need to add to this location variable.

So, we can write weekly delivery times. Now, what will happen is, let us say, we are in the iteration of the first week, so 'i' is one and the week delivery time generated by this function is two weeks, so it will add two weeks to this location, so it will be stock of three. So, at the third location of this array, we will need to add the stock and every time we are ordering a fixed quantity. So, we will add a fixed reorder quantity. And, this value is six for our simulation, but it can be anything for your customized simulation.

But, after trying we need to provide an accept statement as well for this index error, which may happen when we are running towards the end of the iteration for the number of weeks. So, we can provide a print statement here saying this index of those bounds we are inside replenished and demand is still less than stock. Now, one more thing we need to do is, we need to block any further reorders, so to do that, we need to again start a new iteration and block further reorders.

So, we can write a nested 'for loop' here by writing a 'j' variable for the range starting from 'i' + 1 until the order gets delivered which is i + weekly delivery time + 1. Why + 1? Because, this is the range function and it will exclude the last value, so we need to manually add one value, so that the weekly delivery time gets accommodated completely.

Now, let us say that for this number of weeks, we do not need to reorder again. Even if we are not fulfilling the complete consumer demand, we should write reorder of 'j' because we are using the 'j' variable in this nested 'for loop' and we should mention 0. So, this is the same array which stores 99 values as the default, from here. So, 99 value is a placeholder but value 0 tells that the particular week should not initiate a new order and value 1 tells that the particular week initiated a new order. So, this is how we can use a new array.

Now, this can also give you an index error. So, we will again write this try block here and write our except statement which should print a useful line like index errors block reorder demand is less than stock. So, now if you can see that we are writing this whole code until this conditional statement which was checking whether we should place an order or not. So, our order placement and our order tracking is complete here, until this line.

Now, we need to create an 'else' statement that is what if we do not need to order, so what happens then? So, to do that, we can write 'else', this 'else' will be for this ‘if’ statement. So, I am checking the indentation by this highlighted grade line, that my indentation is correct or not. So, we can write that no order will be placed in this week and we can update weekly order cost as 0. Now, everything is done for a particular week. So, what we can do is, we can calculate our cost. So, cost calculations should go beyond this line and we first need to calculate our holding cost for each week which can be assessed using this way by using the ith operator because we are now again in the 'for loop' using the 'i' variable.

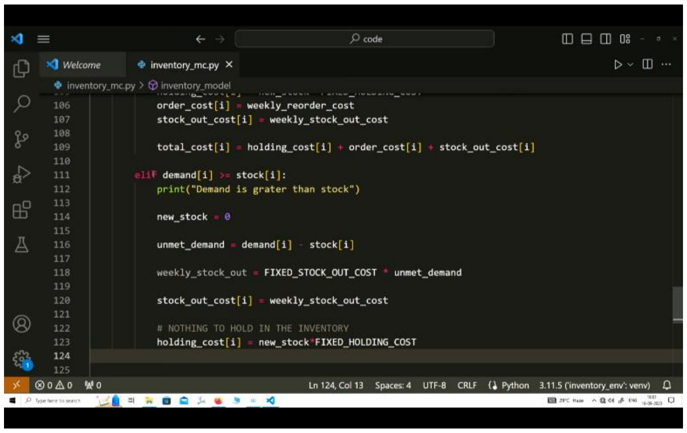
So, each week, if we are holding the new stock, that is when we update our stock details, we found that this is the condition where the demand was less than stock. So, the new stock holds the number of items that are still available in the inventory. So, these are the actual number of items that we are holding, so holding cost will include these new stock variables and it should get multiplied by the fixed holding cost and order cost similarly can be calculated by using the weekly order.

So, this is the weekly order cost and the last is the stock out cost which should be updated if there is a stock out. So, weekly stock out cost holds this value for this week. Now, the total cost for the ith week should be the summation of all these three costs, holding cost for the ith week, order cost for the ith week, and the stock out cost for the ith week.

So, this way we have finally created the complete condition when our demand is less than stock but there is one more condition. So, this was the actual statement and we need to maintain our indentation level. So, the other condition is, if the demand for the ith week is greater than equal to stock of 'i'. So, if the demand is greater, then let us first provide a simple string that the demand is greater than stock which means that the organization cannot fulfill the complete demand based on the inventory levels.

So, we can say that whatever the stock is, we will completely empty the stock from the organization. So, the new stock which holds the current level of the stock after fulfilling the demand from the consumer is zero here and we can also write the unmet_demand to capture the unmet demand and we can write demand of 'i' since the demand is greater here. So, this will give a positive value stock of 'i'. So, unmet demand will capture that. Now, the weekly stock out cost will be calculated for this week because we are not fulfilling the complete demand.

So, we should use the fixed stock out cost and multiply it with the unmet demand. Also, the stock out cost for the 'i'th week should get updated with this weekly stock out cost. Now, since there is nothing to hold in the inventory, the holding cost is zero for this condition. So, we can write the holding_cost of 'i' equals to, let us say, we write the same thing, new stock into fixed holding cost but since new stock is zero, so this will always be zero. So, we can write zero as well.


code
Welcome
inventory_mc.py
inventory_mc.py > inventory_model
Ln 124, Col 13 Spaces: 4 UTF-8 CRLF Python 3.11.5 ('inventory_env': venv)

```python
            order_cost[i] = weekly_reorder_cost
            stock_out_cost[i] = weekly_stock_out_cost

            total_cost[i] = holding_cost[i] + order_cost[i] + stock_out_cost[i]

        elif demand[i] >= stock[i]:
            print("Demand is grater than stock")

            new_stock = 0

            unmet_demand = demand[i] - stock[i]

            weekly_stock_out = FIXED_STOCK_OUT_COST * unmet_demand

            stock_out_cost[i] = weekly_stock_out_cost

            # NOTHING TO HOLD IN THE INVENTORY
            holding_cost[i] = new_stock*FIXED_HOLDING_COST

```

So, we need to still add a few more features in this module and like for this condition. And, in the next lecture, we will go further ahead from this step and we will try to complete this module, then we will develop our new DSS by using the flask application module and we will call this inventory module which we are creating today into that application module which will create our complete DSS system. So, thank you once again and we will meet in the next lecture.