%% [markdown]
#**Day-3 Python Challenge **
LIST PYTHON
A list in Python is a built-in ordered and mutable collection of items, which can hold elements of different data types such as integers, strings, floats,
or even other lists. Lists are defined using square brackets [], and the items are separated by commas.
#
Because lists are mutable, you can add, remove, or change items after the list has been created. Lists also allow duplicate values, and each item in a list
can be accessed using its index, starting from 0.

%% [markdown]
<!-- Key Characteristics:
Ordered: Items have a fixed order and that order will not change unless you change it manually.
#
Mutable: You can modify a list after creating it.
#
Allows Duplicates: Lists can contain the same value more than once.
#
Heterogeneous: Items in a list can be of different data types. -->

%%
#Access List Items
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
print(fruits)

#List items are indexed and you can access them by referring to the index number:

print(fruits[1])

#Negative indexing means start from the end

print(fruits[-1])

#You can specify a range of indexes by specifying where to start and where to end the range.
#When specifying a range, the return value will be a new list with the specified items.
#Note: The search will start at index 2 (included) and end at index 5 (not included).

print(fruits[1:3])
#By leaving out the start value, the range will start at the first item:
print(fruits[:3])
#By leaving out the end value, the range will go on to the end of the list:
print(fruits[2:])
#Range of Negative Indexes:-Specify negative indexes if you want to start the search from the end of the list:
print(fruits[2:4])

%%
#Change Item Value
#To change the value of a specific item, refer to the index number: -->
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]

```
fruits[2]="mango"
print(fruits)

#To change the value of items within a specific range, define a list with the new values, and refer to the range of index
# numbers where you want to insert the new values:
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
fruits[1:4]=["Sani","Saty"]
print(fruits)

                                                        #Insert Items
# To insert a new list item, without replacing any of the existing values, we can use the insert() method.
# The insert() method inserts an item at the specified index:

fruits.insert(3,"Cherry")
print(fruits)
fruits.insert(2,"Bnana")
print(fruits)
fruits.insert(1,"orange")
print(fruits)


# %%
#Add List Items
#To add an item to the end of the list, use the append() method:
fruits.append("orange")
print(fruits)
#To add an item at the specified index, use the insert() method:
fruits.insert(2, "kiwi")
print(fruits)
#To append elements from another list to the current list, use the extend() method:
vegetables = ["carrot", "potato", "cabbage"]
fruits.extend(vegetables)
name=["Sani","Saty","Nirbhay","Nitya","Ramu","Rani","Sita"]
fruits.extend(name)
print(fruits)
#To add elements from another list to the current list, using the + operator:
fruits = fruits + vegetables
print(fruits)
#To add elements from another list to the current list, using the += operator:
fruits += name
print(fruits)
#To add elements from another list to the current list, using the * operator:
fruits = fruits * 2
print(fruits)


# %%
# Remove Specified Item
# The remove() method removes the specified item.
#If there are more than one item with the specified value, the remove() method removes the first occurrence:
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
```

```
fruits.remove("banana")
print(fruits)
fruits.remove("kiwi")
print(fruits)

#The pop() method removes the specified index.
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
fruits.pop(5)
print(fruits)
#If you do not specify the index, the pop() method removes the last item.
fruits.pop()
print(fruits)

# The clear() method empties the list.
# The list still remains, but it has no content.
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
fruits.clear()
print(fruits)
# The del keyword removes the specified index:
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
del fruits[0]
print(fruits)


# %%
# Sort List Alphanumerically
# The sort() method sorts the list alphanumerically, ascending, by default:
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
fruits.sort()
print(fruits)
#You can also specify ascending or descending order:
fruits.sort(reverse=True)
print(fruits)
#Sort List Numerically
# The sort() method can also sort numbers:
numbers = [100, 50, 65, 82, 23]
numbers.sort()
print(numbers)


# %%
# Copy Lists
fruits = ["apple", "banana", "cherry","orange", "kiwi", "melon", "mango"]
frut=fruits.copy()
print(frut)

# %% [markdown]
#                                                          # Tuple
# # Tuples are used to store multiple items in a single variable.
#
# # Tuple is one of 4 built-in data types in Python used to store collections of data,
the other 3 are List, Set, and Dictionary, all with different qualities and usage.
#
# # A tuple is a collection which is ordered and unchangeable.
```

```
#
# # Tuples are written with round brackets.
#
# # Example:-
# # Create a Tuple:
# # thistuple = ("apple", "banana", "cherry")
# # print(thistuple)
#
# # Tuple Items
# # Tuple items are ordered, unchangeable, and allow duplicate values.
#
# # Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
#
# # Ordered
# # When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
#
# # Unchangeable
# # Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
#
# # Allow Duplicates
# # Since tuples are indexed, they can have items with the same value:
#
#

# %%
thistuple = ("apple", "banana", "cherry")
print(thistuple)
#Tuples allow duplicate values:
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
#Tuple Length
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
#Create Tuple With One Item
# To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.
thistuple = ("apple",)
print(type(thistuple))




# %%
                                                # Access Tuple Items

thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple[1])
#Negative Indexing
print(thistuple[-1])
print(thistuple[2:4])
print(thistuple[:3])
```

```
#This example returns the items from the beginning to, but NOT included, "apply":
print(thistuple[2:])
print(thistuple[2:4])
#Negative Indexing
# Specify Negative Indexes
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple[-1])
#Range of Negative Indexes
print(thistuple[-4:-1])



# %% [markdown]
#                                                                 # Change Tuple Values
# # Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.
# # But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

# %%
x = ("apple", "banana", "cherry")
y=list(x)
print(y)
y[1] = "kiwi"
x= tuple(y)
print(x)

# %% [markdown]
# Add Items
# Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.
#
# 1. Convert into a list: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.
#
# 2. Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:
#
#
#

# %%
#Example-1
x = ("apple", "banana", "cherry")
y=list(x)
y.append("orange")
x = tuple(y)
print(x)

#Example-2
x = ("apple", "banana", "cherry")
y=("orange", "kiwi", "melon")
z = x + y
```

```
print(z)

# %% [markdown]
#

# %% [markdown]
# Remove Items
# Note: You cannot remove items in a tuple.
#
# Tuples are unchangeable, so you cannot remove items from it, but you can use the same
workaround as we used for changing and adding tuple items:

# %%
x=('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon')
y=list(x)
y.remove("banana")
x=tuple(y)
print(x)



# %%
                                                    # Join Tuples
x=('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon')
y=('orange', 'kiwi', 'melon')
z=x+y
print(z)
#The count() method returns the number of times a specified value occurs in a tuple:
x = ('apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'banana')
print(x.count("banana"))
#The index() method finds the first occurrence of a specified value.
print(x.index("cherry"))
#If you want to multiply the content of a tuple a given number of times, you can use the
* operator:
x = ('apple', 'banana', 'cherry')
y=x*3
print(y)



# %% [markdown]
#                                                          **  Set**
# Sets are used to store multiple items in a single variable.
#
# A set is a collection which is unordered, unchangeable*, and unindexed.
#
# * Note: Set items are unchangeable, but you can remove items and add new items.
#
# Sets are written with curly brackets.

# %%
thisset = {"apple", "banana", "cherry"}
print(thisset)

# %% [markdown]
```

```
#                                                    **    ** Set Items****
#
# Set items are unordered, unchangeable, and do not allow duplicate values.
#
# **Unordered**
# Unordered means that the items in a set do not have a defined order.
#
# Set items can appear in a different order every time you use them, and cannot be
referred to by index or key.
#
# **Unchangeable**
# Set items are unchangeable, meaning that we cannot change the items after the set has
been created.
#
# Once a set is created, you cannot change its items, but you can remove items and add
new items.
#
# **Duplicates Not Allowed**
# Sets cannot have two items with the same value.

# %%
thisset = {"apple", "banana", "cherry"}
print(thisset)

# Access Items
# You cannot access items in a set by referring to an index or a key.

# But you can loop through the set items using a for loop, or ask if a
# specified value is present in a set, by using the in keyword.

thisset = {"apple", "banana", "cherry"}
for x in thisset:
    print(x)

print("banana" in thisset)
print("banana" not in thisset)

# %%
# Add Items
# Once a set is created, you cannot change its items, but you can add new items.

# To add one item to a set use the add() method.

thisset = {"apple", "banana", "cherry"}
thisset.add("Orange")
print(thisset)
# Add Sets
# To add items from another set into the current set, use the update() method.
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)

# %%
# Remove Item
```

```
# To remove an item in a set, use the remove(), or the discard() method.
# Create a set
thisset = {"apple", "banana", "cherry"}

# Print original set
print("Original set:", thisset)

# Remove "banana" using remove()
thisset.remove("banana")
print("After remove('banana'):", thisset)

# Try removing "banana" again using discard()
thisset.discard("banana")  # No error even if "banana" is not in the set
print("After discard('banana'):", thisset)



# %% [markdown]
# **Join Sets**
# There are several ways to join two or more sets in Python.
#
# The union() and update() methods joins all items from both sets.
#
# The intersection() method keeps ONLY the duplicates.
#
# The difference() method keeps the items from the first set that are not in the other
set(s).
#
# The symmetric_difference() method keeps all items EXCEPT the duplicates.
#
#

# %%
thisset = {"apple", "banana", "cherry"}
num = {5, 6, 7, 8, 22, 44, 65, 76, 87}
num2 = {11, 22, 33, 44, 55, 66, 77}

# | is union operator (same as union())
print(thisset | num)  # Mix of strings and integers

# union of all three sets
num3 = thisset.union(num, num2)

# printing the original thisset (unchanged)
print(num3)

#Join a Set and a Tuple
thisset = {"apple", "banana", "cherry"}
num = (5, 6, 7, 8, 22, 44, 65, 76, 87)
print(thisset.union(num))

# Update
# The update() method inserts all items from one set into another.

# The update() changes the original set, and does not return a new set.
```

```
thisset = {"apple", "banana", "cherry"}
num = (5, 6, 7, 8, 22, 44, 65, 76, 87)
thisset.update(num)
print(thisset)

#The intersection() method will return a new set, that only contains the items that are present in both sets.
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3=set1.intersection(set2)
print(set3)

#The difference() method will return a new set that will contain only the items from the first set that are
#not present in the other set.
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7}
c=A.difference(B)
print(c)

C=A-B
print(C)

#The symmetric_difference() method will keep only the elements that are NOT present in both sets
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7}
C=A.symmetric_difference(B)
print(C)



# %% [markdown]
# **Dictionary**
# Dictionaries are used to store data values in key:value pairs.
#
#  A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
#
# Dictionaries are written with curly brackets, and have keys and values
#
# **Ordered or Unordered**
# When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
#
# Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.
#
# **Changeable**
# Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
#
# **Duplicates Not Allowed**
# Dictionaries cannot have two items with the same key:
```

```
#

# %%
dic={
    "brand":"ford",
    "model":"mustand",
    "year":1989
}
print(dic)
print(dic["brand"])
print(dic["year"])


#Dictionaries cannot have two items with the same key:
dic={
    "brand":"ford",
    "model":"mustand",
    "year":1989,
    "year":7468,
    "colors": ["red", "white", "blue"]
}
print(dic)

#Dictionary Length
print(len(dic))
print(dic["colors"][2])
d = dic.get("colors", "Not Found")
print(d)

dic["color"] = "yellow"

#The keys() method will return a list of all the keys in the dictionary.
print(dic.keys())

#The values() method will return a list of all the values in the dictionary.
print(dic.values())

#The items() method will return each item in a dictionary, as tuples in a list.
print(dic.items())



# %%
# Change Values
# You can change the value of a specific item by referring to its key name:

car={
    "brand":"ford",
    "model":"mustand",
    "year":1989,
    "year":7468,
    "colors": ["red", "white", "blue"]
}
car["year"]=1760
print(car)
```

```
# The update() method will update the dictionary with the items from the given argument.
# The argument must be a dictionary, or an iterable object with key:value pairs.
car.update({"brand":"mahindra"})
print(car)

# %%
# Adding Items
# Adding an item to the dictionary is done by using a new index key and assigning a
value to it:
car={
    "brand":"ford",
    "model":"mustand",
    "year":1989,
    "year":7468,
    "colors": ["red", "white", "blue"]
}
car["colors"]="red"
print(car)

car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
car.update({"color": "red"})

# %%
# Removing Items
# There are several methods to remove items from a dictionary:

car={
    "brand":"ford",
    "model":"mustand",
    "year":1989,
    "year":7468,
    "colors": ["red", "white", "blue"]
}
car.pop("model")
print(car)

# %%
#- Create an inventory system tracking items and quantities with a dictionary

# Initial Inventory
inventory = {
    "apple": 10,
    "banana": 5,
    "orange": 8
}

# Function to display inventory
def display_inventory():
    print("\nCurrent Inventory:")
```

```
    for item, qty in inventory.items():
        print(f"{item}: {qty}")
    print()

# Function to add/update item
def add_item(item, quantity):
    if item in inventory:
        inventory[item] += quantity
        print(f"{quantity} added to existing item '{item}'.")
    else:
        inventory[item] = quantity
        print(f"New item '{item}' added with quantity {quantity}.")

# Function to remove item
def remove_item(item):
    if item in inventory:
        del inventory[item]
        print(f"Item '{item}' removed from inventory.")
    else:
        print(f"Item '{item}' not found.")

# Function to update quantity
def update_quantity(item, quantity):
    if item in inventory:
        inventory[item] = quantity
        print(f"Quantity of '{item}' updated to {quantity}.")
    else:
        print(f"Item '{item}' not found.")

# ------------------------------
# Sample Usage
# ------------------------------
display_inventory()

add_item("apple", 5)
add_item("mango", 12)
update_quantity("banana", 10)
remove_item("orange")

display_inventory()
```