List:

Used for saving any type of data sepeared by comma…

Eg: list=["hello",23,3.4 ,False,45]

Indexing of list: 0 to n-1

Slicing of list: [Start,Stop,Step] # 0 to n-1

Eg: metrics[0:5]

['Accuracy', 'precision', 'Recall', 'F1score', 'AUC\_ROC\_Score']

Eg: metrics[0:]

['Accuracy', 'precision', 'Recall', 'F1score', 'AUC\_ROC\_Score']

metrics[1:-3]

['precision']

metrics[1:1]

**[ ] #Empty List**

metrics[0:4:1]## use step

['Accuracy', 'precision', 'Recall', 'F1score']

metrics[4:0:-1]## use step

['AUC\_ROC\_Score', 'F1score', 'Recall', 'precision']

metrics[0:4:2]

['Accuracy', 'Recall']

metrics[:4:-1]

**[ ]**

**List Operations:**Concatenation

print(metrics+["Rosh carporate","tarun"]) # concates at end off the list previous elements

['Accuracy', 'precision', 'Recall', 'F1score', 'AUC\_ROC\_Score', 'Rosh carporate', 'tarun']

# adds in the end of the list…

metrics ##just added but not store in metrics variable then how to do that

['Accuracy', 'precision', 'Recall', 'F1score', 'AUC\_ROC\_Score']

List Methods:

append()

Use the append() method to permanently add an item to the end of a list.

append() method takes the element which you want to add to the list as an argument

lis.append(2)

lis.append("PariShree")  ## appends as  last element of list

lis

[1,

2,

3,

4,

'ArithmeticError',

'amity',

'hello',

'hi',

['a', 'b'],

[1, 2],

2,

'PariShree']

##pop()

Use pop() to "pop off" an item from the list.

By default pop() takes off the element at the last index, but you can also specify which index to pop off.

pop() takes the index as an argument and returns the elenent which was popped off.

lis.pop()

PariShree

lis.pop(5)

amity

## extend()

To add more than one element in the list..

lis.extend([2,'ji']) #how many times it runs it add those elements that many times end of list

lis

[1, 2, 3, 4, 'ArithmeticError', 'hello', 'hi', ['a', 'b'], [1, 2], 2, 2, 'ji']

Remove Method:

To remove the elements from lists…

lis.remove(2)

lis

[1, 3, 4, 'ArithmeticError', 'hello', ['a', 'b'], [1, 2], 2, 2, 2, 'hi', 2]

##clear()

Removes all the elements from the list

lira.clear()  
lira

[]

Tuple:

The elements inside the parenthesis is called a tuple.

Eg:

# Create a tuple

ab=("raju","ravi",1,2,3,7.8)

print(ab)

type(ab)

Ouput:

('raju', 'ravi', 1, 2, 3, 7.8)

tuple

Tuple Indexing:

print(ab)

('raju', 'ravi', 1, 2, 3, 7.8)

O/p:

('raju', 'ravi', 1, 2, 3, 7.8)

('raju', 'ravi', 1, 2, 3, 7.8)

# Grab the element at index 0, which is the FIRST element

ab[0]

'raju'

o/p:

raju

# Grab the element at the index -3, which is the THIRD LAST element

print(ab[-3])

o/p:

2

Tuple Operations:

## concatenation

a=(1,2,3,7.8)

print(a+ab)

o/p:

(1, 2, 3, 7.8, 'raju', 'ravi', 1, 2, 3, 7.8)

### multiplication with integer

print(a\*2)

o/p:

(1, 2, 3, 7.8, 1, 2, 3, 7.8)

Count( ):

##count()

Returns the number of times a specified value occurs in a tuple

abc=(1,1,2, 2,2, 3, 9, 143, 11.5, 'aditya', 'swathi', 'sujana')

abc.count(1)

o/p:  
2

Sets:

Which are represented in square braces!

### Key Properties of Sets:

1. **\*\*Uniqueness\*\***: Sets only contain unique elements. If you try to add a duplicate, it will be ignored.

2. **\*\*Unordered\*\***: Elements in a set have no specific order. The order of elements when iterated is not guaranteed.

3. **\*\*Mutable\*\***: Elements can be added or removed from a set after its creation.

4. **\*\*Iterability\*\***: Sets can be iterated using loops, allowing you to access each element.

s = {1,23}

type(s)

o/p:

set

d = {1:2 ,4:"b"}

type(d)

o/p:

dict

s1 = {}

type(s1)

o/p:

dict

s2 =set()

type(s2)

o/p:

set

s3 ={1,2, "price",(1,23,4)}

print(s3)

o/p:

{1, 2, 'price', (1, 23, 4)}

l =[1,2,34,5]

l1 =set(l)

type(l1)

o/p:

set

# uniqueness

s = {1,2,3,43,43,34,343,43,43}

print(s)

o/p:

{1, 2, 3, 34, 343, 43}

# unordered

s1 = {4,3,2,1,35,35 ,4}

print(s1)

o/p:

{1, 2, 3, 4, 35}

### Where and When to Use Sets

1. **\*\*Removing Duplicates\*\***:

    - Use sets when you need to store a collection of unique elements and automatically handle duplicates.

2. \*\*Fast Membership Testing\*\*:

    - Sets offer faster membership testing (`in` operator) compared to lists or tuples.

3. \*\*Mathematical Operations\*\*:

    - Sets are ideal for performing mathematical operations like union, intersection, difference, and symmetric difference.

4. \*\*Handling Large Datasets\*\*:

    - Sets are highly efficient for operations that involve large datasets, where you need to compare, combine, or de-duplicate data.

\*\*\*Sets are mutable !!!  
l =[1,3,42,3,453,23,23,23,25,34,3,53,54,54,43]

l1 =set(l )

type(l1)

print(l1)

o/p:

{1, 34, 3, 453, 42, 43, 53, 54, 23, 25}

1.add(element)

l1.add(2)

print(l1)

o/p:

{1, 34, 3, 2, 453, 42, 43, 53, 54, 23, 25}

2. **\*\*`remove(element)`\*\***

    - Removes a specific element from the set. Raises a `KeyError` if the element is not found.

Eh:

Print(s)

o/p:

{1, 2, 3, 4, 6, 76}

s.remove(76)

o/p:

{1,2,3,4,5,6}

3. **\*\*`discard(element)`\*\***

    - Removes a specific element from the set, but does not raise an error if the element is not found.

s.discard(76)

4. **\*\*`pop()`\*\***

    - Removes and returns an arbitrary element from the set. Raises a `KeyError` if the set is empty.

s.add(5)

s.add(6)

s.pop() o/p:

6

s.pop() 0/p:

5

5. **\*\*`clear()`\*\***

    - Removes all elements from the set, leaving it empty.

s.clear()

### Operators on Sets

1. **\*\*Union (`|`)\*\***:

    - Combines elements from both sets.

set1 ={1,2,3}

set2 ={3,5,6}

print(set1 | set2)

o/p:

{1, 2, 3, 5, 6}

2. **\*\*Intersection (`&`)\*\***:

    - Finds common elements between sets.

set1 ={1,2,3}

set2 ={3,5,6}

print(set1 &set2)

o/p:

{3}

3. **\*\*Difference (``)\*\***:

    - Returns elements in the first set but not in the second.

set1 ={1,2,3}  
set2 ={3,5,6}  
print(set1 -set2)

o/p:

{1, 2}

4. **\*\*Symmetric Difference (`^`)\*\***:

    - Returns elements in either set but not in both.

s3 ={3,7,8}

set1.union(set2,s3)

o/p:

{1, 2, 3, 5, 6, 7, 8}

set1.intersection(set2)

o/p:

{3}

set1.difference(set2)

o/p:

{1, 2}

set1.symmetric\_difference(set2)  
o/p:

{1, 2, 5, 6}

# Introduction to Dictionaries ✅

> Constructing a Dictionary

> A dictionary object is constructed using curly braces {key1:value1,key2:value2,key3:value3}

> what is Dictionary : **\*\* mapping of one to one relationship b/w key ,value pair is called dictionay\*\***

> what are mappings? Mappings are a collection of objects that are stored by a key, unlike a sequence that stored objects

# Make a dictionary with {} and : to signify a key and a value

marvel\_dict = {'Name':'Thor'   ,  'Place':'Asgard',  'Weapon' : 'Hammer',     1:2,    3 : 'power',    'super\_heros' : ['Ironman','Captain America'],  'abc' : {1:2, 4:5}}

marvel\_dict.keys()

o.p:

dict\_keys(['Name', 'Place', 'Weapon', 1, 3, 'super\_heros', 'abc'])

marvel\_dict["Name"]

o/p:  
'Thor'

marvel\_dict["super\_heros"]

o/p:

['Ironman', 'Captain America']

marvel\_dict ['Weapon']

o/p:

'Hammer'

marvel\_dict["abc"]

o.p:  
{1: 2, 4: 5}

type(marvel\_dict)

o/p:  
dict

Dict Methods:

\*\*\*Keys()

list(marvel\_dict.keys())  
o/p:

['Name',

'Place',

'Weapon',

1,

3,

'super\_heros',

'abc',

'Designation',

'Salary',

'ravi']

# values()

> values() method returns the list of values in the dictionary object

print(marvel\_dict)

o/p:

{'Name': 'Thor', 'Place': 'Asgard', 'Weapon': 'Hammer', 1: 2, 3: 'power', 'super\_heros': ['Ironman', 'Captain America'], 'abc': {1: 2, 4: 5}}

marvel\_dict.values()

o/p:  
dict\_values(['Thor', 'Asgard', 'Hammer', 2, 'power', ['Ironman', 'Captain America'], {1: 2, 4: 5}])

list(marvel\_dict.values())

o/p:

['Thor',

'Asgard',

'Hammer',

2,

'power',

['Ironman', 'Captain America'],

{1: 2, 4: 5}]