MODULE - 2

1] - Lists -

① ~~Slicing~~ → List is a value that contains multiple values in an ordered sequence.

→ It looks like - ['cats', 'dog', 'mice']

→ It begins with an opening square bracket & closing square bracket [].

→ Values inside list are enclosed within ' ' & seperated by ','.

eg:-

```
>>> ['cat', 'bat', 'rat']
>>> spam = ['cat', 'bat', 'rat']
>>> spam
['cat', 'bat', 'rat']
```

* Slicing and Indexing :-

* Indexing :-

• Indexes are integer values that tells us at what position is a value in the list placed.

• eg :- spam = ['cat', 'bat', 'rat']

spam[0] spam[1] spam[2]

• We can use indexes to check the position of a value or obtain a value from a given particular position.

eg:-

```
>>> 'Hello' + spam[0]
Hello cat.
```

→ If we use an integer value that exceeds the number of values in the list as index, the python gives an IndexError.

→ Indexes can only be integer values.

→ Lists can also contain multiple lists as values and we can use multiple indexing for this.

~~eg:- >>> spam = ['cat', 'bat'], [10, 20, 30]~~

~~>>> spam[0][1]~~

~~'cat', 20 ['cat', 'bat'] ['cat', 'bat']~~

~~Here~~

eg:- >>> spam = [['cat', 'bat'], [10, 20, 30]]

spam[0] spam[1]

In this case, if we want to access a value from the indexes we use:-

```
>>> spam[0][1]
```

list value — value inside the selected list

o/p ⇒ 'bat'

```
>>> spam[1][2]
```

o/p ⇒ 30.

• Negative indexes -

→ The integer value -1 refers to the last index in the list.

→ By using negative indexes, we can access the list values in the reverse order [right to left]

→ eg:- >>> spam = [10, 20, 30, 40]
>>> spam[-1]
40.

Slicing

→ A slice can be used to access several values from a list, to form a new list.

→ A slice is also typed between [ ] but it has 2 integer values.

eg:- spam [1 : 4]

index where slice starts including the index itself.

where index ends but does not include the index (4)

Hence

>>> spam = ['cat', 'bat', 'rat', 'elephant', 'cheetah']
0 1 2 3 4

>>> spam = [1 : 4]

o/p → ['bat', 'rat', 'elephant'].

>>> spam = [2 : 3]

o/p → ['rat'].

→ We can also leave out one or both indexes on either side of the colon in the slice.

→ Here :-

① Leaving out the 1st index means using '0' or starting from beginning.

② Leaving out the 2nd index means slicing until the end and also including the last index.

eg :-
```
>>> spam = ['cat', 'bat', 'rat']
>>> spam[1:]
'cat', 'bat', rat
>>> spam[:2]
'cat', 'bat'
>>> spam[:]
'cat', 'bat', 'rat'
```

→ To get the length of a list, we use

len()

```
>>> spam = ['cat', bat', 'rat']
>>> len(spam)
3.
```

→ Using indexes, we can also change values in a list and even delete them.

eg :- spam[1] = 'dog'

will change the value of the ~~posit~~ existing index [1] to new value 'dog'

```
>>> spam = ['cat', 'rat', 'bat']
>>> spam[0] = 'dog'
>>> spam
['dog', 'rat', 'bat'].
```

```
>>> spam = ['cat', 'rat', 'bat']
>>> spam[0] = spam[1]
>>> spam
'rat', 'rat', 'bat'.
```

```
>>> spam = ['cat', 'rat', 'bat']
>>> spam[-1] = 'buffalo'
>>> spam
['cat', 'rat', 'bat', 'buffalo']
```

→ To delete a value in a list using indexes:-
we use the <u>del</u> statement.

```
>>> spam = [1, 2, 3, 4, 5]
>>> del spam[2]
>>> spam
[1, 2, 4, 5].
```

```
>>> spam = ['cat', 'rat', 'bat']
>>> del spam[2]
>>> spam
['cat', 'rat']
>>> del spam[-1]
>>> spam
['cat'].
```

- To concatenate 2 lists ⇒ '+' operator is used

```
>>> [1,2,3] + ['A','B','C']
[1, 2, 3, 'A', 'B', 'C']
```

- To replicate a list ⇒ '*' operator is used

```
>>> ['X','Y','Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

- The 'in' and 'not in' operators :-

→ These are used to check whether a value is or isn't present in the list.

→ They evaluate to a Boolean value - True / False

→ eg :-

```
>>> 'Hi' in ['Hello', 'Hi', 'Bye']
True.
>>> spam = ['Hello', 'Hi', 'Bye']
>>> 'Hi' in spam
True.
>>> 'cat' in spam
False.
```

* Augmented Assignment operators :-

→ Augmented assignment statement	Equivalent assignment statement
① spam = spam + 1	⇒ spam += 1
② spam = spam - 1	⇒ spam -= 1
③ spam = spam * 1	⇒ spam *= 1
④ spam = spam / 1	⇒ spam /= 1
⑤ spam = spam % 1	⇒ spam %= 1

→ It is a shortcut to assign values to variables.

eg:
```
>>> spam = 'Hello'
>>> spam += 'world'
>>> spam
'Hello world'
```

→ '+=' can also do string/list concatenation as shown above.

→ '*=' can do string/list replication as shown below :-
```
>>> spam = ['Hi']
>>> spam *= 3
>>> spam
['Hi', 'Hi', 'Hi'].
```

* Methods -

→ A method is same as a function but it is "called on" a value.

→ It comes after a value, seperated by a '.'

→ Each data type has its own set of methods.

(*) List Methods:-

1. index()
2. append() & insert()
3. remove()
4. sort()

① index() (Finding a value in a List)

→ This is used to find the index of a given value in a list.

→ If the value exists, it is returned and if not, it causes a ValueError.

eg:- >>> spam = ['hello', 'hi', 'yo']
>>> spam.index('hello')
0
>>> spam.index('yo')
2

→ when there are duplicates of a certain value, the index of the first appearance of that value is returned.

eg:-
```
>>> spam = ['cat', 'dog', 'bear', 'cat']
>>> spam.index('cat')
0.
```

② append() and insert() - (Inserting values)

→ These methods are used to add a new value to a list.

→ The append() method adds an argument at the end of the list

→ The insert() method adds a value at any index in the list that we want.

⇒ It is represented with 2 arguments inside the (). The 1st is the index of the new value & the 2nd is the value itself that is to be inserted.

eg:-
```
# for append():-
>>> spam = ['cat', 'dog']
>>> spam.append('bat')
>>> spam
['cat', 'dog', 'bat']

# for insert():-
>>> spam = ['cat', 'dog']
>>> spam.insert(1, 'bat')
>>> spam
['cat', 'bat', 'dog'].
```

③ remove() - (Removing values)

→ used to remove values from a list.
→ if we try to delete a value that doesn't exist in the list, then a ValueError occurs.
→ if there are multiples/duplicates of a value then only the value which appears 1st is deleted.
→ The 'del' statement is good when we know the index of the value to be deleted.
→ when we use remove(), we just have to mention the value itself and don't need to know its index.

eg:-
```
>>> spam = ['cat', 'dog', 'rat']
>>> spam.remove('cat')
>>> spam
['dog', 'rat']
```

④ sort() - (sorting values)

→ List with numbers or strings can be sorted with this method.
→ Numbers are sorted in ascending order.
→ Strings are sorted alphabetically.
(but in ASCIIbetical order (upper case))

eg:-
```
>>> spam = [-1, 6, 5, 29, -3, -4]
>>> spam.sort()
>>> spam
[-4, -3, -1, 5, 6, 29].

>>> spam = ['cats', 'dogs', 'ants', 'zebra']
>>> spam.sort()
>>> spam
['ants', 'cats', 'dogs', 'zebra'].
```

→ we can also sort the values in reverse order by passing True with reverse keyword :-

```
>>> spam = ['cat', 'dog', 'rat']
>>> spam.sort(reverse = True)
>>> spam
('rat', 'dog', 'cat')
```

→ Things to note about sort()

- we cannot return a value using sort. we can only sort the list in place.
- we cannot sort lists having both number and string values together.
- sort() uses ASCIIbetical order i.e, uppercase alphabets are sorted first & then the lowercase.

eg:
```
>>> spam = ['a', 'b', 'A', 'Z']
>>> spam.sort()
['A', 'Z', 'a', 'b'].
```

* Mutable & Immutable data types

Mutable → can be modified / changed

Immutable → cannot be modified / changed

→ Strings are immutable.

→ lists are mutable because they can be modified - added, removed, changed.

→ A Tuple is a immutable version of a list. It is similar to lists but it is denoted in ( ) instead of [ ] and values inside tuples cannot be changed / modified.

* References

→ We know that variables can store strings as well as integer values.

→ We can differentiate between two variables on the basis of their values

eg:-
```
>>> spam = 100
>>> cheese = spam
>>> spam = 42
>>> spam
42
>>> cheese
100.
```

→ Here we see that even if we equate cheese to spam, the values of both variables remain different

→ But, this is <u>not</u> the case in <u>lists</u>.

→ In lists, values are not stored as actual, ~~it~~ ~~is~~ ~~to~~ their references are.

i.e:-

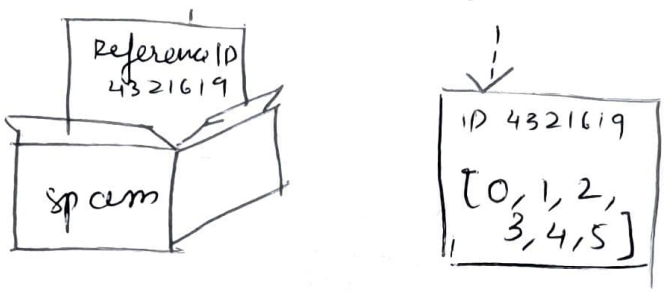


Fig:- [0, 1, 2, 3, 4, 5] is a list that is referenced by spam

→ A reference to actual values are stored in lists.

→ This property of lists allows us to modify lists.

→ eg:-

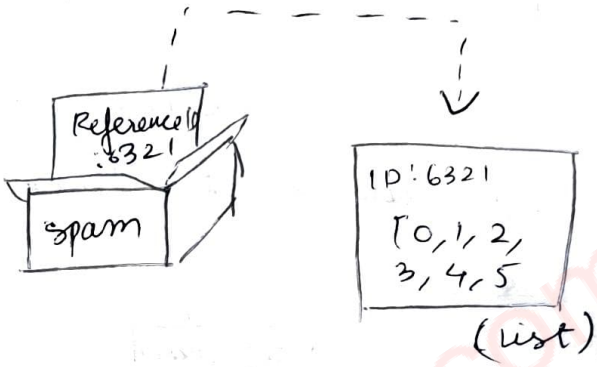
```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = spam
>>> cheese[1] = 'Hello'
>>> spam
[0, 'Hello', 2, 3, 4, 5]
>>> cheese
[0, 'Hello', 2, 3, 4, 5].
```

→ i.e, values stored in cheese & ~~list~~ spam refer to the same list :—

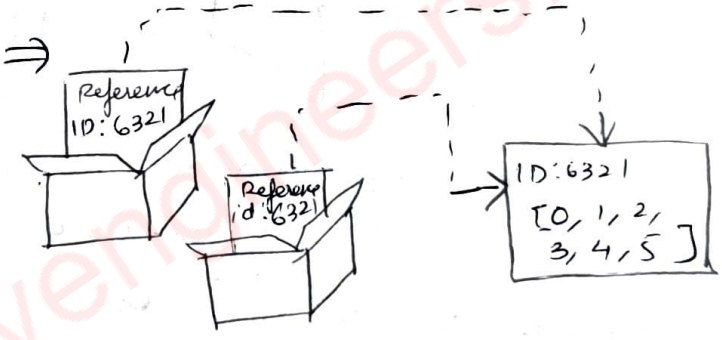
List variables don't actually contain lists, they contain references to the lists.

→ The diagrammatic representation for the above program :-

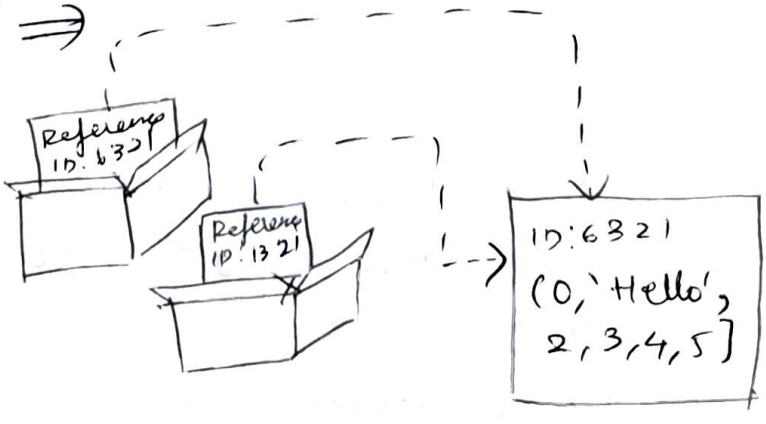
(i) spam = [0, 1, 2, 3, 4, 5] ⇒



(ii) cheese = spam ⇒



(iii) cheese[1] = 'Hello' ⇒



* The copy() and deepcopy() Function.

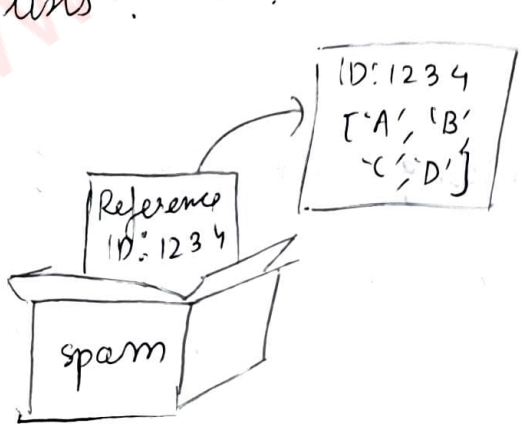
→ If a function modifies a list or dictionary, we may not want these changes in the original list of dictionary.

→ For this, python has a module named copy that provides copy() and deepcopy() functions.

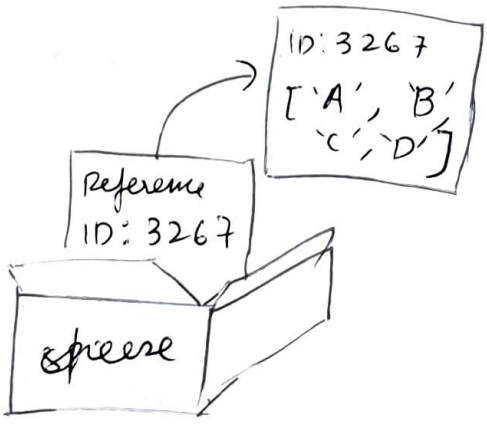
→ copy() ⇒ (copy.copy())

⇒ can be used to make a duplicate copy of a mutable value like list or dictionary. It copies the actual list and not just its reference.

⇒ By doing so, the reference id numbers will no longer be same for the two variables because now they will refer to different lists. i.e:-



spam



cheese = copy.copy(spam)

⇒ Now we can make changes independently.

→ If a list contains ~~deepcopy~~ other inner lists, then we use copy.deepcopy() function instead of copy.copy().

→ The deepcopy() function will copy inner lists as well.

~~2] METHOD~~

2] - DICTIONARIES

→ collection of many values.

→ Indexes for dictionaries can use many different data types.

→ Indexes are called keys and are associated with values to be collectively called as a key-value pair enclosed in { }

→ {'key': 'value'}.

eg:- >>> mycat = {'color': black', 'size': 'fat'}

→ we can access dictionary values using keys:

```
>>> mycat ['size']
fat
>>> 'my cat has' + mycat ['color'] + 'fur'
'My cat has black fur'.
```

* Difference b/w lists & Dictionaries.

Dictionaries	Lists
① items are unordered	① items are ordered
② order of items does not matter in determining if two dictionaries are same.	② Order of items matters in determining if two lists are same.
③ Trying to access a key that does not exist in dictionary will cause a KeyError	③ Trying to access a list value that doesn't exist will cause an IndexError

* Methods :-

→ ① keys()
② values()
③ items()
} → return keys, values & items.

④ get()
⑤ setdefault()
} → check whether a key exists in a dictionary.

→ The for loop is used to iterate over keys, values and key-value pairs using keys(), values() and items() respectively as follows:-

① keys()

→ returns only keys :-

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k in spam.keys():
        print(k)
```

o/p ⇒ color
age

② values()

→ returns only values :-

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
        print(v)
```

o/p ⇒ red
42

③ items()

→ returns keys and their respective values :-

```
>>> spam = {'color': 'red', 'age': 42}
>>> for i in spam.items()
        print(i)
```

o/p ⇒ ('color', 'red')
('age', '42')

④ get()

→ It takes 2 arguments:

- key of the value to retrieve
- fallback value if that key doesn't exist

eg:- >>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I brought ' + str(picnicItems.get('cups', 0)) + 'cups')

o/p ⇒ I brought 2 cups.

>>> 'I brought' + str(picnicItems.get('eggs', 0)) + 'eggs')

o/p ⇒ I brought 0 eggs.

⑤ setdefault()

→ This is used to set a value in a dictionary for a certain key only if that key does not already have a value.

→ It contains

- 1st argument is the key to check for
- 2nd is the value to set if the key does not exist. If the key exists, the setdefault() method doesn't act and returns the original existing value.

```
>>> spam = {'name': 'pooka', 'age': 5}
>>> spam.setdefault('color': 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'pooka'}
>>> spam.setdefault('color': 'white')
>>> spam
{'color': 'black', 'age': 5, 'name': 'pooka'}
```

(remains same because this time color existed)

* Program that counts no. of occurences of each letter in a string :-

```
message = 'Hello world!'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

print(count).
```

→ this programs loops over each character in the message and counts how often each appears.

→ the setdefault() call ensures that the key is in the count dictionary and does not throw a KeyError when count[character] = count[character]+1 is executed.

output:-

{' ': 1, '!': 1, '#': 1, 'd': 1, 'e': 1, 'l': 3, 'o': 2, 'r': 2, 'w': 1}

→ To print the same thing in a cleaner way, we use pretty printing:- pprint() from the pprint module().

The only change will be to import the pprint module and mention pprint.pprint(count) in the last line.

→ The o/p will be in a vertical order:-

{' ': 1,
'!': 1,
'H': 1,
'd': 1,
'e': 1,
'l': 3,
'o': 2,
'r': 2,
'w': 1}

* Nested dictionaries and Lists

→ we can have programs that contain dictionaries and lists inside other dictionaries and lists.

→ eg:- To see who is bringing what and what is the total:-

```
allGuests = { 'Alice' : { 'apples : 5'
                          'pretzels : 12' },
              'Bob' : { 'hams' : 3
                        'apples : 2 },
              'Carol' : { 'cups' : 3
                          'apple pie' : 1 }
            }

def totalBrought (guests , item)
    numBrought = 0
    for k, v in guests.item():
        numBrought = numBrought + V.get (item,
    return numBrought .

print ('Number of total things:')
print ('Apples' + str(totalBrought (allGuests , 'apples'))
print ('pretzels' + str (totalBrought (allGuests , 'pretzels'))
```

O/p :- Number of total things

Apples 7

pretzels 12.

3]- Strings :-

→ begin and end with single quote.

→ Escape characters are used such as (\) to use ' & " " together.

eg :- >>> spam = 'Say hi to Bob\'s mother.'

>>> spam

'Say hi to Bob's mother'

→ Different escape characters are -

escape character	prints as.
\'	⇒ single quote.
\"	⇒ double quote
\t	⇒ Tab
\n	⇒ new line.
\\	⇒ Backslash.

→ Indexing in strings is same as lists.

H	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

→ in & not in operators also work the same as lists

>>> 'Hello' in 'Hi'

False.

* String Methods :-

① upper( ) ⇒ changes to uppercase

② lower( ) → changes to lower case.

③ isupper( ) ⇒ boolean value (T/F) ~~for~~ if the string is in uppercase

④ islower( ) ⇒ boolean value (T/F) if the string is in lowercase.

⑤ is X ⇒ used to return boolean True or False according to the beginning i.e. -

(i) isalpha( ) ⇒ returns true if string consists only letters and is not blank.

(ii) isalnum ~~isdecm~~( ) ⇒ returns true if string consists only numbers and letters and not blank.

(iii) isdecimal( ) ⇒ returns true if string consists only numbers and is not blank.

(iv) isspace( ) ⇒ only spaces, tabs, new line & not blank.

(v) istitle( ) ⇒ consists words that begin with only uppercase letter followed by only lower case letters.

(6) startswith( ) ⇒ True if string starts with a given condition

(7) endswith( ) ⇒ True if string ends with a given condition.

(8) join( ) ⇒ join 2 strings together.

(9) split( ) ⇒ returns list of strings.

(10) rjust( ) ⇒ right justify.

(11) ljust( ) ⇒ left justify.

(12) center( ) ⇒ center justify.

(13) strip( ) ⇒ returns new string without any whitespace characters at the beginning or end.

(14) rstrip( ) ⇒ returns string without any whitespace characters from ~~left~~ right

(15) lstrip( ) ⇒ without whitespace characters from right.

[Go through examples for these methods from text book]