# UNIT IV

# Part A

**1. What is the difference between selection sort and insertion sort.**  
In selection sort, select the minimum value from the unordered list and place it in the ordered list (index).  
Insertion sort always maintains a sorted sublist in the lower positions of the list. Each new item is then “inserted” back into the previous sublist such that the sorted sublist is one item larger.

**2. Write a python code to display the sum of even numbers upto 100.**

print(sum(d for d in range(101) if d % 2 == 0))

**3. How do you view all the keys of a dictionary.**  
keys() return all the keys in the dictionary.  
Example:  
>>> d = {‘math’:56, ‘phy’: 72, ‘chem’: 84}

>>> d.keys( )  
[‘math’, ‘phy’, ‘chem’]

**4. What is meant by tuple unpacking.**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)

**5. Write a function to count the number of occurrences of a key element in a list.**

def **count(L, key):**  
 c = 0  
 for element in L:  
 if element == key:  
 c += 1  
 return c

**Part B**

**1. Explain and implement selection sort algorithm**

In selection sort, select the minimum value from the unordered list and place it in the ordered list (index).

For example, consider the list to be sorted,

**L =** [12, 23, 15, 7, 3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | sublist (U) | The smallest in the sublist | Index ( j ) of the smallest in  L[ i : ] | L after swapping the smallest L[j] with L[i] |
| 0 | [12, 23, 15, 7, 3] | 3 | 4 | [3, 23, 15, 7, 12] |
| 1 | [ 23, 15, 7, 12] | 7 | 3 | [3, 7, 15, 23, 12] |
| 2 | [15, 23, 12] | 12 | 4 | [3, 7, 12, 23, 15] |
| 3 | [23, 15] | 15 | 4 | [3, 7, 12, 15, 23] |

**After swapping:**

**L =** [3, 7, 12, 15, 23]

**Algorithm:**

**Step 1:** Get started from the first element (i = 0): **Step1.1:** Get the unsorted sub list (U)

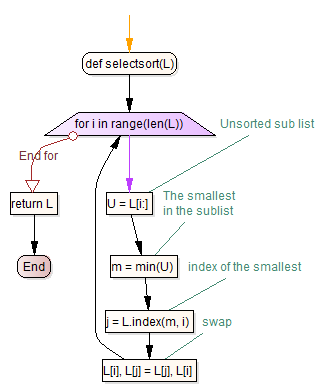
**Step1.2:**  Find the smallest in the sublist (m)

**Step 1.3:** Find the index of the smallest (j)

**Step 1.4:** Swap the smallest with the element at index (i)

**Step 2:** Increment the index (i) and repeat step 1 until list is sorted

**Flowchart:**



**Program:**

def selectsort(L):

for i in range(len(L)):

# Unsorted sub list

U = L[i:]

# The smallest in the sublist

m = min(U)

# index of the smallest

j = L.index(m, i)

# swap

L[i], L[j] = L[j], L[i]

return L

L = [12, 4, 7, 56, 2]

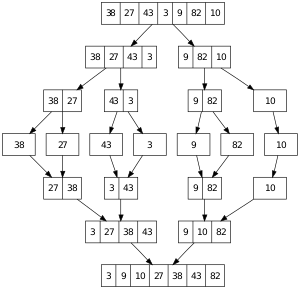
selectsort(L)

print(L)

**Output:**[2, 4, 7, 12, 56]

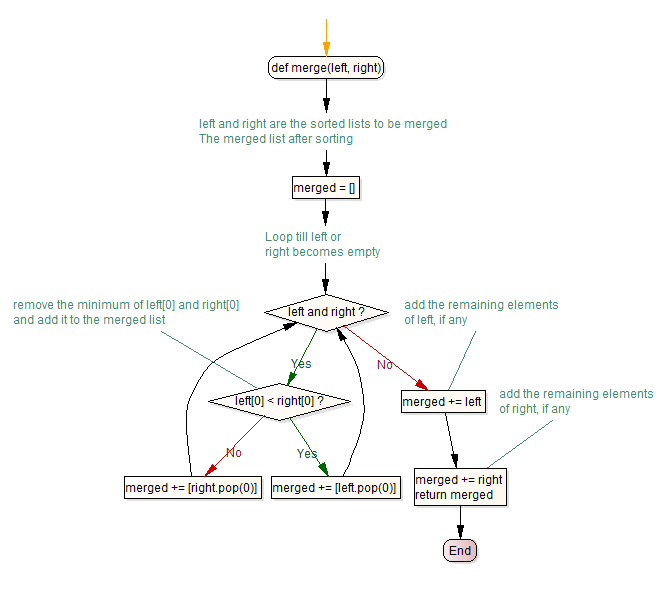
**2. Describe and implement merge sort algorithm**

Merge sort algorithm is the example for divide and conquer algorithm. It repeatedly splits the list and then merge the sorted sublists together. An example is illustrated below.



Recursively split the list into left and right halves, till it contains single element. Merge the ‘left’ and ‘right’ portions in the sorted order to the way up.

**Algorithm for merge( ):**

**Step 1:** Get thesorted lists`left` and `right` to be merged.  
**Step 2:** Initialize the empty `merged` list  
**Step 3:** Loop till `Left` or `right` becomes empty:  
 Step 3.1: Remove the minimum of the left[0] and right[0]  
 and add it to the merged list  
**Step 4:** Add the remaining elements of left/ right to the `merged` list  
**Step 5:** Return the `merged` list  
****

**Program:**

def mergesort(numbers):

# check for the empty list or the list with single element

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

return numbers

# recursive splitting (Divide)

else:

mid = len(numbers)//2

left = mergesort(numbers[:mid])

right = mergesort(numbers[mid:])

return merge(left, right)

def merge(left, right):

# left and right are the sorted lists to be merged

# The merged list after sorting

merged = []

# Loop till left or right becomes empty

while left and right:

# remove the minimum of left[0] and right[0]

# and add it to the merged list

if left[0] < right[0]:

merged += [left.pop(0)]

else:

merged += [right.pop(0)]

# add the remaining elements of left, if any

merged += left

# add the remaining elements of right, if any

merged += right

return merged

**3. Illustrate list loop and list mutability with suitable examples.**

**List mutability**

An immutable object has a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered.

Only exceptions are lists, sets and dictionaries, which are mutable (changeable). The id of the object is the same even after the value is changed.

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

>>> id(mylist)

39075392

>>> mylist[1] = 100

>>> mylist

[1, 100, 3]

>>> id(mylist)

39075392

If the list is passed as the argument, any change to the list in the function affects the list in the calling stack as well.

>>> def change(L):

L[0] = 'python'

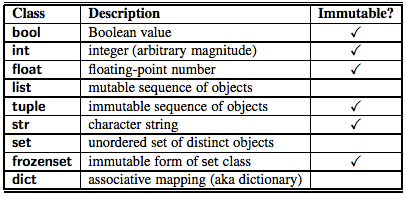
>>> mylist = [12, 3, 45, 2]

>>> change(mylist)

>>> mylist

['python', 3, 45, 2]

This is due to list mutability. All other datatypes in python such as int, bool, str and tuple are immutable.



The function chop takes a list, modifies it by removing the first and last elements.

>>> def chop(arglist):

del arglist[0]

del arglist[-1]

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

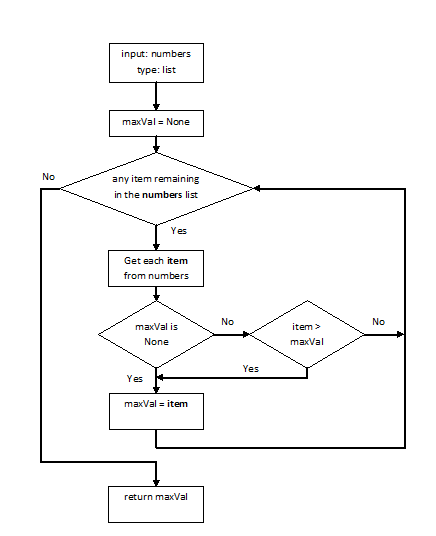
>>> chop(mylist)

>>> mylist

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

**List Loop**

List is the collection of iterable items. Using for construct, you can process each element in the list. For example, find the maximum number in the list using list loop.



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))

The following list loop

for elem in [ 1, 2, 3, 'abc', 99]:

print (elem\*2)

produces the output:

1  
4  
6  
‘abcabc’  
198

Find the sum of odd numbers in the list using list loop.

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

>>> sumval = 0

>>> for element in mylist:

if element % 2 != 0:

sumval += element

>>> sumval

16

List comprehension is the pythonic way (one liner) to write the list loop. It gives the shorter and cleaner code. The above code can be rewritten using list comprehension as,

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

**4. Differentiate list, tuple and dictionary**

# List

List is the ordered sequence. We extensively use list to store and manipulate data in everyday computing.

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

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

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

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

You access the element in the list using index.

>>> my\_friends[1]  
‘kapil’

# Tuple

List is the **mutable sequence** (append, remove, insert, pop, reverse, sort, extend and copy methods modify the list).

>>> mylist.insert(3,10)

# insert 34 at position 3

>>> mylist.append(24)

>>> mylist.remove(34)

Tuple is the **immutable sequence**. Only common methods for tuple and list are index() and count().

# Dictionaries

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

The objects returned from dict.keys(), dict.values(), and dict.items() are called dictionary views. The association of a key and a value is called a key-value pair or sometimes an item.

Example:  
>>> d = {‘math’:56, ‘phy’: 72, ‘chem’: 84}

>>> d[‘math’]  
56

# Part C

1. Write a program that reads a string and returns a table of the letters of the alphabet in alphabetical order which occur in the string together with the number of times each letter occurs. Case should be ignored.

A sample output of the program when the user enters the data

**“ThiS is String with Upper and lower case Letters”,**

would look like:

|  |  |
| --- | --- |
| a | 2 |
| c | 1 |
| d | 1 |
| e | 5 |
| g | 1 |
| h | 2 |

The function ‘**histogram**’ is defined using dictionary to find the frequency of each alphabet in the list.

**def** histogram(s):

d = dict()

**for** c **in** s:

d[c] = 1 + d.get(c, 0)

**return** d

The following function prints the dictionary formed using histogram in the sorted order (alphabetical order).

**def** print\_hist(h):

**for** c **in** sorted(h.keys()):

**print** c, h[c]

Test program:

S = “ThiS is String with Upper and lower case Letters”

freq = histogram(S.lower())  
print\_hist(freq)

Output:

|  |  |
| --- | --- |
| a | 2 |
| c | 1 |
| d | 1 |
| e | 5 |
| g | 1 |
| h | 2 |

2. Write a function called "**remove\_duplicates**" that takes a list and returns a newlist with only the unique elements from the original. Hint: they don’t have to be in the same order.

The following function removes the duplicate entries in the list.

def remove\_duplicates(L):  
 return list(set(L))

Test Program:  
L = [12, 3, 12, 3, 12, 12, 3, 12, 3]  
L = remove\_duplicates(L)  
print(L)  
  
Output:  
[3, 12]

Alternatively, you may also use the dictionary to remove the duplicate elements in the list.  
  
def remove\_duplicates(L):  
 d = dict()  
 for x in L:  
 d[x] = None  
 return list(d.keys())

Test Program:  
L = [12, 3, 12, 3, 12, 12, 3, 12, 3]  
L = remove\_duplicates(L)  
print(L)  
  
Output:  
[3, 12]

Alternatively, you may create a new list with distinct elements.

def remove\_duplicates(L):  
 N = list()  
 for x in L:  
 if x not in N:  
 N.append(x)  
 return N

Test Program:  
L = [12, 3, 12, 3, 12, 12, 3, 12, 3]  
L = remove\_duplicates(L)  
print(L)  
  
Output:  
[3, 12]

3. To check whether a word is in the word list, you could use the "in" operator, but it would be slow because it searches through the words in order. Because the words are in alphabetical order, we can speed things up with a bisection search. You start in the middle and check to see whether the word you are looking for comes before the word in the middle of the list. If so, then you search the first half of the list the same way. Otherwise you search the second half. Write a function called "**bisect**" that takes a sorted list and a target value and returns the index of the value in the list, if it’s there, or None if it’s not.

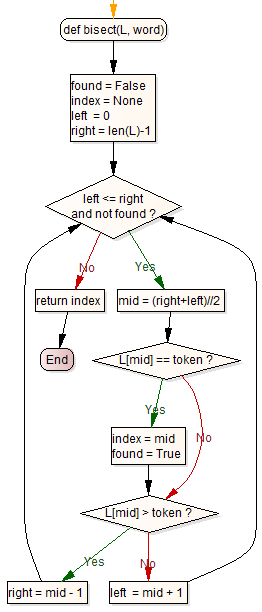
**Algorithm** (bisect or binary search)  
  
**Step1:** Choose the middle element in the list

**Step2:**  If it matches the middle element, its position in the list is returned.

**Step 3:** If the target value is less than or greater than the middle element, the search

continues in the lower or upper half of the array, respectively, eliminating the

other half from consideration



def bisect(L, word):

found = False

index = None

left = 0

right = len(L)-1

while left <= right and not found:

mid = (right+left)//2

if L[mid] == token:  
 index = mid

found = True

if L[mid] > token:

right = mid - 1

else:

left = mid + 1

return index

4. Two words “interlock” if taking alternating letters from each forms a new word. For example, “shoe” and “cold” interlock to form “schooled.” Write a program that finds all pairs of words that interlock.

For example, if all the three words "shoe", "cold" and "schooled" are present in the words.txt, print them.

**schooled shoe cold**

def split\_word(word):

word1 = word[::2]

word2 = word[1::2]

return (word1, word2)

def find\_interlocked(word\_dict):

for word in word\_dict:

split0, split1 = split\_word(word)

if (split0 in word\_dict and split1 in word\_dict):

print word, split0, split1

fd = open('words.txt', 'r')

word\_list = fd.read().splitlines()

word\_dict = {word: None for word in word\_list}

find\_interlocked(word\_dict)