# Lab – 8 Tuple and Dictionary

Tasks:

1. Write a function to loop through and count the occurrence of all numbers from the numbers list. Display all numbers with occurrence greater than 1 in ascending order of the number (key).

# your code goes here

# Output Generation. You are not allowed to modify the following codes

def main():

numbers = [

951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725,

547, 544, 615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390,

984, 592, 236, 105, 942, 941, 386, 462, 47, 418, 907, 344, 236,

375, 823, 566, 597, 978, 328, 615, 953, 345, 399, 162, 758, 219,

918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217, 815,

67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445,

742, 717, 958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440,

380, 126, 721, 328, 753, 470, 743, 527

]

count\_multiple\_occurrence(numbers)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

The occurrence of 219 is 2

The occurrence of 236 is 2

The occurrence of 328 is 2

The occurrence of 566 is 2

The occurrence of 615 is 2

The occurrence of 753 is 2

The occurrence of 984 is 2

Hints:

The dictionary used for counting should be changed to tuple to get a sorted version of a dictionary

Answer:

# your code goes here

def count\_multiple\_occurrence(numbers):

counts = dict()

#counting

for i in numbers:

counts[i] = counts.get(i, 0) + 1

#display

for number, count in sorted(list(counts.items())):

if count > 1:

print("The occurrence of",number,"is",count)

# Output Generation. You are not allowed to modify the following codes

def main():

numbers = [

951, 402, 984, 651, 360, 69, 408, 319, 601, 485, 980, 507, 725,

547, 544, 615, 83, 165, 141, 501, 263, 617, 865, 575, 219, 390,

984, 592, 236, 105, 942, 941, 386, 462, 47, 418, 907, 344, 236,

375, 823, 566, 597, 978, 328, 615, 953, 345, 399, 162, 758, 219,

918, 237, 412, 566, 826, 248, 866, 950, 626, 949, 687, 217, 815,

67, 104, 58, 512, 24, 892, 894, 767, 553, 81, 379, 843, 831, 445,

742, 717, 958, 609, 842, 451, 688, 753, 854, 685, 93, 857, 440,

380, 126, 721, 328, 753, 470, 743, 527

]

count\_multiple\_occurrence(numbers)

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. Write a function to loop through and count the occurrence of all words from the string. Display all numbers with descending order on the number or counts (value)

def main():

line = "the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car"

# your code goes here

# You are not allowed to modify the following codes

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

There are 24 words with 11 in difference

Word: "the" --- Counts: 7

Word: "car" --- Counts: 3

Word: "and" --- Counts: 3

Word: "tent" --- Counts: 2

Word: "ran" --- Counts: 2

Word: "clown" --- Counts: 2

Word: "on" --- Counts: 1

Word: "into" --- Counts: 1

Word: "fell" --- Counts: 1

Word: "down" --- Counts: 1

Word: "after" --- Counts: 1

Hints:

1. The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently.
2. You should perform display after all counting is completed.

Answer:

def main():

line = "the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car"

# your code goes here

counts = dict()

words = line.split()

for word in words:

counts[word] = counts.get(word,0) + 1

print ("There are " + str(len(words)) + " words with "+str(len(counts))+" in difference")

list\_counts = list()

for word, count in counts.items():

list\_counts.append((count, word))

list\_counts.sort(reverse=True)

for i in list\_counts:

print ("Word: \""+i[1]+"\" --- Counts: "+str(i[0]))

# You are not allowed to modify the following codes

if \_\_name\_\_ == "\_\_main\_\_":

main()

Answer2:

def main():

line = "the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car"

# your code goes here

counts = dict()

words = line.split()

for word in words:

counts[word] = counts.get(word,0) + 1

print ("There are " + str(len(words)) + " words with "+str(len(counts))+" in difference")

for word, count in sorted(counts.items(), key=lambda item: item[1], reverse=True):

print ("Word: \""+word+"\" --- Counts: "+str(count))

# You are not allowed to modify the following codes

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. Write a product selling system to help a mini store to record all selling of store’s product. The system always accepts user’s inputs of product name, product price and product quantity. Only the product name (formatted in small letters no matter on user’s input), quantity and total would be recorded in dictionary. After that, the statistics of all products would be shown.

INDEX\_QUANTITY = 0

INDEX\_TOTAL\_SALES = 1

def main():

products = dict()

print ("Welcome to product selling system")

while True:

name = input("Input product name: ").lower()

price = float(input("Input selling price for current order: "))

quantity = int(input("Input quantity for current order: "))

# your code goes here

# You are not allowed to modify the following codes

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

Welcome to product selling system

Input product name: **Apple**

Input selling price for current order: **3**

Input quantity for current order: **4**

Statistics of product selling

Item: apple - Quantity sold: 4 - Total sales amount: 12.0

--------------------------------------------------------------------------------

Input product name: **Orange**

Input selling price for current order: **2**

Input quantity for current order: **5**

Statistics of product selling

Item: apple - Quantity sold: 4 - Total sales amount: 12.0

Item: orange - Quantity sold: 5 - Total sales amount: 10.0

--------------------------------------------------------------------------------

Input product name: **Cherry**

Input selling price for current order: **1**

Input quantity for current order: **20**

Statistics of product selling

Item: apple - Quantity sold: 4 - Total sales amount: 12.0

Item: orange - Quantity sold: 5 - Total sales amount: 10.0

Item: cherry - Quantity sold: 20 - Total sales amount: 20.0

--------------------------------------------------------------------------------

Input product name: **apple**

Input selling price for current order: **4**

Input quantity for current order: **2**

Statistics of product selling

Item: apple - Quantity sold: 6 - Total sales amount: 20.0

Item: orange - Quantity sold: 5 - Total sales amount: 10.0

Item: cherry - Quantity sold: 20 - Total sales amount: 20.0

--------------------------------------------------------------------------------

Input product name:

Hints:

1. The value of dictionary should contain both total quantity and total sales amount. You can use either List or Tuple to implement sales records.
2. The values of quantity and total sales should be retrieved by index constants instead of numbers (i.e. 0, 1) in your codes. i.e.

quantity = products [ name ] [ INDEX\_QUANTITY ]

total\_sales = products [ name ] [ INDEX\_TOTAL\_SALES ]

given that “name” is the key value for dictionary, such as apple, orange or cherry in the above cases.

1. You should perform display after all counting is completed.

Answer:

INDEX\_QUANTITY = 0

INDEX\_TOTAL\_SALES = 1

def main():

products = dict()

print ("Welcome to product selling system")

while True:

name = input("Input product name: ").lower()

price = float(input("Input selling price for current order: "))

quantity = int(input("Input quantity for current order: "))

# your code goes here

if name not in products:

products[name] = [quantity, price \* quantity]

else:

products[name][INDEX\_QUANTITY] += quantity

products[name][INDEX\_TOTAL\_SALES] += price \* quantity

print("Statistics of product selling")

for i in products:

print ("Item: " + i + " - Quantity sold: " + str(products[i][INDEX\_QUANTITY]) + " - Total sales amount: " + str(products[i][INDEX\_TOTAL\_SALES]))

print ("-" \* 80)

# You are not allowed to modify the following codes

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. Write a program to compute the sales for a coffee shop. This coffee shop sells **four types of coffee** (and maybe more later) with different prices. The corresponding coffee name and prices is represented by the following tuple.

tuple **COFFEE\_NAME\_AND\_PRICES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | |  |  | | --- | --- | | 0 | “Cappuccino” | | 1 | 25 | |
| 1 | |  |  | | --- | --- | | 0 | “Espresso” | | 1 | 20 | |
| 2 | |  |  | | --- | --- | | 0 | “Latte” | | 1 | 15 | |
| 3 | |  |  | | --- | --- | | 0 | “Mocha” | | 1 | 30 | |

1. Complete the function **display\_coffee\_shop\_menu()** and obtain the user’s selected coffee index as input. Return the user’s input to the function caller.

After a customer indicate the type of coffee s/he wants, s/he may pay extra money for options such as “large cup” and/or “cold drink” when buying coffee. The user is allowed to enter the quantity and choose the preferred options by answering Y or N. Each **sales record** of above information is stored using a **tuple**.

The customer may buy several types of coffees for a particular purchase. Hence the purchase may contain a list of sales records. The purchase list is represented by a list **currnet\_sales\_list** which contains many sales tuples as representation by the following diagram.

list **current\_sales\_list**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | sales tuple 1   |  |  | | --- | --- | | 0 | Coffee no (0-4) | | 1 | Quantity (Integer) | | 2 | Large cup (Y/N) | | 3 | Cold (Y/N) | |
| 1 | sales tuple 2   |  |  | | --- | --- | | 0 | Coffee no (0-4) | | 1 | Quantity (Integer) | | 2 | Large cup (Y/N) | | 3 | Cold (Y/N) | |
| : | : |
| n | sales tuple n   |  |  | | --- | --- | | 0 | Coffee no (0-4) | | 1 | Quantity (Integer) | | 2 | Large cup (Y/N) | | 3 | Cold (Y/N) | |

1. Write a function **compute\_sales()** which accepts four parameters (integer coffee no, integer quantity, string large cup Y/N, string cold Y/N). This function does the following:
   * get the coffee type (by coffee no.), quantity, and any extra options of this coffee.
   * compute the amount of this coffee.
   * the coffee price can be obtained by the following statement:

coffee\_price = COFFEE\_NAME\_AND\_PRICES[coffee\_no][INDEX\_COFFEE\_PRICE]

where coffee\_no should be an integer between 0 – 3, which is inputted by user.

1. Print the strings in the current order summary as given in the sample output below and calculate the price of each coffee and total price of all coffees by using the **compute\_sales()** function.

A dictionary **total\_cups\_of\_coffee\_sold** is used to accumulate the quantity sold for each type of coffee. The coffee name is used as the key for this dictionary rather than using coffees’ ID and is represented by the following diagram.

dictionary **cups\_of\_coffee\_sold**

|  |  |
| --- | --- |
| “Cappuccino” | 2 |
| “Espresso” | 3 |
| “Latte” | 1 |
| “Mocha” | 4 |

1. Complete the list of number of coffees sold for each type of coffee in the statistic part after customer confirms the order by pressing “Enter” key.

Some of the code have been done for you:

import sys

COFFEE\_NAME\_AND\_PRICES = (("Cappuccino",25),("Espresso",20),("Latte ",15),("Mocha ",30))

# Index represented in tuple COFFEE\_NAME\_AND\_PRICES

INDEX\_COFFEE\_NAME = 0

INDEX\_COFFEE\_PRICE = 1

# Price of extra option

LARGE\_CUP\_PRICE = 5

COLD\_PRICE = 3

# Index represented in sales tuple. The current\_sales\_list should contains sales tuples of all unconfirmed order items

INDEX\_COFFEE\_NO = 0

INDEX\_QUANTITY = 1

INDEX\_LARGE\_CUP = 2

INDEX\_COLD = 3

# (i) Complete this function

def display\_coffee\_shop\_menu():

return 0

# (ii) Complete this function

def compute\_sales(coffee\_no, quantity, large\_cup, cold):

return 0

def main():

current\_sales\_list = list()

cups\_of\_coffee\_sold = dict()

total\_number\_sales = 0

highest\_sales\_amount = 0

lowest\_sales\_amount = sys.maxsize

total\_sales\_amount = 0

print("Welcome to Coffee Shop System.")

while True:

input\_coffee = display\_coffee\_shop\_menu()

#confirm order

if (input\_coffee == ""):

if len(current\_sales\_list) == 0:

print("Current Sales Order is empty")

continue

else:

# (iv) Complete this part

total = 0

for i in current\_sales\_list:

price = compute\_sales(i[INDEX\_COFFEE\_NO], i[INDEX\_QUANTITY], i[INDEX\_LARGE\_CUP], i[INDEX\_COLD])

total += price

# compute total sales

total\_number\_sales+=1

total\_sales\_amount+=total

if total > highest\_sales\_amount:

highest\_sales\_amount = total

if total < lowest\_sales\_amount:

lowest\_sales\_amount = total

print("\nStatistics of Coffee Shop:")

print ("Total number sales = "+str(total\_number\_sales))

print ("Lowest Sales Amount = $"+str(lowest\_sales\_amount))

print ("Highest Sales Amount = $"+str(highest\_sales\_amount))

print ("Total Sales Amount = $"+str(total\_sales\_amount))

print ("Average Sales Amount = $"+str(total\_sales\_amount / total\_number\_sales))

current\_sales\_list.clear()

else:

# for user to input coffee type, quantity, large or not, cold or not

input\_coffee = int(input\_coffee)

input\_quantity = int(input("Please input quantity:"))

input\_large\_cup = input("Large Cup required? +$5.00 (Y / N):").upper()

input\_cold = input("Cold required? +$3.00 (Y / N):").upper()

current\_sales\_record = (input\_coffee, input\_quantity, input\_large\_cup, input\_cold)

current\_sales\_list.append(current\_sales\_record)

print("\nCurrent Order Summary:")

current\_sales\_total\_string = "Total: $"

total = 0

# i is tuple in the format (0, 3, 'Y', 'Y')

# compute\_sales for each coffee

# (iii) Complete this part

if \_\_name\_\_ == "\_\_main\_\_":

main()

A sample output is shown below:

===================== RESTART: C:/Python38/lab08-q4.py =======================

Welcome to Coffee Shop System.

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):0

Please input quantity:2

User’s input

Large Cup required? +$5.00 (Y / N):n

Cold required? +$3.00 (Y / N):N

Small letter or capital letter of y/n should also be acceptable

Current Order Summary:

Cappuccino 2 cups: $50

Total: $50

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):

Statistics of Coffee Shop:

User pressed Enter to confirm order here

Total number sales = 1

Lowest Sales Amount = $50

Highest Sales Amount = $50

Total Sales Amount = $50

Average Sales Amount = $50.0

List of number of cups coffee sold:

Cappuccino: 2

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):1

Please input quantity:3

Large Cup required? +$5.00 (Y / N):y

Cold required? +$3.00 (Y / N):n

Current Order Summary:

Espresso 3 cups with option LARGE CUP: $75

Total: $75

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):2

Please input quantity:1

Large Cup required? +$5.00 (Y / N):n

Cold required? +$3.00 (Y / N):y

Current Order Summary:

Espresso 3 cups with option LARGE CUP: $75

Latte 1 cups with option COLD: $18

Total: $93

Two items are inputted by user

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):

Statistics of Coffee Shop:

User pressed Enter to confirm order here

Total number sales = 2

Lowest Sales Amount = $50

Total number of sales should be 2 instead of 3 in this case

Highest Sales Amount = $93

Total Sales Amount = $143

Average Sales Amount = $71.5

List of number of cups coffee sold:

Cappuccino: 2

Espresso: 3

Latte: 1

Coffee Shop Menu:

No. | Coffee Type | Price

0 | Cappuccino | $25

1 | Espresso | $20

2 | Latte | $15

3 | Mocha | $30

Please input your choice. Press "Enter" to confirm order (0 – 3):

Answers:

import sys

COFFEE\_NAME\_AND\_PRICES = (("Cappuccino",25),("Espresso",20),("Latte ",15),("Mocha ",30))

# Index represented in tuple COFFEE\_NAME\_AND\_PRICES

INDEX\_COFFEE\_NAME = 0

INDEX\_COFFEE\_PRICE = 1

# Price of extra option

LARGE\_CUP\_PRICE = 5

COLD\_PRICE = 3

# Index represented in sales tuple. The current\_sales\_list should contains sales tuples of all unconfirmed order items

INDEX\_COFFEE\_NO = 0

INDEX\_QUANTITY = 1

INDEX\_LARGE\_CUP = 2

INDEX\_COLD = 3

def display\_coffee\_shop\_menu():

print("\nCoffee Shop Menu:")

print("No.\t| Coffee Type\t| Price")

index = 0

for i in COFFEE\_NAME\_AND\_PRICES:

print(index,"\t| "+i[INDEX\_COFFEE\_NAME]+"\t| $"+str(i[INDEX\_COFFEE\_PRICE]))

index+=1

input\_coffee = input("Please input your choice. Press \"Enter\" to confirm order (0 – "+str(len(COFFEE\_NAME\_AND\_PRICES)-1)+"):")

return input\_coffee

def compute\_sales(coffee\_no, quantity, large\_cup, cold):

sales = 0

#options

if large\_cup == "Y":

sales += LARGE\_CUP\_PRICE

if cold == "Y":

sales += COLD\_PRICE

sales += COFFEE\_NAME\_AND\_PRICES[coffee\_no][INDEX\_COFFEE\_PRICE]

sales \*= quantity

return sales

def main():

current\_sales\_list = list()

cups\_of\_coffee\_sold = dict()

total\_number\_sales = 0

highest\_sales\_amount = 0

lowest\_sales\_amount = sys.maxsize

total\_sales\_amount = 0

print("Welcome to Coffee Shop System.")

while True:

input\_coffee = display\_coffee\_shop\_menu()

#confirm order

if (input\_coffee == ""):

if len(current\_sales\_list) == 0:

print("Current Sales Order is empty")

continue

else:

total = 0

for i in current\_sales\_list:

price = compute\_sales(i[INDEX\_COFFEE\_NO], i[INDEX\_QUANTITY], i[INDEX\_LARGE\_CUP], i[INDEX\_COLD])

total += price

cups\_of\_coffee\_sold[COFFEE\_NAME\_AND\_PRICES[i[INDEX\_COFFEE\_NO]][INDEX\_COFFEE\_NAME]] = cups\_of\_coffee\_sold.get(COFFEE\_NAME\_AND\_PRICES[i[INDEX\_COFFEE\_NO]][INDEX\_COFFEE\_NAME],0) + i[INDEX\_QUANTITY]

# compute total sales

total\_number\_sales+=1

total\_sales\_amount+=total

if total > highest\_sales\_amount:

highest\_sales\_amount = total

if total < lowest\_sales\_amount:

lowest\_sales\_amount = total

print("\nStatistics of Coffee Shop:")

print ("Total number sales = "+str(total\_number\_sales))

print ("Lowest Sales Amount = $"+str(lowest\_sales\_amount))

print ("Highest Sales Amount = $"+str(highest\_sales\_amount))

print ("Total Sales Amount = $"+str(total\_sales\_amount))

print ("Average Sales Amount = $"+str(total\_sales\_amount / total\_number\_sales))

print ("List of number of cups coffee sold:")

for key in cups\_of\_coffee\_sold:

print("\t" + key + ": " + str( cups\_of\_coffee\_sold [ key ] ))

current\_sales\_list.clear()

else:

# for user to input coffee type, quantity, large or not, cold or not

input\_coffee = int(input\_coffee)

input\_quantity = int(input("Please input quantity:"))

input\_large\_cup = input("Large Cup required? +$5.00 (Y / N):").upper()

input\_cold = input("Cold required? +$3.00 (Y / N):").upper()

current\_sales\_record = (input\_coffee, input\_quantity, input\_large\_cup, input\_cold)

current\_sales\_list.append(current\_sales\_record)

print("\nCurrent Order Summary:")

current\_sales\_total\_string = "Total: $"

total = 0

# i is tuple in the format (0, 3, 'Y', 'Y')

# compute\_sales for each coffee

for i in current\_sales\_list:

sales\_record\_string = COFFEE\_NAME\_AND\_PRICES[i[INDEX\_COFFEE\_NO]][INDEX\_COFFEE\_NAME].strip() + " "

sales\_record\_string += str(i[INDEX\_QUANTITY]) + " cups"

#options

if i[INDEX\_LARGE\_CUP] == "Y" or i[INDEX\_COLD] == "Y":

sales\_record\_string += " with option"

if i[INDEX\_LARGE\_CUP] == "Y":

sales\_record\_string += " LARGE CUP"

if i[INDEX\_LARGE\_CUP] == "Y" and i[INDEX\_COLD] == "Y":

sales\_record\_string += " and"

if i[INDEX\_COLD] == "Y":

sales\_record\_string += " COLD"

price = compute\_sales(i[INDEX\_COFFEE\_NO], i[INDEX\_QUANTITY], i[INDEX\_LARGE\_CUP], i[INDEX\_COLD])

total += price

sales\_record\_string += ": $" + str(price)

print (sales\_record\_string)

print (current\_sales\_total\_string + str(total))

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. Write an inventory management system to help an office to keep records on borrowing items staff. The system always accepts user’s inputs of item number, number of quantity the staff would like to borrow, and the staff / borrower’s name. The borrower’s name and item number will form a tuple together and act as a key in dictionary, while all quantity of items borrowed by that borrower will act as the value to this key. After the borrowing procedure finished, a table showing all borrower’s borrowing record would be shown.

# Items allowed for borrowing in this system.

ITEMS = ("Apple MacBook Pro",

"ALFA Network AC1200 Wireless Adapter",

"NVIDIA Jetson Nano Developer Kit",

"Linksys WRT1900AC Dual-Band Wi-Fi Router",

"DJI RoboMaster EP")

# Index represented in user item tuple inside key part in   
# user\_borrow\_record dictionary

INDEX\_USER\_BORROW\_RECORD = 0

INDEX\_ITEM\_BORROW\_RECORD = 1

def main():

user\_borrow\_record = dict()

while True:

# Your Implementation starts here

###################################################

# Display Item(s) menu for user selection

# ask input from user

# create the key used in dictionary

# assign corresponding value for corresponding key

# display all records

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

Item(s) in stock:

Item No. | Item Name

0. | Apple MacBook Pro

1. | ALFA Network AC1200 Wireless Adapter

2. | NVIDIA Jetson Nano Developer Kit

3. | Linksys WRT1900AC Dual-Band Wi-Fi Router

4. | DJI RoboMaster EP

Please input the item no. to borrow (0 – 4): **0**

Please input the quantity to borrow: **2**

Please input borrower's name: **Kelvin Yip**

Item(s) Borrowers had borrowed:

Item No. | Item Name | Borrower | Qty. Borrowed

0. | Apple MacBook Pro | Kelvin Yip | 2

Item(s) in stock:

Item No. | Item Name

0. | Apple MacBook Pro

1. | ALFA Network AC1200 Wireless Adapter

2. | NVIDIA Jetson Nano Developer Kit

3. | Linksys WRT1900AC Dual-Band Wi-Fi Router

4. | DJI RoboMaster EP

Please input the item no. to borrow (0 – 4): **4**

Please input the quantity to borrow: **3**

Please input borrower's name: **Cow Leung**

Item(s) Borrowers had borrowed:

Item No. | Item Name | Borrower | Qty. Borrowed

0. | Apple MacBook Pro | Kelvin Yip | 2

4. | DJI RoboMaster EP | Cow Leung | 3

Item(s) in stock:

Item No. | Item Name

0. | Apple MacBook Pro

1. | ALFA Network AC1200 Wireless Adapter

2. | NVIDIA Jetson Nano Developer Kit

3. | Linksys WRT1900AC Dual-Band Wi-Fi Router

4. | DJI RoboMaster EP

Please input the item no. to borrow (0 – 4): **0**

Please input the quantity to borrow: **4**

Please input borrower's name: **Kelvin Yip**

Item(s) Borrowers had borrowed:

Item No. | Item Name | Borrower | Qty. Borrowed

0. | Apple MacBook Pro | Kelvin Yip | 6

4. | DJI RoboMaster EP | Cow Leung | 3

Item(s) in stock:

Item No. | Item Name

0. | Apple MacBook Pro

1. | ALFA Network AC1200 Wireless Adapter

2. | NVIDIA Jetson Nano Developer Kit

3. | Linksys WRT1900AC Dual-Band Wi-Fi Router

4. | DJI RoboMaster EP

Please input the item no. to borrow (0 – 4):

Hints:

1. The key of dictionary should be a tuple containing both the borrower’s name and item number the borrower borrowed. i.e. key = (input\_borrower\_name, input\_item\_no)
2. The value of dictionary should contain quantity of the items user borrowed plus the item this user had borrowed previously. i.e.

user\_borrow\_record[key] = user\_borrow\_record.get(key, 0) + input\_quantity

Answers:

# Items allowed for borrowing in this system.

ITEMS = ("Apple MacBook Pro",

"ALFA Network AC1200 Wireless Adapter",

"NVIDIA Jetson Nano Developer Kit",

"Linksys WRT1900AC Dual-Band Wi-Fi Router",

"DJI RoboMaster EP")

# Index represented in user item tuple inside key part in

# user\_borrow\_record dictionary

INDEX\_USER\_BORROW\_RECORD = 0

INDEX\_ITEM\_BORROW\_RECORD = 1

def main():

user\_borrow\_record = dict()

while True:

# Your Implementation starts here

###################################################

# Display Item(s) menu for user selection

print("\nItem(s) in stock:")

print(f"Item No. | Item Name")

index = 0

for item in ITEMS:

print (f"{index:>7}. | "

f"{item}")

index += 1

# ask input from user

input\_item\_no = int(input("Please input the item no. to borrow (0 – "+str(len(ITEMS)-1)+"): "))

input\_quantity = int(input("Please input the quantity to borrow: "))

input\_borrower\_name = input("Please input borrower's name: ")

# create the key used in dictionary

key = (input\_borrower\_name, input\_item\_no)

# assign corresponding value for corresponding key

user\_borrow\_record[key] = user\_borrow\_record.get(key, 0) + input\_quantity

# display all records

print ("\nItem(s) Borrowers had borrowed:")

print (f"Item No. | {'Item Name':<43}| {'Borrower':<11}| Qty. Borrowed")

for key in user\_borrow\_record:

item\_index = key[INDEX\_ITEM\_BORROW\_RECORD]

item\_name = ITEMS[item\_index]

borrower\_name = key[INDEX\_USER\_BORROW\_RECORD]

print (f"{item\_index:>7}. | "

f"{item\_name:<43}| "

f"{borrower\_name:<11}| "

f"{user\_borrow\_record[key]:>13}")

if \_\_name\_\_ == "\_\_main\_\_":

main()