UNIT IV

COMPOUND DATA: LISTS, TUPLES, DICTIONARIES

Lists, list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters; **Tuples**, tuple assignment, tuple as return value; **Dictionaries**: operations and methods; advanced list processing - list comprehension, **Illustrative programs:** selection sort, insertion sort, merge sort, quick sort.

Lists

❖ List is an ordered sequence of items. Values in the list are called elements / items.

❖ It can be written as a list of comma-separated items (values) between **square brackets[ ]**.

Items in the lists can be of different data types.

Eg: a=*[10, 20, 30, 40]; b=[10, 20, "abc", 4.5]*

The following list contains a string, a float, an integer, and (lo!) another list:

['spam', 2.0, 5, [10, 20]]

A list within another list is nested. A list that contains no elements is called an empty list; you can create one with empty brackets, [].

As you might expect, you can assign list values to variables:

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print cheeses, numbers, empty
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

Operations on list:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Updating
6. Membership
7. Comparison

operations	examples	description

create a list	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> print(a)
	[2, 3, 4, 5, 6, 7, 8, 9, 10]
		in this way we can create a list at compile time

Indexing	>>> print(a[0])
	2
	>>> print(a[8])
	10
	>>> print(a[-1])
	10
		Accessing the item in the position 0
		Accessing the item in the position 8
		Accessing a last element using negative indexing.

>>> print(a[0:3])

Slicing [2, 3, 4]

>>> print(a[0:]) Printing a part of the list.

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

Concatenation	>>>b=[20,30] >>> print(a+b) [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30]	Adding and printing the items of two lists.
Repetition	>>> print(b*3) [20, 30, 20, 30, 20, 30]	Create a multiple copies of the same list.

Updating	>>> print(a[2])
	4
	>>> a[2]=100
	>>> print(a)
	[2, 3, 100, 5, 6, 7, 8, 9, 10]
		Updating the list using index value.

Membership	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in list. Otherwise returns false.

Comparison	>>> a=[2,3,4,5,6,7,8,9,10]
	>>>b=[2,3,4]
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

List slices:

- List slicing is an operation that extracts a subset of elements from an list and packages them as another list.

Syntax:

Listname[start:stop]

Listname[start:stop:steps]

- default start value is 0
- default stop value is n-1
- [:] this will print the entire list
- [2:2] this will create a empty slice

slices	example	description

a[0:3]	>>> a=[9,8,7,6,5,4]
	>>> a[0:3]
	[9, 8, 7]
		Printing a part of a list from 0 to 2.

a[:4]	>>> a[:4] [9, 8, 7, 6]	Default start value is 0. so prints from 0 to 3
a[1:]	>>> a[1:] [8, 7, 6, 5, 4]	default stop value will be n-1. so prints from 1 to 5
a[:]	>>> a[:] [9, 8, 7, 6, 5, 4]	Prints the entire list.

slices	example	description
a[2:2]	>>> a[2:2] [ ]	print an empty slice

a[0:6:2]	>>> a=[9,8,7,6,5,4]
	>>> a[0:6:2]
	[9, 7, 5]
	>>> a[0:6:3]
	[9,6]
		Slicing list values with step size 2.(from index[0] to 2nd element and from that position to next 2nd element

List methods:

Python provides methods that operate on lists.

syntax: **list name.method name(element/index/list)**
example

syntax description

1	a.append(element)	>>> a=[1,2,3,4,5]
		>>> a.append(6)
		>>> print(a)
		[1, 2, 3, 4, 5, 6]
			Add an element to the end of the list

2	a.insert(index,element)	>>> a.insert(0,0)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6]
			Insert an item at the defined index

3 a.extend(b) >>> a=[1,2,3,4,5]
>>> b=[7,8,9]
>>> a.extend(b)
>>> print(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8,9]

Add all elements of a list to the another list

4	a.index(element)	>>>a=[0, 1, 2, 3, 8,5, 6, 7, 8,9] >>> a.index(8) 4	Returns the index of the first matched item

5	sum()	>>> a=[1,2,3,4,5]
		>>> sum(a)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8,9]
			Sort items in a list in ascending order

6	a.reverse()	>>> a.reverse()
		>>> print(a)
		[8, 7, 6, 5, 4, 3, 2, 1, 0]
			Reverse the order of items in the list

		>>>a=[8, 7, 6, 5, 4, 3, 2, 1, 0]	
7	a.pop()	>>> a.pop() 0 >>>print(a) =[8, 7, 6, 5, 4, 3, 2, 1]	Removes and returns an element at the last element
8	a.pop(index)	>>> a.pop(0) 8 >>>print(a) [7, 6, 5, 4, 3, 2, 1, 0]	Remove the particular element and return it.

9	a.remove(element)	>>>a=[7, 6, 5, 4, 3, 2, 1]
		>>> a.remove(1)
		>>> print(a)
		[7, 6, 5, 4, 3, 2]
			Removes an item from the list

	a.count(element)	>>>a=[7, 6, 5, 4, 3, 2,6] >>> a.count(6) 2	Returns the count of number of items passed as an argument

	a.copy()	>>>a=[7, 6, 5, 4, 3, 2]
		>>> b=a.copy()
		>>> print(b)
		[7, 6, 5, 4, 3, 2]
			Returns a copy of the list

	len(list)	>>>a=[7, 6, 5, 4, 3, 2] >>> len(a) 6	return the length of the length
	sum(list)	>>>a=[7, 6, 5, 4, 3, 2] >>> sum(a) 27	return the sum of element in a list

14 max(list) >>> max(a) return the maximum element in a list.

15	a.clear()	>>> a.clear()
		>>> print(a)
		[ ]
			Removes all items from the list.

16	del(a)	>>> del(a)
		>>> print(a)
		Error: name 'a' is not defined
			delete the entire list.

List loops:

1. **For loop**
2. **While loop**
3. **Infinite loop**

List using For Loop:

- The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.

- Iterating over a sequence is called traversal.
- Loop continues until we reach the last item in the sequence.
- The body of for loop is separated from the rest of the code **using indentation**.

Syntax:

for val in sequence:	

Accessing element ~~output~~

a=[10,20,30,40,50] for i in a: print(i)	10 20 30 40 50
Accessing index	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(i)	0 1 2 3 4
Accessing element using range:	**output**
a=[10,20,30,40,50] for i in range(0,len(a),1): print(a[i])	10 20 30 40 50

List using While loop

- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Syntax:
while
(condition):
body of

while

Sum of elements in list			**Output:**

a=[1,2,3,4,5]
i=0
sum=0
while i<len(a):
  sum=sum+a[i]
  i=i+1
print(sum)
			15

Infinite Loop

A loop becomes infinite loop if the condition given never becomes false. It keeps on running. Such loops are called infinite loop.

Example	**Output:**

a=1
while (a==1):
  n=int(input("enter the number"))
  print("you entered:" , n)
	Enter the number 10
	you entered:10
	Enter the number 12
	you entered:12
	Enter the number 16
	you entered:16

Mutability:

- Lists are mutable. (can be changed)
- Mutability is the ability for certain types of data to be changed without entirely recreating it.
- An item can be changed in a list by accessing it directly as part of the assignment statement.
- Using the indexing operator (square brackets[ ]) on the left side of an assignment, one of the list items can be updated.

Example	**description**

>>> a=[1,2,3,4,5]
>>> a[0]=100
>>> print(a)
[100, 2, 3, 4, 5]
	changing single element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[100,100,100]
>>> print(a)
[100, 100, 100, 4, 5]
	changing multiple element

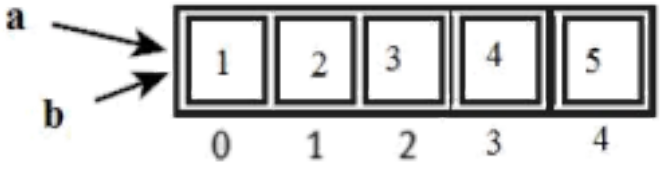
>>> a=[1,2,3,4,5]
>>> a[0:3]=[ ]
>>> print(a)
[4, 5]
	The elements from a list can also be removed by assigning the empty list to them.

>>> a=[1,2,3,4,5]
>>> a[0:0]=[20,30,45]
>>> print(a)
[20,30,45,1, 2, 3, 4, 5]
	The elements can be inserted into a list by squeezing them into an empty slice at the desired location.

Aliasing(copying):

- Creating a copy of a list is called aliasing.
- When you create a copy both the list will be having same memory location.
- changes in one list will affect another list.
- **Alaising refers to having different names for same list values.**

Example	Output:
a= [1, 2, 3 ,4 ,5] b=a print (b) a is b a[0]=100 print(a) print(b)	[1, 2, 3, 4, 5] True [100,2,3,4,5] [100,2,3,4,5]



❖ In this a single list object is created and modified using the subscript operator. ❖

When the first element of the list named **"a"** is replaced, the first element of the list named "**b"** is also replaced.

❖ This type of change is what is known as a **side effect**. This happens because after the assignment **b=a**, the variables **a** and **b** refer to the exact same list object. ❖ They are **aliases** for the same object. This phenomenon is known as **aliasing**. ❖ To prevent aliasing, a new object can be created and the contents of the original can be copied which is called **cloning.**

Clonning:

- ❖ To avoid the disadvantages of copying we are using cloning.
  - ❖ Creating a copy of a same list of elements with two different memory locations is called cloning.

❖ Changes in one list will not affect locations of aother list.

❖ Cloning is a process of making a copy of the list without modifying the original list.

1. Slicing
2. list()method
3. copy() method

clonning using Slicing

```
>>>a=[1,2,3,4,5]
>>>b=a[:]
>>>print(b)
[1,2,3,4,5]
>>>a is b
False #because they have different memory location
```

clonning using List( ) method

```
>>>a=[1,2,3,4,5]
>>>b=list
>>>print(b)
[1,2,3,4,5]
>>>a is b
false
>>>a[0]=100
>>>print(a)
>>>a=[100,2,3,4,5]
>>>print(b)
>>>b=[1,2,3,4,5]
```

clonning using copy() method

```
a=[1,2,3,4,5]
>>>b=a.copy()
```

```
>>> print(b)
[1, 2, 3, 4, 5]
>>> a is b
False
```

List as parameters:

- In python, arguments are passed by reference.
- If any changes are done in the parameter which refers within the function, then the changes also reflects back in the calling function.
- When a list to a function is passed, the function gets a reference to the list.
- Passing a list as an argument actually passes a reference to the list, not a copy of the list.
- Since lists are mutable, changes made to the elements referenced by the parameter change the same list that the argument is referencing.

Example 1`:	**Output**
def remove(a): a.remove(1) a=[1,2,3,4,5] remove(a) print(a)	[2,3,4,5]

Example 2:	**Output**

def inside(a):
  for i in range(0,len(a),1):
  a[i]=a[i]+10
  print("inside",a)
a=[1,2,3,4,5]
inside(a)
print("outside",a)
	inside [11, 12, 13, 14, 15]
	outside [11, 12, 13, 14, 15]

Example 3	**output**
def insert(a): a.insert(0,30) a=[1,2,3,4,5] insert(a) print(a)	[30, 1, 2, 3, 4, 5]

Tuple:

- A tuple is same as list, except that the set of elements is enclosed in parentheses instead of square brackets.

❖ A tuple is an immutable list. i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.

❖But tuple can be converted into list and list can be converted in to tuple.

methods	example	description

list( )	>>> a=(1,2,3,4,5)
	>>> a=list(a)
	>>> print(a)
	[1, 2, 3, 4, 5]
		it convert the given tuple into list.

tuple( )	>>> a=[1,2,3,4,5]
	>>> a=tuple(a)
	>>> print(a)
	(1, 2, 3, 4, 5)
		it convert the given list into tuple.

Benefit of Tuple:

❖ Tuples are faster than lists.

❖ If the user wants to protect the data from accidental changes, tuple can be used.

❖ Tuples can be used as keys in dictionaries, while lists can't.

Operations on Tuples:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Membership
6. Comparison

Operations	examples	description
Creating a tuple	>>>a=(20,40,60,"apple","ball")	Creating the tuple with elements of different data types.
Indexing	>>>print(a[0]) 20 >>> a[2] 60	Accessing the item in the position 0 Accessing the item in the position 2
Slicing	>>>print(a[1:3]) (40,60)	Displaying items from 1st till 2nd.
Concatenation	>>> b=(2,4) >>>print(a+b) >>>(20,40,60,"apple","ball",2,4)	Adding tuple elements at the end of another tuple elements

Repetition	>>>print(b*2) >>>(2,4,2,4)	repeating the tuple in n no of times

Membership	>>> a=(2,3,4,5,6,7,8,9,10)
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in tuple. Otherwise returns false.

Comparison	>>> a=(2,3,4,5,6,7,8,9,10)
	>>>b=(2,3,4)
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

Tuple methods:		

❖ Tuple is immutable so changes cannot be done on the elements of a tuple once it is assigned.

methods	**example**	**description**

a.index(tuple)	>>> a=(1,2,3,4,5)
	>>> a.index(5)
	4
		Returns the index of the first matched item.

a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple

min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

Tuple Assignment:

- Tuple assignment allows, variables on the left of an assignment operator and values of tuple on the right of the assignment operator.
- Multiple assignment works by creating a tuple of expressions from the right hand side, and a tuple of targets from the left, and then matching each expression to a target.
- Because multiple assignments use tuples to work, it is often termed tuple assignment.

Uses of Tuple assignment:

- It is often useful to swap the values of two variables.

Example:

Swapping using temporary variable:	Swapping using tuple assignment:
a=20 b=50 temp = a a = b b = temp print("value after swapping is",a,b)	a=20 b=50 (a,b)=(b,a) print("value after swapping is",a,b)

Multiple assignments:

Multiple values can be assigned to multiple variables using tuple assignment.

```
>>>(a,b,c)=(1,2,3)
>>>print(a)
1
>>>print(b)
2
>>>print(c)
3
```

Tuple as return value:

- A Tuple is a comma separated sequence of items.
- It is created with or without ( ).
- A function can return one value. if you want to return more than one value from a function. we can use tuple as return value.

Example1:	Output:

def div(a,b):
  r=a%b
  q=a//b
  return(r,q)
a=eval(input("enter a value:"))
b=eval(input("enter b value:"))
r,q=div(a,b)
print("reminder:",r)
print("quotient:",q)
	enter a value:4
	enter b value:3
	reminder: 1
	quotient: 1

Example2:	**Output:**

def min_max(a):
  small=min(a)
  big=max(a)
  return(small,big)
a=[1,2,3,4,6]
small,big=min_max(a)
print("smallest:",small)
print("biggest:",big)
	smallest: 1
	biggest: 6

Tuple as argument:

- The parameter name that begins with * gathers argument into a tuple.

Example:	**Output:**
def printall(*args): print(args) printall(2,3,'a')	(2, 3, 'a')

Dictionaries:

- Dictionary is an unordered collection of elements. An element in dictionary has a key: value pair.
- All elements in dictionary are placed inside the curly braces i.e. { }
- Elements in Dictionaries are **accessed via keys** and not by their position.
- The values of a dictionary can be any data type.
- Keys must be immutable data type (numbers, strings, tuple)

Operations on dictionary:

1. Accessing an element
2. Update
3. Add element
4. Membership

	Operation		**Example**		**Description**

s		

Creating a dictionary	>>> a={1:"one",2:"two"}
	>>> print(a)
	{1: 'one', 2: 'two'}
		Creating the dictionary with elements of different data types.

accessing an element	>>> a[1]
	'one'
	>>> a[0]
	KeyError: 0
		Accessing the elements by using keys.

Update	>>> a[1]="ONE"
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Assigning a new value to key. It replaces the old value by new value.

add element	>>> a[3]="three"
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
		Add new element in to the dictionary with key.

membership	a={1: 'ONE', 2: 'two', 3: 'three'} >>> 1 in a True >>> 3 not in a False	Returns True if the key is present in dictionary. Otherwise returns false.

Methods in dictionary:

Method	Example	Description
a.copy( )	a={1: 'ONE', 2: 'two', 3: 'three'} >>> b=a.copy() >>> print(b) {1: 'ONE', 2: 'two', 3: 'three'}	It returns copy of the dictionary. here copy of dictionary 'a' get stored in to dictionary 'b'
a.items()	>>> a.items() dict_items([(1, 'ONE'), (2, 'two'), (3, 'three')])	Return a new view of the dictionary's items. It displays a list of dictionary's (key, value) tuple pairs.

a.keys()	>>> a.keys() dict_keys([1, 2, 3])	It displays list of keys in a dictionary
a.values()	>>> a.values() dict_values(['ONE', 'two', 'three'])	It displays list of values in dictionary

a.pop(key)	>>> a.pop(3)
	'three'
	>>> print(a)
	{1: 'ONE', 2: 'two'}
		Remove the element with *key* and return its value from the dictionary.

setdefault(key,value)	>>> a.setdefault(3,"three")
	'three'
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three'}
	>>> a.setdefault(2)
	'two'
		If key is in the dictionary, return its value. If key is not
		present, insert key with a value of dictionary and return dictionary.

a.update(dictionary)	>>> b={4:"four"}
	>>> a.update(b)
	>>> print(a)
	{1: 'ONE', 2: 'two', 3: 'three', 4: 'four'}
		It will add the dictionary with the existing
		dictionary

fromkeys()	>>> key={"apple","ball"}
	>>> value="for kids"
	>>>
	d=dict.fromkeys(key,value)
	>>> print(d)
	{'apple': 'for kids', 'ball': 'for kids'}
		It creates a dictionary from key and values.

len(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>>lena(a) 3	It returns the length of the list.
clear()	a={1: 'ONE', 2: 'two', 3: 'three'} >>>a.clear() >>>print(a) >>>{ }	Remove all elements form the dictionary.
del(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>> del(a)	It will delete the entire dictionary.

Difference between List, Tuples and dictionary:

List	Tuples	Dictionary

A list is mutable	A tuple is immutable	A dictionary is mutable
Lists are dynamic	Tuples are fixed size in nature	In values can be of any data type and can repeat, keys must be of immutable type
List are enclosed in brackets[ ] and their elements and size can be changed	Tuples are enclosed in parenthesis ( ) and cannot be updated	Tuples are enclosed in curly braces { } and consist of key:value
Homogenous	Heterogeneous	Homogenous
Example: List = [10, 12, 15]	Example: Words = ("spam", "egss") Or Words = "spam", "eggs"	Example: Dict = {"ram": 26, "abi": 24}
Access: print(list[0])	Access: print(words[0])	Access: print(dict["ram"])

Can contain duplicate elements	Can contain duplicate elements. Faster compared to lists	Cant contain duplicate keys, but can contain duplicate values
Slicing can be done	Slicing can be done	Slicing can't be done
Usage: ❖List is used if a collection of data that doesnt need random access. ❖ List is used when data can be modified frequently	Usage: ❖Tuple can be used when data cannot be changed. ❖ A tuple is used in combination with a dictionary i.e.a tuple might represent a key.	Usage: ❖Dictionary is used when a logical association between key:value pair. ❖ When in need of fast lookup for data, based on a custom key. ❖ Dictionary is used when data is being constantly modified.

Advanced list processing:

List Comprehension:

- ❖ List comprehensions provide a concise way to apply operations on a list.
- ❖ It creates a new list in which each element is the result of applying a given operation in a list.
- ❖ It consists of brackets containing an expression followed by a “for” clause, then a list. ❖ The list comprehension always returns a result list.

Syntax

list=[expression for item in list if conditional]

List Comprehension	Output
>>>L=[x**2 for x in range(0,5)] >>>print(L)	[0, 1, 4, 9, 16]
>>>[x for x in range(1,10) if x%2==0]	[2, 4, 6, 8]
>>>[x for x in 'Python Programming' if x in ['a','e','i','o','u']]	['o', 'o', 'a', 'i']
>>>mixed=[1,2,"a",3,4.2] >>> [x**2 for x in mixed if type(x)==int]	[1, 4, 9]
>>>[x+3 for x in [1,2,3]]	[4, 5, 6]
>>> [x*x for x in range(5)]	[0, 1, 4, 9, 16]
>>> num=[-1,2,-3,4,-5,6,-7] >>> [x for x in num if x>=0]	[2, 4, 6]

>>> str=["this","is","an","example"]
>>> element=[word[0] for word in str]
>>> print(element)
	['t', 'i', 'a', 'e']

Nested list:

List inside another list is called nested list.

Example:

```
>>> a=[56,34,5,[34,57]]
>>> a[0]
56
>>> a[3]
[34, 57]
>>> a[3][0]
34
>>> a[3][1]
57
```

Programs on matrix:

Matrix addition	**Output**

a=[[1,1],[1,1]]
b=[[2,2],[2,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(b)):
  c[i][j]=a[i][j]+b[i][j]
for i in c:
  print(i)
	[3, 3]
	[3, 3]

Matrix multiplication	**Output**

a=[[1,1],[1,1]]
b=[[2,2],[2,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(b)):
  for k in range(len(b)):
  c[i][j]=a[i][j]+a[i][k]*b[k][j]
for i in c:
  print(i)
	[3, 3]
	[3, 3]

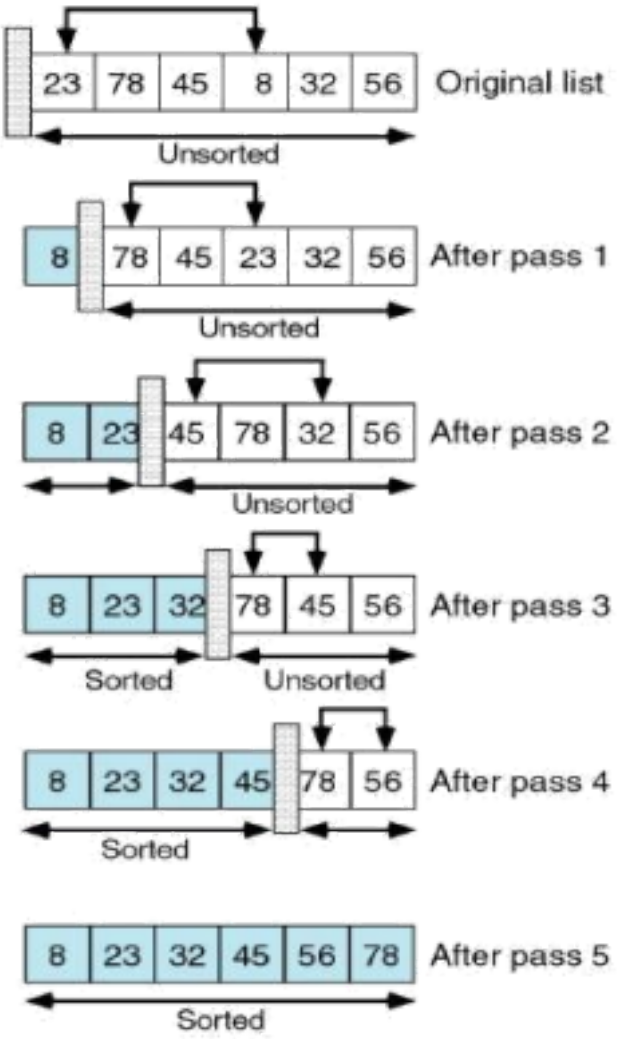
Matrix transpose	**Output**

a=[[1,3],[1,2]]
c=[[0,0],[0,0]]
for i in range(len(a)):
  for j in range(len(a)):
  c[i][j]=a[j][i]
for i in c:
  print(i)
	[1, 1]
	[3, 2]

Illustrative programs:

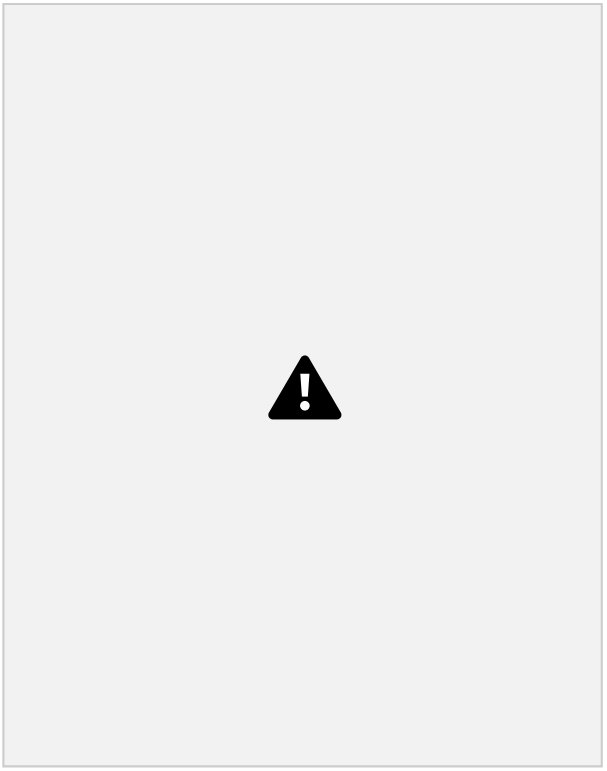
Selection sort	**Output**

a=input("Enter list:").split()
a=list(map(eval,a))
for i in range(0,len(a)):
  smallest = min(a[i:])
  sindex= a.index(smallest)
  a[i],a[sindex] = a[sindex],a[i]
print (a)
	Enter list:23 78 45 8 32 56
	[8,2 3, 32, 45,56, 78]



Insertion sort	**output**

a=input("enter a list:").split()
a=list(map(int,a))
for i in a:
  j = a.index(i)
  while j>0:
  if a[j-1] > a[j]:
  a[j-1],a[j] = a[j],a[j-1]
  else:
  break
  j = j-1
print (a)
	enter a list: 8 5 7 1 9 3
	[1,3,5,7,8,9]



Merge sort	**output**

def merge(a,b):
  c = []
  while len(a) != 0 and len(b) != 0:
  if a[0] < b[0]:
  c.append(a[0])
  a.remove(a[0])
  else:
  c.append(b[0])
  b.remove(b[0])
  if len(a) == 0:
  c=c+b
  else:
  c=c+a
  return c
 	
def divide(x):
  if len(x) == 0 or len(x) == 1:
  return x
  else:
  middle = len(x)//2
  a = divide(x[:middle])
  b = divide(x[middle:])
  return merge(a,b)
 	
x=[38,27,43,3,9,82,10]
c=divide(x)
print(c)
	[3,9,10,27,38,43,82]

Histogram Output def histogram(a): ****
for i in a: *****
sum = '' *******
while(i>0): ********
sum=sum+'#' ********** i=i-1
print(sum)
a=[4,5,7,8,12]
histogram(a)

Calendar program Output import calendar enter year:2017
y=int(input("enter year:")) enter month:11 m=int(input("enter month:")) November 2017 print(calendar.month(y,m)) Mo Tu We Th Fr Sa Su 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

Write these programs both in Observation and Record:

1. Access the elements of a tuple in multiple ways, such as indexing, negative indexing, range. 2. Concatenate two tuples together
3. Create a tuple and perform following the functions on tuple
i. Nesting ii. Slicing iii. Deleting iv. Update v. Sort vi. Reverse
4. Create a tuple and perform following the operations on tuple

1. in
2. not in
3. *
4. +
5. []
6. [:]

5. Perform the following Mutable operations on List
a. append b. extend c. insert d. del d. remove e. reverse f. sort

6. Perform the following
1. concatenate or join two different lists in a new list.
2. replicate a list to the specified number of times
3. returns the smallest element and largest element in a list
4. return the number of times the specified item occurs in the list.
5. Find the sum of the elements in the list

7.Perform the following Operations on a Dictionary

- Accessing An Element
- Replacing An Element
- Removing An Element

8. Create a dictionary and add the elements to the dictionary

9. Create a dictionary and Iterate through each key in a dictionary using a for loop. 10. Demonstarte the usage of Built-in Dictionary Methods

d.clear(), d.get(<key>[, <default>]), d.items(), d.keys(), d.values(), d.pop(<key>[, <default>])

d.popitem(), d.update(<obj>)