# 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

**6 Define Python List with an example.**

A List is the built-in 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']

**7 What is aliasing of a list?**  
If an object is referred by more than one variable name, it is aliased.

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

>>> b = a

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

In this example, ‘b’ is the alias for ‘a’, both referring to the same object. id(a) is thus equal to id(b). Thus, a change in ‘b’ gets reflected in ‘a’ as well.

>>> b[1] = 100

>>> a

[1, 100, 3]

**8 Illustrate list comprehension with an example.**

List comprehension is the pythonic way (one liner) to write the list loop. It gives the shorter and cleaner code.

Example: Find the sum of odd numbers in the list.

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

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

>>> sumval

16

**9 Illustrate negative indexing in list with an example.**

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[-2]

34

>>> mylist[2:-2]

[12, 72]

>>> mylist[::-1]

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

>>> mylist[::-2]

[21, 72, 48]

**10. Define mutable and immutable data type.**  
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.

**11. What is iterable**  
 An object capable of returning its members one at a time is called iterable.

**12. What is compound data**  
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.

**13. What is cloning of a List?**

Clone is the copy of the list. When a value in the clone is modified, it won’t affect the original list.

Example:

a = [2, 3, 4]

b = a[:]

b[0] = 100

The change in the clone(b) doesn’t affect the values in the original list (a).

**14. Define mutable and immutable data type.**  
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.

**15. What is iterable**  
 An object capable of returning its members one at a time is called iterable.

**16. What is compound data**  
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.

**17. How do you concatenate two lists**  
 Using concatenation operator (+)

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

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

>>> part1 + part2

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

**18. How do you reverse a list using slice.**

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

>>> mylist[::-1]

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

**19. Define dictionary**  
Dictionary is the unordered sequence. The elements are accessed using key. A dictionary is an associative array, where arbitrary keys are mapped to values.

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

>>> days['jan']

31

**20. What are the ordered sequences in python**  
Tuples and Lists are the ordered sequences of the list. The elements are accessed using index

**21. Differentiate append and extend methods**

**list.append(x)** Add an item to the end of the list. Any sublist is appended as is.

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

>>> mylist.append([2,3])

>>> mylist

[12, 12, 34, 34, 34,**[2, 3]**]  
>>> mylist[5]  
[2, 3]

**list.extend(x)** flattens the sublist, before adding its elements to the list.

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

>>> mylist.extend([2,3])

>>> mylist

[12, 12, 34, 34, 34, 2, 3]  
>>> mylist[5]  
2

**22. Illustrate 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)

**23. Illustrate the use of tuple to return multiple values from a function**

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

>>> def divmod(a, b):

return (a // b, a % b)

**24. When do we use dictionary instead of a list**

Dictionaries are well suited, when the data is labeled (i.e, the data has the field name). Dictionaries store key-values pair.

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

>>> days['jan']

Lists are used to store the collection of unlabeled items.  
Search of the key in the dictionary is faster than in the list.

**25. How do you merge two dictionaries?**

>>> dict1 = {‘jan’:31, ‘feb’: 28, ‘mar’: 31}  
>>> dict2 = {‘apr’:30, ‘may’: 31}  
>>> dict1.update(dict2)  
>>>dict1  
{'jan':31, 'feb':28, 'mar':31, 'apr':30, 'may':31}

**Part B Questions**

1. Explain and implement selection sort algorithm

2. Describe and implement merge sort algorithm

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

4. Differentiate list, tuple and dictionary

5. Write a program that reads a string and returns the number of times each alphabet letter occurs in the string (histogram)

6. Write a function to “remove\_duplicates" from a list

7. Write the program to perform binary search (bisect)

8. Illustrate various List methods with suitable examples

9. Explain List slicing with examples

10. Write the program to find the greatest in the list

11. Write the program to perform insertion sort

**Part B Answers**

**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)

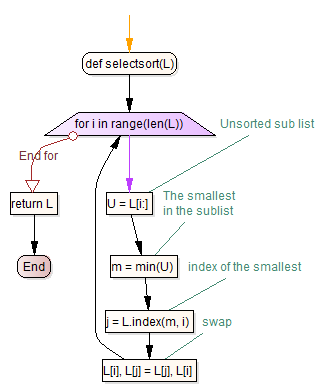
**Step1.2:**  Find the smallest in the sublist (m)

**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]

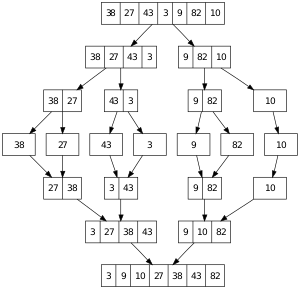
selectsort(L)

print(L)

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

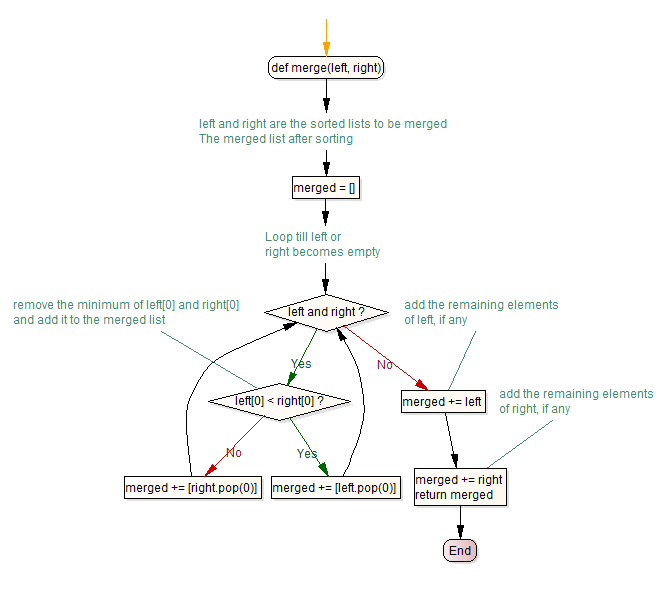
**2. Describe and implement merge sort algorithm**

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):

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

return numbers

else:

mid = len(numbers)//2

left = mergesort(numbers[:mid])

right = mergesort(numbers[mid:])

return merge(left, right)

def merge(left, right):

merged = []

while left and right:

if left[0] < right[0]:

merged += [left.pop(0)]

else:

merged += [right.pop(0)]

merged += left

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'

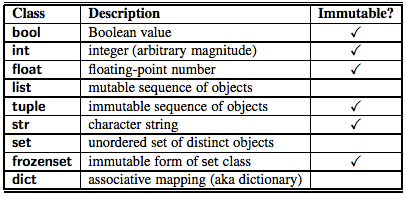
>>> mylist = [12, 3, 45, 2]

>>> 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]

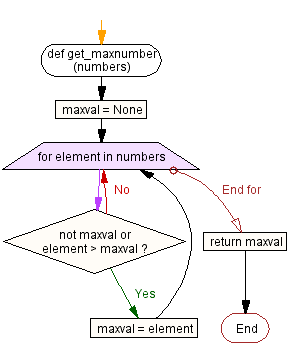
>>> chop(mylist)

>>> 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

**5. Write a program that reads a string and returns the number of times each alphabet letter occurs in the string (histogram)**

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 |

**6. Write a function to “remove\_duplicates" from a list**

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]

**7. Write the program to perform binary search** (bisect)

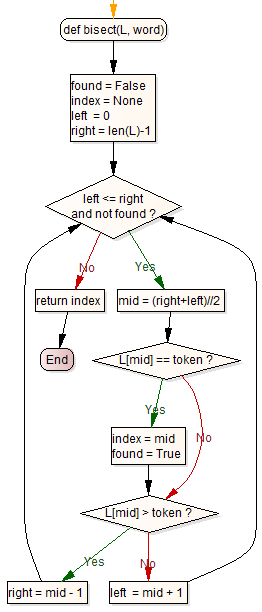
**Algorithm** (bisect or binary search)  
  
**Step1:** Choose the middle element in the list

**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

**8. Illustrate various List methods with suitable examples**

### 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

[12, 12, 34, 34, 34]

>>> mylist.insert(3,10)

# insert 34 at position 3

>>> mylist

[12, 12, 34, 10, 34, 34]

**list.append(x)**

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

>>> mylist

[12, 12, 34, 34, 34]

>>> mylist.append([2,3])

>>> mylist

[12, 12, 34, 34, 34,[2,3]]

**list.extend(L)**

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

>>> mylist

[12, 12, 34, 34, 34]

>>> mylist.extend([2,3])

>>> mylist

[12, 12, 34, 34, 34, 2, 3]

**list.remove(x)**

Remove the first item from the list whose value is x.

>>> mylist

[12, 12, 34, 34, 34]

>>> mylist.remove(34)

>>> mylist

[12, 12, 34, 34]

**list.pop([i])**

Remove the item at the given position in the list, and return it. If i is not mentioned, the last element is popped out.

>>> mylist

[12, 12, 34, 34, 34]

>>> mylist.pop(2)

34

>>> mylist

[12, 12, 34, 34]

>>> mylist.pop()

>>> mylist

[12, 12, 34]

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).

**9. Explain List slicing with examples**

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.

A slice is an object usually containing a portion of a sequence. A slice is created using the subscript notation, [] with colons between numbers when several are given, such as in variable\_name[1:3:5]. The bracket (subscript) notation uses slice objects internally.

| **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]

Slice can be passed as an object as well.

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

>>> start, end, step = 1, None, 2

>>> items = slice(start, end, step)

>>> mylist[items]

[48, 72, 21]

**10. Write the program to find the greatest in the list**

A python function get\_maxnumber  takes as argument a list of numbers

**Algorithm**

Function: **get\_maxnumber**

**Parameters**: numbers (list)

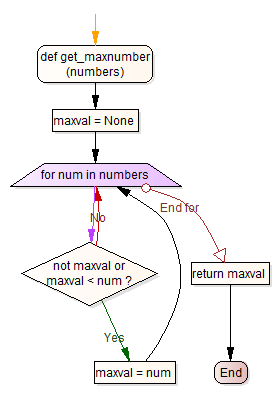
Step 1: initialize the maximum value

Step 2: Loop through the entries in the list

Step 2a: If maxval is not set, update with number

Step 2b: If maxval is less than number, update with number

Step 3: Return the max value



def get\_maxnumber(numbers):

maxval = None

for num in numbers:

if not maxval or maxval < num:

maxval = num

return maxval

The function returns the maximum number in the list. The sequence of steps in the program is:

#1 - initialize the maximum value

#2 - Loop through the entries in the list

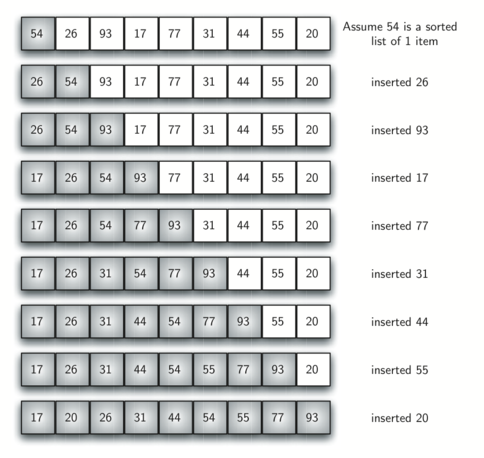
#3 - If maxval is not set, update with number

#3 - If maxval is less than number, update

#4 - Return the max value

**11. Write the program to perform insertion sort**

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.



**Insertion sort:**

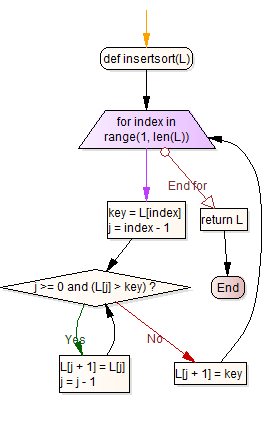
**Step1:** Get the List(L) to be sorted

**Step2:** Iterate index from 1 to len(L)

**Step2.1:**  j = index -1  
 key = L[index]   
**Step 2.2:** Until j > 0 and L[j] > key:  
 Step 2.2.1: L[j+1] = L[j]  
 Step 2.2.2: Decrement j by 1

**Step 2.3:** Insert the key at L[j+1]

**Step 3:** Return the sorted list (L)



def insertsort(L):

for index in range(1, len(L)):

key = L[index]

j = index - 1

while j >= 0 and (L[j] > key):

L[j + 1] = L[j]

j = j - 1

L[j + 1] = key

return L

**OUTPUT**

>>> insertsort([5, 4, 0, 29, 2])

[ 0, 2, 4, 5, 29 ]