**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.

**Objective:**

To use Python data structures –- lists, tuples, dictionaries

**Outcome:**

Represent compound data using Python lists, tuples, dictionaries

**4 COMPOUND DATA**

Primitive data types are basic data types such as int, bool and float. Compound data is any data type which is constructed using primitive data types and other compound data types. Python offers different compound data types (sequences) such as lists, tuples and dictionaries.

**4.1 LISTS**

List is the collection (bag) of objects. We extensively use list to store and manipulate data in everyday computing.

**Examples**

1. List of web pages matching the keyword (google)
2. List of friends (facebook)
3. List of products prices (amazon)
4. List of tasks to do
5. List of grocery items to be purchased
6. List of students enrolled in a class

The objects in the list can be of same type or of different types.

>>> grocery = ['bread', 'butter', 'milk']

>>> absentees = [3, 14, 24, 35, 37, 41]

>>> movie\_review = ['enthiran', {'5-rating':344, '4-rating': 28, '3-rating':0}]

>>> my\_friends = ['akil', 'kapil', 'dhoni']

>>> my\_favorite\_menu = ['idli','dhosa','pongal']

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: list() or list(iterable)

**4.1.1 LIST OPERATIONS**

**repeat (\*)**

>>> mylist = [1, True, 'python']

>>> mylist \* 2

[1, True, 'python',1, True, 'python']

**concatenate (+)**

>>> part1 = ['python','is']

>>> part2 = ['all', 'purpose', 'language']

>>> part1 + part2

['python','is','all', 'purpose', 'language']

**empty list**

>>> a = []

>>> not a

True

**index**

>>> mylist = [12, 48, 12, 72, 34, 21]

>>> mylist[1]

48

>>> mylist[0]

12

**Exercises**

1. What is the output?

>>> a = 10

>>> mylist = [a]\*5

>>> mylist[3]

1. What is the output?

>>> mylist1 = ['In', 'python']

>>> mylist2 = ['explicit','is','better']

>>> mylist = mylist1 + mylist2

>>> mylist += ['than','implicit']

>>> mylist

**4.1.2 LIST SLICES**

We can select the specific subset from the list using slicing. We can either use a positive index (forward) or negative index(reverse) to refer the particular element or slice in the list.

| **Forward index** | **0** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- | --- |
| mylist | 12 | 48 | 12 | 72 | 34 | 21 |
| Reverse index | -6 | -5 | -4 | -3 | -2 | -1 |

**Example**

>>> mylist = [12, 48, 12, 72, 34, 21]

>>> mylist

[12, 48, 12, 72, 34, 21]

>>> mylist[2]

12

>>> mylist[-2]

34

>>> mylist[3]

72

List may be sliced into part, from start till end.

mylist[start:end:step]

The elements are picked in steps from start. If step is not mentioned, it is taken as 1 as default. The element at end is not included.

**Example**

>>>mylist = [12, 48, 12, 72, 34, 21]

>>> mylist[1:3]

[48, 12]

>>> mylist[2:-2]

[12, 72]

>>> mylist[0:3]

[12, 48, 12]

>>> mylist[:3]

[12, 48, 12]

>>> mylist[3:]

[72, 34, 21]

# Elements at odd indices

>>> mylist[::2]

[12, 12, 34]

# In reverse order

>>> mylist[::-1]

[21, 34, 72, 12, 48, 12]

>>> mylist[::-2]

[21, 72, 48]

**4.1.3 LIST METHODS**

**count(x)**

return the number of times x appears in the list.

>>> mylist = [12, 12, 34, 34, 34]

>>> mylist.count(34)

3

**index(x)**

return: the index of first occurence of x

>>> mylist.index(34)

2

**insert(index,x)**

insert an item at a given position(index).

>>> mylist.index(3,34)

# insert 34 at 3

list.append(x) Add an item to the end of the list; equivalent to a[len(a):] = [x].

list.extend(L) Extend the list by appending all the items in the given list; equivalent to a[len(a):] = L.

list.remove(x) Remove the first item from the list whose value is x.

list.pop([i]) Remove the item at the given position in the list, and return it.

list.sort() Sort the items of the list in place

list.reverse() Reverse the elements of the list, in place.

Associated methods and attributes of a list may be viewed with dir(mylist).

Exercises:

1. What is the error?

>>> mylist = [12, 48, 34, 72, 56]

>>> mylist.pop(2)

>>> mylist.append(mylist.index(34))

1. What is the output?

>>> mylist = [12, 48, 34, 72, 56]

>>> mylist.remove(34)

>>> mylist.insert(2,2)

>>> mylist.sort()

>>> mylist.reverse()

>>> mylist.append(mylist.count(2))

>>> mylist

**4.1.4 LIST LOOP**

List is the collection of iterable items. Using for loop, you can process each element in the list.

**Example**

Find the maximum number in the list

def get\_maxnumber(numbers):

maxval = None

for element in numbers:

if not maxval or element > maxval:

maxval = element

return maxval

# test

mylist = [1, 5, 67, 34, 128]

print(get\_maxnumber(mylist))

**Exercise**

1. Find the sum of N numbers (using List)
2. Create list with the following pattern for the input num:

Example:

num = 4 mylist = [4, 8, 12, 16, 12, 8, 4]

num = 3 mylist = [ 3, 6, 9, 6, 3]

1. Create list with the following pattern for the input num:

Example

num = 4 mylist = [1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15]

num = 3 mylist = [1, 2, 4, 5, 7, 8]

1. Write a function to find the factorial of ‘n’?
2. Find the sum of ‘n’ terms of the series

f = 0! + 1! + 2! + … + n! (n >= 0)

1. Find whether n is the factorial number

**4.1.6 Aliasing**

If an object is referred by more than one variable name, it is aliased.

>>> a = [1, 2, 3]

>>> b = a

>>> id(a), id(b)

(140143212216136, 140143212216136)

[](https://github.com/ashok-cs/PSP/raw/master/img/aliasing.jpg)

As list is mutable, a change by one reference is reflected in other reference, as both refer to the same list object.

>>> b[1] = 100

>>> a

[1, 100, 3]

**Exercise**

What is the output?

>>> a = [12,'python',True]

>>> b = a

>>> b[2] = False

>>> id(a) == id(b)

**4.1.7 Cloning Lists**

**L.copy()**

create a shallow copy of L

>>> a = [2,3,4,[5,6]]

>>> b = a.copy()

>>> b[3][1] = 8

>>> a

[2, 3, 4, [5, 8]]

>>> id(b[3]),id(a[3])

(140522783809160, 140522783809160)

**deepcopy**

In shallow copy, the nested sublists are not cloned (same id). In deep copy, they are cloned (different id).

>>> from copy import deepcopy

>>> a = [2,3,4,[5,6]]

>>> b = deepcopy(a)

>>> b[3][1] = 8

>>> a

[2, 3, 4, [5, 6]]

>>> id(b[3]),id(a[3])

(140522755061704, 140522783746376)

**Exercise**

1. Modify the program to get the desired output

>>> old\_stock = [['item1',23],['item2',34],['item3',45]]

>>> new\_stock = old\_stock.copy()

# Add 10 to each item

>>> for i in range(3):

new\_stock[i][1] += 10

# old\_stock should not be changed

**4.1.8 List parameters**

When the list is passed to a function as parameter, the parameter refers to the same object. Hence any change in the function gets reflected in the calling stack as well.

**Example**

def fun(mylist):

mylist[2] = 'python'

del mylist[3:]

# Test

olist=[1,2,3,4,5,6,7]

print("before calling:",olist)

fun(olist)

print("after calling:",olist)

ouput:

before calling: [1, 2, 3, 4, 5, 6, 7]

after calling: [1, 2, 'python']

Write the function ‘chop’ that takes a list, modifies it by removing the first and last elements and returns None.

>>> def chop(arglist):

del arglist[0]

del arglist[-1]

>>> mylist = [1, 2, 3, 4, 5, 6, 7, 8]

>>> chop(mylist)

>>> mylist

[2, 3, 4, 5, 6, 7]

**Exercise**

1. Write a function cat\_num which takes a list, say, [1,2,3,4,5] and modifies to [11,22,33,44,55] (concatenates each element itself) and returns None.

**4.2 Tuples**

List is the mutable sequence (append, remove, insert, pop, reverse, sort, extend and copy methods modify the list). Tuple is the immutable sequence. Only common methods for tuple and list are index() and count().

**4.2.1 Tuple Assignment**

Multiple variables can be assigned using tuple assignment (tuple unpacking). Parentheses are optional.

>>> (a,b,c) = (12,34,48)

>>> a

12

>>> a,b,c

(12, 34, 48)

**Exercise**

1. What is the output

>>> a,b,c = 10, 00, 000

>>> (a, b, c)\*2

>>> a,b,c

**4.2.2 Tuple as return value**

Mutiple variables can be returned from the function using tuple. Parantheses are optional.

>>> def myswap(num1, num2):

return (num2, num1)

>>> a,b = 12,34

>>> a,b = myswap(a,b)

>>> a,b

(34, 12)

**Exercise**

1. Write the function quotient\_reminder to return quotient and reminder of a/b

**4.3 Dictionaries**

Lists and tuples are ordered sequence. The elements are accessed using index. Dictionary is the unordered sequence. The elements are accessed using key.

>>> days = {'jan':31, 'feb':28, 'mar':31}

>>> days['jan']

31

**4.3.1 Operations and methods**

In dictionaries, the elements are stored as “key-value” pair. keys() return all the keys in the dictionary. values() return all the values in the dictionary. All the items are iterable in dictionary.

>>> for mon in days:

days[mon]

31

28

31

**Example**

Find the number of donors – blood group wise.

def blood\_donors(dataset):

# empty dictionary

donors = {}

for blood\_group in dataset:

if blood\_group in donors:

donors[blood\_group] += 1

else:

donors[blood\_group] = 1

return donors

#global frame

dataset = [ 'O+', 'B+', 'B-', 'O-', 'O+',

'B+', 'B+', 'O+', 'O+']

print(blood\_donors(dataset))

**Output**

{'B+': 3, 'O+': 4, 'O-': 1, 'B-': 1}

**Exercise**

1. Write the function letters\_freq to find the frequency of letters in a string. Return the result as the dictionary.
2. Find the capital for the given country from the imported dictionary capital

from country import capital

def find\_capital(country):

# your code

1. Find the country for the given capital.

from country import capital

def find\_country(capital):

# your code

1. Find the countries for the given capitals.

from country import capital

def find\_countries(capitals):

# your code

Example: input = [‘New Delhi’,’Washington DC’] output = [‘India’,’US’]

**4.4 Advanced List Processing**

**4.4.1 List Comprehension**

List comprehension is the pythonic way (one liner) to write the list loop.

**Example**

Find the sum of odd numbers in the list.

>>> mylist = [1, 2, 3, 4, 5, 6, 7, 8]

>>> sumval = 0

>>> for element in mylist:

if element % 2 != 0:

sumval += element

>>> sumval

16

>>> 1+3+5+7

16

This can be written in one line using list comprehension.

>>> sumval = sum([d for d in mylist if d % 2 != 0])

>>> sumval

16

Find the pass percentage

>>> marks = [ 84, 65, 59, 45, 34, 12, 98, 29]

>>> pass\_count = len([d for d in marks if d>=50])

>>> total = len(marks)

>>> pass\_count/total

0.5

Example: Remove duplicates from the list (using dictionary)

>>> mylist = [12, 12, 34, 12, 34, 12]

>>> mylist = list({d:1 for d in mylist})

>>> mylist

[12, 34]

>>>

**4.5 Illustrative Programs**

**4.5.1 Selection sort**

**Exercises**

1. Assume that first number in the list is minimum. Exchange, if first> second Example

input = [12,3,15,7,23] output = [3,12,15,7,23]

1. Assume that first element in the list is minimum. Compare it with every other element. Exchange if it is greater. (index selected = 0)

### Example

[12,23,15,7,3] As 12<23, don’t exchange.

[12,23,15,7,3] As 12<15, don’t exchange.

[12,23,15,7,3] As 12>7, exchange

[7,23,15,12,3] As 7>3, exchange

[3,23,15,12,7] Stop.

1. Now, the first element is the minimum. Now, bring the next minimum value in the list as the second element. (index selected = 1)

### Example

[3,23,15,12,7] As 23>15, exchange

[3,15,23,12,7] As 15>12, exchange

[3,12,23,15,7] As 12>7, exchange

[3,7,23,15,12] stop

If we continue to place the subsequent minimum values, we get the sorted list.

| **selected index (outer loop)** | **numbers** |
| --- | --- |
| Before sorting | 12 3 45 17 15 |
| 0 | 3 12 45 17 15 |
| 1 | 3 12 45 17 15 |
| 2 | 3 12 15 45 17 |
| 3 | 3 12 15 17 45 |

Selected index: 2 sorted in steps

| **After inner iteration (j)** | **numbers** |
| --- | --- |
| before sorting | 3 12 45 17 15 |
| 3 | 3 12 17 45 15 |
| 4 | 3 12 15 45 17 |

**Algorithm**

1. Select an index (i) successively from 0 to len(numbers)-2
2. Compare numbers[i] with each element in the remaining list
3. Swap numbers[i] with the element whenever numbers[i] is larger

**Pseudocode**

selection\_sort(numbers):

N=len(numbers)

for index in range(N-1):

for j in range(index+1,N):

If numbers[index] > numbers[j]:

swap (numbers[index], numbers[j])

**Implementation**

def selection\_sort(numbers):

N = len(numbers)

for index in range(N-1):

for j in range(index+1,N):

if numbers[index] > numbers[j]:

numbers[index], numbers[j] = numbers[j], numbers[index]

print("Selected index:",index, numbers)

#Test

mylist=[12,3,45,17,15]

print("Before sorting:",mylist)

selection\_sort(mylist)

print("After sorting:",mylist)

**Output**

Before sorting: [12, 3, 45, 17, 15]

Selected index: 0 [3, 12, 45, 17, 15]

Selected index: 1 [3, 12, 45, 17, 15]

Selected index: 2 [3, 12, 15, 45, 17]

Selected index: 3 [3, 12, 15, 17, 45]

After sorting: [3, 12, 15, 17, 45]

**4.5.2 Insertion sort**

**Exercises**

1. Consider the second element in the list num. Insert at index 0, if element < first. hint: use insert()
2. Remove element if it is inserted. hint: use pop() or remove
3. Now num[0:1] is in sorted order. Now, consider the third element in the list (num[2]). Compare with first two elements. Insert at 0, if element is less than first. Insert at 1, if element is less than second. Remove num[2], if it is inserted.

Subsequently, the list num gets sorted.

| **i** | **position to be inserted** | **num** |
| --- | --- | --- |
| 1 | 0 | 12 3 45 17 15 |
| 2 | 2 | 3 12 45 17 15 |
| 3 | 2 | 3 12 45 17 15 |
| 4 | 2 | 3 12 17 45 15 |
| sorted | 3 12 15 17 45 |  |

**Pseudocode**

insertion\_sort(num):

for i in range(1,len(num)):

element = num[i]

inserted = False

for j in range(i):

if element < num[j]

insert element at j

and break loop

if inserted:

remove element from i

**Implementation**

def insertion\_sort(num):

for i in range(1,len(num)):

element = num[i]

for j in range(i):

if element < num[j]:

print(num,"insert",element,"at",j)

num.insert(j,element)

num.pop(i+1)

break

#Test

mylist = [12,3,45,72,15]

insertion\_sort(mylist)

print(mylist)

**Pseudocode (v2)**

insertion\_sort(num):

for i in range(1,len(num)):

element=num[i]

j=i

while j > 0 and num[j-1] > element:

num[j]= num[j-1]

num[j]= element

Implementation (v2)

def insertion\_sort(num):

for i in range(1,len(num)):

element = num[i]

for j in range(i):

if element < num[j]:

print(num,"insert",element,"at",j)

num.insert(j,element)

num.pop(i+1)

break

# Test

mylist = [12,3,45,17,15]

insertion\_sort(mylist)

print(mylist)

Output

[12, 3, 45, 17, 15] insert 3 at 0

[3, 12, 45, 17, 15] insert 17 at 2

[3, 12, 17, 45, 15] insert 15 at 2

[3, 12, 15, 17, 45]

**4.5.3 Merge sort**

It is divide recursively and conquer approach.

**Exercise**

1. Consider left and right lists of size 1. Merge them in a sorted order. Example:

left = [12] right = [3]

merged = [3,12]

1. Now consider the two sorted lists of unspecified size. Merge them in a sorted order. Example:

left = [12,45] right = [3,17]

merged = [3,12,17,45]

1. Divide the list num into left and right halves.
2. Recursively divide, till the partition size is 1 Example:

num = [12,3,45,17,15]

left = [12,3]

left = [12]

right = [3]

right = [45, 17, 15]

left = [45]

right = [17,15]

left = [17]

right = [15]

**Algorithm**

1. Divide the list recursively to left and right halves, till the partition size is 1
2. Merge the left and right halves in the sorted order

**Algorithm for merge**

1. Remove the minimum of two lists left and right and add it to the merged list till left or right becomes empty.
2. Append the remaining elements of left and right to merged list

Note: Both left and right are in sorted order, before merging.

Pseudo code

merge\_sort(num)

return divide(num)

divide(num)

if num is empty or len(num) is 1:

return num

mid = len(num)/2

left = divide(num[:mid])

right = divide(num[mid:])

merge(left,right)

merge(left,right)

merged\_list = [ ]

while left and right are not empty:

if left[0] < right[0]:

Pop left[0] and add it to merged\_list

else:

Pop right[0] and add it to merged\_list

Append remaining left and right to merged\_list

**Implementation**

def merge\_sort(num):

return divide(num)

def divide(num):

print(num)

if not num or len(num) == 1:

return num

else:

mid = len(num)//2

print("divide left:", end=' ')

left = divide(num[:mid])

print("divide right:", end=' ')

right = divide(num[mid:])

return merge(left,right)

def merge(left, right):

merged\_list = []

print("merging:",left,right,end=' ')

while left and right:

if left[0] < right[0]:

merged\_list += [left.pop(0)]

else:

merged\_list += [right.pop(0)]

merged\_list += left

merged\_list += right

print("merged:",merged\_list)

return merged\_list

# Test

mylist=[12,3,45,17,15]

print("Before sorting:",mylist)

mylist = merge\_sort(mylist)

print("After sorting:",mylist)

**Output**

Before sorting: [12, 3, 45, 17, 15]

[12, 3, 45, 17, 15]

divide left: [12, 3]

divide left: [12]

divide right: [3]

merging: [12] [3] merged: [3, 12]

divide right: [45, 17, 15]

divide left: [45]

divide right: [17, 15]

divide left: [17]

divide right: [15]

merging: [17] [15] merged: [15, 17]

merging: [45] [15, 17] merged: [15, 17, 45]

merging: [3, 12] [15, 17, 45] merged: [3, 12, 15, 17, 45]

After sorting: [3, 12, 15, 17, 45]

**4.5.4 Quick Sort**

**Exercises**

1. Select last element of the list num as pivot.
2. Find from the front, which element is larger than or equal to pivot (num[front]) Find from the rear next to pivot, which element is smaller than pivot (num[rear]) Swap num[front] and num[rear] if front < rear
3. Repeat step 2 till front <= rear
4. Now the first half of num holds values smaller than pivot. Second half of num excluding pivot holds vlaues larger than pivot. Now, front points to the start of the larger partition. Swap pivot and num[front]. to bring pivot to the middle. Example num = [12,3,17,45,15,12]

**Algorithm**

1. Pick last element as a pivot from the num list
2. Divide num into small and large partitions which contain elements smaller or larger than pivot
3. Recursively divide till partition size becomes 1

**Pseudocode**

Qsort(num,firt,last):

pivot=last

front=first

rear=last-1

while front < rear:

increment front till num[front]< pivot

decrement rear till num[rear] >= pivot

if front <rear:

swap num[front],num[rear]

else:

break

swap num[front],pivot

Qsort(num,first, front-1) # partition small recursively

Qsort(num,front+1,last) # partition large recursively

**Implementation**

def quick\_sort(num):

if len(num)<=1:

return

Qsort(num,0,len(num)-1)

def Qsort(num,first,last):

print(num[first:last+1])

if first >= last:

return

pivot=last

front=first

rear=last-1

print("pivot=",num[pivot])

while front <= rear:

while num[front] < num[pivot] and front <= last:

front += 1

while num[rear] >= num[pivot] and rear >= first:

rear -= 1

if front < rear :

num[front],num[rear] = num[rear],num[front]

else:

break

num[front],num[pivot] = num[pivot],num[front]

if first <= front-1:

print("partition small", end=' ')

Qsort(num,first, front-1)

if front+1 <= last:

print("partition large", end=' ')

Qsort(num,front+1,last)

# Test

num=[12,3,17,45,15,12]

quick\_sort(num)

print(num)

Output

[12, 3, 17, 45, 15, 12]

pivot= 12

partition small [3]

partition large [17, 45, 15, 12]

pivot= 12

partition large [45, 15, 17]

pivot= 17

partition small [15]

partition large [45]

[3, 12, 12, 15, 17, 45]