#Case Study 1: Managing a User Data List

# Scenario: You are managing a list of user data, which contains numerical and textual

# information. The list allows for slicing, modification,

# merging with other data, and performing product calculations on numeric values.

# Questions:

# 1. Extracting a Subset: How would you slice the list to get the first three elements of the user's data?

# for slicing you need to give index as [start:End:Step]

# here start: is from which index you want to start the slicing , by default index is starts from 0

# end: till which index you want to slice ,it will exclude the last element

#step: it represents the diffrence between the next index to move by defaults it 1

# l=[10,20,30,56,78]

# print(l[:3])

# output:- [10, 20, 30]

# 2. Modifying Data: How can you update the second element of a user's data list, for example changing the age or score?

# by default index of list is starts from 0 hence 2nd elements is denoted at index 1

# then we will retrive the elemnts at index 1 in list

# age=[10,20,30,40]

# print(age[1])

# output:-20

# 3. Merging Data: How can you merge two lists, one containing user data and the other containing additional information like preferences?

```
# 1)with help of + operator you can merge

# 2)with help of extend()method also you can merge the two lists , it will change the original list

# l=["lavanya","neha","Sarika","Nandita"]

# p=["data science",'Data analytics','Statistics']

# print(l+p)

# l.extend(p)

# print(l)
```

# output:-['lavanya', 'neha', 'Sarika', 'Nandita', 'data science', 'Data analytics', 'Statistics']

#   ['lavanya', 'neha', 'Sarika', 'Nandita', 'data science', 'Data analytics', 'Statistics']

# 4. Calculating the Product: How would you compute the product of numerical values in the user data list, such as multiplying all the user's scores or amounts?

```
# import math

# l=[10,20,30,40]

# print(math.prod(l))
```

# output:-240000

# 5. Adding Dynamic Data: How would you prompt the user to input their age or score and add that to an existing list of user data?

```
# you can get dynamic data from user's with help of input() method

# by default input method type is string you can typeconversion it as you need

# with help of append method you can add new element in list at end of list

# l=[10,20,30]

# l.append(int(input("Enter score")))

# print(l)
```

Lavanya

# output:-Enter score67

# [10, 20, 30, 67]

# 6. Displaying Data: How can you use an f-string to display a formatted version of the list of user data with dynamic values?

```
# l=[10,20,30,40]

# print(f'{l[0]} is lowest score the list\n{l[-1]} is highest score in list')
```

# output:-

# 10 is lowest score the list

# 40 is highest score in list

Case Study 2: Employee Records with Tuples

Scenario: You have an employee records system that stores immutable employee data using tuples.

This system allows for slicing data, updating records by creating new tuples, merging employee records, and

calculating specific metrics (e.g., salary product).

Questions:

1. Extracting Employee Information: How would you slice the tuple to get the name and role of an employee?

for slicing you need to give index as [start:End:Step]

here start: is from which index you want to start the slicing , by default index is starts from 0

end: till which index you want to slice ,it will exclude the last element

step: it represents the diffrence between the next index to move by defaults it 1

```
t=("lavanya","data scienctist","yash","IIT Teacher")

print(t[:2])
```

output:-('lavanya', 'data scienctist')

2. Updating Employee Records: Given that tuples are immutable, how would you "update" the role of an employee by creating a new tuple?

as tuple is immutable then you can't update it

if you want to update the you have to convert the tuple into list

then update the value and again convert the list to tuple

```
t_role=('DS',"DA","pythone","Sql Devloper")

print(t_role)

t_role=list(t_role)

t_role[2]="python Devloper"
```

```
t_role=tuple(t_role)
print(t_role)
```

output:-

```
('DS', 'DA', 'pythone', 'Sql Devloper')
('DS', 'DA', 'python Devloper', 'Sql Devloper')
```

3. Combining Employee Data: How would you merge two tuples containing employee details into a single tuple?

by using + operator you can merge two tuples

```
e1=(1,"Mira","Python devloper")
e2=(2,"lava","Data scientist")
e3=e1+e2
print(e3)
```

output:-

```
(1, 'Mira', 'Python devloper', 2, 'lava', 'Data scientist')
```

4. Calculating Salaries: How would you calculate the product of salaries from two tuples containing employee salary data?

```
import math
sal=(100,200,300,400,900)
print(math.prod(sal))
```

output:-2160000000000

5. User Interaction for Record Update: How would you ask the user for input (e.g., new job title) and then update the tuple with that information, while leaving other details unchanged?

```
new_job_title=input("enter you new role")

t=list(("DA","DS"))

t.append(new_job_title)

t=tuple(t)

print(t)
```

output:-

```
enter you new role python specialist

('DA', 'DS', ' python specialist')
```

6. Displaying Employee Details: How can you display the employee tuple in a readable format using an f-string?

```
t=(10,"lavanya","Data scientist")

print(f'id: {t[0]} name: {t[1]} who appointed in x company as role {t[2]}')
```

output:-

```
id: 10 name: lavanya who appointed in x company as role Data scientist
```

# Case Study 3: Managing a Fruit Inventory with Sets

# Scenario: You are working with a set-based inventory system where each item is unique.

# You need to handle adding, removing, and merging sets,

# as well as performing set operations like intersection (finding common elements), and user input.

# Questions:

# 1. Adding Inventory Items: How would you add a new fruit to the set of inventory items?

# if you want to add only one element with help of add() you can add new element is set

```
# fruit={"Mango","Apple"}

# fruit.add("Cherry")

# print(fruit)
```

# output:-

# {'Cherry', 'Apple', 'Mango'}

# 2. Removing Items from Inventory: How can you remove a specific fruit from the set if it’s no longer available?

```
# with help of discard or remove you can remov ethe specified item from set

# only difference is that discard will never through error if item is not found which is remove does
```

```
# fruit={"Mango","Apple"}

# fruit.discard("Mango")

# print(fruit)
```

# output:-{'Apple'}

# 3. Combining Inventory: How would you merge two sets of fruits to create a combined inventory with unique fruits from both sets?

#note: you can't merge set by + oprator

#with help of update you can merge the sets , update will change the original set

#with update you can give any iterables like list tuple etc

```
# fruit={"Mango","Apple"}

# fruits2={"Tamato","Apple","Carrot"}

# # fruit3=fruit+fruits2 # this will not work gives you Type Error

# fruit.update(fruits2)

# print(fruit)
```

# output:-

# {'Carrot', 'Tamato', 'Apple', 'Mango'}

# 4. Finding Common Items: How would you perform a "product" operation by finding the common fruits between two sets

# (e.g., inventory and shopping list)?

```
# inventory={"Tamato","Apple","Carrot"}

# shopping_list={"Mango","Apple"}

# print(inventory.intersection(shopping_list))
```

Output:-{“Apple”}

# 5. Simulating Slicing: Since sets are unordered, how can you convert the set to a list and slice it to get the first two fruits?

# you can convert set into list help of list function

```
# s=list({"Tamato","Apple","Carrot"})

# print(s[:2])
```

# output:- ['Carrot', 'Apple']

Lavanya

```
# 6. Updating the Inventory: How would you prompt the user for input to add a new fruit to
the inventory set,

# and display the updated set using an f-string?

# s=set()

# f=input("enter new fruit ")

# s.add(f)

# print(s)

# output:-

# enter new fruit mango

# {'mango'}
```

# Case Study: Online Store Inventory Management System

# Scenario:

# You are tasked with developing an inventory management system for an online store.

# The store manages its product inventory using Python data structures.

# The product information includes categories, stock quantities, and prices.

# The system needs to handle operations like updating stock, calculating total sales, merging product lists, and managing user interactions.

# In this scenario, you will work with lists, tuples, and sets to manage different aspects of the inventory system.

# Questions:

# 1. Managing Product Categories (List Slicing and Merging):

#   The store keeps a list of product categories.

#   You need to slice the list to display the first three categories and

#   merge the available categories with new categories coming from a supplier.

#   How would you slice the list to retrieve the first three product categories?

#   How would you merge the current list of product categories with a new list of categories provided by a supplier?

```
# categories=["Kitechen","Electronis","Assesrings","Clothing"]

# supplier=["Jwellary","Homedecores"]

# print(categories[:3]+supplier)
```

# output:-['Kitechen', 'Electronis', 'Assesrings', 'Jwellary', 'Homedecores']

Lavanya

# 2. Updating Product Information (List Mutation):

#   The store needs to update the stock quantity of a particular product.

#   The list contains product dictionaries, where each dictionary has keys like 'name', 'category', and 'stock'.

#   How would you find the dictionary for a product by its name and update its stock?

#   How would you adjust the stock for a product when a new shipment arrives?

```
# shipment_arrives=100
# d={'Mixer':{
#            'category':'Kitechen',
#            'stock':50}
#    }
# d['Mixer']['stock'] += shipment_arrives
# print(d['Mixer'])
```

# output:-{'category': 'Kitechen', 'stock': 150}

# 3. Calculating Total Sales (Product Calculation with List):

#   Each product in the store has a price and a stock quantity.

#   You are asked to calculate the total sales value by multiplying the price of each product by its stock quantity

#   and summing these values for all products.

#   How would you calculate the total sales value from a list of product dictionaries, each containing 'price' and 'stock' keys?

```
# d={
#     'Mixer':{
```

```
#           'price':1000,
#           'stock':50,
#           },
#    'T-shirt':{
#           'price':500,
#           'stock':150},
#    'Laptop':{
#           'price':25000,
#           'stock':10},
#    'AC':{
#           'price':2000,
#           'stock':30}
# }
# l=list(d.keys())
# print(l)
# Totals_Salaes=[]
# Totals_Salaes.extend([d[l[0]]['price'] * d[l[0]]['stock'],d[l[1]]['price'] * d[l[1]]['stock'],
#                    d[l[2]]['price'] * d[l[2]]['stock'],d[l[3]]['price'] * d[l[3]]['stock']])
# print("Total_Sales: ",sum(Totals_Salaes))
```

# output:-Total_Sales: 435000

# 4. Handling Product Identifiers (Tuple Manipulation and Merging):

#   The product identifiers (product IDs) are stored as tuples.

#   You are given a tuple of IDs for products that are currently in stock,

#   and a tuple for products that are out of stock.

#   How would you merge these two tuples into one list of product IDs, ensuring no duplicates?

#   How would you retrieve the first five product IDs from the list of available product IDs?

```
# c_PID=tuple(range(10,16))

# O_PID=tuple(range(1,4))

# P_IDS=list(c_PID)+list(O_PID)

# P_IDS=list(set(P_IDS))

# print(P_IDS[:6])
```

# output:-[1, 2, 3, 10, 11, 12]

# 5. Removing Out-of-Stock Products (Set Mutation):

#   The store wants to track which products are out of stock using a set.

#   When a product is out of stock, it is added to a set. If a product is restocked,

# it is removed from the out-of-stock set and added to the in-stock set.

#   How would you remove a product from the out-of-stock set and add it to the in-stock set when the product is restocked?

#   How would you add a product to the out-of-stock set if the stock runs out?

```
# in_stock={'Mixers','cloths','laptops','AC'}

# o_stock={'Tv','Jwellary'}

# print("********************Items in Stock********************")

# print("In_stock: ",in_stock)

# print("Out_stock: ",o_stock)

# item_runs_out='AC'
# in_stock.discard(item_runs_out)
# o_stock.add(item_runs_out)
# print("***************Items in Stock after stock runs out************")
# print("In_stock: ",in_stock)
# print("Out_stock: ",o_stock)
# item_restock='Tv'
# o_stock.discard(item_restock)
# in_stock.add(item_restock)
# print("****************Items after Restoked*******************")
# print("In_stock: ",in_stock)
# print("Out_stock: ",o_stock)
```

# output:-

```
# ********************Items in Stock*******************
# In_stock:  {'cloths', 'laptops', 'Mixers', 'AC'}
# Out_stock:  {'Jwellary', 'Tv'}
# ***************Items in Stock after stock runs out************
# In_stock:  {'cloths', 'laptops', 'Mixers'}
# Out_stock:  {'Jwellary', 'Tv', 'AC'}
# ****************Items after Restoked*******************
# In_stock:  {'laptops', 'Mixers', 'cloths', 'Tv'}
# Out_stock:  {'Jwellary', 'AC'}
```

# 6. Performing Set Operations on Inventory (Set Operations and User Input):

#   The store tracks two sets of products: one set of "promotional" products

# and another set of "on-sale" products. You need to perform the following operations:

#   - Find the products that are both promotional and on sale (intersection).

#   - Combine all products that are either promotional or on sale (union).

#   - Remove the promotional products from the on-sale set (difference).

#   How would you perform these operations on the sets and display the results using f-strings?

```
# promotional={"Mixer",'cloths', 'laptops', 'AC'}

# on_sale={'Jwellary', 'AC','TV','Mixer'}

# print(f'products that are both promotional and on sale: {promotional.intersection(on_sale)}')

# print(f'all products that are either promotional or on sale: {promotional.union(on_sale)}')

# print(f'Products are only on sale set on in promotional: {on_sale.difference(promotional)}')
```

# output:-

# products that are both promotional and on sale: {'AC', 'Mixer'}

# all products that are either promotional or on sale: {'TV', 'AC', 'Jwellary', 'Mixer', 'cloths', 'laptops'}

# Products are only on sale set on in promotional: {'Jwellary', 'TV'}

# 7. Dynamic Product Search (User Input and f-string):

#   The store wants to allow customers to search for products based on category.

# A customer inputs a category name, and the program returns all products in that category.

# The list contains product dictionaries with keys 'name' and 'category'.

#   How would you prompt the user for a category, and display all products from that category in a formatted list using an f-string?

```
# product=    {

#        'Kitechn':['Mixer','Owen','Stow','Kitech_set'],

#        'Cloths':['T-shirts','Shots','Kurtis','Sarees'],

#        'Electronis':['phones','Laptops','Smart-Waches','Tvs'],
#        'HomeDecore':['Doormate','Wallhanging','WallStickers','Designers-curtuns']
#    }
# category=input("Enter Category: ")
# print(product[category])
```

# output:-

# Enter Category: HomeDecore

# ['Doormate', 'Wallhanging', 'WallStickers', 'Designers-curtuns']