UNIT IV : 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
* mergesort
* histogram

LIST INTRODUCTION, LIST CREATION, LIST ACCESSING, LIST METHODS : REFER TO LIST AS ARRAYS TOPIC IN III UNIT, IT IS THE SAME.

LIST OPERATIONS

* few operations that can be performed with a list.

1) concatenation : joins two lists by using '+' operator

eg)
```
l1 = [1,2,3,4,5]
l2 = [6,7,8,9,10].
l3 = l1 + l2
print(l3)
```

o/p: [1,2,3,4,5,6,7,8,9,10]

2) repetition : The '*' operator repeats a list a given no. of. times

eg)
```
l1 = [1,2]
l2 = l1 * 3
print(l2)
```

o/p: [1,2,1,2,1,2].

: checks if the value is p
list.

eg) l = ['a', 'e', 'i', 'o',
print ('a' in l)

o/p : True

4) not in : checks if the value is not present in the list.

eg) l = ['a', 'e', 'i', 'o', 'u']
print ('a' not in l)

o/p: False

5) max() : returns maximum value in the list.

eg) l = [20, 10, 30, 40]
print (max(l))

o/p: 40

6) min() : returns minimum value in the list.

eg) l = [20, 10, 30, 40]
print (min(l))

o/p: 10

7) sum() : adds the values in the list has numbers

eg) l = [10, 20, 30, 40, 50]
print (sum(l))

8) list() : converts an iterable (tuple, string, dictionary) to a list.

eg) fruit = "banana"
l = list(fruit)
print(l)

o/p : ['b', 'a', 'n', 'a', 'n', 'a']

9) sorted() : returns a new sorted list. The original list is not sorted.

eg) l1 = [3, 4, 1, 2, 9, 8]
l2 = sorted(l1)
print(l2)

o/p : [1, 2, 3, 4, 7, 8]

LIST SLICES

* List can also be sliced. slice operator also works on lists.

syntax

list_var = List [Beg : end : step]

* If 'Beg' index is omitted, then slice starts at the beginning
* 'end' index goes upto end - 1, if it is omitted, then slice goes to the end.
* step indicates the no. of elements to move forward, the default step value is 1.
* If both 'beg' and 'end' index are omitted, the slice is a copy of the whole list.

eg(i)

```
t = ['a', 'b', 'c', 'd', 'e', 'f']
l = t[1:3]
print(l)
```

O/P.

['b', 'c']

ii) t = ['a', 'b', 'c', 'd', 'e', 'f']

```
l = t[:4]          # slice 'begen' index a.
print(l)
```

o/p.

['a', 'b', 'c', 'd']

iii) t = ['a', 'b', 'c', 'd', 'e', 'f']

```
l = t[a:]          # slice 'end' index omitted
print(l)
```

o/p

['d', 'e', 'f']

iv) t = ['a', 'b', 'c', 'd', 'e', 'f']

```
l = t[:]           # both index omitted
print(l)
```

o/p

['a', 'b', 'c', 'd', 'e', 'f']

* a slice operator on the left side of an assignment can update multiple elements:

eg) t = ['a', 'b', 'c', 'd', 'e', 'f']

```
t[1:3] = ['x', 'y']
print(t)
```

o/p: ['a', 'x', 'y', 'd', 'e', 'f']

LIST LOOP

⇒ for ... in

* The for.. in statement can be used to loop the list.

eg) l = [10, 20, 30, 40]

```
for x in l:
    print(x, end=" ")
```

O/P : 10 20 30 40

⇒ using the range() function

* range() function with for can also be used to loop the list.

eg) l = [10, 20, 30, 40]

```
for i in range(len(l)):
    print(l[i], end=" ")
```

O/P : 10 20 30 40

⇒ using the enumerate() function

* the enumerate() function returns an enumerate object which contains the index and the value of all the items of the list as a tuple.

eg) l = [10, 20, 30, 40]

```
for index, x in enumerate(l):
    print(x, "is at index:", index)
```

O/P

```
10 is at index: 0
20 is at index: 1
30 is at index: 2
40 is at index: 3
```

⇒ using an iterator

* an iterator can be created using the built-in iter() function.
* The iterator is used to loop over the elements of the list.
* The iterator fetches the value and automatically points to the next element using next() function

eg)

```
l = [10, 20, 30, 40]
it = iter(l)
for i in range(len(l)):
    print(next(it), end=" ")
```

O/P 10 20 30 40

MUTABILITY:

* List are mutable, which means the list can be updated.

eg) l = [10, 20, 30, 40]

```
l[1] = 50
print(l)
```

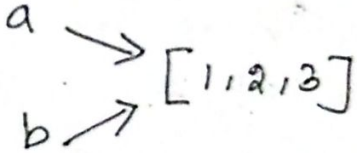
O/P

[10, 50, 30, 40]

ALIASING:

* A object with more than one reference has more than one name, which is said to be aliasing

```
>>> a = [1,2,3]
>>> b = a
>>> b is a
True
```



* Both a and b variable refers to the same object.
* The association of a variable with an object is called a reference.

```
>>> b[0] = 4
>>> print(a)
[4,2,3]
>>> print(b)
```

CLONING

* creating a copy of the original list.
* The slice operation is used to clone a list
* The original list is not modified.

eg)

```
>>> a = [1,2,3]
>>> b = a[:]        #creates a copy
>>> a == b
  True
>>> a is b          #reference not same
  False
>>> b[0] = 4
>>> print(a)
  [1, 2, 3]
>>> print (b)
  [4, 2, 3]
```

NESTED LIST

* A list within another list is called nested list

eg) $l = [\overset{0}{10}, \overset{1}{20}, \overset{2}{[30,40]}, \overset{3}{50}]$

```
print(l[2])
print(l[2][0])
```

o/p: [30,40]

[30]

LIST PARAMETERS

* List can be passed as a parameter to a function, the function gets a reference to the list.

* If the function modifies a list parameter, the change is reflected, in the function call module.

eg)
```
def sample(t):    # fn definition
    t[0]=90

l=[10,20,30,40]
print(l)
sample(l)    # fn call
print(l)
```

o/p: [10,20,30,40]

[90,20,30,40]

* The list variable 'l' and parameter 't' are ali
for the same object.

Advanced List Processing

* The use of map(), filter() and reduce() functions

map()

* map() function applies a particular function to every element of a list.

syntax:

```
map(function, sequence)
```

* After applying the specified function on the sequence, map() function returns the modified list.

eg.

```
def add(x):
    x = x+2
    return x

l = [1,2,3,4,5]
print("list before applying map:", l)
new_l = list(map(add, l))
print("mapped list:", new_l)
```

O/P:

```
list before applying map : [1,2,3,4,5]
mapped list : [3,4,5,6,7]
```

1)

e filter() function constructs a m
ose elements of the list for which a function
eturns True.

Syntax

filter (function, sequence)

eg.

```
def check(x):
    if (x%2==0 or x%4==0):
        return True

evens = list(filter(check, range(2,22)))
print(evens)
```

o/p

[2,4,6,8,10,12,14,16,18,20]

reduce()

* The reduce() function returns a single value which is generated by calling the function on the first two items of the sequence, then on the result and the next item and so on

syntax

reduce (function, sequence)

* reduce() is available in the functools.

eg.

```
import functools
def add(x,y):
    return x+y
l = functools.reduce(add, [1,2,3,4,5])
print(l)
```

o/p: 15

LIST COMPREHENSION

* List comprehension is used to create lists in a concise way.
* A new list can be created where each element is obtained by applying some operations to each member of the given sequence or iterable.

syntax

list = [expression for<variable> in sequence]

* The expression is evaluated for every item in sequence
* List comprehension is also used to create a subsequence of those elements that satisfy a certain condition.

```
cubes = [ i**3  for i i:
print(cubes)
```

P:

[1, 8, 27, 64, 125]

eg) program that illustrates filtering using list comprehension

```
t = "Python Book"
l = [s for s in t  if s.isupper()]
print(l)
```

O/P

['P', 'B']

TUPLE :

* Tuple datatype is similar to list, it also consist of a sequence of values separated by commas and enclosed within parenthesis. The elements in the tuple can be of different data type (int, float, str)
* Tuple is a sequence of immutable objects, which means the values in a tuple cannot be changed.

eg) t = ('a', 'b', 'c', 10, 5.6)

⇒ creating a tuple

1) creating an empty tuple

```
eg) t = ()
    print(t)
```

o/p.

```
()
```

2) creating a tuple with a single element

```
eg) t = ('a',)
    print(t)
```

o/p

```
('a',)
```

[note: To create a tuple with a single element, should include a final comma].

creating a tuple with more th

eg) t = ('a', 'b', 'c', 10, 8.9)
print(t)

o/p

('a', 'b', 'c', 10, 8.9)

⇒ Accessing values in tuple

* Elements in the tuple are indexed starting from '0' and accessed using indexing.

eg) t = ('a', 'b', 'c', 'd', 'e')
print (t[0])

o/p
a

* for... in statement can be used to iterate thro' the tuple

eg) t = (10, 20, 30)
for x in t :
print (x, end=" ")

o/p : 10 20 30

⇒ tuple slicing

* slicing operation similar to list and the slice operator selects a range of elements.

eg) t = ('a', 'b', 10, 8.5)
print (t[1:3])

o/p : ('b', 1)

⇒ immutability

* since tuples are immutable, the elements in the tuple cannot be modified.

eg) t = (10, 20, 30, 40)
t[0] = 40
print(t)

o/p : TypeError: 'tuple' object does not support item assignment.

⇒ tuple operation

* len() : returns the no. of elements in tuple

eg) t = (1, 2, 3, 4, 5, 6)
print (len(t))

o/p
6

* concatenation : '+' operator

eg) t1 = (1, 2, 3)
t2 = (4, 5, 6)
t3 = t1 + t2
print(t3)

o/p : (1, 2, 3, 4, 5, 6)

* repetition : '*' operator

eg) t1 = (1, 2)
print (t1 * 3)

o/p : (1, 2, 1, 2, 1, 2)

3. membership : in and not in

eg) t = (1, 2, 3, 4)
print (5 not in t)

o/p : True

* comparison : all relational operators (==, !=, >, <, >=, <=) can be applied on tuples

eg) t1 = (1, 2, 3, 4)
t2 = (4, 1, 2, 3)
print (t1 > t2)

o/p : False

[note: compares the corresponding element in the given tuples)

* maximum : returns the maximum element in tuple

eg) t = (1, 2, 3, 4)
print (max (t))

o/p : 4

* minimum : returns the minimum element in tuple

eg) t = (1, 2, 3, 4)
print (min (t))

o/p : 1

* tuple() : converts a sequence into a tuple

eg) str = "hello"
t = tuple(str)
print(t)

o/p : ('h', 'e', 'l', 'l', 'o')

tuple assignment ✓ (1)

* It allows a tuple of variables on the left side of the assignment operator and its corresponding values on the right side.

eg) a, b, c = 1, 2, 3
a, b = b, a
print (a, b, c)

o/p : 2 1 3

* The no. of var on the left and the no. of values on the right should be same

eg) a, b, c = 2+4, 9%6, 4-2
print (a, b, c)

o/p : 6 3 2

* expressions are evaluated before assignment

* A tuple can be assigned to another tuple

eg) t = (2, 4, 9) # tuple packing
(a, b, c) = t # tuple unpacking
print (a, b, c)
print (t)

o/p
2 4 9
(2, 4, 9)

* tuple packing: values on the left are 'packed' together in a tuple

Tuple as return values (7)

* Function can always return only a single value, but by making the value a tuple, more than one value can be grouped together and returned from function.

eg)
```
def circle(r):
    c = 2 * 3.14 * r
    a = 3.14 * r * r
    return (c,a)  # returns multiple values

r = int(input())
(circum, area) = circle(r)
print(circum, area)
```

O/P : 2
12.56 12.56

DICTIONARIES

* Dictionary is a datatype that stores values as a pair of key and value.
* Each key is separated from its value by a colon(:) and consecutive items are separated by commas.
* The entire items in a dictionary are enclosed in curly brackets { }.
* key must be unique and be of any immutable datatype (strings, number or tuples).
* value of a key can be of any type.
* dictionary are mutable datatype.
* Dictionary represents mapping and not sequence.

Syntax

```
dict_name = {key_1 : value_1 , . . . , key_n : value_n}
```

eg

```
dt = { 1 : "x" , 2 : "y" }
print (dt)
```

o/p

```
{1 : 'x' , 2 : 'y' }
```

creating a dictionary

* creating empty dictionary

eg) dt = { }
print(dt)

o/p
{ })

* creating new dictionary with no items using the built-in function dict()

eg) dt = dict()
print(dt)

o/p
{ }

* creating dictionary by specifying key-value pair.

eg) dt = {'name': 'Monty', 'roll': 1117 }
print(dt)

o/p
{'name': 'Monty', 'roll': 1117 }

⇒ Accessing values

* The elements of a dictionary are not indexed instead the keys are used to look up the corresponding values.

eg) dt = {'name': 'monty', 'roll': 1117}

```
print (dt['roll'])
```

O/P

1117

* If the key is not in the dictionary, then a KeyError is generated.

* using for loop, to loop over a dictionary to access only key, only values and even both.

eg) dt = {'name': 'monty', 'roll': 1117}

```
for key in dt:
    print(key, end=" ")
for val in dt.values():
    print(val, end=" ")
For key, val in dt.items():
    print(key, val, end=" ")
```

o/p

name roll

monty 1117

name monty roll 1117

⇒ Adding / Modifying an item in a dictionary

syntax

dict_name[key] = value

eg)

```
dt = {'name' : 'monty', 'roll' : 1117}
dt['roll'] = 1115        # modify
dt['mark'] = 195         # Adding
print(dt)
```

o/p

{'name' : 'monty', 'roll' : 1115, 'mark' : 195}

⇒ deleting items

* 'del' keyword can delete items in the dictionary as well as delete the dictionary itself.
* clear() function can remove all items in the dictionary.

eg)

```
dt = {'name':'monty', 'roll':1117}
del dt['roll']      # using 'del' keyword
print(dt)
dt.clear()
print(dt)           # using clear() function
```

o/p:

```
{'name':'monty'}
{}
```

OPERATIONS AND METHODS

⇒ len() : returns the length of dictionary, i.e., no. of items

eg)
```
dt = {'name':'monty', 'roll':1117}
print(len(dt))
```

o/p: 2

⇒ clear() : deletes all entries in the dictionary

eg)
```
dt = {'name':'monty', 'roll':1117}
dt.clear()
print(dt)
```

o/p: {}

() : returns a reference of the dictionary, doesn't create a duplicate copy.

eg) dt = {'name': 'monty', 'roll': 117}

```
dt2 = dt.copy()
dt2['roll'] = 1115
print(dt)
```

o/p:

{'name': 'monty', 'roll': 1115}

⇒ keys() : returns a list of keys in the dictionary

eg) dt = {'name': 'monty', 'roll': 1117}

```
print(dt.keys())
```

o/p:

['name', 'roll']

⇒ values() : returns a list of values in dictionary

eg) dt = {'name': 'monty', 'roll': 1117}

```
print(dt.values())
```

o/p:

['monty', 1117]

⇒ items() : returns a list of tuples (key-value

eg) dt = {'name' : 'monty', 'roll' : 1117}

print (dt.items())

o/p :

[('name', 'monty'), ('roll', 1117)]

⇒ get (key) : returns the value for the key passed as argument.

eg) dt = {'name' : 'monty', 'roll' : 1117}.

print (dt.get ('name'))

o/p

monty

⇒ in and not in operator : checks whether a given key is present in dictionary or not

eg) dt = {'name' : 'monty', 'roll' : 1117}

print ('name' in dt)

print ('name' not in dt)

o/p.

True

False

⇒ update() : Adds the key-value one dictionary to the key-value pairs of another dictionary

eg) dt1 = {'name' : 'monty', 'roll' : 1117}

dt2 = {'marks' : 195}

dt1.update(dt2)

print(dt1)

O/P

{'name' : 'monty', 'roll' : 1117, 'marks' : 195}

⇒ has_key(key) : returns True if the key is present in the dictionary and False otherwise.

eg) dt = {'name' : 'monty', 'roll' : 1117}

print(dt.has_key('roll'))

O/P

True

ILLUSTRATIVE PROGRAMS

SELECTION SORT (X)

* Sorting is an operation in which all the elements of a list are arranged either in ascending / descending order.

Procedure for selection sort

* It finds the smallest element in the list and exchanges it with the element present at the head (first) of the list.
* Now the list is divided into two parts sorted and unsorted.
* The first element forms the sort part and rest of the list forms the unsorted part.
* Again the smallest element in the unsorted part is exchanged with the element at the head of unsorted part.
* procedure repeated till list is sorted
* 2 important steps in selection sort are selection and exchange.
* For 'n' elements, n-1 pass is required.

eg) [25] 57 48 37 (12)

⇒ select the smallest element

⇒ exchange with the element at the head

12 | [57] 48 37 (25)

sorted unsorted

12 25 | [48] 37 (57)

12 25 37 | (48) 57

12 25 37 48 | 57

12 25 37 48 57 ⇒ sorted list

program

```
l = [25, 57, 48, 37, 12]
for i in range(len(l)):
    smallest = min(l[i:])
    ind = l.index(smallest)
    l[i], l[ind] = l[ind], l[i]
print(l)
```

o/p

[12, 25, 37, 48, 57]

INSERTION SORT

procedure (1)

* Given a list of numbers, it divides the list in two part - sorted part and unsorted part.
* The first element becomes the sorted part and the rest of the list becomes the unsorted part
* Then it picks up one element from the front of the unsorted part and inserts it at its proper position in the sorted part of the list.
* This insertion action is repeated till the unsorted part diminishes.

eg) 12 1 8 10 5 3

⇒ list divided into sorted and unsorted array

12 | (1) 8 10 5 3

sorted unsorted

⇒ pick the 1st element from unsorted list and insert in the proper position in sorted list.

1 12 | (8) 10 5 3

sorted unsorted

1 8 12 | (10) 5 3

1 8 10 12 | (5) 3

1 5 8 10 12 | (3)

1 3 5 8 10 12

program

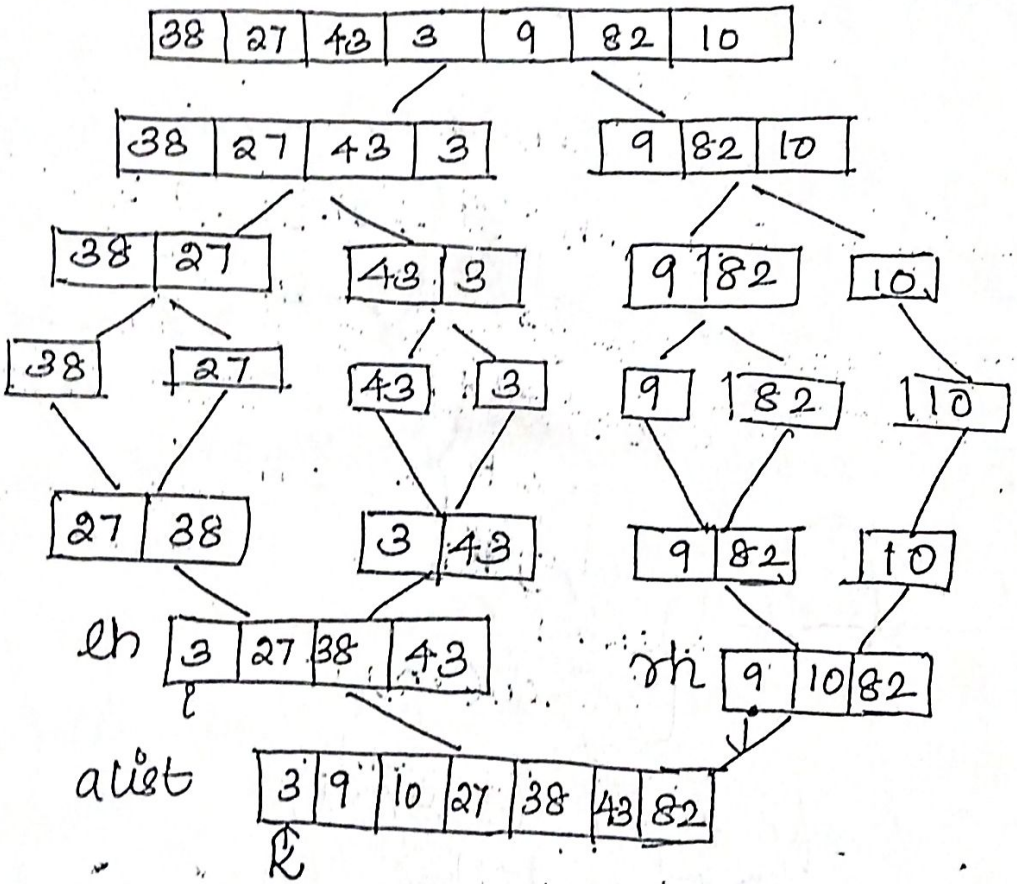
```
l = [12,1,8,10,5,3]
for i in range (1,len(l)):
    first = l[i]
    j = i
    while ((j>0) and l[j-1]>first):
        l[j] = l[j-1]
        j = j-1
    l[j] = first
print(l)
```

o/p

[1,3,5,8,10,12]

MERGE SORT

* merge sort is a divide and conquer algorithm
* It divides the input array into two halves, and the process is repeated again on the two halves and so on till the size of the array becomes 1
* Then merge and sort recursively, each sub-array to form a single sorted list.



program:

```
def merge(alist):
    if(len(alist)>1):
        mid = len(alist)//2
        lh = alist[:mid]
        rh = alist[mid:]
        merge(lh)
        merge(rh)
        i=j=k=0
        while(i<len(lh) and j<len(rh)):
            if(lh[i]<rh[j]):
                alist[k]=lh[i]
                i=i+1
            else:
                alist[k]=rh[j]
                j=j+1
            k=k+1
        while(i<len(lh)):
            alist[k]=lh[i]
            i=i+1
            k=k+1
        while(j<len(rh)):
            alist[k]=rh[j]
            j=j+1
            k=k+1
alist=[38,27,43,3,9,82,10]
merge(alist)
print("sorted list:",alist)
```

o/p: sorted list: [3,9,10,27,38,43,82]

HISTOGRAM

* It is a statistical term for a collection of frequencies.

* The following example prints the frequency of characters in a given string using dictionary.

Program

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] = d[c]+1
    return d

h = histogram('parrot')
print(h)
for x in h:
    print(x, h[x])
```

P:

{'a': 1, 'p': 1, 'r': 2, 't': 1, 'o': 1}

('a', 1)

('p', 1)

('r', 2)

('t', 1)

('o', 1)