Exercises - Section 4: Lecture 12 – Creating Subscripted Variables – Solutions - Part 3

For each of the following descriptions, create the necessary notation and Python code for the data and/or variables.

DESCRIPTION	MATH	GUROBIPY
1. The height of each person (data)	h_i for each person i *NOTE: Of course, you can choose whatever letter you want for the data and the subscript. Here, I chose h for height, but you could've used a (as the first letter of the alphabet) or some other letter; similarly, I chose i to be the subscript for people, but you could've used p for "people", or any other letter. This note applies for every other exercise as well.*	`P = 100 # Number of people` `h = [get_height(p) for p in range(P)]` *NOTE: The function* `get_height()` *is assumed to have been defined elsewhere in your Python code. Similar assumptions are made in the other answers below.*
2. The weight of each person (data)	w_i	`w = [get_weight(p) for p in range(P)]`
3. The height and weight of each person (data)	h_i, w_i for each person i *or* a_{ij} where i is the person and j is the attribute number (for example, j=1 is height and j=2 is weight). *NOTE: The second version is easier for quickly writing Python code to read a matrix of data, but it's harder to keep track of which attribute number means which thing.*	`h = [get_height(p) for p in range(P)]` `w = [get_weight(p) for p in range(P)]`
4. The height of each person at each age (ages 1, 2, 3, …) (data)	h_{it} for each person i and each age t	`h = {(p, get_age(p)): get_height(p) for p in range(P)}`

	Here, the subscripts are i for the person and t for the age (time) of the measurement.	
5. The distance between every pair of the largest 50 cities in the United States (data)	d_{ij} for each pair of cities i and j *NOTE: Notice that because distance is symmetric, the distance from city i to city j is the same as the distance from city j to city i, so $d_{ij} = d_{ji}$.*	`d = {(i,j): get_distance(i,j) for i in cities for j in cities if j!=i}` *NOTE: The phrase* `if j!=i` *means that no data is recorded from a city to itself (e.g, the distance from New York to New York). If your data included values of 0 for those distances, then you could remove the "if..." phrase.*
6. Cost per share of each investment (data)	c_i for each investment i	`c=[get_investment(i) for i in investments]`
7. Amount of money invested in each investment (data)	a_i for each investment i	`a=[get_amount(i) for i in investments]`
8. Amount of money invested in each investment (variable)	x_i for each investment i *NOTE: The answers to this question and the previous question could be the same; I used different letters only because of the tradition of starting data with a and variables with x. The practical difference between the two is that when the amount of money invested is data, it's something that the optimization model can't suggest changes to – the value just is what it is (perhaps that money was invested in the past) – but when it's a variable, the optimization model can suggest what value the variable should have (perhaps as a suggestion for how to invest money going forward).*	`m = gp.Model()` `x=m.addVars(investments, lb=0, name="x")`
9. Number of sweatshirts purchased (variable) if size is the only important attribute	x_s for each size s *Here, s is the subscript for the size, with different values*	`x = m.addVars(size, lb=0, ub=len(size), vtype=GRB.INTEGER, name="x")`

	of s for small, medium, large, etc.	*NOTE:* `GRB.INTEGER` *means that the variable is required to take an integer value (it's not possible to purchase a fraction of a sweatshirt).*
10. Number of sweatshirts purchased of each color (variable)	x_c for each color c *Here, c is the subscript for the color, with different values of c for green, red, blue, etc.*	`x = m.addVars(colors, lb=0, ub=len(colors), vtype=GRB.INTEGER, name="x")`
11. Number of sweatshirts purchased of each color and size (variable), if both attributes are important (for example,	x_{sc} for each size s and each color c *Now, there are two subscripts, one for size and one for color, so x_{sc} is the number of sweatshirts of size s and color c purchased.*	`k=[(s,c) for s in size for c in color]` `x=m.addVars(k, lb=0, ub=len(colors)*len(size), vtype=GRB.INTEGER, name="x")`
12. An airline wants to hire flight attendants who live in each city. The airline will then assign each flight attendant to a daily schedule that allows the flight attendant to return home each evening. Specify notation and Python code for (1) data showing the minimum number of flight attendants the airline needs to hire in each city, (2) a variable for the number of flight attendants the airline will hire in each city, and (3) a constraint to require the airline to hire enough flight attendants in each city.	*In the following solution, the subscript i represents the city.* Data: m_i for all cities i Variables: x_i for all cities i Constraints: $x_i \geq m_i$ for all cities i	`m = [get_m(i) for i in cities]` `x = m.addVars(cities, name="x")` `m.addConstrs((x[i] >= m[i] for i in cities), name="lb")` *NOTE: An alternative way to write the constraint in gurobipy is:* `for i in cities:` `  m.addConstr(x[i] >= m[i], name=f"lb{i}")` *In this alternative writing, each constraint is added separately.*
13. An airline wants to hire flight attendants who prefer to work on each aircraft type (Boeing 747, Airbus A380, etc.). The airline will then assign each flight attendant to a daily schedule that allows the flight attendant to	Now, the subscript i represents the aircraft type. Data: m_i for all aircraft types i Variables: x_i for all aircraft types i	*NOTE: Notice that this solution is exactly the same as for #12 except the set of cities is replaced by the set of aircraft types.* `m = [get_m(i) for i in aircraft]`

work on the preferred aircraft. Specify notation and Python code for (1) data showing the minimum number of flight attendants the airline needs to hire for each aircraft type, (2) a variable for the number of flight attendants the airline will hire for each aircraft type, and (3) a constraint to require the airline to hire enough flight attendants for each aircraft type.	Constraints: $x_i \geq m_i$ for all aircraft types i	`x = m.addVars(aircraft, name="x")` `m.addConstrs((x[i] >= m[i] for i in aircraft), name="lb")` *NOTE: An alternative way to write the constraint in gurobipy is:* `for i in aircraft:` `  m.addConstr(x[i] >= m[i], name=f"lb{i}")` *In this alternative writing, each constraint is added separately.*
14. An airline wants to hire flight attendants who live in each city and who prefer to work on each aircraft type that the airline operates in that city. The airline will then assign each flight attendant to a daily schedule that allows the flight attendant to work on the preferred aircraft and return home each evening. Specify notation and Python code for (1) data showing the minimum number of flight attendants the airline needs to hire in each city for each aircraft type, (2) a variable for the number of flight attendants the airline will hire in each city for each aircraft type, and (3) a constraint to require the airline to hire enough flight attendants in each city for each aircraft type.	*Now, two subscripts are required. Below, i will represent the city and j will represent the aircraft type (you could use other letters).* Data: m_{ij} for all cities i and aircraft types j Variables: x_{ij} for all cities i and aircraft types j Constraints: $x_{ij} \geq m_{ij}$ for all cities i and aircraft types j *NOTE: It's also fine to switch the order of the subscripts and put the subscript for aircraft type first, as long as the change is consistent:* Data: m_{ji} for all aircraft types j and cities i	`m = {(i,j): get_m(i,j) for i in cities for j in aircraft}` `x = m.addVars(m, name="x")` `m.addConstrs((x[i,j] >= m[i,j] for i in cities for j in aircraft), name="lb")` *or* `for i in cities:` `  for j in aircraft:` `  m.addConstr(x[i,j] >= m[i,j])`

	Variables: x_{ji} for all aircraft types j and cities i Constraints: $x_{ji} \geq m_{ji}$ for all aircraft types j and cities i	
15. Cost of hiring a flight attendant in each city (data; can be different from city to city because the cost of living in each city is different)	c_i for all cities i	`c = [get_c(i) for i in cities]`
16. Cost of hiring a flight attendant who prefers each aircraft type, in each city	c_{ij} for all cities i and aircraft types j	`c = {(i,j): get_c(i,j) for i in cities for j in aircraft}`
17. Total cost of hiring all flight attendants, assuming cost differs both by aircraft type and by home city	$\sum_i \sum_j c_{ij} x_{ij}$	`S = gp.quicksum(c[i,j] * x[i,j] for i in cities for j in aircraft)`
18. Total cost of hiring all flight attendants, if only city is taken into account for hiring and cost	$\sum_i c_i x_i$	`S = gp.quicksum(c[i] * x[i] for i in cities)`
19. Total cost of hiring all flight attendants, if both city and aircraft type are important for hiring but cost depends only on home city	$\sum_i \left(c_i \sum_j x_{ij} \right)$ *or* $\sum_i \sum_j c_i x_{ij}$ *NOTE: In either of these two solutions, the variable has two subscripts but the cost only requires one, because it only depends on the flight attendant's home city. Compare with #20, where the cost depends on both city and aircraft type.*	`S = gp.quicksum(c[i] * x[i,j] for i in cities for j in aircraft)` *or* `S = gp.quicksum(c[i] * gp.quicksum(x[i,j] for j in aircraft) for i in cities)` *NOTE: In either of these two solutions, the variable has two subscripts but the cost only requires one, because it only depends on the flight attendant's home city. Compare with #20, where the cost depends on both city and aircraft type.*
20. Total cost of hiring all flight attendants, if both city and aircraft type are important for hiring, and cost	$\sum_i \sum_j c_{ij} x_{ij}$	`S = gp.quicksum(c[i,j] * x[i,j] for i in cities for j in aircraft)`

depends on both city and aircraft type	*NOTE: Now there's only one possible solution; because the cost depends on both city and aircraft type (two subscripts), it has to be within the second sum.* *Compare with #19, where the cost only depends on city, not aircraft type.*	*NOTE: Now there's only one possible solution; because the cost depends on both city and aircraft type (two subscripts), it has to be within the second sum.* *Compare with #19, where the cost only depends on city, not aircraft type.*

In the next set of exercises, think about which parts of the description are relevant, and write the math and gurobipy code including only relevant factors. Different people might have slightly different answers here depending on how you envision the situation; the important thing is to get used to thinking about what features might be relevant, just like one might consider in determining features to use in a data science model.

DESCRIPTION	**MATH**	**GUROBIPY**
21. Cost of hiring a flight attendant who prefers each aircraft type and does or doesn't speak Spanish, in each city (assume all three factors are relevant)	c_{ijk} for all cities i and aircraft types j and Spanish abilities k *NOTE: The new subscript k, for being Spanish-speaking, will either be 0 (no) or 1 (yes), or "N" (no) or "Y" (yes), or something similar; whatever is used, it'll have just two values… unless there are more than two categories for which the cost might be different – for example, if the cost for "not at all", "some, but not fluent", and "fluent" can all be different.*	`c = {(i,j,k): get_c(i,j,k) for i in cities for j in aircraft_type for k in Spanish}`
22. Cost of hiring a flight attendant who prefers each aircraft type and does or doesn't speak Spanish, in each city, if Spanish-speaking ability is not relevant for any costs or constraints or decisions in the model	c_{ij} for all cities i and aircraft types j *If the last subscript's values would be irrelevant, there's no need to include it.*	`c = {(i,j): get_c(i,j) for i in cities for j in aircraft}`
23. Cost of hiring a flight attendant who prefers each aircraft type, has each hair color, is each different height, speaks or doesn't speak Ancient	c_{ij} for all cities i and aircraft types j *Assuming that hair color, height, Ancient Greek fluency,*	`c = {(i,j): get_c(i,j) for i in cities for j in aircraft}`

Greek, and does or doesn't have a pet dog, in each city	*and dog ownership are irrelevant for this planning question, there's no need to include subscripts for them.*	
24. Cost of hiring a flight attendant who prefers each aircraft type, wants to work each different number of hours per week (from 30 to 60), has each hair color, does or doesn't have a pet dog, speaks or doesn't speak Spanish (relevant), in each city	c_{ijhs} for all cities i and aircraft types j and hours h and Spanish abilities s *A reasonable assumption is that pets and hair color are not relevant, but the other attributes might be.*	`c = {(i,j,h,s): get_c(i,j,h,s) for i in cities for j in aircraft_type for h in hours for s in Spanish}`

NOTES: