Data structures or Collections or Iterables

What is collection in python?

Collection type represents more than one value or object.
Collections allows to group individual objects into one object.

1. Perform aggregate operations
2. By grouping data, we can transfer or send data from one place to another place
3. It allows referring data with one name (OR) does not require create multiple variables to store more than one value.

Python collection types are classified into 3 categories

1. Sequences
2. Sets
3. Mapping

This classification is done based on the way how data is organized.

1. Sequences
    a. List
    b. Tuple
    c. String
    d. Range
    e. Bytes
    f. Bytearray
2. Sets
    a. Set
    b. Frozenset
3. Mapping
    a. Dict

Note: Collections are dynamic in size.
Collections are heterogeneous (Which allows various types of objects/data)

Sequences

- Sequences are ordered collections.
- In sequences data is organized in sequential order.
- In sequences insertion order is preserved.
- Sequences are index based collection. This index is used as subscript for reading and writing data.

List

- List is a mutable sequence data type
- After Crating list changes can be done.
- List can be homogeneous or heterogeneous.
- Each value of the list is called element/item.
- List is an index based collection/sequence.
- Lists are mutable sequences, typically used to store collections of homogeneous items

Application development list is used to group individual objects where insertion ordered is preserved and reading and writing is done sequentially or randomly.

"list" class or data type is used for representing or creating list object.

How to create list?

Lists may be constructed in several ways:

- Using a pair of square brackets to denote the empty list: []
- Using square brackets, separating items with commas: [a], [a, b, c]
- Using a list comprehension: [x for x in iterable]
- Using the type constructor or function: list() or list(iterable)

Using a pair of square brackets to denote the empty list: []

```
>>> a=[]
```

```
>>> print(a,type(a))
[] <class 'list'>
```

Using square brackets, separating items with commas: [a], [a, b, c]

```
>>> b=[10]
>>> print(b,type(b))
[10] <class 'list'>
>>> c=[10,20,30,40,50]
>>> print(c,type(c))
[10, 20, 30, 40, 50] <class 'list'>
>>> stud=[101,"naresh","python",5000]
>>> print(stud,type(stud))
[101, 'naresh', 'python', 5000] <class 'list'>
```

Using the type constructor or function: list() or list(iterable)

```
>>> a=int()
>>> print(a,type(a))
0 <class 'int'>
>>> b=float()
>>> print(b,type(b))
0.0 <class 'float'>
>>> c=list()
>>> print(c,type(c))
[] <class 'list'>
>>> d=int("25")
>>> print(d,type(d))
25 <class 'int'>
>>> e=list(range(1,6))
>>> print(e,type(e))
[1, 2, 3, 4, 5] <class 'list'>
>>> f=list(range(10,110,10))
>>> print(f,type(f))
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100] <class 'list'>
>>> g=list("PYTHON")
>>> print(g,type(g))
```

```
['P', 'Y', 'T', 'H', 'O', 'N'] <class 'list'>
>>> x=10
>>> y=20
>>> z=30
>>> list1=[x,y,z]
>>> print(list1)
[10, 20, 30]
>>> list2=[input(),input(),input()]
10
20
30
>>> print(list2)
['10', '20', '30']
```

How to read content of list or sequence?

Content of list can be read in different ways

1. Indexing
2. Slicing
3. For loop
4. Iterator
5. Enumerate

Indexing

What is index?

Index an integer value.

Every element or item within sequence is identified with a unique number called index

Index is element position.

Using index a programmer can read elements sequentially or randomly.

This index used as subscript for reading and writing.

The index can be +ve or –ve

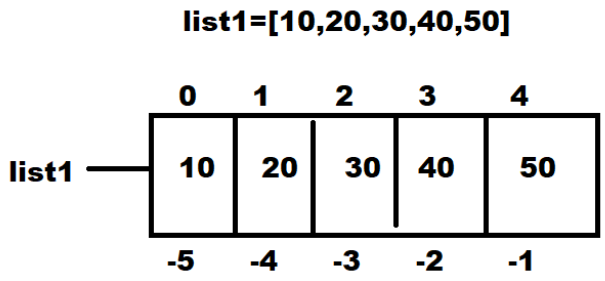
+ve index starts at 0, when elements read in forward direction (L-R)

-ve index starts at -1, when elements read in backward direction (R-L)

Syntax: sequence-name[index]

Given index must within range
If given index is not within range, python raises IndexError

Note: index is used for reading one value.



list1[0] --> 10
list1[1] --> 20
list1[2] --> 30
list1[3] --> 40
list1[4] --> 50
list1[5] --> IndexError
list1[-1] --> 50
list1[-2] --> 40
list1[-3] --> 30
list1[-4] --> 20
list1[-5] --> 10
list1[-6] --> IndexError

len(): It is a predefined function in python, this function returns count of values/length of list.

Example:

```
list1=[10,20,30,40,50]
print(list1[0],list1[1],list1[2],list1[3],list1[4])
print(list1[-1],list1[-2],list1[-3],list1[-4],list1[-5])

list2=list(range(10,310,10))
print(list2)

print("Forward Direction")
for i in range(30): # 0 1 2 3 4 5 6 7 8 9 ... 29
    print(list2[i],end=' ')

print()
```

```
print("Backward Direction")
for i in range(-1,-31,-1):
    print(list2[i],end=' ')

print()
print(len(list2))
print("Forward Direction")
for i in range(len(list2)):
    print(list2[i],end=' ')

print()
print("Backward Direction")
for i in range(-1,-(len(list2)+1),-1):
    print(list2[i],end=' ')
```

Output

```
10 20 30 40 50
50 40 30 20 10
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300]
Forward Direction
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300
Backward Direction
300 290 280 270 260 250 240 230 220 210 200 190 180 170 160 150 140 130 120 110 100 90 80 70 60 50 40 30 20 10
30
Forward Direction
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300
Backward Direction
300 290 280 270 260 250 240 230 220 210 200 190 180 170 160 150 140 130 120 110 100 90 80 70 60 50 40 30 20 10
```

Example:

```
student_names=['naresh','suresh','ramesh','kishore','rajesh','kiran','raman']
```

```
index=int(input("Enter Student Position/Index to Read :"))

if (index>=0 and index<len(student_names) ) or (index>=-
len(student_names) and index<=-1) :
    name=student_names[index]
    print(f'StudentName {name}')
else:
    print("Invalid Index")
```

Output

```
Enter Student Position/Index to Read :-1
StudentName raman
>>>
Enter Student Position/Index to Read :-4
StudentName kishore
>>>
Enter Student Position/Index to Read :-10
Invalid Index
```

Example:

```
name="NARESH"
marks=[60,70,80]
total=0

for i in range(len(marks)):
    total=total+marks[i]

avg=total/len(marks)

print(f'Name {name}')
print(f'Marks {marks}')
print(f'Total Marks {total}')
print(f'Avg Marks {avg:.2f}')
```

Output

```
Name NARESH
```

Marks [60, 70, 80]
Total Marks 210
Avg Marks 70.00

Example of count of even and odd:

```
list1=[1,5,9,2,6,7,8,12,17,23,65,87,45,23,89,87,69,54,34,23,13,89]

even_count=0
odd_count=0

for i in range(len(list1)):
    if list1[i]%2==0:
        even_count=even_count+1
    else:
        odd_count=odd_count+1

print(f'List is {list1}')
print(f'Even Count {even_count}')
print(f'Odd Count {odd_count}')
```

Output

```
List is [1, 5, 9, 2, 6, 7, 8, 12, 17, 23, 65, 87, 45, 23, 89, 87, 69, 54, 34, 23,
13, 89]
Even Count 6
Odd Count 16
```

Example:

```
# Program to count +ve,-ve,zeros
list1=[1,2,3,-1,4,-4,-6,0,0,-12,-25,-9,3,6,7,0,0,12,56,-34,0,0]

pcount=0
ncount=0
zcount=0

for i in range(len(list1)):
    if list1[i]>0:
        pcount+=1
```

```
    elif list1[i]<0:
        ncount+=1
    else:
        zcount+=1

print(f'list is {list1}')
print(f'+ve Count {pcount}')
print(f'-ve Count {ncount}')
print(f'zero count {zcount}')
```

Output

```
list is [1, 2, 3, -1, 4, -4, -6, 0, 0, -12, -25, -9, 3, 6, 7, 0, 0, 12, 56, -34, 0, 0]
+ve Count 9
-ve Count 7
zero count 6
```

Example:

```
# Program to count prime numbers
list1=[5,2,3,-1,4,-4,-6,0,0,-12,-25,-9,3,6,7,0,0,12,56,-34,0,0]

pcount=0

for i in range(len(list1)): # 0 1 2 3 4 5 6 7
    num=list1[i]
    c=0
    for j in range(1,num+1):
        if num%j==0:
            c=c+1
    if c==2:
        print(num)
```

Output

```
5
2
3
3
```

Slicing
Slicing allows reading more than one value from sequence.
This slicing is done using two approaches

1. Slice operator
2. Slice object

Slice Operator
Slice operator is used for generating multiple indexes to read multiple values. Slice operator internally uses range for generating indexes.

Syntax: sequence-name[start:stop:step]

Slice operator read all the values and return inside list.

Example:

```
>>> list1=[10,20,30,40,50,60,70,80,90,100]
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list2=list1[0:10:1]
>>> print(list2)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list3=list1[0:10:2]
>>> print(list3)
[10, 30, 50, 70, 90]
>>> list4=list1[0:10:3]
>>> print(list4)
[10, 40, 70, 100]
>>> print(list4)
[10, 40, 70, 100]
```

Syntax-1: sequence-name[::]
Default start=0, stop=len(sequence),step=+1

```
>>> list1=[10,20,30,40,50,60,70,80,90,100]
```

```
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list2=list1[::]
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Syntax-2: sequence-name[::step]
Start,stop values are taken based on step value

If step is +ve, start=0,stop=len(sequence) → Forward
If step is –ve, start=-1,stop=-(len(sequence)+1) → Backward

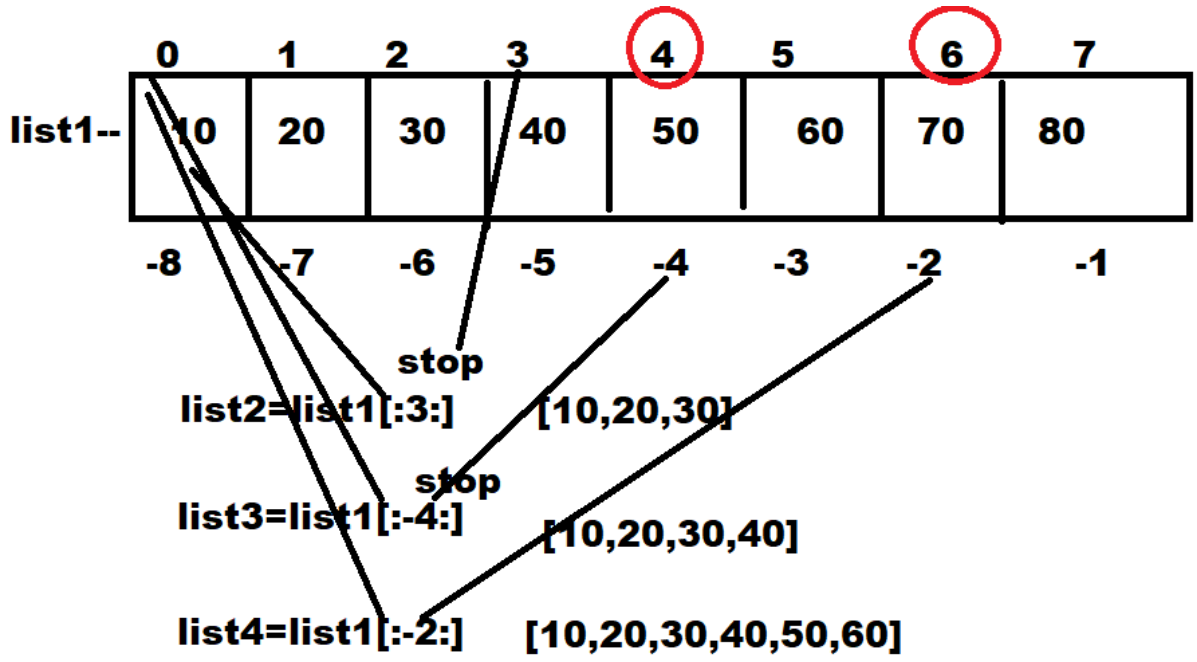
```
>>> list1=list(range(10,60,10))
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list2=list1[::1]
>>> print(list2)
[10, 20, 30, 40, 50]
>>> list3=list1[::2]
>>> print(list3)
[10, 30, 50]
>>> print(list3)
[10, 30, 50]
>>> list4=list1[::-1]
>>> print(list4)
[50, 40, 30, 20, 10]
>>> list5=list1[::-2]
>>> print(list5)
[50, 30, 10]
```

Syntax-3: sequence-name[start::]
In this syntax default stop=len(sequence),step=+1

```
list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list2=list1[3:]
```

```
>>> print(list2)
[40, 50]
>>> list3=list1[-3:]
>>> print(list3)
[30, 40, 50]
```

Syntax4: sequence-name[:stop:]
Default start=0,step=+1




```
>>> list1=list(range(10,210,10))
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170,
180, 190, 200]
>>> list2=list1[:10:]
>>> print(list2)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list3=list1[:-10:]
>>> print(list3)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Syntax-5: sequence-name[start:stop]

Default step is +1

```
>>> list1=list(range(10,110,10))
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list2=list1[2:8]
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> print(list2)
[30, 40, 50, 60, 70, 80]
>>> list3=list1[2:-2]
>>> print(list3)
[30, 40, 50, 60, 70, 80]
>>> list4=list1[-2:2]
>>> print(list4)
[]
>>> list5=list1[-2:2:-1]
>>> print(list5)
[90, 80, 70, 60, 50, 40]
```

What is difference between indexing and slicing?

Indexing is used for reading one value	Slicing is used for reading more than one value

Slice object

Slice object is created using slice() function.

Syntax: slice(stop)
Syntax: slice(start,stop,step)

Slice object is reusable, once slice object is created; it can be used with one or more than one sequence.

Example:

```
slice1=slice(2)
slice2=slice(-1,-4,-1)
```

```
list1=list(range(10,110,10))
list2=list(range(10,60,10))

print(list1)
print(list2)

list3=list1[slice1]
list4=list2[slice1]

print(list3)
print(list4)

list5=list1[slice2]
list6=list2[slice2]
print(list5)
print(list6)
```

Output

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
[10, 20, 30, 40, 50]
[10, 20]
[10, 20]
[100, 90, 80]
[50, 40, 30]
```

for loop

for is an iterator , which iterate values from iterables
iterating is nothing reading each value from iterable/collection.

Syntax:

```
for variable-name in iterable:
        statement-1
        statement-2
```

Example:

```
list1=list(range(1,11))
print(list1)
```

```
for x in list1:
    print(x)
```

Output

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
1
2
3
4
5
6
7
8
9
10
```

Example

```
list1=[3,9,12,56,23,98,45,12,6,8,9,3,5,12,15,76,34,12,67,87,98]

# count values which are >50
count=0
for value in list1:
    if value>50:
        print(value,end=' ')
        count=count+1

print(f'\nCount is {count}')

# count values in given range
start=int(input("Enter Start Value "))
end=int(input("Enter End Value "))
count=0
for value in list1:
    if value>=start and value<=end:
        print(value,end=' ')
        count=count+1
```

```
print(f'\nCount is {count}')

# count even numbers and odd numbers in list

ecount=0
ocount=0
for value in list1:
    if value%2==0:
        ecount+=1
    else:
        ocount+=1

print(f'Even Count {ecount}')
print(f'Odd Count {ocount}')
```

Output

```
56 98 76 67 87 98
Count is 6
Enter Start Value 10
Enter End Value 50
12 23 45 12 12 15 34 12
Count is 8
Even Count 11
Odd Count 10
```

Iterator object

Iterator object is used to iterate/read values from iterable(collection)

What is difference between iterable and iterator?

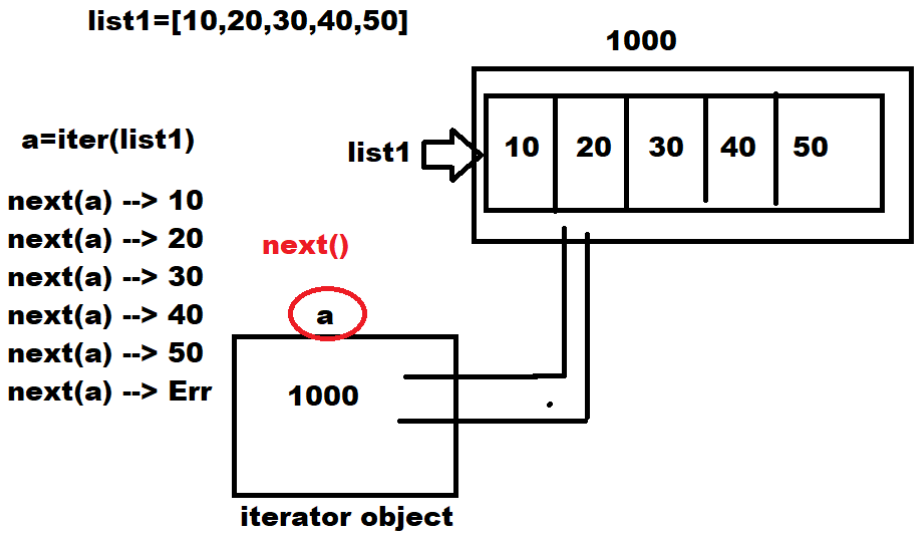
Iterator	**Iterable**
This object is used to read data from iterables. Iterator does not have any data.	Iterable is a collection which contains data

Iter()

It is a predefined function in python, this function returns iterator object.

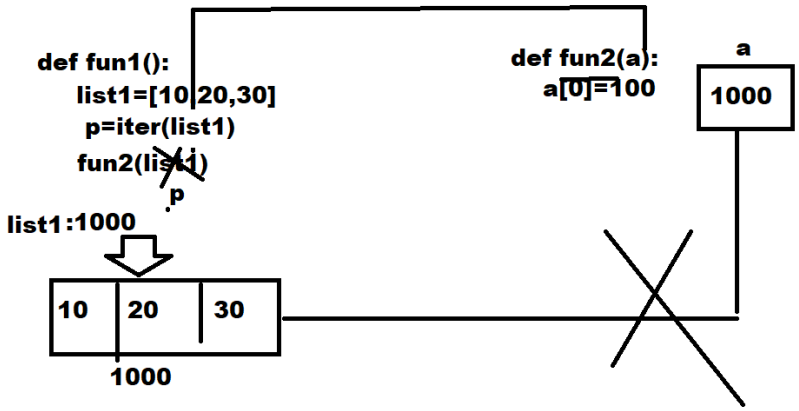
Syntax: iter(iterable)

The job of iterator object is reading but not doing any changes.



In application development iterators are used with,

1. Non index based collections
2. To convert mutable collections to immutable



Example:

```
list1=list(range(10,110,10))
print(list1)
```

```
a=iter(list1)
value1=next(a)
value2=next(a)
value3=next(a)
print(value1,value2,value3)

for value in a:
    print(value)
```

Output

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
10 20 30
40
50
60
70
80
90
100
```

Enumerate object

Enumerate is also iterator object, this returns two values

1. Start/count
2. Value read from iteable

It returns these two values as one tuple.

Syntax: enumerate(iterable,start=0)

```
>>> list1=[10,20,30,40,50]
>>> e=enumerate(list1)
>>> next(e)
(0, 10)
>>> next(e)
(1, 20)
```

```
>>> next(e)
(2, 30)
>>> next(e)
(3, 40)
>>> next(e)
(4, 50)
>>> names=["naresh","suresh","ramesh","kishore"]
>>> e=enumerate(names,start=101)
>>> next(e)
(101, 'naresh')
>>> next(e)
(102, 'suresh')
>>> next(e)
(103, 'ramesh')
>>> next(e)
(104, 'kishore')
>>> sales=[10000,20000,30000,40000,50000]
>>> e=enumerate(sales,start=1995)
>>> next(e)
(1995, 10000)
>>> next(e)
(1996, 20000)
>>> next(e)
(1997, 30000)
>>> next(e)
(1998, 40000)
>>> next(e)
(1999, 50000)
```

Mutable Operations of list

List is a mutable object or data type, after creating list changes can be done. These mutable operations can be done using mutable methods of list.

1. append() : This method is used for adding value at the end of list

Syntax: list-name.append(value)

Example:

```
>>> list1=[]
>>> print(list1)
[]
>>> list1.append(10)
>>> list1.append(20)
>>> list1.append(30)
>>> print(list1)
[10, 20, 30]
>>> list2=[]
>>> list2.append("naresh")
>>> list2.append("suresh")
>>> print(list2)
['naresh', 'suresh']
>>> list3=[]
>>> list3.append(10,20,30)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    list3.append(10,20,30)
TypeError: list.append() takes exactly one argument (3 given)
```

Example:

```
# Write a program to append n integer values into side list

list1=[]
n=int(input("Enter how many values?"))

for i in range(n):
    value=int(input("Enter any value "))
    list1.append(value)

print(list1)
```

Output

```
Enter how many values?3
```

```
Enter any value 100
Enter any value 200
Enter any value 300
[100, 200, 300]
```

Example:

```
# Write a program to read scores of n players and find min,max,total

scores=[]
n=int(input("Enter how many players?"))

for i in range(n):
    score=int(input("Enter Score "))
    scores.append(score)

print(f'Scores are {scores}')

# Calculating Total
total=0
for s in scores:
    total=total+s

print(f'Total Score is {total}')

# Maximum score
max_score=0
for s in scores:
    if s>max_score:
        max_score=s

print(f'Maximum Score is {max_score}')

# Minimum Score
min_score=scores[0]
for s in scores:
    if s<min_score:
        min_score=s
```

```
print(f'Minimum Score is {min_score}')
```

Example:

```
# Write a program to input name, n subject marks and calculate total marks,avg marks,result

name=input("Enter Name ")
n=int(input("Enter how many subjects ?"))

marks=[]
for i in range(n):
    s=int(input("Enter Marks "))
    marks.append(s)

total=0
for s in marks:
    total+=s

avg=total/n
result="pass"
for s in marks:
    if s<40:
        result="fail"
        break

print(f"{name}\t{marks}\t{total}\t{avg:.2f}\t{result}")
```

How to append more than one value?

Appending of more than one value is done using slicing

Syntax: list-name[len(list-name):]=iterable

Example

```
list1=[10,20,30]
```

```
print(f'Before append list is {list1}')
list1[len(list1):]=[40,50,60]
print(f'After append list is {list1}')
list1[len(list1):]=range(70,110,10)
print(f'After append list is {list1}')
list1[len(list1):]="NIT"
print(f'After append list is {list1}')

sales1=[1000,2000,3000,4000]
sales2=[5000,7000,9000,10000,56000]
print(sales1)
sales1[len(sales1):]=sales2
print(sales1)
```

Output

```
Before append list is [10, 20, 30]
After append list is [10, 20, 30, 40, 50, 60]
After append list is [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
After append list is [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 'N', 'I', 'T']
[1000, 2000, 3000, 4000]
[1000, 2000, 3000, 4000, 5000, 7000, 9000, 10000, 56000]
```

Replacing or updating values of list

Replacing content of list is done using

1. indexing
2. slicing

Using index only one value is replaced.
Using slicing more than one value can be replaced

Syntax1: list-name[index]=value
Syntax2: list-name[start:stop:step]=iterable

```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list1[0]=100
```

```
>>> print(list1)
[100, 20, 30, 40, 50]
>>> list1[-1]=900
>>> print(list1)
[100, 20, 30, 40, 900]
```

Example:

```
# Python program to interchange first and last elements in a list

list1=[10,20,30,40,50]
print(f'Before swaping {list1}')

temp=list1[0]
list1[0]=list1[-1]
list1[-1]=temp

print(f'After swaping {list1}')

list1[0],list1[-1]=list1[-1],list1[0]
print(f'After swaping {list1}')
```

Output

```
Before swaping [10, 20, 30, 40, 50]
After swaping [50, 20, 30, 40, 10]
After swaping [10, 20, 30, 40, 50]
```

Example:

'''Given a list in Python and provided the positions of the elements, write a program to swap the two elements in the list.

Examples:

Input : List = [23, 65, 19, 90], pos1 = 1, pos2 = 3
Output : [19, 65, 23, 90]

Input : List = [1, 2, 3, 4, 5], pos1 = 2, pos2 = 5
Output : [1, 5, 3, 4, 2]'''

```
list1=[23, 65, 19, 90]
pos1=1
pos2=3
print(f'Before swaping {list1}')

list1[pos1-1],list1[pos2-1]=list1[pos2-1],list1[pos1-1]
print(f'After swaping {list1}')
```

Example:

```
list1=[]
n=int(input("Enter how many values?"))

for i in range(n):
    value=int(input("Enter value "))
    list1.append(value)

pos1=int(input("Pos1 :"))
pos2=int(input("Pos2 :"))
print(f'Before Swaping {list1}')

list1[pos1-1],list1[pos2-1]=list1[pos2-1],list1[pos1-1]
print(f'After Swaping {list1}')
```

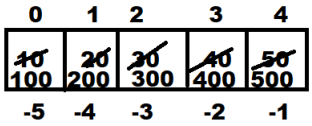
Replacing more than one value

Replacing multiple values are done using slicing

Syntax:

```
list-name[start:stop:step]=iterable
```

list1=[10,20,30,40,50]



list1[2:-2]=[300]

list1[0:2]=[100,200]

list1[-2:]=[400,500]

Example:

```
>>> list1=list(range(10,110,10))
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> list1[:3]=[1,2,3]
>>> print(list1)
[1, 2, 3, 40, 50, 60, 70, 80, 90, 100]
>>> list1[-3:]=[8,9,10]
>>> print(list1)
[1, 2, 3, 40, 50, 60, 70, 8, 9, 10]
>>> list1[3:-3]=[4,5,6,7]
>>> print(list1)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list1[::2]=[10,20,30,40,50]
>>> print(list1)
[10, 2, 20, 4, 30, 6, 40, 8, 50, 10]
>>> list1[::-2]=[1,2,3,4,5]
>>> print(list1)
[10, 5, 20, 4, 30, 3, 40, 2, 50, 1]
```

Example:

```
# Write a program sort the elements of list in ascending order

n=int(input("Enter how many values?"))
list1=[]

for i in range(n):
```

```
    value=int(input("Enter any value"))
    list1.append(value)

print(f'Before Sorting {list1}')

for i in range(n):
    for j in range(n-1):
        if list1[j]<list1[j+1]:
            list1[j],list1[j+1]=list1[j+1],list1[j]


print(f'After Sorting {list1}')
```

Output

```
Enter how many values?5
Enter any value4
Enter any value2
Enter any value1
Enter any value5
Enter any value2
Before Sorting [4, 2, 1, 5, 2]
After Sorting [5, 4, 2, 2, 1]
```

sort() : it is a method of list, which sort the elements in ascending or descending order. It is mutable method, this method sort elements in place (same list).

Syntax:

```
sort(key=None,reverse=False)
```

by default sort method of list, sort elements in ascending order
in order to sort elements in descending order, the value of reverse must be True

```
<list-name>.sort(reverse=True)
```

Key is nothing but, the function which is applied to each value of the list before comparing.

<list-name>.sort(key=function)
<list-name>.sort(key=function, reverse=True)

Example:

```
>>> list1=[4,2,5,2,1]
>>> print(list1)
[4, 2, 5, 2, 1]
>>> list1.sort()
>>> print(list1)
[1, 2, 2, 4, 5]
>>> list1.sort(reverse=True)
>>> print(list1)
[5, 4, 2, 2, 1]
>>> list1=["a","b","c","A","B","C"]
>>> print(list1)
['a', 'b', 'c', 'A', 'B', 'C']
>>> list1.sort()
>>> print(list1)
['A', 'B', 'C', 'a', 'b', 'c']
>>> list1.sort(key=str.upper)
>>> print(list1)
['A', 'a', 'B', 'b', 'C', 'c']
>>> list1=["45","41","65","54","87","51","42"]
>>> print(list1)
['45', '41', '65', '54', '87', '51', '42']
>>> list1.sort()
>>> print(list1)
['41', '42', '45', '51', '54', '65', '87']
>>> list1.sort(key=int)
>>> print(list1)
['41', '42', '45', '51', '54', '65', '87']
```

sorted() : This method after sorting elements, it returns sorted elements in new sequence.

Syntax: <sequence-name>=sorted(sequence-name)

```
>>> list1=[4,1,5,3,2]
>>> print(list1)
[4, 1, 5, 3, 2]
>>> list2=sorted(list1)
>>> print(list1)
[4, 1, 5, 3, 2]
>>> print(list2)
[1, 2, 3, 4, 5]
>>> list3=sorted(list1,reverse=True)
>>> print(list3)
[5, 4, 3, 2, 1]
```

Example:

```
# Python program to find second largest number in a list

n=int(input("Enter how many values ?"))
list1=[]


for i in range(n):
    value=int(input("Enter any value "))
    list1.append(value)

print(f'List is {list1}')
list1.sort()
first_max=list1[-1]
c=0
for value in list1:
    if value==first_max:
        c=c+1

print(f'Second Maximum is {list1[-(c+1)]}')
```

Output

```
Enter how many values ?5
Enter any value 10
Enter any value 20
Enter any value 30
Enter any value 40
Enter any value 40
List is [10, 20, 30, 40, 40]
Second Maximum is 30
```

Example:

```
# Write a program to remove duplicate values from list

list1=[10,20,30,10,10,20,20,30,40,10,10,10,20,30]
list2=[]

for value in list1:
    if value not in list2:
        list2.append(value)

print(list1)
print(list2)
```

Example:

```
# Write a program to remove duplicate values from list

list1=[10,20,30,10,10,20,20,30,40,10,10,10,20,30]
list2=list(set(list1))

print(list1)
print(list2)
```

Output

```
[10, 20, 30, 10, 10, 20, 20, 30, 40, 10, 10, 10, 20, 30]
[40, 10, 20, 30]
```

Example:

```
# Python | Program to print duplicates from a list of integers
```

```
list1=[10,20,20,30,40,40,50,60,60,70,80]

# Remove Duplicates
list2=[]
for value in list1:
    if value not in list2:
        list2.append(value)

print(list1)
print(list2)
# Read values from list2
for value in list2:
    c=0
    for x in list1:
        if value==x:
            c=c+1
    if c>1:
        print(value)
```

Output

```
[10, 20, 20, 30, 40, 40, 50, 60, 60, 70, 80]
[10, 20, 30, 40, 50, 60, 70, 80]
20
40
60
```

How to input more than one value in one line?

How to input more than one value in one line?

input() function is used to input one value of type string.
split() this function split string into number of sub string using default separator (space)
map() this function is used apply a function to collection of value exists in collection or iterable

Example:

```
s=input("Enter value ")
```

```
print(s,type(s))
list1=s.split()
print(list1)
list2=list(map(int,list1))
print(list2)
```

Output

```
Enter value 10 20 30 40 50
10 20 30 40 50 <class 'str'>
['10', '20', '30', '40', '50']
[10, 20, 30, 40, 50]
```

https://www.hackerrank.com/challenges/find-second-maximum-number-in-a-list/problem?isFullScreen=false

```
n=int(input())
s=input()
list1=list(map(int,s.split()))
list1.sort()
fmax_value=list1[-1]
c=list1.count(fmax_value)
print(list1[-(c+1)])
```

count() method of list/sequence

This method returns count of given value

Syntax: <list-name>.count(value)

```
>>> list1=[1,2,3,1,1,1,4,4,2,2,2,2,2]
>>> list1.count(2)
6
>>> list1.count(1)
4
>>> list1.count(100)
0
>>> list1.count(10)
0
```

Example:

```
# Write a program to count each value exists within list

list1=[1,2,3,1,2,3,4,5,1,1,2,2,3,4,5,4,5,4,4,4,4,4,5,5,7]

list2=[]
for value in list1:
    if value not in list2:
        list2.append(value)

print(list1)
print(list2)

for value in list2:
    c=list1.count(value)
    print(f'{value}--->{c}')
```

Output

```
[1, 2, 3, 1, 2, 3, 4, 5, 1, 1, 2, 2, 3, 4, 5, 4, 5, 4, 4, 4, 4, 4, 5, 5, 7]
[1, 2, 3, 4, 5, 7]
1--->4
2--->4
3--->3
4--->8
5--->5
7--->1
```

Example:

```
# Write a program to find median.
# input format
# first line contain n
# second line contain values

#output
# print median

n=int(input())
```

```
list1=list(map(int,input().split()))
if n%2!=0:
    m=n//2
    print(list1[m])
else:
    m=n//2
    value1=list1[m]
    value2=list1[m-1]
    print((value1+value2)/2)
```

Output

```
6
1 2 3 4 5 6
3.5
```

Removing or deleting elements from list

Removing or deleting elements or values from list is done different ways

1. Using del keyword
2. Remove method
3. Pop method
4. Clear method

Using del keyword

This keyword is used to delete one or more than one value or element.

Syntax1: del <list-name>[index]
Syntax2: del <list-name>[start:stop:step]

Syntax1 is used for deleting one value/element
Syntax2 is used for deleting more than one value

```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
```

```
del list1[0]
>>> print(list1)
[20, 30, 40, 50]
>>> del list1[-2]
>>> print(list1)
[20, 30, 50]
>>> del list1[8]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    del list1[8]
IndexError: list assignment index out of range
```

Example:

```
# Write a program to delete an element from list

list1=[10,10,10,30,40,50,10,30,10,10,20]
value=10

l=len(list1)
index=0

while index<l:
    if list1[index]==value:
        del list1[index]
        l=l-1
        continue
    else:
        index=index+1

print(list1)
```

Output

```
[30, 40, 50, 30, 20]
```

Example of deleting multiple values using slicing

```
>>> list1=[10,20,30,40,50,60,70,80,90,100]
```

```
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> del list1[:2]
>>> print(list1)
[30, 40, 50, 60, 70, 80, 90, 100]
>>> del list1[-2:]
>>> print(list1)
[30, 40, 50, 60, 70, 80]
>>> del list1[2:-2]
print(list1)
[30, 40, 70, 80]
>>> list2=list(range(10,110,10))
>>> print(list2)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>> del list2[::2]
>>> print(list2)
[20, 40, 60, 80, 100]
>>> del list2[::-2]
>>> print(list2)
[40, 80]
```

remove method

this method remove given value from list.

Syntax: list-name.remove(value)

Example:

```
>>> list1=[10,10,20,30,40,10,10,20,30]
>>> print(list1)
[10, 10, 20, 30, 40, 10, 10, 20, 30]
>>> list1.remove(10)
>>> print(list1)
[10, 20, 30, 40, 10, 10, 20, 30]
>>> list1.remove(100)
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
```

```
    list1.remove(100)
ValueError: list.remove(x): x not in list
```

Example:

```
list1=[10,10,20,30,40,10,10,20,30]
value=10

while True:
    if value in list1:
        list1.remove(value)
    else:
        break

print(list1)


list1=[10,10,20,30,40,10,10,20,30]

c=list1.count(value)
for i in range(c):
    list1.remove(value)
```

Output

```
[20, 30, 40, 20, 30]
[20, 30, 40, 20, 30]
```

clear()

This method removes all the values from list or empty list

```
>>> list1=[10,20,30,40,50]
>>> del list1[::]
>>> print(list1)
[]
>>> list2=list(range(10,110,10))
>>> print(list2)
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

```
>>> list2.clear()
>>> print(list2)
[]
```

pop()
pop()

This method remove last element from list.
Before removing, it returns last element and remove.
pop() is method belong a data structure called STACK.
STACK is data structure which follows Last In First Out (LIFO), the element added last is removed first.

```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> x=list1.pop()
>>> print(x)
50
>>> print(list1)
[10, 20, 30, 40]
>>> y=list1.pop()
>>> print(y)
40
>>> print(list1)
[10, 20, 30]
```

Stack allows two operations

1. Push -> appending element or value to list
2. Pop → removing element or value from list

Example:

```
# Write a program to implement stack data structure

stack=[]

while True:
```

```
    print("1.Push")
    print("2.Pop")
    print("3.View")
    print("4.Exit")
    opt=int(input("Enter Your Option "))
    if opt==1:
        value=int(input("Enter any value "))
        stack.append(value)
        print("value pushed inside stack")
    elif opt==2:
        if len(stack)==0:
            print("Stack is empty ")
        else:
            value=stack.pop()
            print(f'Value poped from stack is {value}')
    elif opt==3:
        print(f'Stack is {stack}')
    elif opt==4:
        break
    else:
        print("Invalid Option")
```

Output

```
1.Push
2.Pop
3.View
4.Exit
Enter Your Option 1
Enter any value 10
value pushed inside stack
1.Push
2.Pop
3.View
4.Exit
Enter Your Option 1
```

Implementation of Queue Data structure

Queue data structure follows FIFO (First in First Out), The element added first is removed is first.

Queue allows two operations

1. adding
2. removing

Example:

```
# Write a program to implement Queue data structure in python

queue=[]
while True:
    print("1. Adding ")
    print("2. Removing ")
    print("3. View ")
    print("4. Exit")
    opt=int(input("Enter Your Option :"))
    match(opt):
        case 1:
            value=int(input("Enter Value "))
            queue.append(value)
            print(f'{value} is added with queue')
        case 2:
            if len(queue)==0:
                print("queue is empty")
            else:
                value=queue[0]
                del queue[0]
                print(f'{value} is removed from queue')
        case 3:
            print(f'Queue is {queue}')
        case 4:
            break
        case _:
            print("invalid option")
```

Output

1. Adding
2. Removing
3. View
4. Exit
Enter Your Option :1
Enter Value 10
10 is added with queue
1. Adding
2. Removing
3. View
4. Exit

Syntax: pop(index)

This removes given index position value.

Example:

```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list1.pop(2)
30
>>> print(list1)
[10, 20, 40, 50]
```

Inserting values within list

Inserting is nothing adding element or value in between the list of values.
This inserting is done using various approaches

1. using insert method
2. using slicing

Syntax of insert method
list-name.insert(index,value)

```
>>> list1=[10,20,30,40,50]
```

```
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list1.insert(0,99)
>>> print(list1)
[99, 10, 20, 30, 40, 50]
>>> list1.insert(3,88)
>>> print(list1)
[99, 10, 20, 88, 30, 40, 50]
>>> list1.insert(-1,77)
>>> print(list1)
[99, 10, 20, 88, 30, 40, 77, 50]
>>> list1.insert(20,100)
>>> print(list1)
[99, 10, 20, 88, 30, 40, 77, 50, 100]
>>> list1.insert(-20,200)
>>> print(list1)
[200, 99, 10, 20, 88, 30, 40, 77, 50, 100]
```

Using slicing

Slicing allows inserting more than one value.

Syntax: list-name[start:stop]=iterable

Start index and stop index must be same

```
>>> list1=[10,20,30,40,50]
print(list1)
[10, 20, 30, 40, 50]
>>> list1[0:0]=[77,88,99]
>>> print(list1)
[77, 88, 99, 10, 20, 30, 40, 50]
>>> list1[3:3]=[1,2,3,4,5]
>>> print(list1)
[77, 88, 99, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50]
>>> list1[-2:-2]=[11,22,33]
>>> print(list1)
[77, 88, 99, 1, 2, 3, 4, 5, 10, 20, 30, 11, 22, 33, 40, 50]
```

```
>>> list1[0:3]=[1,2,3]
>>> print(list1)
[1, 2, 3, 1, 2, 3, 4, 5, 10, 20, 30, 11, 22, 33, 40, 50]
```

extend()
This method is used to append more than one value.

Syntax: list-name.extend(iterable)

```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list1.extend([60,70,80])
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80]
>>> list1.extend("NIT")
>>> print(list1)
[10, 20, 30, 40, 50, 60, 70, 80, 'N', 'I', 'T']
```

What is difference between append and extend methods of list?

append method is used to add one element or value
extend method is used to add more than one value or element

s.reverse() reverses the items of s in place

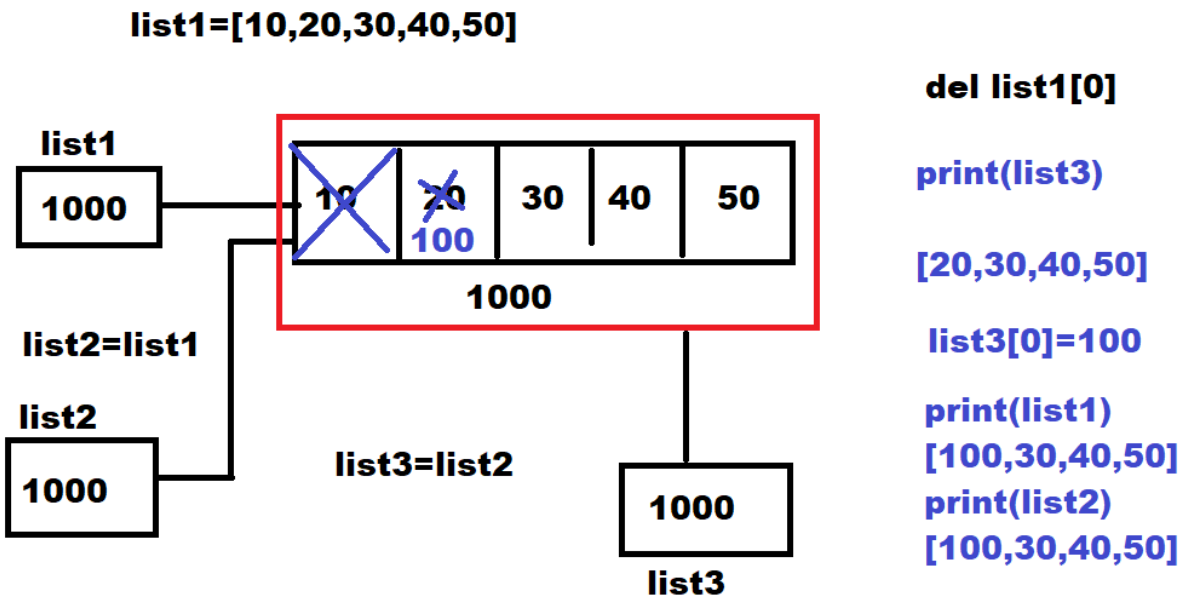
```
>>> list1=[10,20,30,40,50]
>>> print(list1)
[10, 20, 30, 40, 50]
>>> list1.reverse()
>>> print(list1)
>>> list2=list1[::-1]
>>> print(list2)
```

Creating copy of the list

There are various approaches of creating copy of the list

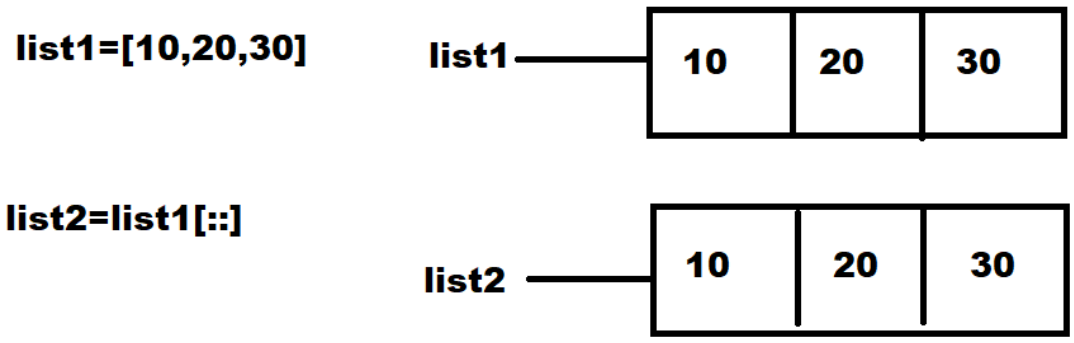
1. Alias copy
2. Slicing
3. Shallow copy
4. Deep copy

Alias copy
Assigning a list address or list to another variable is called alias copy.



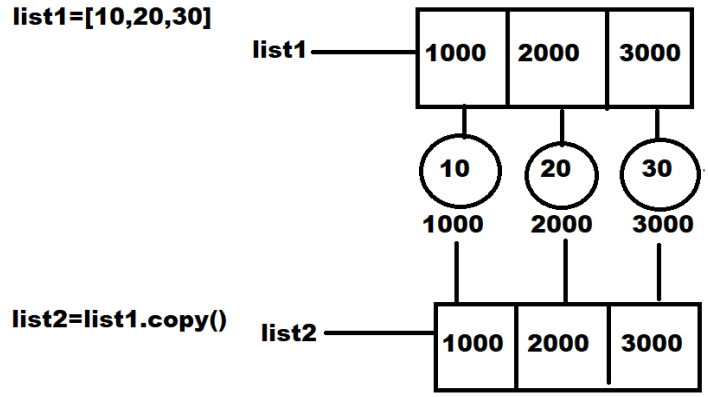
In above example list2 and list3 are alias of list1

2. slicing: list slicing allows to create copy of the list

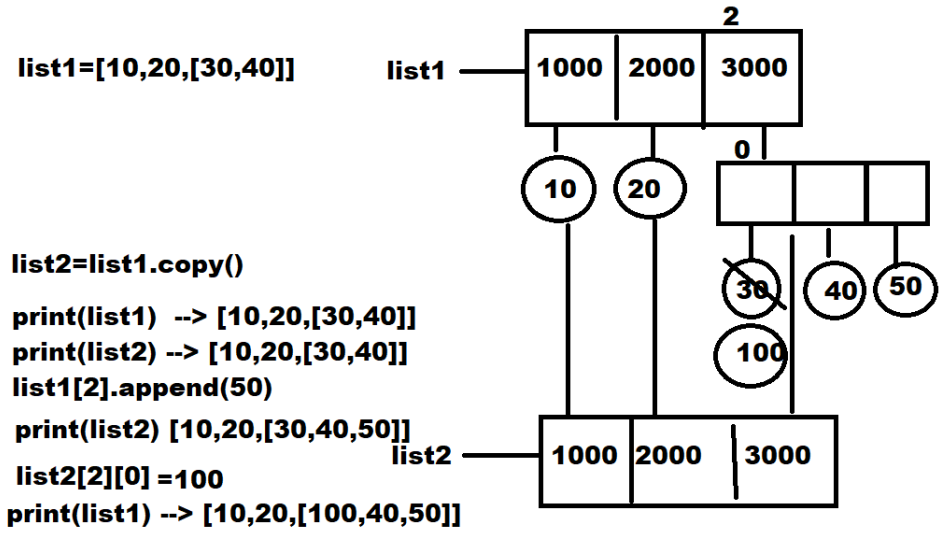


3. Shallow copy

The copy method provided by list creates shallow copy.
In shallow copy, copy() method create new list object by copying reference/address of objects found in existing list.



Objects exists within list are mutable, the changes done using one list can be applied to another list.

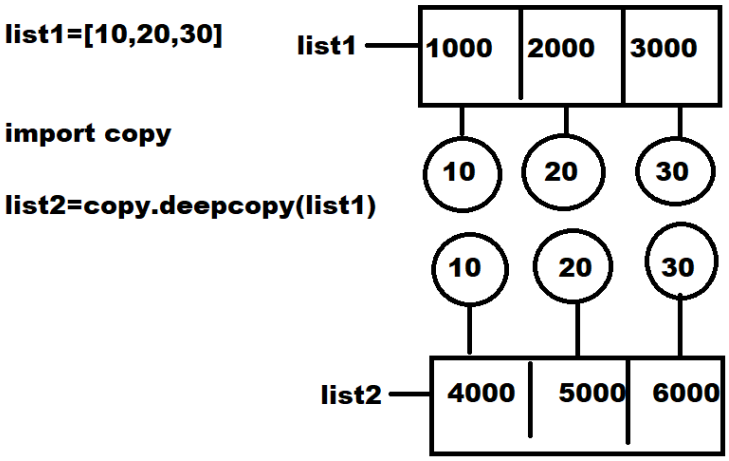


4.**Deepcopy**

Deep copy is an object copy. In deep copy a new list object is created by copying objects found in existing list. This deep copy is performed using "copy" module.

Syntax: copy.deepcopy(list-name)

"copy" is module-name
"deepcopy" is function name



What is difference between shallow copy and deep copy?

Shallow copy	Deep copy
Shallow copy is faster	Deep copy is slower as compared to shallow copy
It stores the references of an object to the original memory address	It only stores the copy of object's value.
It reflects changes made in copied object to original object	It does not reflect any change to the new object
Both copied and original object point to same memory location	Copied and original object does not point to same memory location.

Example of shallow copy

```
>>> list1=[10,20,[30,40]]
>>> list2=list1.copy()
>>> print(list1)
[10, 20, [30, 40]]
>>> print(list2)
[10, 20, [30, 40]]
>>> list1[2][0]=100
```

```
>>> print(list1)
[10, 20, [100, 40]]
>>> print(list2)
[10, 20, [100, 40]]
>>> list2[2].append(50)
>>> print(list2)
[10, 20, [100, 40, 50]]
>>> print(list1)
[10, 20, [100, 40, 50]]
```

Example of deep copy

```
>>> list1=[10,20,[30,40]]
>>> import copy
>>> list2=copy.deepcopy(list1)
>>> print(list1)
[10, 20, [30, 40]]
>>> print(list2)
[10, 20, [30, 40]]
>>> list1[2][0]=100
>>> print(list1)
[10, 20, [100, 40]]
>>> print(list2)
[10, 20, [30, 40]]
>>> list2[2].append(200)
>>> print(list2)
[10, 20, [30, 40, 200]]
>>> print(list1)
[10, 20, [100, 40]]
```

Operators used on list

\+ → This operator is used for concatenation of list

```
>>> list1=[10,20,30]
>>> list2=[40,50,60]
```

```
>>> list3=list1+list2
>>> print(list1)
[10, 20, 30]
>>> print(list2)
[40, 50, 60]
>>> print(list3)
[10, 20, 30, 40, 50, 60]
>>> list4=[1,2,3]
>>> list5=list1+list2+list4
>>> print(list5)
[10, 20, 30, 40, 50, 60, 1, 2, 3]
```

*→ This operator is used to repeat a sequence or list number of times

```
>>> list1=[0]*10
>>> print(list1)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> list2=[1]*20
>>> print(list2)
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>
```

in operator : this operator is used for searching value/element within list

```
>>> list1=[10,20,30,40,50]
>>> 10 in list1
True
>>> 100 in list1
False
```

https://www.hackerrank.com/challenges/python-lists/problem?isFullScreen=true

```
list1=[]
n=int(input())
for i in range(n):
```

```
cmd=input().split()
if cmd[0]=="insert":
   list1.insert(int(cmd[1]),int(cmd[2]))
elif cmd[0]=="print":
    print(list1)
elif cmd[0]=="sort":
    list1.sort()
elif cmd[0]=="reverse":
    list1.reverse()
elif cmd[0]=="pop":
    list1.pop()
elif cmd[0]=="append":
    list1.append(int(cmd[1]))
elif cmd[0]=="remove":
    list1.remove(int(cmd[1]))
```

Nested List

Creating list inside list is called nested list.

Nested list is used for representing table data, which consist of number of rows and columns (OR) matrix.

Defining list as an element inside list is called nested list.