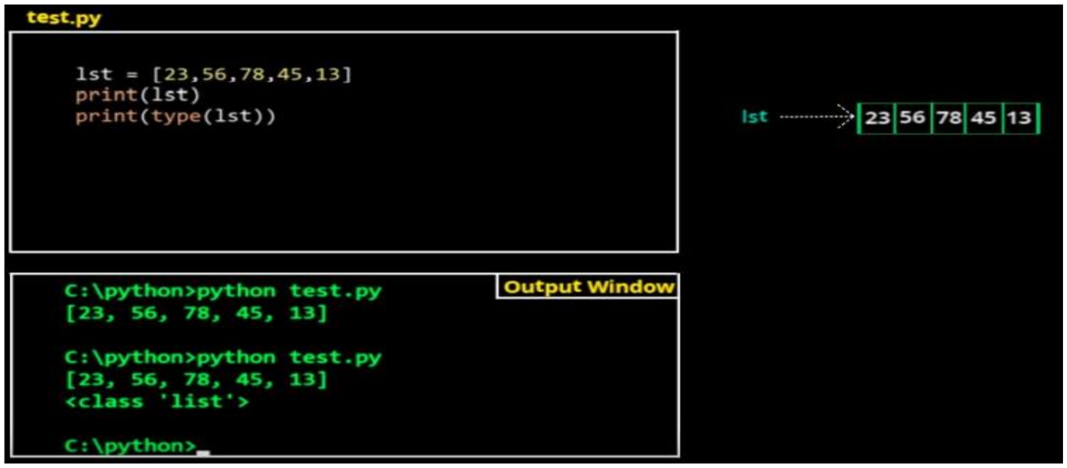
Built In Data Structure/ Advanced Data Type

List :- A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.



```
lst = [23,56,78,45,13]
print(lst)
print(type(lst))
```

```
C:\python>python test.py
[23, 56, 78, 45, 13]

C:\python>python test.py
[23, 56, 78, 45, 13]
<class 'list'>

C:\python>_
```

Few Points regarding list

1. We can access single elements from list by using positive indices or negative.

Positive Indexing

- Starts from **0** for the first element.
- Increases by **1** for each subsequent element.

Negative Indexing

- Starts from **-1** for the last element.
- Decreases as you move left.

[Note: Indexing is used to access list elements. Each list element has an index corresponding to its position in the list. The first item on the list has an index of 0, the second has an index of 1, and so on. The list's final element has an index that is one less than the list's length.]

Information Classification: General

Accessing List Elements

```
# creating empty list
l=[]
l=list()

l=list(range(1,10,2))# starts from 1 till 9 with incremented by 2
#range(start:end:steps)# l=[1,3, 5, 7, 9]
print(l)
list1=l[0:10:2] # kind of copy
# list1=[0, 2, 4, 6, 8]
print(list1)

# convert string to list
s='rajeev shivansh'
s=s.split()
print(s) # ['rajeev', 'shivansh']

# Accessing list element
l=[10, 20, 30, 'Veer', True, 30.7, 20+50j]
print(l[0])  # 10
print(l[3])  # Veer
print(l[-1]) # last item - (20+50j)
print(l[-2]) # 30.7
```

List slicing is used to extract a portion (sublist) of a list using the syntax:

list[start:stop:step]

- start: The index where the slice starts (inclusive). Default is 0.
- stop: The index where the slice ends (exclusive). Default is last item.
- step: The interval between elements (default is 1).

```
# Slicing in list
# by default step value is always 1 , start value-0, end- end of list
l2=[10, 20, 'preet', 20, 70.5, True, 30+70j]
print(l2[::]) # op-[10, 20, 'preet', 20, 70.5, True, (30+70j)]
print(l2[3::3]) # op- [20, (30+70j)]
print(l2[1:9:3]) # op- [20, 70.5]
print(l2[-1:-4:-2]) # op- [(30+70j), 70.5]
print(l2[-2::]) # [True, (30+70j)]
```

A **nested list** is a list inside another list. It allows creating multi-dimensional lists (e.g., matrices).

```
# Nested list -list inside list
lst2=[20, 30, 50, [77, 20, 60.8, ['free', True, 80,[10]]]]
# Accessing item from nested list
# indexing starts from 0
# again for sublist - indexing starts from 0
# first traverse in outer list to that index then again idexing starts from 0
print(len(lst2)) # It will give the number item present in list
# sublist will treat as 1 item for outer list
# here 0 index-20, 1 index-20, 2 index-50, 3 index-[77, 20, 60.8,['free'. True,
80, [10]]]
# if u want to fetch list data first traverse till that index
print(lst2[3]) # op-[77, 20, 60.8,['free'. True, 80, [10]]] Now traverse in this
list
# again indexing starts from 0 index-77, 1st index-20, 2nd index- 60.8, 3rd
index-['free', True, 80,[10]]
print(lst2[3][3]) # op- ['free', True, 80, [10]] Now traverse in this list
# indexing - 0 index- 'free', 1 index- True, 2 index- 80, 3 index-[10] # this
also sublist which has only one value 0 index-10
print(lst2[3][3][2]) # op- 80
print(lst2[3][3][3][0]) # 10

# Accessing list item of nested list
# No. of sub lists = no. of loops & loop-1 = if else condition
for item in lst2:
    if type(item) == list:  # Check if the item is a list
        for sub_item in item:
            if type(sub_item) == list:  # Check if sub_item is a list
                for sub_sub_item in sub_item:
                    if type(sub_sub_item) == list:  # Check if sub_sub_item is a
list
                        for sub_sub_sub_item in sub_sub_item:
                            print(sub_sub_sub_item, end=" ")
                    else:
                        print(sub_sub_item, end=" ")
            else:
                print(sub_item, end=" ")
    else:
        print(item, end= " ")
```

2. Lists are mutable, which means we can append elements and remove whenever needed. We can also change the data of existing list.

```
# Addind data to list - append, extend
```

```python
# append can add only one element at the end of list
l=[10, 20, 80, 90, 80, 90]
l.append(8)
print(l)
l.append([20, 30]) # u can pass multiple data as list but it will store at the end
as list
# l.append(20, 30, 40) # u will get error bcoz it accepts 1 argument

# extend can add one list at the end of list
list1=[30, 70]
l.extend(list1)
print(l)
l.extend([40, 80, 90])
print(l)

# insert(position, value)
list1.insert(1, "hamper")
print(list1)
print(list1.index("hamper")) # give the index position of value # return the first
occurence of value
```

3. Removing elements from the list

```python
# Removing data from list - remove(value), pop()
l=[60, 90, 80, 70.6, 'yet']
l.pop() # Remove last item from list
print(l)
# l.pop(index) # Remove item from specied position
l.pop(2)

# remove(value) # we are passing one value if that value is not presnt will give
you error else it will remove that value
# l.remove(9) # error if value not present
print(l)
l.remove(60) # value is present in list, it will remove that element
# if duplicate value is present it will remove the first occurence of value
print(l)
l.clear() # op- []  It removes elements from the list
l2=[9, 8, 7, 65, 98, 10]
del l2[3:] # delete all elements from index 3
print(l2)
```

```
```

Functions of List-

```
# Functions of list-

lst_fun=[10, 20, 30, 80, 30, 50]
# count(value) - Returns the count of occurrences of x.
print(lst_fun.count(30))

# Sorting and Reversing
lst_fun.sort() # Sort element of list-Ascending to Descending
print(lst_fun)
lst_fun.reverse() # Descending to Ascending
print(lst_fun)
lst_fun.sort(reverse=True)
print(lst_fun)

# Copying list
lst=[10, 20, 30, 80]
lst2=lst.copy() # Returns a shallow copy of the list.
print(lst2)

# len(lst)  Returns the length (number of elements).    len(lst)
# max(lst)  Returns the maximum element.    max(lst)
# min(lst)  Returns the minimum element.    min(lst)
# sum(lst)  Returns the sum of all elements.    sum(lst)
```

Operations on List -

```
# Operations on list-

```

Information Classification: General

```
# Concatenation/ Merging - Adding one list to another list like append function
lst1=[10, 20, 30, 40]
lst2=[10]
lst3=lst1+lst2
print(lst3)

# Muliplication / Replicating
lst3=lst1*3
print(lst3)

# Memebership Operator
print(20 in lst) # return boolean value - True or False
print(lst1 is lst2) # comparing memory location
```

Zip Function - The zip() function in Python **combines** multiple lists, tuples, or other iterables **element by element**, creating pairs (or tuples) of corresponding elements.

```
# zip function - both list should contain same elements - Pair up tupe
lst1=[5, 12, 15, 7]
lst2=[14,5,7,90]
print(list(zip(lst1,lst2)))
print(tuple(zip(lst1, lst2)))

lst3=[]

for i,j in zip(lst1,lst2):
    lst3.append(i+j)

print(lst3)

lst1=['A', 'app', '', 'da', 'kee', 't', 'doc', 'a']
lst2=['n', 'le', 'a', 'y', 'ps', 'he', 'tor', 'way']
lst3=[]
for i, j in zip(lst1, lst2):
    lst3.append(i+j)
```

The any() and all() functions are used to check conditions in iterables like lists and tuples.
The any() function **returns True if at least one** element in the iterable is True, otherwise, it returns False.

Information Classification: General

The all() function **returns True only if all** elements in the iterable are True. If any element is False, it returns False.

```
# any and all function
lst=[10, 20, 30, -40, -50]
# using function
print(all([i>0 for i in lst]))
print(any([i>0 for i in lst]))
def any(lst):
    for i in lst:
        if i > 0:
            return True
    return False

def all(lst):
    for i in lst:
        if i<0:
            return False

    return True
print(any([-10, -20, -30, -40, -40]))
print(all([10, 20, 30, 40, -50]))

marks=[45, 75, 30, 80, 90]
print(all([x>35 for x in marks ]))
print(any([x>35 for x in marks ]))
```

List Comprehension - It is a concise way to create lists in Python. It allows you to generate a new list by applying an expression to each item in an iterable (like a list, tuple, or range) in a single line of code.

[expression for item in iterable if condition]

- **expression**: The operation to perform on each item.
- **item**: Variable representing each element in the iterable.
- **iterable**: The data structure being iterated over.
- **condition** (optional): A filter to include only certain items.

```
# List Comprehension

# [expression for item in iterable if condition]
# expression: The operation to perform on each item.
# item: Variable representing each element in the iterable.
```

Information Classification: General

```
# iterable: The data structure being iterated over.
# condition (optional): A filter to include only certain items.

lst4=[x for x in range(1, 10)]
print(lst4)
print([x*x for x in range(1, 11)])
evens = [x for x in range(10) if x % 2 == 0]
print(evens)
pairs = [(x, y) for x in range(3) for y in range(3)]
print(pairs)  # Op: [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2,
1), (2, 2)]
numbers = [x if x % 2 == 0 else "Odd" for x in range(5)]
print(numbers)
```

Tuple : - A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

Like lists in tuple also we can access single elements with both positive and negative indices.

```
# Empty tuple
t5=10,20, 30, 40
print(t5)
t=tuple()
t=()
print(t)
t=10,
print(t)
t2=10
print(t2)

# converting into tuple
t1=tuple(t)
print(t1[::])
print(len(t1))
print(max(t5))

# tuple packing
a=10
b=20
c=30
print(a, b ,c)
```

```
tup=a,b,c
print(tup)

# tuple unpacking
c, d, e=tup
# variables equal to no. of values in tuple
print(c, d ,e)

t=tuple(x**2 for x in range(1, 20, 2))
print(t)
```

Tuples are immutable, meaning they cannot be modified.

They support **indexing, slicing, concatenation, and repetition**.

Functions like **min(), max(), sum(), count(), index()** work on tuples.

Tuples can store **nested tuples and different data types**.

Use tuples when data should not be changed, e.g., coordinates, database record.

Information Classification: General