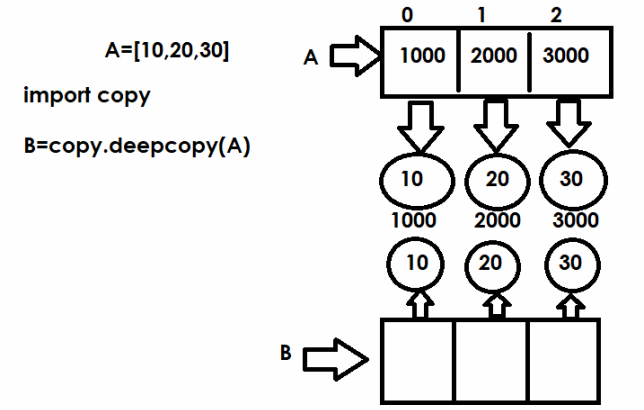
Deep Copy
Deep copy is object copy, in deep copy a new list created by copying objects found in existing list.
This deep copy is implemented using **"copy"** module. It is standard module which comes with python software
This module provides a predefined function called **deepcopy()**

Syntax:
Import copy
<variable-name>=copy.deepcopy(list-name)



In deep copy, if list having mutable objects, the changes done using one list is not applied to another list.

Example:
```
import copy
A=[[10,20],[30,40]]
B=A.copy() # Shallow Copy
print(A)
print(B)
A[0][0]=99
print(A)
print(B)

X=[[10,20],[30,40]]
Y=copy.deepcopy(X) # Deep Copy
print(X)
print(Y)
X[0][0]=99
print(X)
```

```
print(Y)
Y[0].append(88)
print(Y)
print(X)
```

Output

```
[[10, 20], [30, 40]]
[[10, 20], [30, 40]]
[[99, 20], [30, 40]]
[[99, 20], [30, 40]]
[[10, 20], [30, 40]]
[[10, 20], [30, 40]]
[[99, 20], [30, 40]]
[[10, 20], [30, 40]]
[[10, 20, 88], [30, 40]]
[[99, 20], [30, 40]]
```

What is difference between shallow copy and deep copy in python?

Shallow Copy	**Deep Copy**
It creates a new object, but instead of creating copies of the nested objects, it inserts references to the original nested objects. This means if you modify a nested object in the copied object, the change will also be reflected in the original object.	It creates a new object and recursively copies all nested objects as well. The copied object and the original object are completely independent, meaning changes made to one will not affect the other.
Shallow copy is done using copy method of list	Deep copy is done using deepcopy function of copy module

Operators used with list data type

1. + → This operator is used for concatenation of two lists

```
A=[10,20,30,40,50]
B=[60,70,80]
C=A+B

print(A)
print(B)
print(C)
```

Output

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

2. *Operator → This operator is used to repeat element/value of sequence/list n times

```
>>> A=[0]*10
>>> print(A)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> B=[None]*5
>>> print(B)
[None, None, None, None, None]
>>> C=[[0,0,0]]*3
>>> print(C)
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> D=["python"]*4
>>> print(D)
['python', 'python', 'python', 'python']
>>> E=[int(input())]*5
10
>>> print(E)
[10, 10, 10, 10, 10]
```

3. **in** → This operator is used for search given value within sequence or iterable. “in” is called membership operator, this operators returns Boolean value (True/False)

Example:

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

Example:

```
# Write a program to input unique values into list

A=[]
for i in range(10):
    value=int(input("Enter Value:"))
    if value not in A:
        A.append(value)
    else:
        print(f'{value} exists')

print(A)
```

Output

```
Enter Value:10
Enter Value:20
Enter Value:30
Enter Value:40
Enter Value:50
Enter Value:10
10 exists
Enter Value:50
50 exists
Enter Value:90
Enter Value:100
Enter Value:30
30 exists
[10, 20, 30, 40, 50, 90, 100]
```

Tuple data type

What is tuple?

Tuple is an immutable sequence data type, After creating tuple object changes cannot done. Tuple does not provides mutable operations.

1. append()
2. Insert()
3. Remove()
4. Extend()
5. Pop()

6. Clear()
7. Sort()
8. Del keyword

Where tuple is used in application development?
In application development tuple is used,

1. To represent immutable sequence
2. It is used to represent data for hash based collections like set, dictionary
3. It is used by enumerate for generating values from iterable

How to create tuple?
Tuple is created in different ways

1. Empty tuple is created using empty ()

```
>>> t1=()
>>> print(t1,type(t1))
() <class 'tuple'>
>>> t1.append(10)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    t1.append(10)
AttributeError: 'tuple' object has no attribute 'append'
>>> a=[]
>>> a.append(10)
>>> a
[10]
>>> del a[0]
>>> a
[]
```

2. Tuple with single value/element is created using parenthesis (value,)

```
>>> x=(10)
>>> print(x,type(x))
10 <class 'int'>
>>> y=(10,)
>>> print(y,type(y))
(10,) <class 'tuple'>
>>> z=("naresh")
```

```
>>> print(z,type(z))
naresh <class 'str'>
>>> p=("naresh",)
>>> print(p,type(p))
('naresh',) <class 'tuple'>
```

A tuple which consist of single object or element is called singleton tuple.

```
>>> q=10,
>>> print(q,type(q))
(10,) <class 'tuple'>
```

3. Tuple can be created with multiple values using parenthesis (value1,value2,value3,value4,value5,….)

Tuple can be homogeneous (similar type values) or Heterogeneous (Different types of values)

```
>>> stud1=(101,"naresh","python",5000)
>>> print(stud1,type(stud1))
(101, 'naresh', 'python', 5000) <class 'tuple'>
>>> stud1[1]="suresh"
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    stud1[1]="suresh"
TypeError: 'tuple' object does not support item assignment
>>> t1=(10,20,30,40,50)
>>> print(t1,type(t1))
(10, 20, 30, 40, 50) <class 'tuple'>
>>> t2=10,20,30,40,50
>>> t3=101,"naresh","manager",90000
>>> print(t2,type(t2))
(10, 20, 30, 40, 50) <class 'tuple'>
>>> print(t3,type(t3))
(101, 'naresh', 'manager', 90000) <class 'tuple'>
```

4. A tuple is created using tuple() function/type. This is used for converting other iterables/collections into tuple type

Syntax1: tuple() : This creates empty tuple

Syntax2: tuple(iterable) : This converts existing iterable/collections into tuple type

```
>>> t1=tuple()
>>> print(t1,type(t1))
() <class 'tuple'>
>>> t2=tuple(10)
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    t2=tuple(10)
TypeError: 'int' object is not iterable
>>> t2=tuple("PYTHON")
>>> print(t2,type(t2))
('P', 'Y', 'T', 'H', 'O', 'N') <class 'tuple'>
>>> t3=tuple(range(10,60,10))
>>> print(t3,type(t3))
(10, 20, 30, 40, 50) <class 'tuple'>
>>> t4=tuple([10,20,30])
>>> print(t4,type(t4))
(10, 20, 30) <class 'tuple'>
```