**Index**

|  |  |
| --- | --- |
| 1 | Write a Python program to check a number is an Armstrong Number. |
| 2 | Write a Python program to print all Prime numbers in an Interval. |
| 3 | Write a Python program to check if a given number is Fibonacci number? |
| 4 | Write a Python program to print cube sum of first n natural numbers. |
| 5 | Write a Python program to print all odd numbers in a range. |
| 6 | Write a Python program to interchange first and last elements in a list. |
| 7 | Write a Python program to check if element exists in list in Python. |
| 8 | Write a Python program to find sum of elements in list. |
| 9 | Write a Python program to find second largest number in a list |
| 10 | Write a Python program to print duplicates from a list of integers |

**AIM: Illustrate the use of Flowcharts and Algorithms.**

**Experiment No. 01**

**Flowchart:**

A flowchart is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

**Importance of Flowchart:**

Flow charts are an important tool for the improvement of processes. By providing a graphical representation, it helps one to identify the different elements of a process and understand the interrelationships among the various steps. Flow charts may also be used to gather information and data about a process as an aid to decision making or performance evaluation.

**Flowchart Symbols:**

Whether you're trying to read a flowchart or creating a flowchart, knowing the most common symbols and flowchart conventions is going to make it a lot easier. Here, we've got the four flowcharting symbols you've got to know, plus a rundown on some more intermediate process symbols if you're looking for extra credit.

**1. The Oval:**

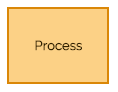
An End or a Beginning



The oval, or terminator, is used to represent the start and end of a process.

**2. The Rectangle:**

A Step in the Flowcharting Process

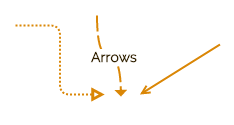


The rectangle is your go-to symbol once you've started flowcharting. It represents any step in the process you’re diagramming and is the workhorse of the flowchart diagram. Use rectangles to capture process steps like basic tasks or actions in your process.

**3. The Arrow:**

Indicate Directional Flow

LM/CS104/01



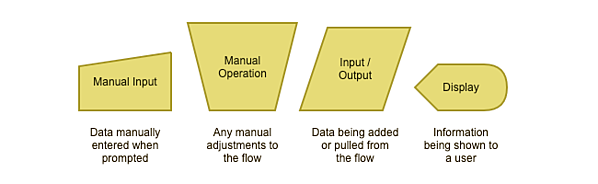
The arrow is used to guide the viewer along their flowcharting path. And while there are many different types of arrow tips to choose from, we recommend sticking with one or two for your entire flowchart. This keeps your diagram looking clean, but also allows you to emphasize certain steps in your process.

**4. The Diamond:**

Indicate a Decision

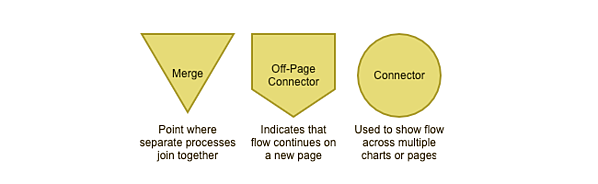


The diamond symbolizes that a decision is required to move forward. This could be a binary, this-or-that choice or a more complex decision with multiple choices. Make sure that you capture each possible choice within your diagram.

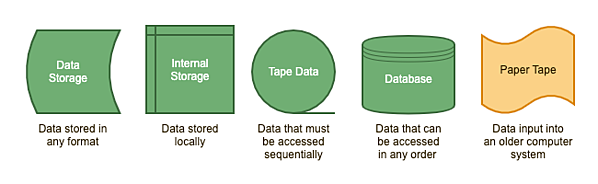
**5.Input & Output:**

Input and output symbols show where and how data is coming in and out throughout your process.

**6. Merging & Connecting:**

 Merging and connector symbols make it easier to connect flowcharts that span multiple pages.

**7. Data Symbols**

Data symbols clarify where the data your flowchart references is being stored.

**Guidelines for preparing a flowchart:**

**1. Use Consistent Design Elements:**

Shapes, lines and texts within a flowchart diagram should be consistent.

**2. Keep Everything on One Page:**

It is good practice to make sure that the flowchart fits on a single page and the text remains readable. When a diagram becomes too large to fit on a page, it's advisable to divide it into multiple charts and connect them with hyperlinks.

**3. Flow Data from Left to Right:**

Structuring a flowchart from left to right makes the information easier to read and comprehend.

**4. Use a Split Path Instead of a Traditional Decision Symbol:**

The use of a split-path continues the left-to-right process flow, and it's easy to see and understand without explanation.

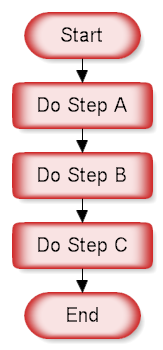
**5. Place Return Lines Under the Flow Diagram:**

Since we naturally read text from the top of the page down, it is logical that return lines should be placed under the flowchart rather than above. If two return lines are needed, they shouldn't overlap.

**Structure of Flowchart:**

Whether you are flowcharting software programs or business processes, using only these structures will make it easier to find and correct errors in your charts. Each structure has a simple flow of control with one input and one output. These structures can then be nested within each other. Any chart can be drawn using only these structures. You do not have to use GOTO or draw spaghetti diagrams just because you are drawing a flowchart. You can draw structured flowcharts.

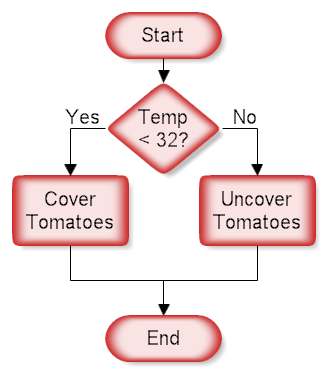
**Sequence:**



The flowchart above demonstrates a sequence of steps. The reader would start at the Start shape and follow the arrows from one rectangle to the other, finishing at the End shape.  A sequence is the simplest flowcharting construction. You do each step in order.   
  
If your charts are all sequences, then you probably don't need to draw a flowchart. You can type a simple list using your word processor. The power of a flowchart becomes evident when you include decisions and loops.

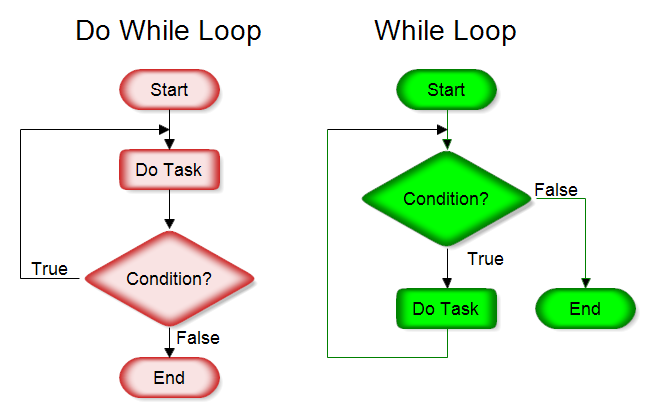
LM/CS104/01

**Decision:**



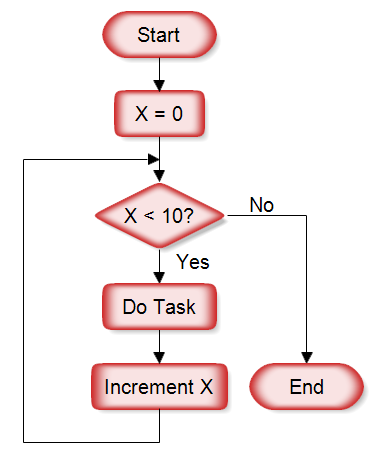
This structure is called a decision, "If Then.. Else" or a conditional. A question is asked in the decision shape. Depending on the answer the control follows either of two paths. In the chart above, if the temperature is going to be less than freezing (32 degrees Fahrenheit) the tomatoes should be covered.

**Repetition:**



LM/CS104/01

This structure allows you to repeat a task over and over. The red chart above on the left does the task and repeats doing the task until the condition is false. It always does the task at least once. The green chart on the right checks the condition first and continues doing the task while the condition is true. In the green chart the task may not be done at all. You can also have the conditions reversed and your loop is still a structured design loop.



The above chart is a "For Loop." In this example the task is performed 10 times as X counts from 0 to 10. Depending on the condition, the task may not be performed at all.

There is also a "For Each" structure that is like the for loop, but has no counter. It will go through each item of a collection and do the task. You don't have to know the length of the collection or use a counter. It is essentially saying "do this for every item in the collection".

**Algorithm:**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

**Characteristics of an Algorithm**

* Not all procedures can be called an algorithm. An algorithm should have the following characteristics −
* Unambiguous − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
* Input − An algorithm should have 0 or more well-defined inputs.
* Output − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
* Finiteness − Algorithms must terminate after a finite number of steps.
* Feasibility − Should be feasible with the available resources.
* Independent − An algorithm should have step-by-step directions, which should be independent of any programming code.

**Example:**

**Problem** − Design an algorithm to add two numbers and display the result.

Step 1 − START

Step 2 − declare three integers a, b & c

Step 3 − define values of a & b

Step 4 − add values of a & b

Step 5 − store output of step 4 to c

Step 6 − print c

Step 7 − STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

Step 1 − START ADD

Step 2 − get values of a & b

Step 3 − c ← a + b

Step 4 − display c

Step 5 – STOP

LM/CS104/01

**Experiment No. 02**

**Aim: Write a program to demonstrate basic data type in python.**

**Description:** - Data types in Python Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

**Source Code: -**

a = 5

print(a, "is of type", type(a))

a = 2.0

print(a, "is of type", type(a))

a = 1+2j

print(a, "is complex number?", isinstance(1+2j,complex))

**Output:-**

5 is of type <class ‘int’>

2.0 is of type <class ‘float’>

(1+2j) is complex number? True

**AIM:** Illustrate the use of operators. Write a program to compute distance between two points taking input from the user.

**Experiment No. 03**

## Python Operators

Operators are used to perform operations on variables and values.

In the example below, the + operator is used to add together two values:

print(10 + 5)

**Output:** 15

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

**Python Arithmetic Operators**

Table 4.1 shows Arithmetic operators used with numeric values to perform common mathematical operations:

Table 4.1

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

## Python Assignment Operators

Assignment operators are used to assign values to variables:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| //= | x //= 3 | x = x // 3 |
| \*\*= | x \*\*= 3 | x = x \*\* 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

## Python Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Python Logical Operators:

Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| or | Returns True if one of the statements is true | x < 5 or x < 4 |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) |

## Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Returns True if both variables are the same object | x is y |
| is not | Returns True if both variables are not the same object | x is not y |

## Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y |

## Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

Program:

# Python3 program to calculate

# distance between two points

import math

# Function to calculate distance

def distance(x1 , y1 , x2 , y2):

# Calculating distance

return math.sqrt(math.pow(x2 - x1, 2) +

math.pow(y2 - y1, 2) \* 1.0)

# Drivers Code

print("%.6f"%distance(3, 4, 4, 3))

Output:-

1.414214

**Experiment No. 04**

**Aim:** Write a Program for checking whether the given number is an even number or not.

**Description:**

**Control Flow Statements**

A program’s control flow is the order in which the program’s code executes. The control flow of

a Python program is regulated by conditional statements, loops, and function calls.

**The if Statement**

Often, you need to execute some statements only if some condition holds, or choose statements

to execute depending on several mutually exclusive conditions. The Python compound statement

if, which uses if, elif, and else clauses.

Here’s the syntax for the if statement:

if expression:

statement(s)

elif expression:

statement(s)

elif expression:

statement(s)

...

else:

statement(s)

The elif and else clauses are optional. Note that unlike some languages, Python does not have a

switch statement, so you must use if, elif, and else for all conditional processing.

**Source Code:**

# Python program to check if the input number is odd or even.

# A number is even if division by 2 gives a remainder of 0.

# If the remainder is 1, it is an odd number.

num = int(input("Enter a number: "))

if (num % 2) == 0:

print("{0} is Even".format(num))

else:

print("{0} is Odd".format(num))

Output:

Enter a number: 2

2 is Even

**Experiment No. 05**

**Aim: To write a Python Program to find the square root of a number by Newton’s Method.**

**Algorithm:**

1. Define a function named newtonSqrt().

2. Initialize approx as 0.5\*n and better as 0.5\*(approx.+n/approx.)

3. Use a while loop with a condition better!=approx to perform the following,

i. Set approx.=better

ii. Better=0.5\*(approx.+n/approx.)

4. Print the value of approx..

**Program:**

def newtonSqrt(n):

approx = 0.5 \* n

better = 0.5 \* (approx + n/approx)

while better != approx:

approx = better

better = 0.5 \* (approx + n/approx)

return approx

print('The square root is' ,newtonSqrt(100))

**Sample Output:**

The square root is 10

**AIM: To write a Python program to find first n prime numbers.**

**Experiment No. 6**

**Algorithm:**

1. Read the value of n

2. for num in range(0,n + 1), perform the following

3. if num%i is 0 then break

else print the value of num

4. Repeat step 3 for i in range(2,num)

**Program:**

n = int(input("Enter the upper limit: "))

print("Prime numbers are")

for num in range(0,n + 1):

# prime numbers are greater than 1

if num > 1:

for i in range(2,num):

if (num % i) == 0:

break

else:

print(num**)**

**Output:**

$python main.py

Enter the upper limit: 20

Prime numbers are

2

3

5

7

11

13

17

19

**AIM: To write a Python program to multiply matrices.**

**Experiment No. 7**

**Algorithm:**

1. Define two matrices X and Y

2. Create a resultant matrix named ‘result’

3. for i in range(len(X)):

i. for j in range(len(Y[0])):

a) for k in range(len(Y))

b) result[i][j] += X[i][k] \* Y[k][j]

4. for r in result, print the value of r

**Program:**

X = [[12,7,3],

[4 ,5,6],

[7 ,8,9]]

Y = [[5,8,1,2],

[6,7,3,0],

[4,5,9,1]]

result = [[0,0,0,0],

[0,0,0,0],

[0,0,0,0]]

for i in range(len(X)):

for j in range(len(Y[0])):

for k in range(len(Y)):

result[i][j] += X[i][k] \* Y[k][j]

for r in result:

print(r)

**Output:**

[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23]

**Experiment No. 08**

**Aim: Write a Program to demonstrate list and tuple in python.**

**Description:**

Lists are one of the most powerful tools in python. They are just like the arrays declared in other

languages. But the most powerful thing is that list need not be always homogenous. A single list

can contain strings, integers, as well as objects. Lists can also be used for implementing stacks

and queues. Lists are mutable, i.e., they can be altered once declared.

A tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception

that tuple cannot be changed once declared. Tuples are usually faster than lists.

**Source Code:**

L = [1, "a" , "string" , 1+2]

print L

L.append(6)

print L

L.pop()

print L

print L[1]

**Output** :

[1, 'a', 'string', 3]

[1, 'a', 'string', 3, 6]

[1, 'a', 'string', 3]

**Source Code:**

a

tup = (1, "a", "string", 1+2)

print tup

print tup[1]

**Output:**

(1, 'a', 'string', 3)

a

1

0

**Experiment No. 09**

**AIM:**

**To write a Python Program to perform Linear Search**

**Algorithm:**

1. Read n elements into the list

2. Read the element to be searched

3. If alist[pos]==item, then print the position of the item

4. else increment the position and repeat step 3 until pos reaches the length of the list

**Source Code:**

items = [5, 7, 10, 12, 15]

print("list of items is", items)

x = int(input("enter item to search:")

i = flag = 0

while i < len(items):

if items[i] == x:

flag = 1

break

i = i + 1

if flag == 1:

print("item found at position:", i + 1)

else:

print("item not found")

**Output:**

$python main.py

(list of items is: [5, 7, 10, 12, 15] )

enter item to search: 7

(item found at position:, 2)

**AIM: To write a Python Program to perform binary search.**

Algorithm:

1. Read the search element

2. Find the middle element in the sorted list

3. Compare the search element with the middle element

i. if both are matching, print element found

ii. else then check if the search element is smaller or larger than the middle element

4. If the search element is smaller than the middle element, then repeat steps 2 and 3 for the

left sublist of the middle element

5. If the search element is larger than the middle element, then repeat steps 2 and 3 for the

right sublist of the middle element

6. Repeat the process until the search element if found in the list

7. If element is not found, loop terminates

**Source Code:**

# Python code to implement iterative Binary Search.

# It returns location of x in given array arr

# if present, else returns -1

def binarySearch(arr, l, r, x):

while l <= r:

mid = l + (r - l)/2;

# Check if x is present at mid

if arr[mid] == x:

return mid

# If x is greater, ignore left half

elif arr[mid] < x:

l = mid + 1

# If x is smaller, ignore right half

else:

r = mid - 1

# If we reach here, then the element

# was not present

return -1

# Test array

arr = [ 2, 3, 4, 10, 40 ]

x = 4

# Function call

result = binarySearch(arr, 0, len(arr)-1, x)

if result != -1:

print "Element is present at index % d" % result

else:

print "Element is not present in array"

**Output:**

$python main.py

Element is present at index 2

**Experiment No. 10**

**Aim: To write a Python Program to perform selection sort.**

**Algorithm: nb**

1. Create a function named selection sort

2. Initialise pos=0

3. If alist[location]>alist[pos] then perform the following till i+1,

4. Set pos=location

5. Swap alist[i] and alist[pos]

6. Print the sorted list

**Source Code:**

def selectionSort(alist):

for i in range(len(alist)-1,0,-1):

pos=0

for location in range(1,i+1):

if alist[location]>alist[pos]:

pos= location

temp = alist[i]

alist[i] = alist[pos]

alist[pos] = temp

alist = [54,26,93,17,77,31,44,55,20]

selectionSort(alist)

print(alist)

**Output:**

$python main.py

[17, 20, 26, 31, 44, 54, 55, 77, 93]

**Aim:**

**To write a Python Program to perform insertion sort.**

**Algorithm:**

1. Create a function named insertionsort

2. Initialise currentvalue=alist[index] and position=index

3. while position>0 and alist[position-1]>currentvalue, perform the following till len(alist)

4. alist[position]=alist[position-1]

5. position = position-1

6. alist[position]=currentvalue

7. Print the sorted list

**Program:**

def insertionSort(alist):

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

currentvalue = alist[index]

position = index

while position>0 and alist[position-1]>currentvalue:

alist[position]=alist[position-1]

position = position-1

alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]

insertionSort(alist)

print(alist)

**Output:**

$python main.py

[20, 54, 54, 54, 54, 54, 93, 93, 93]

**Experiment No. 11**

**AIM: To write a Python Program to perform Merge sort.**

**Algorithm:**

1. Create a function named mergesort

2. Find the mid of the list

3. Assign lefthalf = alist[:mid] and righthalf = alist[mid:]

4. Initialise i=j=k=0

5. while i < len(lefthalf) and j < len(righthalf), perform the following

if lefthalf[i] < righthalf[j]:

alist[k]=lefthalf[i]

Increment i

else

alist[k]=righthalf[j]

Increment j

Increment k

6. while i < len(lefthalf),perform the following

alist[k]=lefthalf[i]

Increment i

Increment k

7. while j < len(righthalf), perform the following

alist[k]=righthalf[j]

Increment j

Increment k

8. Print the sorted list

**Program:**

# Python program for implementation of MergeSort

# Merges two subarrays of arr[].

# First subarray is arr[l..m]

# Second subarray is arr[m+1..r]

def merge(arr, l, m, r):

n1 = m - l + 1

n2 = r- m

# create temp arrays

L = [0] \* (n1)

R = [0] \* (n2)

# Copy data to temp arrays L[] and R[]

for i in range(0 , n1):

L[i] = arr[l + i]

for j in range(0 , n2):

R[j] = arr[m + 1 + j]

# Merge the temp arrays back into arr[l..r]

i = 0 # Initial index of first subarray

j = 0 # Initial index of second subarray

k = l # Initial index of merged subarray

while i < n1 and j < n2 :

if L[i] <= R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

# Copy the remaining elements of L[], if there

# are any

while i < n1:

arr[k] = L[i]

i += 1

k += 1

# Copy the remaining elements of R[], if there

# are any

while j < n2:

arr[k] = R[j]

j += 1

k += 1

# l is for left index and r is right index of the

# sub-array of arr to be sorted

def mergeSort(arr,l,r):

if l < r:

# Same as (l+r)/2, but avoids overflow for

# large l and h

m = (l+(r-1))/2

# Sort first and second halves

mergeSort(arr, l, m)

mergeSort(arr, m+1, r)

merge(arr, l, m, r)

# Driver code to test above

arr = [12, 11, 13, 5, 6, 7]

n = len(arr)

print ("Given array is")

for i in range(n):

print ("%d" %arr[i]),

mergeSort(arr,0,n-1)

print ("\n\nSorted array is")

for i in range(n):

print ("%d" %arr[i]),

**Output:**

$python main.py

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

**Experiment No. 12**

**Aim: Create a dictionary and apply the following methods**

**i) Access Elements from Dictionary.**

**ii) Change and Add Dictionary elements,**

**iii) Removing elements from Dictionary**

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

Dictionaries are optimized to retrieve values when the key is known.

**Creating Python Dictionary**

Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.

An item has a key and a corresponding value that is expressed as a pair (**key: value**).

While the values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys

my\_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()

my\_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair

my\_dict = dict([(1,'apple'), (2,'ball')])

As you can see from above, we can also create a dictionary using the built-in dict() function.

1. **Accessing Elements from Dictionary**

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets [] or with the get() method.

If we use the square brackets [], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

**Program:**

# get vs [] for retrieving elements

my\_dict = {'name': 'Jack', 'age': 26}

# Output: Jack

print(my\_dict['name'])

# Output: 26

print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error

# Output None

print(my\_dict.get('address'))

# KeyError

print(my\_dict['address'])

**Output**

Jack

26

None

Traceback (most recent call last):

File "<string>", line 15, in <module>

print(my\_dict['address'])

KeyError: 'address'

1. **Changing and Adding Dictionary elements**

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.

If the key is already present, then the existing value gets updated. In case the key is not present, a new (**key: value**) pair is added to the dictionary.

**Program:**

# Changing and adding Dictionary Elements

my\_dict = {'name': 'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}

print(my\_dict)

**Output**

{'name': 'Jack', 'age': 27}

{'name': 'Jack', 'age': 27, 'address': 'Downtown'}

1. **Removing elements from Dictionary**

We can remove a particular item in a dictionary by using the pop() method. This method removes an item with the provided key and returns the value.

The popitem() method can be used to remove and return an arbitrary (key, value) item pair from the dictionary. All the items can be removed at once, using the clear() method.

We can also use the del keyword to remove individual items or the entire dictionary itself.

**Program:**

# Removing elements from a dictionary

# create a dictionary

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item, return (key,value)

# Output: (5, 25)

print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}

print(squares)

# remove all items

squares.clear()

# Output: {}

print(squares)

# delete the dictionary itself

del squares

# Throws Error

print(squares)

**Output**

16

{1: 1, 2: 4, 3: 9, 5: 25}

(5, 25)

{1: 1, 2: 4, 3: 9}

{}

Traceback (most recent call last):

File "<string>", line 30, in <module>

print(squares)

NameError: name 'squares' is not defined