Collection Data type:

Till now we were storing one data with a single data type to a single variable.

Now we want to store multiple data of same or different data types so we use the Collection data type.

Collection data type

four different types

- List
- Tuple
- set
- Dictionary

How the data is stored depends on individual properties.

LIST

Properties of Lists:

1) List is a Collection data type.

2) List can store same or different types of data.

3) Inside the list each data is called element.

4) Inside a list each element has an index value. (+ve Indexing and -ve Indexing can be used)

5) Lists are ordered (because they have index).

6) Lists are mutable. (After creation we can replace add or delete).

7) Lists can have duplicate elements.

8) Lists are dynamic.

Lists are represented in square brackets.

$$[e_1, e_2, e_3, \cdots]$$

The elements inside need to be seperated by comma's.

List constructor

list( ) : It is also possible to use the list() Constructor when creating a new list.

list (($e_1, e_2, e_3, \cdots$))

* It has double round brackets.

ex:- [1, 2, 1+2j, "S", -1.2]

Note* : different datatypes.

Both indexing and slicing concepts can be used here.

ex:- list1 = ['apple', 'banana', 'orange']

list1 [2]
→ ['orange']

list1 [0:2]
→ ['apple', 'banana']

* list( ) constructor can construct lists from lists, tuples and sets.

Adding elements with replacing :

For this we use indexing and slicing.

To add single element in a particular index position we use indexing.

ex:-

```
l = [11, 12, 13, 'a', 'b']
```

if we want to add 100 to this list at index position of 3 we say

```
l[3] = 100
```

'a' got replaced with 100.

∴ `l = [11, 12, 13, 100, 'b']`

if `l[4] = 100, 200, 300`

↳ just use commas instead of brackets as memory leakage happens.

∴ `l = [11, 12, 13, 100, 100, 200, 300]`

If we want to add element at a range of index positions we use slicing.

```
l[2:4] = 40,
```

↳ Comma is mandatory as machine is expecting iterable (list of elements)

`l = [11, 12, 40, 100, 200, 300]`

* when we are accessing multiple elements and replacing them it is not mandatory to replace

with same number of elements.

* If lesser elements are used to replace then the other elements from the original list will disappear.

Adding elements without replacing :-

three list methods
- insert()
- append()
- extend()

List methods: Methods that can only be used on lists.

we use '.' dot operator here. It first checks if list is there and then applies the method.

Insert() method

Insert is a list method by using which we can add single element without replacement at a specified location.

Here we need to specify the index where element needs to be added and the element to be added.

insert (Index value , Element to be added)

ex:-

```
l = [11, 12, 100, 200, 300]
l.insert(0, "a")
l = ['a', 11, 12, 100, 200, 300]
```

* In insert() the old elements are pushed to the right to create space for the element to be added.

* If we want to add element at the last index position using -ve index '-1' then element cannot be pushed and we get wrong list.

```
l.insert(-1, "b")
l = ['a', 11, 12, 100, 200, 'b', 300]
```

↳ wrong position ✗

Instead we need to use +ve indexing

```
l.insert(5, "b")
```

* Insert can only insert one element at a time so if we want to insert multiple elements we need to perform insert option multiple no.of times.

append() method

append() is a method by using which we can add a single element without replacement at the end of the list.

Here the index value need not be specified as the new element is appended at the end of the list.

ex:- l = ['a', 11, 12, 100, 200, 'b', 300]

l.append (1+2j)

l = ['a', 11, 12, 100, 200, 'b', 300, (1+2j)]

brackets are only added for visualization

* append can only insert one element at a time. If we want to insert multiple elements append needs to be applied multiple number of times.

extend( ) method

This is a list method that can append n number of sequential data elements without replacement at the end of the current list.

var.extend(sequential data)

Sequential data: This is a data that has sequence of elements.

Data type	Sequential data

Numeric data types ✗

like int, float, complex

Strings ✓

Collection data type ✓

ex:- `l = [0, 1, 2, 3, 'a']`

`l.extend ([100, 200, 300])`

↳ these square brackets are must.

`l = [0, 1, 2, 3, 'a', 100, 200, 300]`

`l.extend ("abcde")`

`l = [0, 1, 2, 3, 'a', 100, 200, 300, 'a', 'b', 'c', 'd', 'e']`

↑

extends as single elements and not as a string

* when we want to append another list to the current list we use extend().

Deleting / Removing elements from lists

four ways
- del keyword
- pop() method
- remove() method
- clear() method

del keyword

del keyword can remove a single element from a specified index or a range of elements or complete list.

ex:-

```
l = [11, 12, 13, 100, 200, 300]
```

```
del l[2]
l = [11, 12, 100, 200, 300]
del l[2:5]
l = [11, 12]
del l
```

↳ l is completely deleted.

pop() method

pop() method is a list method by using which we can remove an element from a specified index from the list.

- Here index values need to be specified.
- del keyword can be used anywhere.
- pop() is only used in lists.
- If no index is specified in pop this method removes the last item

ex:-

```
l=[11, 12, 13, 100, 200, 300]
l.pop(0)
    ↳ 11 (output gives what was removed)
l = [12, 13, 100, 200, 400]
l.pop()  → nothing was mentioned.
    ↳ 400 (i.e last element)
l=[12, 13, 100, 200]
```

remove() method

remove() method is a list method by using which we can remove an specified element from the list.

- Here we specify the element to be removed and not the index of the element.
- will only remove the first instance of the element.
- we can only remove one element at a time

ex:- l = [11, 12, 13, 100, 11, 400, 12]

l.remove(11)

l = [12, 13, 100, 11, 400, 12]

l.remove(11)

l = [12, 13, 100, 400, 12]

clear( ) method

clear( ) method is mainly used to remove all the elements from the list.

i.e to create an empty list

l.clear( )

↳ [ ] (empty list)

* empty list - if the list exists but it does not have any elements in the list.
* After clear the list still exists but it is empty.

ex:- l = [11, 12, 13]

l.clear()

l = [ ]

Copying a list

We cannot copy a list simply by typing

list2 = list 1 ×

because list2 will only be a reference/view to the list 1, and changes made in list1 will automatically be made in list2.

There are ways to make a copy. one way is to use a list method called copy().

Copy() method

Copy() method is a list method that will copy and create a new memory location. so new list is created.

ex:- l = [11, 12, 13] (z = l is ×)

z = l.copy() ✓

z = [11, 12, 13]

we can also use list() method to copy a list.

l = [11, 12, 13]

z = list(l)

z = [11, 12, 13]

Exercise

Q) Create a list where inside the list 1st element is first name, 2nd element is last name, 3rd element is age, 4th element is height.

```
l = ["subhadra", "Bhupathiraju", 42, 170]
bio_data = "my name is {}_{} my age is {}
                        my height is {}"
bio_data.format (l[0], l[1], l[2], l[3])
```

↳ my name is subhadra Bhupathiraju my age is 42 my height is 170.

Sort list Alphanumerically

sort( ) method

list objects have a sort( ) method that will sort the list alphanumerically, ascending, by default.

* The sort list needs to be homogenous.

homogenous - same type of data

heterogenous - different data types of data

ex:-

```
l = [4, 7, 2, 6, 8]
l.sort()
l = [2, 4, 6, 7, 8]
```

* If we want to sort in descending order we need to use the keyword argument.

```
reverse = True
l.sort(reverse=True)
l = [8, 7, 6, 4, 2]
```

* by default reverse = False
* Once the list is sorted it will automatically stored. So no need to do l = l.sort( ).

- Alphabets are sorted alphabetically with preference to upper case letters.

```
l = ['a', 'c', 'd', 'A', 'e']
l.sort()
l = ['A', 'a', 'c', 'd', 'e']
```

* Complex cannot logically be sorted.

reverse() method

reverse() method in the list reverses all the elements.

ex:-

```
l = ['a', 'b', 'c', 'd']
l.reverse()
l = ['d', 'c', 'b', 'a']
```

ex:-

```
l = [5, 7, 6, 4]
l.reverse
l = [4, 6, 7, 5]
```

*Count() and index() methods can be applied to both Lists and tuples. find() does not exist.

count() method

count() method is used to count how many number of times that particular element exists in the list [] or Tuple().

```
l.count("element")
```
↳ number of times it exists.

index() method

index() method is used to know the index of an element at the first instance point left to right.

```
l.index("element")
```
↳ index at the first instance.

* lists can be created with single elements.

ex:- `l = [1]`

List methods

- insert()
- append()
- extend()
- remove()
- pop()
- clear()
- copy()
- sort()
- reverse()
- count()
- index()

Tuples

Properties of tuples

1) Tuple is a collection data type.

2) tuples can store same or different types of data.

3) Inside the tuple each data is called element.

4) Inside a tuple each element has an index value. (+ve indexing and -ve indexing can be used)

5) tuples are ordered (because they have index).

6) tuples are immutable (After creation they cannot be changed)

7) tuples can have duplicate elements.

8) tuples are static

tuples are represented in round brackets.

$$(e_1, e_2, e_3, \ldots)$$

The elements inside need to be seperated by comma's.

Tuple Constructor

tuple( ) constructor can also be used to create a new tuple.

tuple((e1, e2, e3, ...))

* It has double round brackets.
* Both indexing and slicing concepts can be used here to access single or sequential elements.
* count( ) and index( ) methods can be used on tuples.
* other tuple methods do not exist because tuples are immutable.

tuples cannot
- replace
- add
- delete

If we try to edit tuple it will give error.

* If we want to create a single element tuple the we cannot mention

t=(e1) x

as python considers this as an integer.

So we should write

t=(e1,)

↳ this comma is must

ex:- t=(1,)

* Empty tuple is ( )
* Converting tuple to list can be done but it is not a recommended practice.
  For this we can use list( ) constructor.
* tuple( ) constructor can construct tuples from lists, tuples and sets.

Tuple unpacking:

If we want to unpack a tuple we assign it to the no.of variables that are equal to the no.of parameters.

ex:- x, y, z, q = (1, 2, 3, 4)

x → 1, y → 2, z → 3, q → 4

The no.of variables and elements have to match or we will get an error.

Instead we can use args (*)

x, y, *z = (1, 2, 3, 4)

x → 1, y → 2, z → [3, 4]

$x, *y, 3 = (1, 2, 3, 4)$

$x \rightarrow 1$ $\quad$ $y \rightarrow [2, 3)$ $\quad$ $3 \rightarrow 4$

Sets

Properties of sets

1) Sets is a collection data type.

2) Sets can store same or different types of data.

3) Inside the set each data is called element.

4) Elements do not have index value inside a set.

5) Sets are unordered (because they do not have index)

6) Sets are mutable (No replace), and unchangeable

Sets items are unchangable, meaning that we cannot change the items after the set has been created (i.e we cannot replace). But we can add new items and delete items.

* we cannot replace because we do not have index to access that particular element.

7) Sets will not allow duplicate values. All the duplicate values will be removed.

8) Sets are mutable dynamic collections of immutable unique elements

* Sets are represented in flower brackets.

$$\{e_1, e_2, e_3, \ldots\}$$

The elements inside need to be seperated by comma's.

we can use a set constructor set() to create a set

set((e1, e2, e3, ...)) → double round brackets

* we cannot access items in a set using index.

Add items: once a set is created, we cannot change its items but we can add items.

- add()
- update()

add() method

add() is a set method by using which we can add a single new element inside a set.

Var. add("element to be added")

we do not know at what position it will be added.

update( ) method

update( ) is a set method by using which we can add multiple sequential data elements at a time.

The object inside the update( ) method does not have to be a set, it can be any other iterable object. (tuples, lists, dictionaries etc)

```
Set1.update(set2)
set1.update("string")
Set 1.update([list])
Set 1.update((tuple))
```

Remove Items : we can remove an item in a set.

- remove()
- discard()
- clear()
- del

remove() method :

remove() is a set method by using which we can remove a single element at a time.

* If the item to remove does not exist remove will raise an error.

```
var.remove("a")
```

discard() method :

discard() method is a set method by using which we can remove a single element at a time.

* If the item to be removed does not exist, discard() will NOT raise an error so the code does not break.

```
var.discard("a")
```

* we can use pop() method but since sets are unordered and pop() removes the last element when index is not specified we do not know which item is removed. before hand.

* The output of the pop method is the removed item.

clear() method:

clear() is a set method that will remove all the elements from a set and create an empty set.

```
Var. clear()
    ↳ set()
```

[ ] → Empty list
( ) → Empty tuple
set() → Empty set
* { } → Empty dictionary

del keyword: del keyword will delete the set completely.

Join sets ( ) we use several ways to join two or more sets in python.

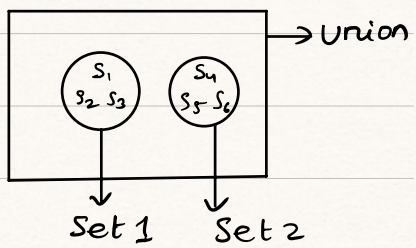
- union( )
- update( )

union( ) method

It joins the two sets and returns a new set containing all the items from both sets.

Set 1 = { S_1, S_2, S_3 }

Set 2 = { S_4, S_5, S_6 }



Set 3 = set 1 . union (set 2)

= { S_1, S_2, S_3, S_4, S_5, S_6 }

* Any duplicates are removed.

update ( ) method

update ( ) method when used between two sets it inserts all the items from one set to the other set.

Set 1 = { S_1, S_2, S_3 }

Set 2 = { S_4, S_5, S_6 }

Set 1 . update (set 2)

↳ {S1, S2, S3, S4, S5, S6}

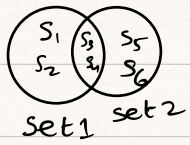
* The update() does not create a new list.
* It deletes all duplicate values.

Keep only Duplicates

Intersection() method:

The intersection() method will return a new set, that only contains items that are present in both sets.

set1 = {S1, S2, S3, S4}
set2 = {S3, S4, S5, S6}



Set3 = Set1.intersection(set2)
= {S3, S4}

Intersection_update method

The intersection_update() method will keep only the items that are present in both sets and update the original with these values.

* This will not create a new set.

set1.intersection_update(set2)

Set1 ⟶ {S3, S4}

set2.intersection_update(set1)

set2 → {S3, S4}

Keep All, But NOT the Duplicates:

difference() method

This difference() method will keep only the elements that are NOT present in both sets.

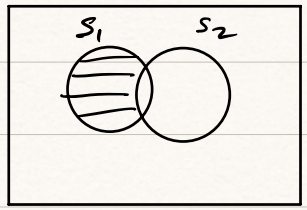
set1 = {S1, S2, S3, S4}

set2 = {S3, S4, S5, S6}

set3 = set1.difference(set2)

= set1 - set2

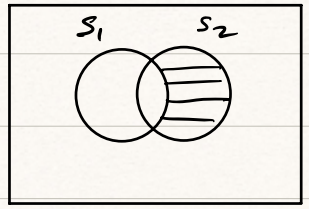
= {S1, S2}



set3 = set2.difference(set1)

= set2 - set1

= {S5, S6}



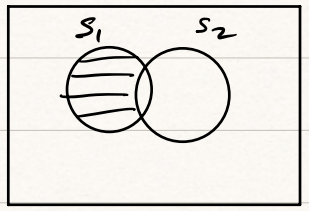
* Creates a new set.

difference_update() method: The difference_update method will keep only the items that are NOT present in both sets and update

the original set with these values.

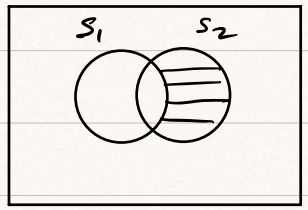
set1.difference_update(set2)

set1 → {S_1, S_2}



set2.difference_update(set1)

set2 → {S_5, S_6}



* Sets only have single elements in it. It cannot have sequential data in it.
i.e sets cannot contain lists, tuples etc.

Set methods

Method	Description
add( )	Adds an element to the set
clear( )	Removes all the elements from the set
copy( )	Returns a copy of the set
difference( )	Returns a set containing the difference between two or more sets.
difference_update( )	Removes the items in this set that are also included in another specified set.
discard( )	Removes the specified item.
intersection( )	Returns a set that is intersection of two other sets
intersection_update( )	Removes items in this set that are not present in other, specified set(s).
isdisjoint( )	Returns whether two sets have a intersection or not.
issubset( )	Returns whether another set contains this set or not.
issuperset( )	Returns whether this set contains another set or not.
pop( )	Removes an element from the set
remove( )	Removes the specified element.

union( )	Return a set containing the union of sets.
update( )	update the set with the union of this set and others.

Dictionary

Properties of Dictionaries

1) dictionary is a collection data type.

2) dictionaries are used to store data values in key: value pairs.

3) A dictionary is a collection which is ordered

* Note: before 3.6 version dictionaries were unordered.

4) Inside the dictionary the key values are our index's.

5) Indexing concept is possible but slicing is not possible as keys are not sequential.

6) Keys cannot be duplicate but values can be duplicate. If duplicate key is used then the previous value in the key will be replaced with the new value.

7) dictionaries are mutable.
keys are immutable but values are mutable.

8) dictionaries are dynamic.

we can create a dictionary using flower brackets.

Each element will have keys and values.

$$\{key_1 : value1 , key_2 : value2 , \ldots\ldots\}$$

Key and value are seperated by (:) colon.

elements are seperated by a comma(,).

* Keys and values can be any data type.
* Advised to use integer and string in the keys. It is illogical to use float and complex.
* values can be anything including a dictionary, i.e int, float, set, list, tuple & dictionary.
* dict() constructor exists in python but it is not used.

<u>dictionaries are mutable</u>

This means that we can change, add or remove items after the dictionary has been created.

To replace, delete or add we can use indexing. Here the index would be the key.

replace

```
var["key1"] = new value
```

delete

```
del var["key1"]
```

here both key and the corresponding value will be deleted

add

```
var["new key"] = "new value"
```

* If the mentioned key is non-existing after the check process then it will add the new key and value to it. New key is added at the end.

* If key is already present the new value will replace the old value.

Dictionary methods:

update() method:

update() method is a dictionary method in which we can replace or add multiple elements inside a dictionary.

* when we use update() method for dictionary it can only take the input as dictionary.

Var.update({key1:value1, key2:value2, ...})

* If keys are same it will replace old values with the new values.
* If keys are different it will add multiple elements at a time.

pop() method

pop() method is used to remove a key that is specified.

Var.pop("key") (here index is key)

Value will also be deleted along with the key.

popitem() method

The popitem() method removes the last inserted item.

Note: In versions before 3.7, a random item is removed instead.

Var.popitem()

* Needs to be used with caution.

keys() method

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

Var.keys()

↳ [keys list]

i.e [key1, key2, key3, ...]

values method

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

Var.values()

↳ [values list]

i.e [value1, value2, ...]

items() method

The items() method will return all the keys & values in the form of a tuple list.

Var. items()
↳ [(key1, value1), (key2, value2)....]

clear() method

The clear() method will clear all the elements and give an Empty dictionary.

Var. clear()
↳ { } Empty dictionary

copy() method

copy() method is used to create a new dictionary by creating the copy of the old one.

$d_2 = d_1.copy()$

* Assignment operator $d_2 = d_1$ will only create a view of the original dictionary.

Nested dictionary

A dictionary can contain dictionaries inside this is called nested dictionaries.

ex:- A dictionary nested inside another dictionary.

For this the $key_1 : value_1$. The $value_1$ here will be another $key_2 : value_2$ so we need double indexing to access the value.

$$Var = \{key_1 : \{key_2 : value_2\}\}$$

$$Var[key_1] = \{key_2 : value_2\}$$

$$Var[key_1][key_2] = value_2$$

** When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and it could mean an increase in efficiency or security.

Dictionary methods

method	Description
clear( )	Removes all the elements from the dictionary.
copy( )	Returns a copy of the dictionary.
fromkeys( )	Returns a dictionary with the specified keys and value.
get( )	Returns the value of the specified key
items( )	Returns a list containing a tuple for each key:value pair.
keys( )	Returns a list containing dictionary keys.
pop( )	Removes the element with the specified key.
popitem( )	Removes the last inserted key:value pair
setdefault( )	Returns the value of the specified key. If the key does not exist, insert the key with specified value.
update( )	Updates the dictionary with the specified key:value pair
values( )	Returns a list of all the values in the dictionary.

Lists

- have index
- Ordered
- Mutable
- Allow Duplicates
- $[e_1, e_2, \cdots]$
- list()
- [] - empty list
- Indexing & slicing possible

Tuples

- have index
- ordered
- immutable
- Allow Duplicates
- $(e_1, e_2, \cdots)$
- tuple()
- () - Empty tuple
- Indexing & slicing possible

sets

- No index
- un ordered
- partially mutable
- Do not Allow Duplicates
- $\{e_1, e_2, \cdots\}$
- set()
- set() Empty set
- Indexing & slicing NOT possible

Dictionaries

- key is the index
- Ordered (version 3.7+)
- mutable
- key cannot be duplicate value can be duplicate
- $\{k_1:v_1, k_2:v_2, \cdots\}$
- dict()
- {} - Empty dictionary
- Slicing is NOT possible Indexing is possible
- Elements are in pairs

join( ) method (string method)

The join() method takes all items in an iterable and joins them in to one string.

* A string must be specified as the seperator.

String . join (iterable)

* Input is any iterable object (i.e list, set or tuple)

"operator". join (var)

operator is better to be empty or space(␣).

l = {'a', 'b', 'c', 'd'}

"" . join (l)

↳ "abcd"

"␣" . join (l)

↳ "a b c d"

* join() will be used a lot in NLP.