

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.

Lists

- ❖ List is an ordered sequence of items. Values in the list are called elements / items.
- ❖ It can be written as a list of comma-separated items (values) between **square brackets []**.
- ❖ Items in the lists can be of different data types.

Operations on list:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Updating
6. Membership
7. Comparison

operations	examples	description
create a list	<pre>>>> a=[2,3,4,5,6,7,8,9,10] >>> print(a) [2, 3, 4, 5, 6, 7, 8, 9, 10]</pre>	in this way we can create a list at compile time
Indexing	<pre>>>> print(a[0]) 2 >>> print(a[8]) 10 >>> print(a[-1]) 10</pre>	<p>Accessing the item in the position 0</p> <p>Accessing the item in the position 8</p> <p>Accessing a last element using negative indexing.</p>
Slicing	<pre>>>> print(a[0:3]) [2, 3, 4] >>> print(a[0:]) [2, 3, 4, 5, 6, 7, 8, 9, 10]</pre>	Printing a part of the list.
Concatenation	<pre>>>>b=[20,30] >>> print(a+b) [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30]</pre>	Adding and printing the items of two lists.
Repetition	<pre>>>> print(b*3) [20, 30, 20, 30, 20, 30]</pre>	Create a multiple copies of the same list.

Updating	<pre>>>> print(a[2]) 4 >>> a[2]=100 >>> print(a) [2, 3, 100, 5, 6, 7, 8, 9, 10]</pre>	Updating the list using index value.
Membership	<pre>>>> a=[2,3,4,5,6,7,8,9,10] >>> 5 in a True >>> 100 in a False >>> 2 not in a False</pre>	Returns True if element is present in list. Otherwise returns false.
Comparison	<pre>>>> a=[2,3,4,5,6,7,8,9,10] >>> b=[2,3,4] >>> a==b False >>> a!=b True</pre>	Returns True if all elements in both elements are same. Otherwise returns false

List slices:

- ❖ List slicing is an operation that extracts a subset of elements from an list and packages them as another list.

Syntax:

Listname[start:stop]

Listname[start:stop:steps]

- ❖ default start value is 0
- ❖ default stop value is n-1
- ❖ [:] this will print the entire list
- ❖ [2:2] this will create a empty slice

slices	example	description
a[0:3]	<pre>>>> a=[9,8,7,6,5,4] >>> a[0:3] [9, 8, 7]</pre>	Printing a part of a list from 0 to 2.
a[:4]	<pre>>>> a[:4] [9, 8, 7, 6]</pre>	Default start value is 0. so prints from 0 to 3
a[1:]	<pre>>>> a[1:] [8, 7, 6, 5, 4]</pre>	default stop value will be n-1. so prints from 1 to 5
a[:]	<pre>>>> a[:] [9, 8, 7, 6, 5, 4]</pre>	Prints the entire list.

a[2:2]	>>> a[2:2] []	print an empty slice
a[0:6:2]	>>> a[0:6:2] [9, 7, 5]	Slicing list values with step size 2.
a[::-1]	>>> a[::-1] [4, 5, 6, 7, 8, 9]	Returns reverse of given list values

List methods:

- ❖ Methods used in lists are used to manipulate the data quickly.
- ❖ These methods work only on lists.
- ❖ They do not work on the other sequence types that are not mutable, that is, the values they contain cannot be changed, added, or deleted.

syntax:

list name.method name(element/index/list)

	syntax	example	description
1	a.append(element)	>>> a=[1,2,3,4,5] >>> a.append(6) >>> print(a) [1, 2, 3, 4, 5, 6]	Add an element to the end of the list
2	a.insert(index,element)	>>> a.insert(0,0) >>> print(a) [0, 1, 2, 3, 4, 5, 6]	Insert an item at the defined index
3	a.extend(b)	>>> b=[7,8,9] >>> a.extend(b) >>> print(a) [0, 1, 2, 3, 4, 5, 6, 7, 8,9]	Add all elements of a list to the another list
4	a.index(element)	>>> a.index(8) 8	Returns the index of the first matched item
5	a.sort()	>>> a.sort() >>> print(a) [0, 1, 2, 3, 4, 5, 6, 7, 8]	Sort items in a list in ascending order
6	a.reverse()	>>> a.reverse() >>> print(a) [8, 7, 6, 5, 4, 3, 2, 1, 0]	Reverse the order of items in the list

7	a.pop()	>>> a.pop() 0	Removes and returns an element at the last element
8	a.pop(index)	>>> a.pop(0) 8	Remove the particular element and return it.
9	a.remove(element)	>>> a.remove(1) >>> print(a) [7, 6, 5, 4, 3, 2]	Removes an item from the list
10	a.count(element)	>>> a.count(6) 1	Returns the count of number of items passed as an argument
11	a.copy()	>>> b=a.copy() >>> print(b) [7, 6, 5, 4, 3, 2]	Returns a shallow copy of the list
12	len(list)	>>> len(a) 6	return the length of the length
13	min(list)	>>> min(a) 2	return the minimum element in a list
14	max(list)	>>> max(a) 7	return the maximum element in a list.
15	a.clear()	>>> a.clear() >>> print(a) []	Removes all items from the list.
16	del(a)	>>> del(a) >>> print(a) Error: name 'a' is not defined	delete the entire list.

List loops:

1. For loop
2. While loop
3. Infinite loop

List using For Loop:

- ❖ The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.
- ❖ Iterating over a sequence is called traversal.
- ❖ Loop continues until we reach the last item in the sequence.
- ❖ The body of for loop is separated from the rest of the code **using indentation.**

Syntax:
for val in sequence:

Accessing element	output
<pre>a=[10,20,30,40,50] for i in a: print(i)</pre>	<pre>1 2 3 4 5</pre>
Accessing index	output
<pre>a=[10,20,30,40,50] for i in range(0,len(a),1): print(i)</pre>	<pre>0 1 2 3 4</pre>
Accessing element using range:	output
<pre>a=[10,20,30,40,50] for i in range(0,len(a),1): print(a[i])</pre>	<pre>10 20 30 40 50</pre>

List using While loop

- ❖ The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- ❖ When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Syntax:
while (condition):
body of while

Sum of elements in list	Output:
<pre>a=[1,2,3,4,5] i=0 sum=0 while i<len(a): sum=sum+a[i] i=i+1 print(sum)</pre>	<pre>15</pre>

Infinite Loop

A loop becomes infinite loop if the condition given never becomes false. It keeps on running. Such loops are called infinite loop.

Example	Output:
<pre>a=1 while (a==1): n=int(input("enter the number")) print("you entered:" , n)</pre>	<pre>Enter the number 10 you entered:10 Enter the number 12 you entered:12 Enter the number 16 you entered:16</pre>

Mutability:

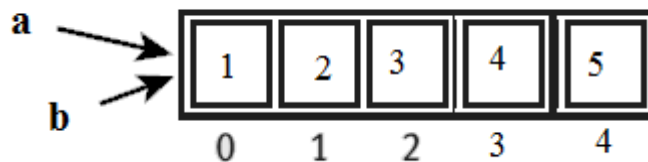
- ❖ Lists are mutable. (can be changed)
- ❖ Mutability is the ability for certain types of data to be changed without entirely recreating it.
- ❖ An item can be changed in a list by accessing it directly as part of the assignment statement.
- ❖ Using the indexing operator (square brackets []) on the left side of an assignment, one of the list items can be updated.

Example	description
<pre>>>> a=[1,2,3,4,5] >>> a[0]=100 >>> print(a) [100, 2, 3, 4, 5]</pre>	changing single element
<pre>>>> a=[1,2,3,4,5] >>> a[0:3]=[100,100,100] >>> print(a) [100, 100, 100, 4, 5]</pre>	changing multiple element
<pre>>>> a=[1,2,3,4,5] >>> a[0:3]=[] >>> print(a) [4, 5]</pre>	The elements from a list can also be removed by assigning the empty list to them.
<pre>>>> a=[1,2,3,4,5] >>> a[0:0]=[20,30,45] >>> print(a) [20,30,45,1, 2, 3, 4, 5]</pre>	The elements can be inserted into a list by squeezing them into an empty slice at the desired location.

Aliasing(copying):

- ❖ Creating a copy of a list is called aliasing. When you create a copy both list will be
- ❖ having same memory location. changes in one list will affect another list.
- ❖ **Alaising refers to having different names for same list values.**

Example	Output:
a= [1, 2, 3 ,4 ,5] b=a print (b) a is b a[0]=100 print(a) print(b)	[1, 2, 3, 4, 5] True [100,2,3,4,5] [100,2,3,4,5]



- ❖ In this a single list object is created and modified using the subscript operator.
- ❖ When the first element of the list named “a” is replaced, the first element of the list named “b” is also replaced.
- ❖ This type of change is what is known as a **side effect**. This happens because after the assignment **b=a**, the variables **a** and **b** refer to the exact same list object.
- ❖ They are **aliases** for the same object. This phenomenon is known as **aliasing**.
- ❖ To prevent aliasing, a new object can be created and the contents of the original can be copied which is called **cloning**.

Cloning:

- ❖ To avoid the disadvantages of copying we are using cloning. creating a copy of a
 - ❖ same list of elements with two different memory locations is called cloning.
 - ❖ Changes in one list will not affect locations of aother list.
 - ❖ Cloning is a process of making a copy of the list without modifying the original list.
1. Slicing
 2. list()method
 3. copy() method

clonning using Slicing

```
>>>a=[1,2,3,4,5]
>>>b=a[:]
>>>print(b)
[1,2,3,4,5]
>>>a is b
False
```

clonning using List() method

```
>>>a=[1,2,3,4,5]
>>>b=list
>>>print(b)
[1,2,3,4,5]
>>>a is b
false
>>>a[0]=100
>>>print(a)
>>>a=[100,2,3,4,5]
>>>print(b)
>>>b=[1,2,3,4,5]
```

clonning using copy() method

```
a=[1,2,3,4,5]
>>>b=a.copy()
>>> print(b)
[1, 2, 3, 4, 5]
>>> a is b
False
```

List as parameters:

- ❖ In python, arguments are passed by reference.
- ❖ If any changes are done in the parameter which refers within the function, then the changes also reflects back in the calling function.
- ❖ When a list to a function is passed, the function gets a reference to the list.
- ❖ Passing a list as an argument actually passes a reference to the list, not a copy of the list.
- ❖ Since lists are mutable, changes made to the elements referenced by the parameter change the same list that the argument is referencing.

Example 1`:

```
def remove(a):
    a.remove(1)
a=[1,2,3,4,5]
remove(a)
print(a)
```

Output

```
[2,3,4,5]
```


Example 2:	Output
<pre>def inside(a): for i in range(0,len(a),1): a[i]=a[i]+10 print("inside",a) a=[1,2,3,4,5] inside(a) print("outside",a)</pre>	<pre>inside [11, 12, 13, 14, 15] outside [11, 12, 13, 14, 15]</pre>
Example 3	output
<pre>def insert(a): a.insert(0,30) a=[1,2,3,4,5] insert(a) print(a)</pre>	<pre>[30, 1, 2, 3, 4, 5]</pre>

Tuple:

- ❖ A tuple is same as list, except that the set of elements is enclosed in parentheses instead of square brackets.
- ❖ A tuple is an immutable list. i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- ❖ But tuple can be converted into list and list can be converted in to tuple.

methods	example	description
list()	<pre>>>> a=(1,2,3,4,5) >>> a=list(a) >>> print(a) [1, 2, 3, 4, 5]</pre>	it convert the given tuple into list.
tuple()	<pre>>>> a=[1,2,3,4,5] >>> a=tuple(a) >>> print(a) (1, 2, 3, 4, 5)</pre>	it convert the given list into tuple.

Benefit of Tuple:

- ❖ Tuples are faster than lists.
- ❖ If the user wants to protect the data from accidental changes, tuple can be used.
- ❖ Tuples can be used as keys in dictionaries, while lists can't.

Operations on Tuples:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Membership
6. Comparison

Operations	examples	description
Creating a tuple	>>>a=(20,40,60,"apple","ball")	Creating the tuple with elements of different data types.
Indexing	>>>print(a[0]) 20 >>> a[2] 60	Accessing the item in the position 0 Accessing the item in the position 2
Slicing	>>>print(a[1:3]) (40,60)	Displaying items from 1st till 2nd.
Concatenation	>>> b=(2,4) >>>print(a+b) >>>(20,40,60,"apple","ball",2,4)	Adding tuple elements at the end of another tuple elements
Repetition	>>>print(b*2) >>>(2,4,2,4)	repeating the tuple in n no of times
Membership	>>> a=(2,3,4,5,6,7,8,9,10) >>> 5 in a True >>> 100 in a False >>> 2 not in a False	Returns True if element is present in tuple. Otherwise returns false.
Comparison	>>> a=(2,3,4,5,6,7,8,9,10) >>>b=(2,3,4) >>> a==b False >>> a!=b True	Returns True if all elements in both elements are same. Otherwise returns false

Tuple methods:

- ❖ Tuple is immutable so changes cannot be done on the elements of a tuple once it is assigned.

methods	example	description
a.index(tuple)	>>> a=(1,2,3,4,5) >>> a.index(5) 4	Returns the index of the first matched item.
a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a) 5	return the length of the tuple

min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

Tuple Assignment:

- ❖ Tuple assignment allows, variables on the left of an assignment operator and values of tuple on the right of the assignment operator.
- ❖ Multiple assignment works by creating a tuple of expressions from the right hand side, and a tuple of targets from the left, and then matching each expression to a target.
- ❖ Because multiple assignments use tuples to work, it is often termed tuple assignment.

Uses of Tuple assignment:

- ❖ It is often useful to swap the values of two variables.

Example:

Swapping using temporary variable:	Swapping using tuple assignment:
<pre>a=20 b=50 temp = a a = b b = temp print("value after swapping is",a,b)</pre>	<pre>a=20 b=50 (a,b)=(b,a) print("value after swapping is",a,b)</pre>

Multiple assignments:

Multiple values can be assigned to multiple variables using tuple assignment.

```
>>>(a,b,c)=(1,2,3)
>>>print(a)
1
>>>print(b)
2
>>>print(c)
3
```

Tuple as return value:

- ❖ A Tuple is a comma separated sequence of items.
- ❖ It is created with or without ().
- ❖ A function can return one value. if you want to return more than one value from a function. we can use tuple as return value.

Example1:	Output:
<pre>def div(a,b): r=a%b q=a//b return(r,q) a=eval(input("enter a value:")) b=eval(input("enter b value:")) r,q=div(a,b) print("remainder:",r) print("quotient:",q)</pre>	<pre>enter a value:4 enter b value:3 remainder: 1 quotient: 1</pre>
Example2:	Output:
<pre>def min_max(a): small=min(a) big=max(a) return(small,big) a=[1,2,3,4,6] small,big=min_max(a) print("smallest:",small) print("biggest:",big)</pre>	<pre>smallest: 1 biggest: 6</pre>

Tuple as argument:

- ❖ The parameter name that begins with * gathers argument into a tuple.

Example:	Output:
<pre>def printall(*args): print(args) printall(2,3,'a')</pre>	<pre>(2, 3, 'a')</pre>

Dictionaries:

- ❖ Dictionary is an unordered collection of elements. An element in dictionary has a key: value pair.
- ❖ All elements in dictionary are placed inside the curly braces i.e. { }
- ❖ Elements in Dictionaries are **accessed via keys** and not by their position.
- ❖ The values of a dictionary can be any data type.
- ❖ Keys must be immutable data type (numbers, strings, tuple)

Operations on dictionary:

1. Accessing an element
2. Update
3. Add element
4. Membership

Operations	Example	Description
Creating a dictionary	<pre>>>> a={1:"one",2:"two"} >>> print(a) {1: 'one', 2: 'two'}</pre>	Creating the dictionary with elements of different data types.
accessing an element	<pre>>>> a[1] 'one' >>> a[0] KeyError: 0</pre>	Accessing the elements by using keys.
Update	<pre>>>> a[1]="ONE" >>> print(a) {1: 'ONE', 2: 'two'}</pre>	Assigning a new value to key. It replaces the old value by new value.
add element	<pre>>>> a[3]="three" >>> print(a) {1: 'ONE', 2: 'two', 3: 'three'}</pre>	Add new element in to the dictionary with key.
membership	<pre>a={1: 'ONE', 2: 'two', 3: 'three'} >>> 1 in a True >>> 3 not in a False</pre>	Returns True if the key is present in dictionary. Otherwise returns false.

Methods in dictionary:

Method	Example	Description
a.copy()	<pre>a={1: 'ONE', 2: 'two', 3: 'three'} >>> b=a.copy() >>> print(b) {1: 'ONE', 2: 'two', 3: 'three'}</pre>	It returns copy of the dictionary. here copy of dictionary 'a' get stored in to dictionary 'b'
a.items()	<pre>>>> a.items() dict_items([(1, 'ONE'), (2, 'two'), (3, 'three')])</pre>	Return a new view of the dictionary's items. It displays a list of dictionary's (key, value) tuple pairs.
a.keys()	<pre>>>> a.keys() dict_keys([1, 2, 3])</pre>	It displays list of keys in a dictionary
a.values()	<pre>>>> a.values() dict_values(['ONE', 'two', 'three'])</pre>	It displays list of values in dictionary
a.pop(key)	<pre>>>> a.pop(3) 'three' >>> print(a) {1: 'ONE', 2: 'two'}</pre>	Remove the element with <i>key</i> and return its value from the dictionary.

setdefault(key,value)	>>> a.setdefault(3,"three") 'three' >>> print(a) {1: 'ONE', 2: 'two', 3: 'three'} >>> a.setdefault(2) 'two'	If key is in the dictionary, return its value. If key is not present, insert key with a value of dictionary and return dictionary.
a.update(dictionary)	>>> b={"4":"four"} >>> a.update(b) >>> print(a) {1: 'ONE', 2: 'two', 3: 'three', 4: 'four'}	It will add the dictionary with the existing dictionary
fromkeys()	>>> key={"apple","ball"} >>> value="for kids" >>> d=dict.fromkeys(key,value) >>> print(d) {'apple': 'for kids', 'ball': 'for kids'}	It creates a dictionary from key and values.
len(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>> len(a) 3	It returns the length of the list.
clear()	a={1: 'ONE', 2: 'two', 3: 'three'} >>>a.clear() >>>print(a) >>>{}	Remove all elements form the dictionary.
del(a)	a={1: 'ONE', 2: 'two', 3: 'three'} >>> del(a)	It will delete the entire dictionary.

Difference between List, Tuples and dictionary:

List	Tuples	Dictionary
A list is mutable	A tuple is immutable	A dictionary is mutable
Lists are dynamic	Tuples are fixed size in nature	In values can be of any data type and can repeat, keys must be of immutable type
List are enclosed in brackets [] and their elements and size can be changed	Tuples are enclosed in parenthesis () and cannot be updated	Tuples are enclosed in curly braces { } and consist of key:value
Homogenous	Heterogeneous	Homogenous
Example: List = [10, 12, 15]	Example: Words = ("spam", "eggs") Or Words = "spam", "eggs"	Example: Dict = {"ram": 26, "abi": 24}
<u>Access:</u> print(list[0])	<u>Access:</u> print(words[0])	<u>Access:</u> print(dict["ram"])

Can contain duplicate elements	Can contain duplicate elements. Faster compared to lists	Cant contain duplicate keys, but can contain duplicate values
Slicing can be done	Slicing can be done	Slicing can't be done
<u>Usage:</u> ❖ List is used if a collection of data that doesnt need random access. ❖ List is used when data can be modified frequently	<u>Usage:</u> ❖ Tuple can be used when data cannot be changed. ❖ A tuple is used in combination with a dictionary i.e.a tuple might represent a key.	<u>Usage:</u> ❖ Dictionary is used when a logical association between key:value pair. ❖ When in need of fast lookup for data, based on a custom key. ❖ Dictionary is used when data is being constantly modified.

Advanced list processing:

List Comprehension:

- ❖ List comprehensions provide a concise way to apply operations on a list.
- ❖ It creates a new list in which each element is the result of applying a given operation in a list.
- ❖ It consists of brackets containing an expression followed by a “for” clause, then a list.
- ❖ The list comprehension always returns a result list.

Syntax

list=[expression for item in list if conditional]

List Comprehension	Output
>>>L=[x**2 for x in range(0,5)] >>>print(L)	[0, 1, 4, 9, 16]
>>>[x for x in range(1,10) if x%2==0]	[2, 4, 6, 8]
>>>[x for x in 'Python Programming' if x in ['a','e','i','o','u']]	['o', 'o', 'a', 'i']
>>>mixed=[1,2,"a",3,4.2] >>> [x**2 for x in mixed if type(x)==int]	[1, 4, 9]
>>>[x+3 for x in [1,2,3]]	[4, 5, 6]
>>> [x*x for x in range(5)]	[0, 1, 4, 9, 16]
>>> num=[-1,2,-3,4,-5,6,-7] >>> [x for x in num if x>=0]	[2, 4, 6]
>>> str=["this", "is", "an", "example"] >>> element=[word[0] for word in str] >>> print(element)	['t', 'i', 'a', 'e']

Nested list:

List inside another list is called nested list.

Example:

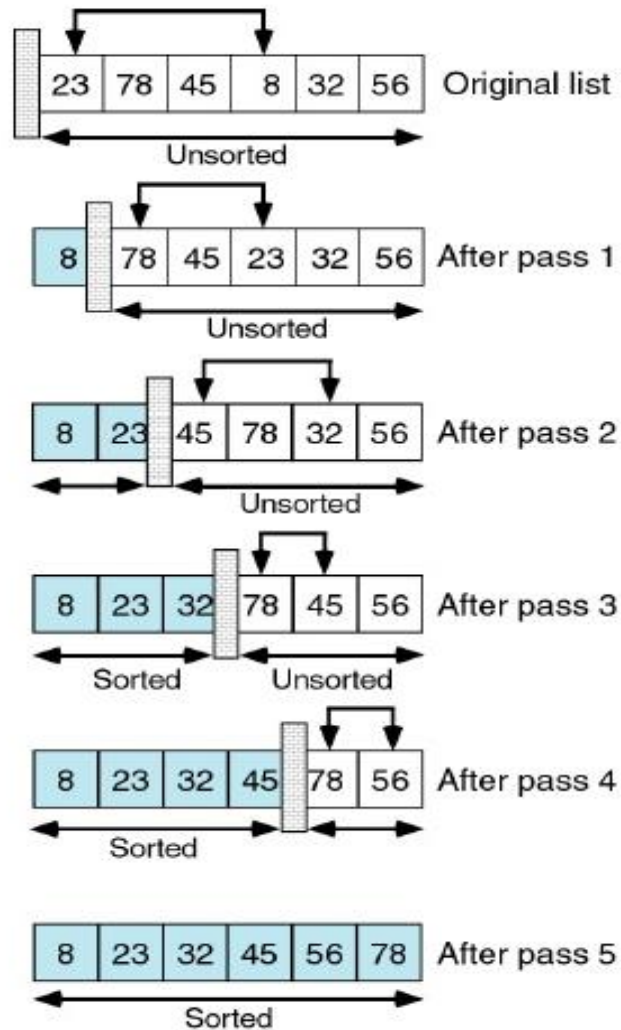
```
>>> a=[56,34,5,[34,57]]
>>> a[0]
56
>>> a[3]
[34, 57]
>>> a[3][0]
34
>>> a[3][1]
57
```

Programs on matrix:

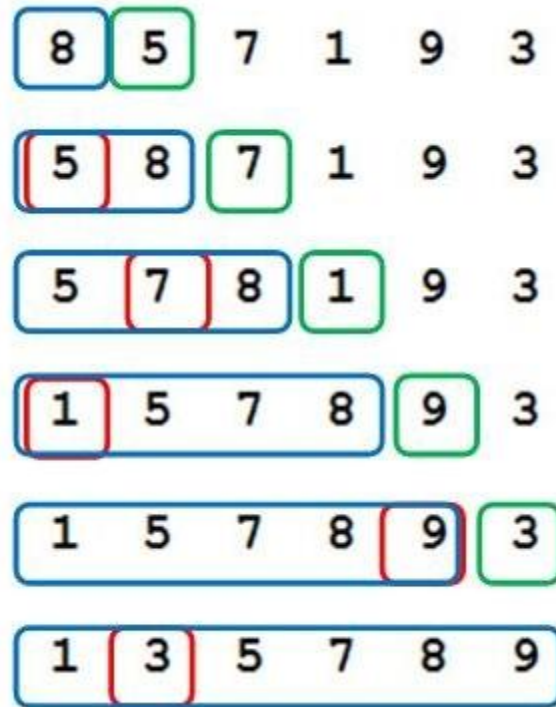
Matrix addition	Output
<pre>a=[[1,1],[1,1]] b=[[2,2],[2,2]] c=[[0,0],[0,0]] for i in range(len(a)): for j in range(len(b)): c[i][j]=a[i][j]+b[i][j] for i in c: print(i)</pre>	<pre>[3, 3] [3, 3]</pre>
Matrix multiplication	Output
<pre>a=[[1,1],[1,1]] b=[[2,2],[2,2]] c=[[0,0],[0,0]] for i in range(len(a)): for j in range(len(b)): for k in range(len(b)): c[i][j]=a[i][j]+a[i][k]*b[k][j] for i in c: print(i)</pre>	<pre>[3, 3] [3, 3]</pre>
Matrix transpose	Output
<pre>a=[[1,3],[1,2]] c=[[0,0],[0,0]] for i in range(len(a)): for j in range(len(a)): c[i][j]=a[j][i] for i in c: print(i)</pre>	<pre>[1, 1] [3, 2]</pre>

Illustrative programs:

Selection sort	Output
<pre> a=input("Enter list:").split() a=list(map(eval,a)) for i in range(0,len(a)): smallest = min(a[i:]) sindex= a.index(smallest) a[i],a[sindex] = a[sindex],a[i] print (a) </pre>	<p>Enter list:23 78 45 8 32 56 [8,2 3, 32, 45,56, 78]</p>



Insertion sort	output
<pre> a=input("enter a list:").split() a=list(map(int,a)) for i in a: j = a.index(i) while j>0: if a[j-1] > a[j]: a[j-1],a[j] = a[j],a[j-1] else: break j = j-1 print (a) </pre>	<p>enter a list: 8 5 7 1 9 3 [1,3,5,7,8,9]</p>

**Merge sort****output**

```
def merge(a,b):
```

```
    c = []
```

```
    while len(a) != 0 and len(b) != 0:
```

```
        if a[0] < b[0]:
```

```
            c.append(a[0])
```

```
            a.remove(a[0])
```

```
        else:
```

```
            c.append(b[0])
```

```
            b.remove(b[0])
```

```
    if len(a) == 0:
```

```
        c=c+b
```

```
    else:
```

```
        c=c+a
```

```
    return c
```

```
def divide(x):
```

```
    if len(x) == 0 or len(x) == 1:
```

```
        return x
```

```
    else:
```

```
        middle = len(x)//2
```

```
        a = divide(x[:middle])
```

```
        b = divide(x[middle:])
```

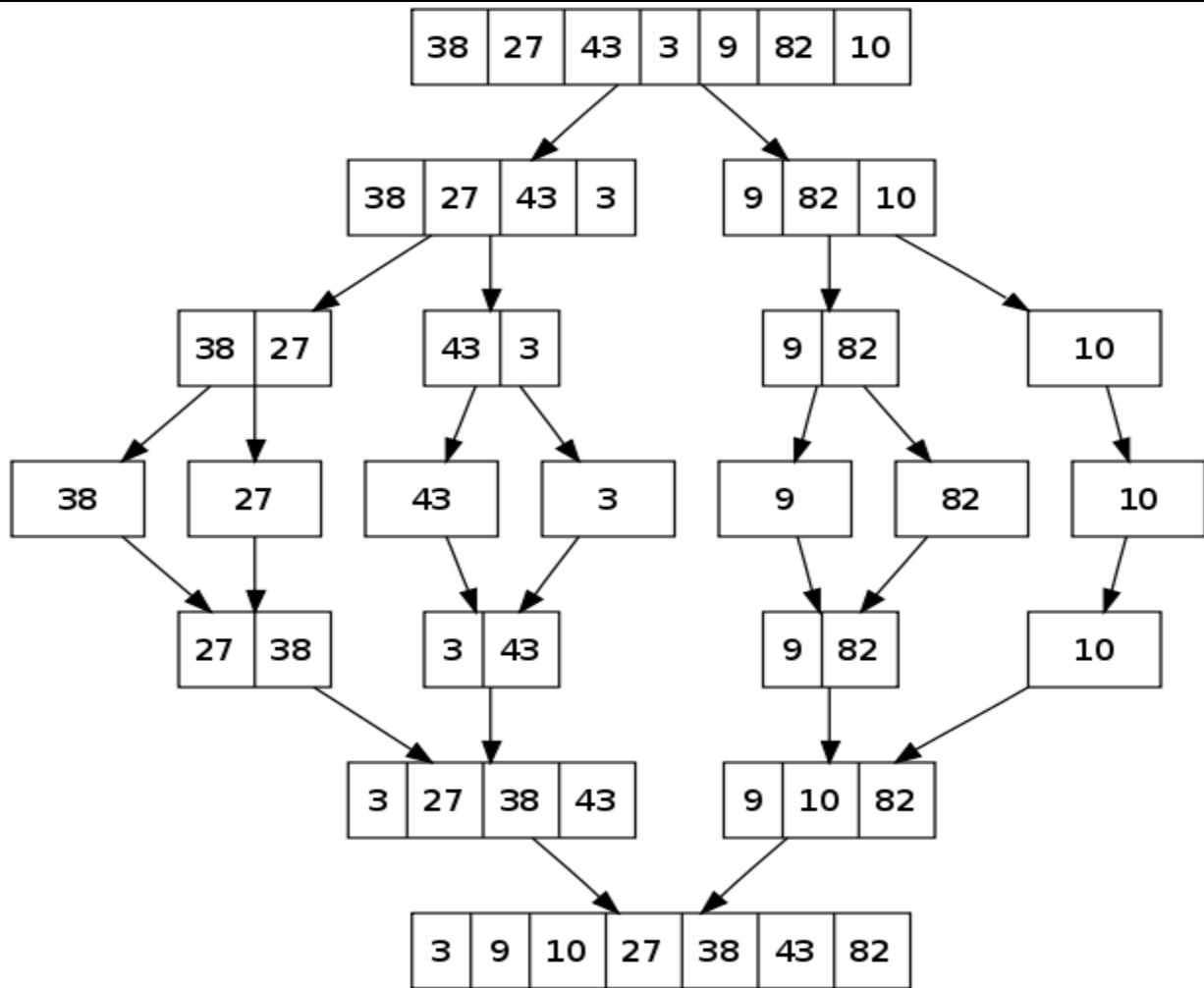
```
        return merge(a,b)
```

```
x=[38,27,43,3,9,82,10]
```

```
c=divide(x)
```

```
print(c)
```

```
[3,9,10,27,38,43,82]
```



Histogram

```

def histogram(a):
    for i in a:
        sum = ""
        while(i>0):
            sum=sum+'#'
            i=i-1
        print(sum)
a=[4,5,7,8,12]
histogram(a)
    
```

Output

```

****
*****
*****
*****
*****
    
```

Calendar program

```

import calendar
y=int(input("enter year:"))
m=int(input("enter month:"))
print(calendar.month(y,m))
    
```

Output

```

enter year:2017
enter month:11
    November 2017
Mo Tu We Th Fr Sa Su
    1 2 3 4 5
    6 7 8 9 10 11 12
    13 14 15 16 17 18 19
    20 21 22 23 24 25 26
    27 28 29 30
    
```

PART - A

1. What is slicing?
2. How can we distinguish between tuples and lists?
3. What will be the output of the given code?
 - a. List=['p','r','i','n','t,']
 - b. Print list[8:]
4. Give the syntax required to convert an integer number into string?
5. List is mutable. Justify?
6. Difference between del and remove methods in List?
7. Difference between pop and remove in list?
8. How are the values in a tuple accessed?
9. What is a Dictionary in Python
10. Define list comprehension
11. Write a python program using list looping
12. What do you meant by mutability and immutability?
13. Define Histogram
14. Define Tuple and show it is immutable with an example.
15. state the difference between aliasing and cloning in list
16. what is list cloning
17. what is deep cloning
18. state the difference between pop and remove method in list
19. create tuple with single element
20. swap two numbers without using third variable
21. define properties of key in dictionary
22. how can you access elements from the dictionary
23. difference between delete and clear method in dictionary
24. What is squeezing in list? give an example
25. How to convert a tuple in to list
26. How to convert a list in to tuple
27. Create a list using list comprehension
28. Advantage of list comprehension
29. What is the use of map () function.

30. How can you return multiple values from function?
31. what is sorting and types of sorting
32. Find length of sequence without using library function.
33. how to pass tuple as argument
34. how to pass a list as argument
35. what is parameter and types of parameter
36. how can you insert values in to dictionary
37. what is key value pair
38. mention different data types can be used in key and value
39. what are the immutable data types available in python
40. What is the use of fromkeys() in dictionary.

PART-B

1. Explain in details about list methods
2. Discuss about operations in list
3. What is cloning? Explain it with example
4. What is aliasing? Explain with example
5. How can you pass list into function? Explain with example.
6. Explain tuples as return values with examples
7. write a program for matrix multiplication
8. write a program for matrix addition
9. write a program for matrix subtraction
10. write a program for matrix transpose
11. write procedure for selection sort
12. explain merge sort with an example
13. explain insertion with example
14. Explain in detail about dictionaries and its methods.
15. Explain in detail about advanced list processing.

GE8151 PROBLEM SOLVING AND PYTHON PROGRAMMING

UNIT I

ALGORITHMIC PROBLEM SOLVING

Algorithms, building blocks of algorithms (statements, state, control flow, functions), notation (pseudo code, flow chart, programming language), algorithmic problem solving, simple strategies for developing algorithms (iteration, recursion). Illustrative problems: find minimum in a list, insert a card in a list of sorted cards, Guess an integer number in a range, Towers of Hanoi.

1.PROBLEM SOLVING

Problem solving is the systematic approach to define the problem and creating number of solutions.

The problem solving process starts with the problem specifications and ends with a Correct program.

1.1 PROBLEM SOLVING TECHNIQUES

Problem solving technique is a set of techniques that helps in providing logic for solving a problem.

Problem Solving Techniques:

Problem solving can be expressed in the form of

1. Algorithms.
2. Flowcharts.
3. Pseudo codes.
4. programs

1.2.ALGORITHM

It is defined as a sequence of instructions that describe a method for solving a problem. In other words it is a step by step procedure for solving a problem.

Properties of Algorithms

- ❖ Should be written in simple English
- ❖ Each and every instruction should be precise and unambiguous.
- ❖ Instructions in an algorithm should not be repeated infinitely.
- ❖ Algorithm should conclude after a finite number of steps.
- ❖ Should have an end point
- ❖ Derived results should be obtained only after the algorithm terminates.

Qualities of a good algorithm

The following are the primary factors that are often used to judge the quality of the algorithms.

Time – To execute a program, the computer system takes some amount of time. The lesser is the time required, the better is the algorithm.

Memory – To execute a program, computer system takes some amount of memory space. The lesser is the memory required, the better is the algorithm.

Accuracy – Multiple algorithms may provide suitable or correct solutions to a given problem, some of these may provide more accurate results than others, and such algorithms may be suitable.

Example:

Example

Write an algorithm to print „Good Morning”

Step 1: Start

Step 2: Print “Good Morning”

Step 3: Stop

2.BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

2.1.Statements:

Statement is a single action in a computer.

In a computer statements might include some of the following actions

- input data-information given to the program
- process data-perform operation on a given input
- output data-processed result

2.2.State:

Transition from one process to another process under specified condition with in a time is called state.

2.3.Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

Sequence:

All the instructions are executed one after another is called sequence execution.

Example:

Add two numbers:

Step 1: Start

Step 2: get a,b

Step 3: calculate $c=a+b$

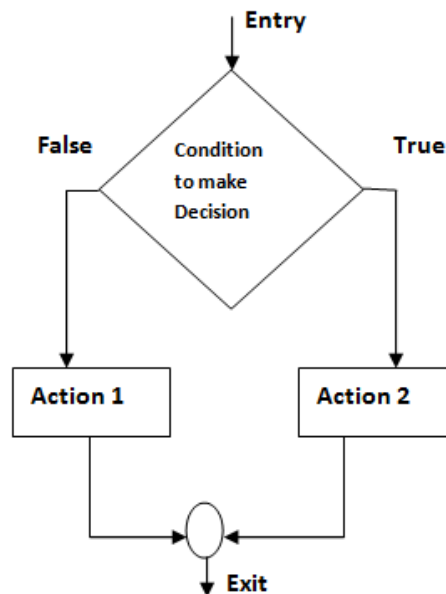
Step 4: Display c

Step 5: Stop

Selection:

A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.

If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.



Example

Write an algorithm to check whether he is eligible to vote?

Step 1: Start

Step 2: Get age

Step 3: if age \geq 18 print "Eligible to vote"

Step 4: else print "Not eligible to vote"

Step 6: Stop

Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called looping or iteration.

Example

Write an algorithm to print all natural numbers up to n

Step 1: Start

Step 2: get n value.

Step 3: initialize $i=1$

Step 4: if $(i \leq n)$ go to step 5 else go to step 7

Step 5: Print i value and increment i value by 1

Step 6: go to step 4

Step 7: Stop

2.4.Functions:

- ❖ Function is a sub program which consists of block of code(set of instructions) that performs a particular task.
- ❖ For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

Benefits of Using Functions

- ❖ Reduction in line of code
- ❖ code reuse
- ❖ Better readability
- ❖ Information hiding
- ❖ Easy to debug and test
- ❖ Improved maintainability

Example:

Algorithm for addition of two numbers using function

Main function()

Step 1: Start

Step 2: Call the function add()

Step 3: Stop

sub function add()

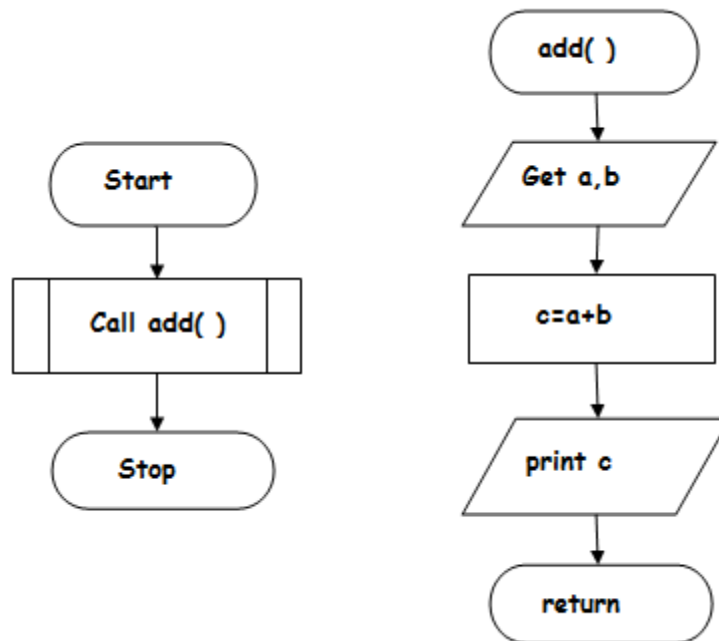
Step 1: Function start

Step 2: Get a, b Values

Step 3: add $c=a+b$

Step 4: Print c

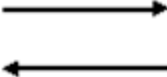






Step 5: Return



3.NOTATIONS

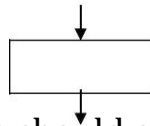
3.1.FLOW CHART

Flow chart is defined as graphical representation of the logic for problem solving. The purpose of flowchart is making the logic of the program clear in a visual representation.

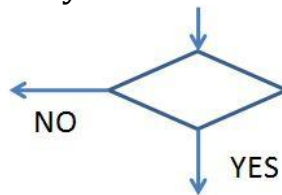
Symbol	Symbol Name	Description
	Flow Lines	Used to connect symbols
	Terminal	Used to start, pause or halt in the program logic
	Input/output	Represents the information entering or leaving the system
	Processing	Represents arithmetic and logical instructions
	Decision	Represents a decision to be made
	Connector	Used to Join different flow lines
	Sub function	used to call function

Rules for drawing a flowchart

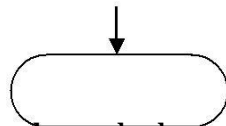
1. The flowchart should be clear, neat and easy to follow.
2. The flowchart must have a logical start and finish.
3. Only one flow line should come out from a process symbol.



4. Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol.



5. Only one flow line is used with a terminal symbol.



6. Within standard symbols, write briefly and precisely.
7. Intersection of flow lines should be avoided.

Advantages of flowchart:

1. **Communication:** - Flowcharts are better way of communicating the logic of a system to all concerned.
2. **Effective analysis:** - With the help of flowchart, problem can be analyzed in more effective way.

3. **Proper documentation:** - Program flowcharts serve as a good program documentation, which is needed for various purposes.
4. **Efficient Coding:** - The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. **Proper Debugging:** - The flowchart helps in debugging process.
6. **Efficient Program Maintenance:** - The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

Disadvantages of flow chart:

1. **Complex logic:** - Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
2. **Alterations and Modifications:** - If alterations are required the flowchart may require re-drawing completely.
3. **Reproduction:** - As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
4. **Cost:** For large application the time and cost of flowchart drawing becomes costly.

3.2.PSEUDO CODE:

- ❖ Pseudo code consists of short, readable and formally styled English languages used for explain an algorithm.
- ❖ It does not include details like variable declaration, subroutines.
- ❖ It is easier to understand for the programmer or non programmer to understand the general working of the program, because it is not based on any programming language.
- ❖ It gives us the sketch of the program before actual coding.
- ❖ It is not a machine readable
- ❖ Pseudo code can't be compiled and executed.
- ❖ There is no standard syntax for pseudo code.

Guidelines for writing pseudo code:

- ❖ Write one statement per line
- ❖ Capitalize initial keyword
- ❖ Indent to hierarchy
- ❖ End multiline structure
- ❖ Keep statements language independent

Common keywords used in pseudocode

The following gives common keywords used in pseudocodes.

1. **//:** This keyword used to represent a comment.
2. **BEGIN,END:** Begin is the first statement and end is the last statement.
3. **INPUT, GET, READ:** The keyword is used to inputting data.
4. **COMPUTE, CALCULATE:** used for calculation of the result of the given expression.
5. **ADD, SUBTRACT, INITIALIZE** used for addition, subtraction and initialization.
6. **OUTPUT, PRINT, DISPLAY:** It is used to display the output of the program.
7. **IF, ELSE, ENDIF:** used to make decision.
8. **WHILE, ENDWHILE:** used for iterative statements.
9. **FOR, ENDFOR:** Another iterative incremented/decremented tested automatically.

Syntax for if else:	Example: Greates of two numbers
IF (condition) THEN statement ... ELSE statement ... ENDIF	BEGIN READ a,b IF (a>b) THEN DISPLAY a is greater ELSE DISPLAY b is greater END IF END
Syntax for For:	Example: Print n natural numbers
FOR(<i>start-value</i> to <i>end-value</i>) DO <i>statement</i> ... ENDFOR	BEGIN GET n INITIALIZE i=1 FOR (i<=n) DO PRINT i i=i+1 ENDFOR END
Syntax for While:	Example: Print n natural numbers
WHILE (condition) DO statement ... ENDWHILE	BEGIN GET n INITIALIZE i=1 WHILE(i<=n) DO PRINT i i=i+1 ENDWHILE END

Advantages:

- ❖ Pseudo is independent of any language; it can be used by most programmers.
- ❖ It is easy to translate pseudo code into a programming language.
- ❖ It can be easily modified as compared to flowchart.
- ❖ Converting a pseudo code to programming language is very easy as compared with converting a flowchart to programming language.

Disadvantages:

- ❖ It does not provide visual representation of the program's logic.
- ❖ There are no accepted standards for writing pseudo codes.
- ❖ It cannot be compiled nor executed.
- ❖ For a beginner, It is more difficult to follow the logic or write pseudo code as compared to flowchart.

Example:

Addition of two numbers:

```
BEGIN
GET a,b
ADD c=a+b
PRINT c
END
```

Algorithm	Flowchart	Pseudo code
An algorithm is a sequence of instructions used to solve a problem	It is a graphical representation of algorithm	It is a language representation of algorithm.
User needs knowledge to write algorithm.	not need knowledge of program to draw or understand flowchart	Not need knowledge of program language to understand or write a pseudo code.

3.3.PROGRAMMING LANGUAGE

A programming language is a set of symbols and rules for instructing a computer to perform specific tasks. The programmers have to follow all the specified rules before writing program using programming language. The user has to communicate with the computer using language which it can understand.

Types of programming language

1. Machine language
2. Assembly language
3. High level language

Machine language:

The computer can understand only machine language which uses 0's and 1's. In machine language the different instructions are formed by taking different combinations of 0's and 1's.

Advantages:

Translation free:

Machine language is the only language which the computer understands. For executing any program written in any programming language, the conversion to machine language is necessary. The program written in machine language can be executed directly on computer. In this case any conversion process is not required.

High speed

The machine language program is translation free. Since the conversion time is saved, the execution of machine language program is extremely fast.

Disadvantage:

- It is hard to find errors in a program written in the machine language.
- Writing program in machine language is a time consuming process.

Machine dependent: According to architecture used, the computer differs from each other. So machine language differs from computer to computer. So a program developed for a particular type of computer may not run on other type of computer.

Assembly language:

- To overcome the issues in programming language and make the programming process easier, an assembly language is developed which is logically equivalent to machine language but it is easier for people to read, write and understand.

- Assembly language is symbolic representation of machine language. Assembly languages are symbolic programming language that uses symbolic notation to represent machine language instructions. They are called low level language because they are so closely related to the machines.

Ex: ADD a, b

Assembler:

Assembler is the program which translates assembly language instruction in to a machine language.

Advantage:

- Easy to understand and use.
- It is easy to locate and correct errors.

Disadvantage

Machine dependent

The assembly language program which can be executed on the machine depends on the architecture of that computer.

Hard to learn

It is machine dependent, so the programmer should have the hardware knowledge to create applications using assembly language.

Less efficient

- Execution time of assembly language program is more than machine language program.
- Because assembler is needed to convert from assembly language to machine language.

High level language

High level language contains English words and symbols. The specified rules are to be followed while writing program in high level language. The interpreter or compilers are used for converting these programs in to machine readable form.

Translating high level language to machine language

The programs that translate high level language in to machine language are called interpreter or compiler.

Compiler:

A compiler is a program which translates the source code written in a high level language in to object code which is in machine language program. Compiler reads the whole program written in high level language and translates it to machine language. If any error is found it display error message on the screen.

Interpreter

Interpreter translates the high level language program in line by line manner. The interpreter translates a high level language statement in a source program to a machine

code and executes it immediately before translating the next statement. When an error is found the execution of the program is halted and error message is displayed on the screen.

Advantages

Readability

High level language is closer to natural language so they are easier to learn and understand

Machine independent

High level language program have the advantage of being portable between machines.

Easy debugging

Easy to find and correct error in high level language

Disadvantages

Less efficient

The translation process increases the execution time of the program. Programs in high level language require more memory and take more execution time to execute.

They are divided into following categories:

1. Interpreted programming languages
2. Functional programming languages
3. Compiled programming languages
4. Procedural programming languages
5. Scripting programming language
6. Markup programming language
7. Concurrent programming language
8. Object oriented programming language

Interpreted programming languages:

An interpreted language is a programming language for which most of its implementation executes instructions directly, without previously compiling a program into machine language instructions. The interpreter executes the program directly translating each statement into a sequence of one or more subroutines already compiled into machine code.

Examples:

Pascal
Python

Functional programming language:

Functional programming language defines every computation as a mathematical evaluation. They focus on the programming languages are bound to mathematical calculations

Examples:

Clean
Haskell

Compiled Programming language:

A compiled programming is a programming language whose implementation are typically compilers and not interpreters.

It will produce a machine code from source code.

Examples:

C
C++
C#
JAVA

Procedural programming language:

Procedural (imperative) programming implies specifying the steps that the programs should take to reach to an intended state.

A procedure is a group of statements that can be referred through a procedure call. Procedures help in the reuse of code. Procedural programming makes the programs structured and easily traceable for program flow.

Examples:

Hyper talk
MATLAB

Scripting language:

Scripting language are programming languages that control an application. Scripts can execute independent of any other application. They are mostly embedded in the application that they control and are used to automate frequently executed tasks like communicating with external program.

Examples:

Apple script
VB script

Markup languages:

A markup language is an artificial language that uses annotations to text that define hoe the text is to be displayed.

Examples:

HTML
XML

Concurrent programming language:

Concurrent programming is a computer programming technique that provides for the execution of operation concurrently, either with in a single computer or across a number of systems.

Examples:

Joule
Limbo

Object oriented programming language:

Object oriented programming is a programming paradigm based on the concept of objects which may contain data in the form of procedures often known as methods.

Examples:

Lava

Moto

4.ALGORITHMIC PROBLEM SOLVING:

Algorithmic problem solving is solving problem that require the formulation of an algorithm for the solution.

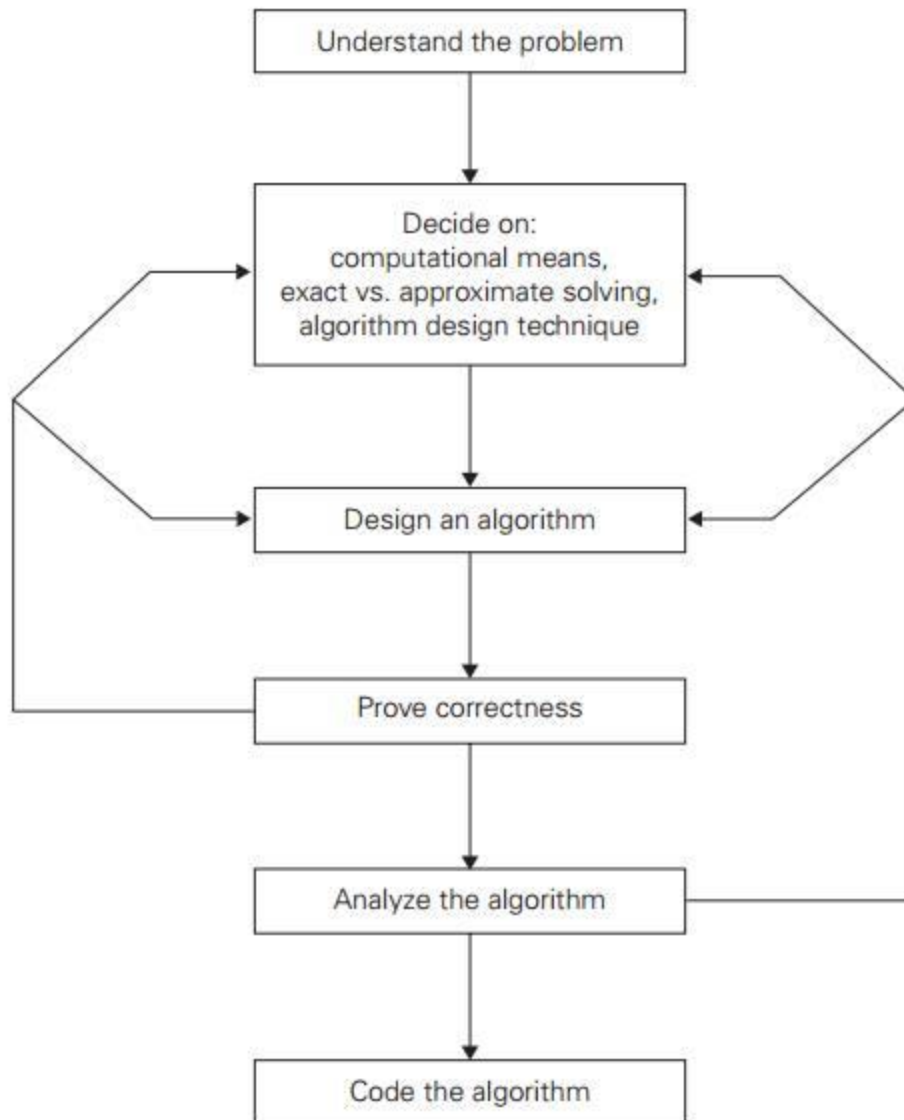


FIGURE 1.2 Algorithm design and analysis process.

Understanding the Problem

- ❖ It is the process of finding the input of the problem that the algorithm solves.
- ❖ It is very important to specify exactly the set of inputs the algorithm needs to handle.
- ❖ A correct algorithm is not one that works most of the time, but one that works correctly for *all* legitimate inputs.

Ascertaining the Capabilities of the Computational Device

- ❖ If the instructions are executed one after another, it is called sequential algorithm.

- ❖ If the instructions are executed concurrently, it is called parallel algorithm.

Choosing between Exact and Approximate Problem Solving

- ❖ The next principal decision is to choose between solving the problem exactly or solving it approximately.
- ❖ Based on this, the algorithms are classified as exact *algorithm* and *approximation algorithm*.

Deciding a data structure:

- ❖ Data structure plays a vital role in designing and analysis the algorithms.
- ❖ Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance
- ❖ Algorithm+ Data structure=programs.

Algorithm Design Techniques

- ❖ An *algorithm design technique* (or “strategy” or “paradigm”) is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- ❖ Learning these techniques is of utmost importance for the following reasons.
- ❖ First, they provide guidance for designing algorithms for new problems,
- ❖ Second, algorithms are the cornerstone of computer science

Methods of Specifying an Algorithm

- ❖ *Pseudocode* is a mixture of a natural language and programming language-like constructs. Pseudocode is usually more precise than natural language, and its usage often yields more succinct algorithm descriptions.
- ❖ In the earlier days of computing, the dominant vehicle for specifying algorithms was a *flowchart*, a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.
- ❖ **Programming language** can be fed into an electronic computer directly. Instead, it needs to be converted into a computer program written in a particular computer language. We can look at such a program as yet another way of specifying the algorithm, although it is preferable to consider it as the algorithm's implementation.

Proving an Algorithm's Correctness

- ❖ Once an algorithm has been specified, you have to prove its *correctness*. That is, you have to prove that the algorithm yields a required result for every legitimate input in a finite amount of time.
- ❖ A common technique for proving correctness is to use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
- ❖ It might be worth mentioning that although tracing the algorithm's performance for a few specific inputs can be a very worthwhile activity, it cannot prove the algorithm's correctness conclusively. But in order to show that an algorithm is incorrect, you need just one instance of its input for which the algorithm fails.

Analysing an Algorithm

1. *Efficiency.*

Time efficiency, indicating how fast the algorithm runs,

Space efficiency, indicating how much extra memory it uses.

2. *simplicity.*

- ❖ An algorithm should be precisely defined and investigated with mathematical expressions.
- ❖ Simpler algorithms are easier to understand and easier to program.
- ❖ Simple algorithms usually contain fewer bugs.

Coding an Algorithm

- ❖ Most algorithms are destined to be ultimately implemented as computer programs. Programming an algorithm presents both a peril and an opportunity.
- ❖ A working program provides an additional opportunity in allowing an empirical analysis of the underlying algorithm. Such an analysis is based on timing the program on several inputs and then analysing the results obtained.

5.SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS:

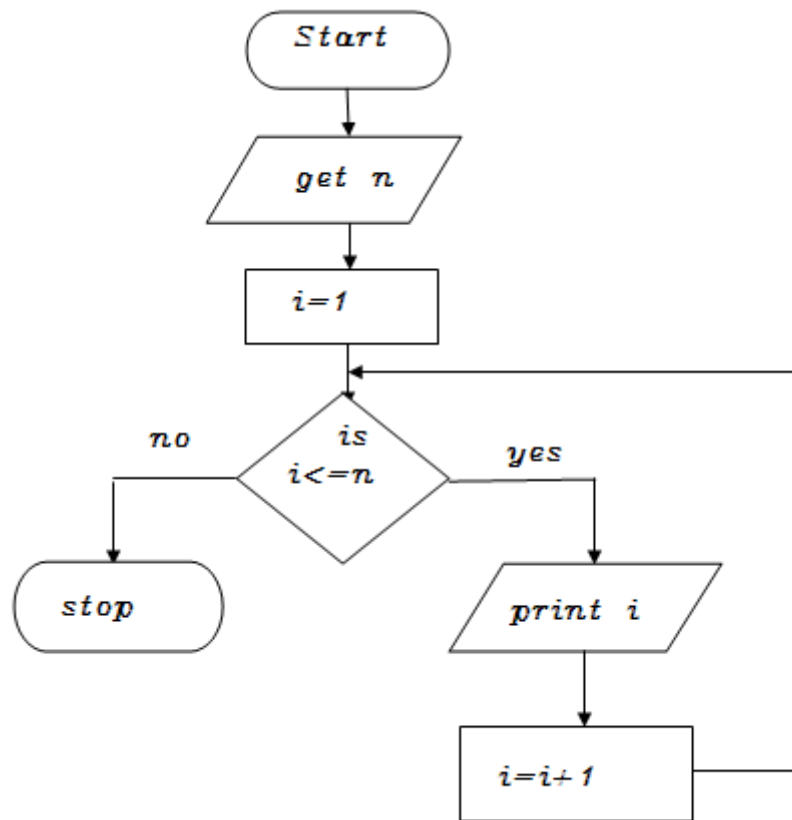
1. iterations
2. Recursions

5.1.Iterations:

A sequence of statements is executed until a specified condition is true is called iterations.

1. for loop
2. While loop

<u>Syntax for For:</u>	<u>Example: Print n natural numbers</u>
<pre>FOR(start-value to end-value) DO statement ... ENDFOR</pre>	<pre>BEGIN GET n INITIALIZE i=1 FOR (i<=n) DO PRINT i i=i+1 ENDFOR END</pre>
<u>Syntax for While:</u>	<u>Example: Print n natural numbers</u>
<pre>WHILE (condition) DO statement ... ENDWHILE</pre>	<pre>BEGIN GET n INITIALIZE i=1 WHILE(i<=n) DO PRINT i i=i+1 ENDWHILE END</pre>



5.2.Recursions:

- ❖ A function that calls itself is known as recursion.
- ❖ Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

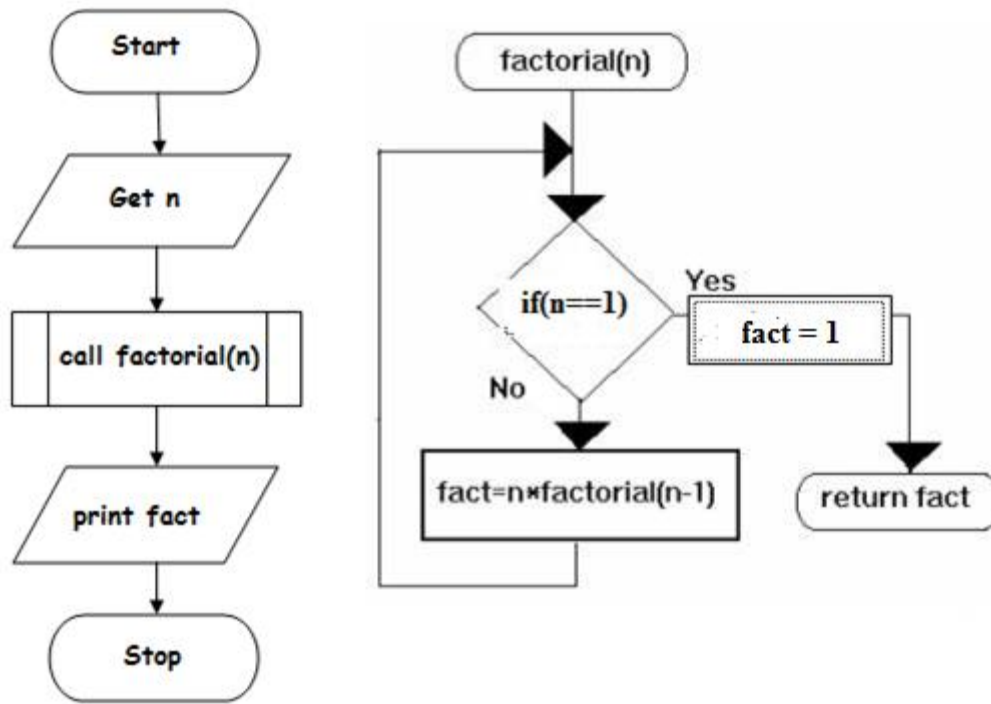
Algorithm for factorial of n numbers using recursion:

Main function:

- Step1: Start
- Step2: Get n
- Step3: call factorial(n)
- Step4: print fact
- Step5: Stop

Sub function factorial(n):

- Step1: if(n==1) then fact=1 return fact
- Step2: else fact=n*factorial(n-1) and return fact



Pseudo code for factorial using recursion:

Main function:

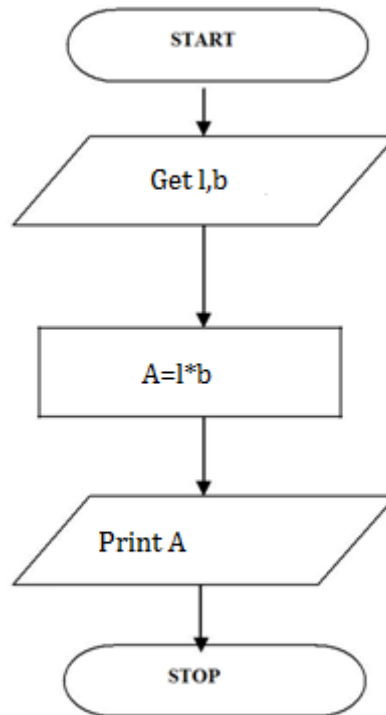
```

BEGIN
GET n
CALL factorial(n)
PRINT fact
BIN
    
```

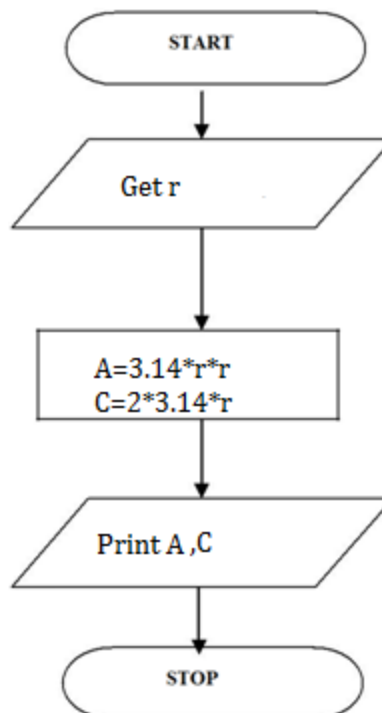
Sub function factorial(n):

```

IF(n==1) THEN
    fact=1
    RETURN fact
ELSE
    RETURN fact=n*factorial(n-1)
    
```

More examples:**Write an algorithm to find area of a rectangle****Step 1:** Start**Step 2:** get l,b values**Step 3:** Calculate $A=l*b$ **Step 4:** Display A**Step 5:** Stop

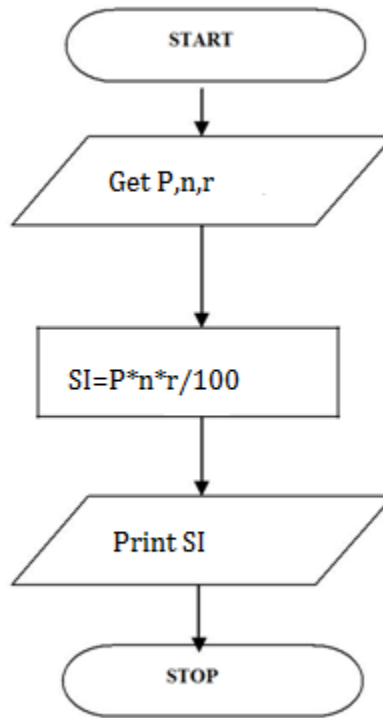
BEGIN
 READ l,b
 CALCULATE $A=l*b$
 DISPLAY A
 END

Write an algorithm for Calculating area and circumference of circle**Step 1:** Start**Step 2:** get r value**Step 3:** Calculate $A=3.14*r*r$ **Step 4:** Calculate $C=2*3.14*r$ **Step 5:** Display A,C**Step 6:** Stop

BEGIN
 READ r
 CALCULATE A and C
 $A=3.14*r*r$
 $C=2*3.14*r$
 DISPLAY A
 END

Write an algorithm for Calculating simple interest

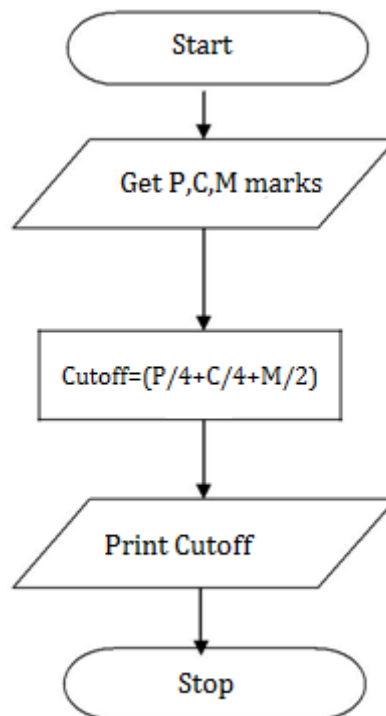
Step 1: Start
Step 2: get P, n, r value
Step3: Calculate
 $SI=(p*n*r)/100$
Step 4: Display S
Step 5: Stop



BEGIN
 READ P, n, r
 CALCULATE S
 $SI=(p*n*r)/100$
 DISPLAY SI
 END

Write an algorithm for Calculating engineering cutoff

Step 1: Start
Step2: get P,C,M value
Step3: calculate
 $Cutoff= (P/4+C/4+M/2)$
Step 4: Display Cutoff
Step 5: Stop

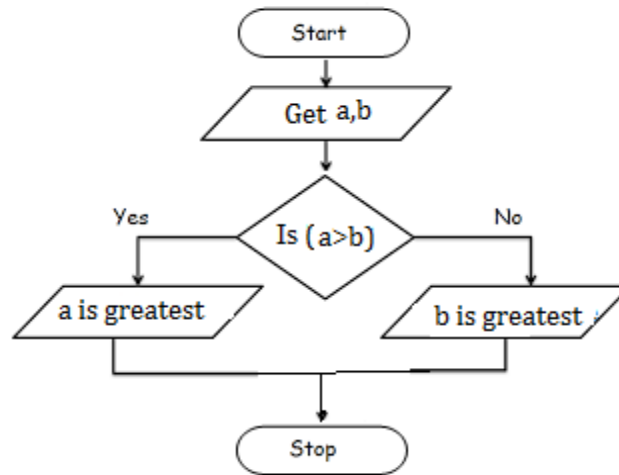


BEGIN
 READ P,C,M
 CALCULATE
 $Cutoff= (P/4+C/4+M/2)$
 DISPLAY Cutoff
 END

To check greatest of two numbers

Step 1: Start
Step 2: get a,b value
Step 3: check if(a>b) print a is greater
Step 4: else b is greater
Step 5: Stop

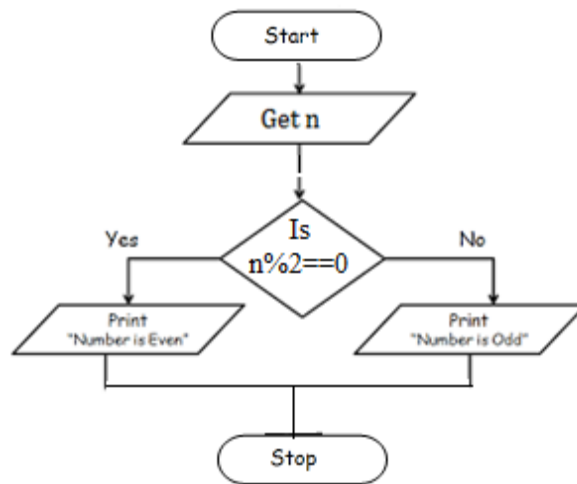
```
BEGIN  
READ a,b  
IF (a>b) THEN  
DISPLAY a is greater  
ELSE  
DISPLAY b is greater  
END IF  
END
```



To check leap year or not

Step 1: Start
Step 2: get y
Step 3: if(y%4==0) print leap year
Step 4: else print not leap year
Step 5: Stop

```
BEGIN  
READ y  
IF (y%4==0) THEN  
DISPLAY leap year  
ELSE  
DISPLAY not leap year  
END IF  
END
```

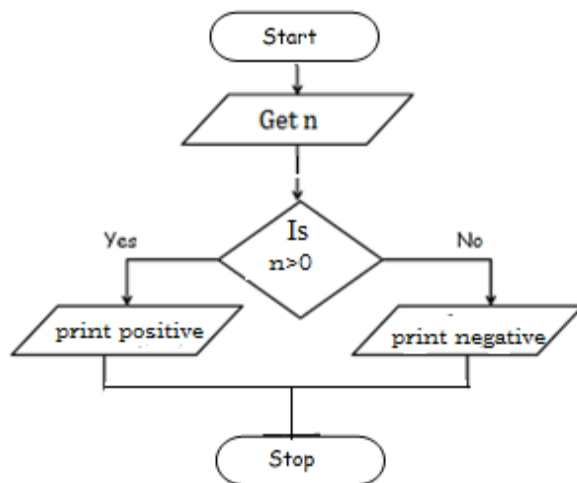


To check positive or negative number

- Step 1:** Start
- Step 2:** get num
- Step 3:** check if(num>0) print a is positive
- Step 4:** else num is negative
- Step 5:** Stop

```

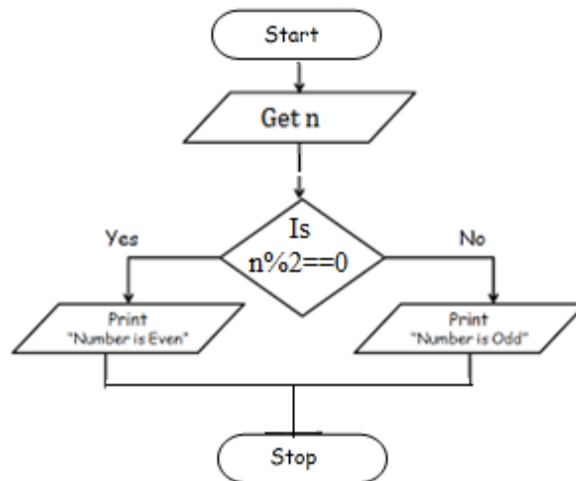
BEGIN
READ num
IF (num>0) THEN
DISPLAY num is positive
ELSE
DISPLAY num is negative
END IF
END
  
```



To check odd or even number

Step 1: Start
 Step 2: get num
 Step 3: check if($\text{num}\%2==0$) print num is even
 Step 4: else num is odd
 Step 5: Stop

```
BEGIN
READ num
IF (num%2==0) THEN
DISPLAY num is even
ELSE
DISPLAY num is odd
END IF
END
```



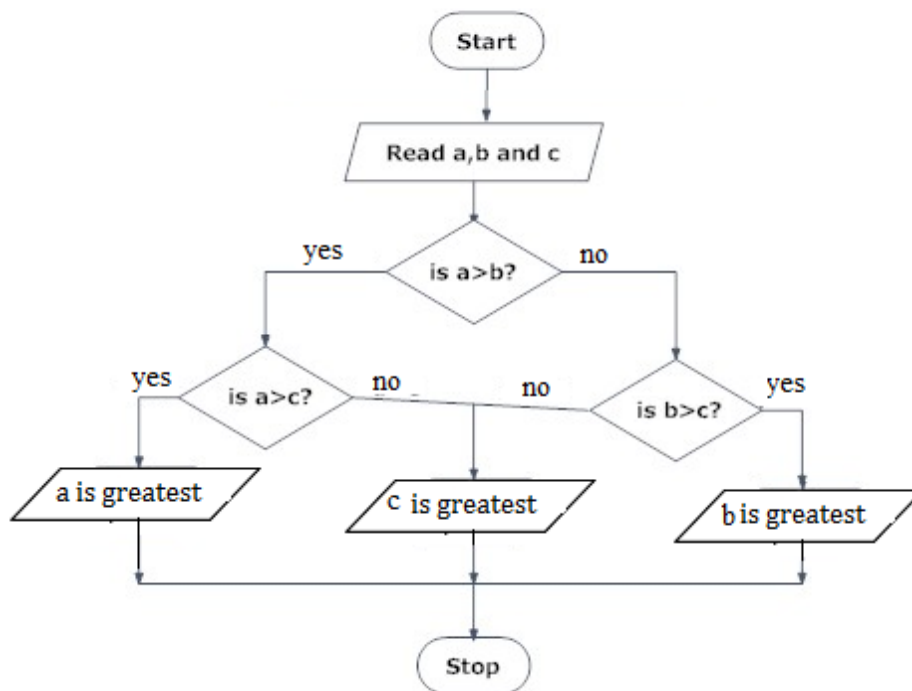
To check greatest of three numbers

Step1: Start
 Step2: Get A, B, C
 Step3: if($A>B$) goto Step4 else goto step5
 Step4: If($A>C$) print A else print C
 Step5: If($B>C$) print B else print C
 Step6: Stop

```

BEGIN
READ a, b, c
IF (a>b) THEN
  IF(a>c) THEN
    DISPLAY a is greater
  ELSE
    DISPLAY c is greater
  END IF
ELSE
  IF(b>c) THEN
    DISPLAY b is greater
  ELSE
    DISPLAY c is greater
  END IF
END IF
END

```



Write an algorithm to check whether given number is +ve, -ve or zero.

Step 1: Start

Step 2: Get n value.

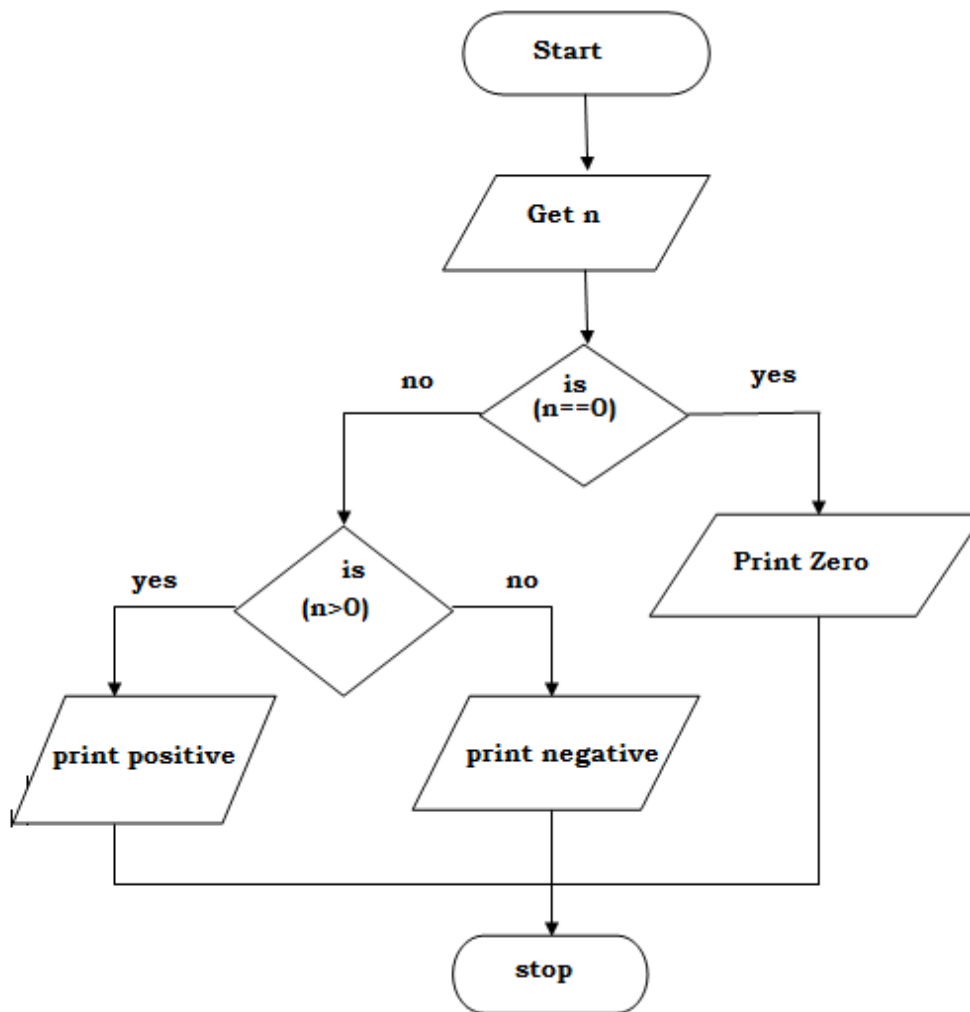
Step 3: if (n ==0) print "Given number is Zero" Else goto step4

Step 4: if (n > 0) then Print "Given number is +ve"

Step 5: else Print "Given number is -ve"

Step 6: Stop

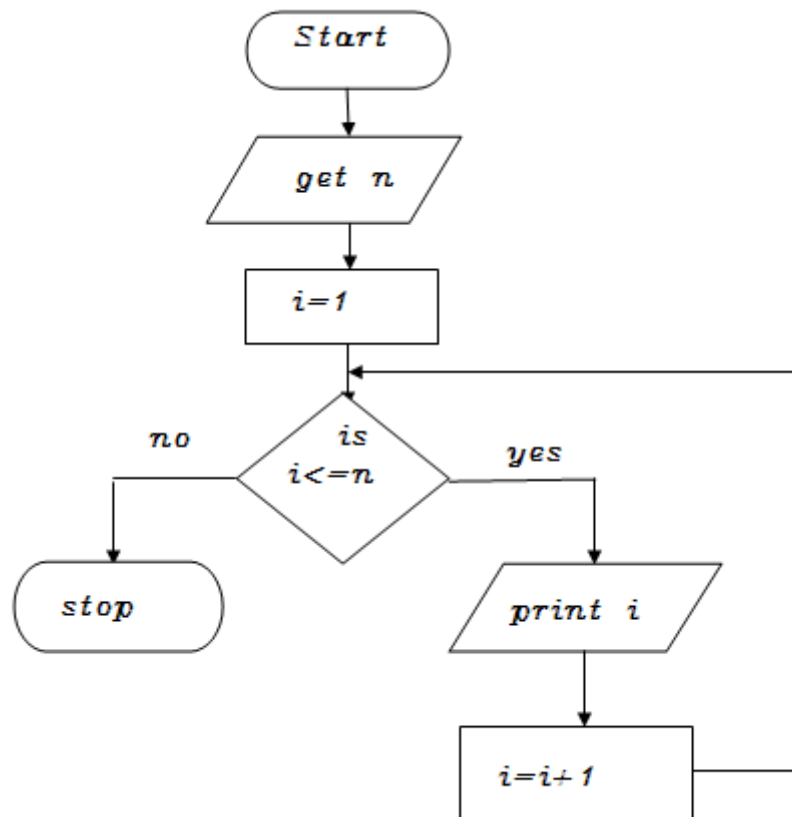
```
BEGIN
GET n
IF(n==0) THEN
  DISPLAY " n is zero"
ELSE
  IF(n>0) THEN
    DISPLAY "n is positive"
  ELSE
    DISPLAY "n is positive"
  END IF
END IF
END
```



Write an algorithm to print all natural numbers up to n

- Step 1:** Start
- Step 2:** get n value.
- Step 3:** initialize i=1
- Step 4:** if (i<=n) go to step 5 else go to step 8
- Step 5:** Print i value
- step 6 :** increment i value by 1
- Step 7:** go to step 4
- Step 8:** Stop

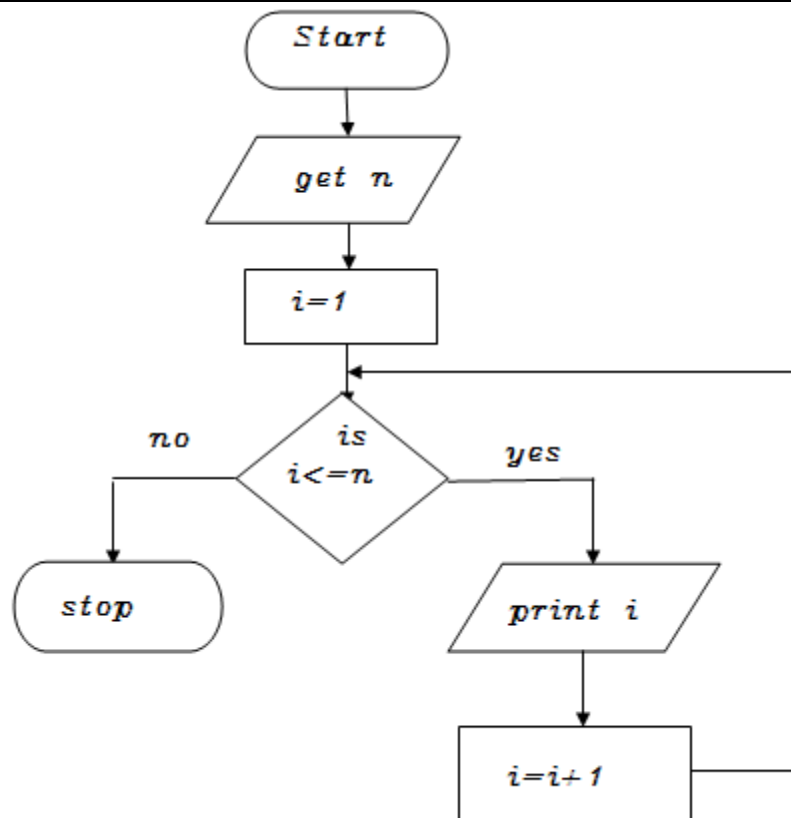
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
    i=i+1
ENDWHILE
END
```



Write an algorithm to print n odd numbers

- Step 1: start
- step 2: get n value
- step 3: set initial value i=1
- step 4: check if(i<=n) goto step 5 else goto step 8
- step 5: print i value
- step 6: increment i value by 2
- step 7: goto step 4
- step 8: stop

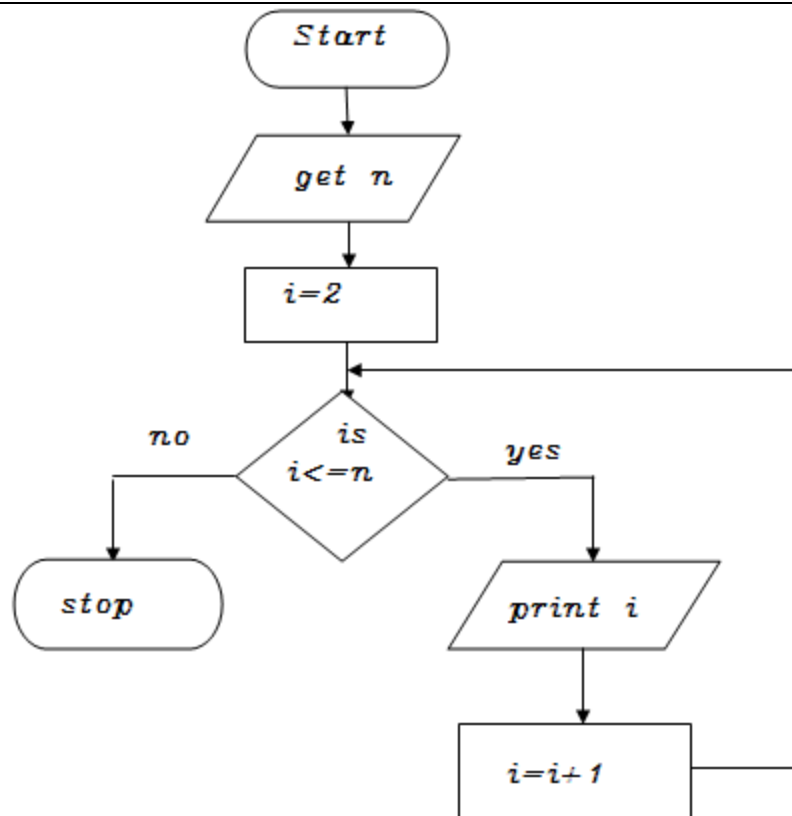
```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print n even numbers

- Step 1: start
- step 2: get n value
- step 3: set initial value i=2
- step 4: check if(i<=n) goto step 5 else goto step8
- step 5: print i value
- step 6: increment i value by 2
- step 7: goto step 4
- step 8: stop

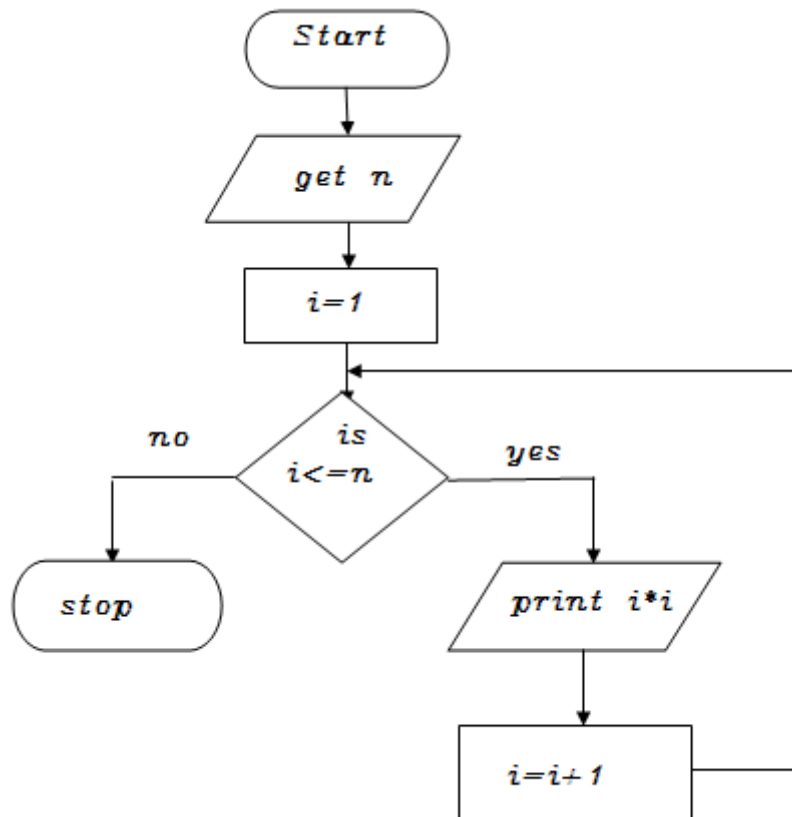
```
BEGIN
GET n
INITIALIZE i=2
WHILE(i<=n) DO
    PRINT i
    i=i+2
ENDWHILE
END
```



Write an algorithm to print squares of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

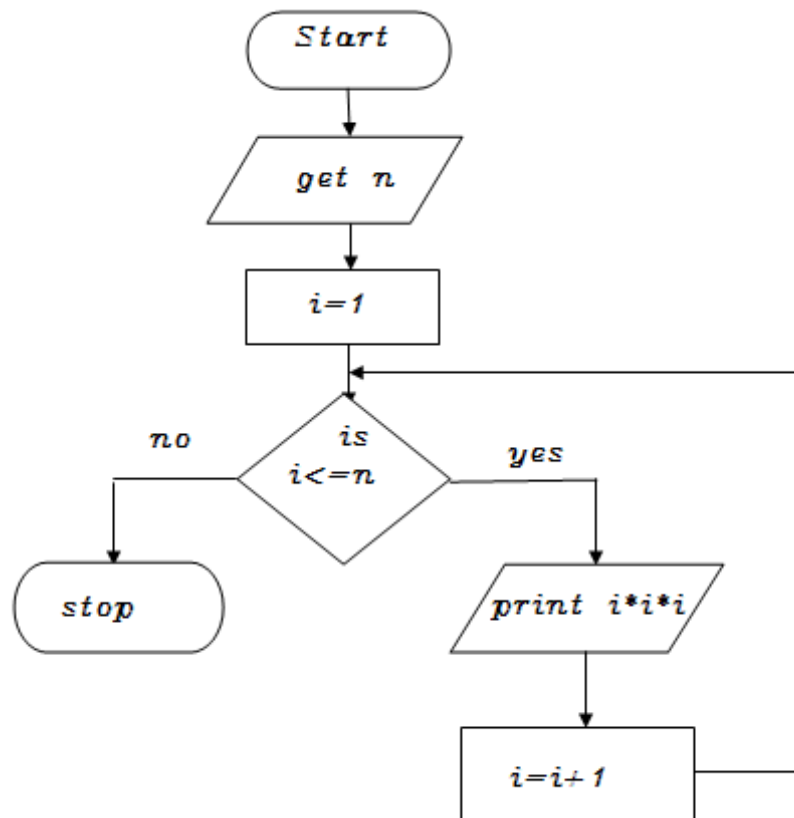
```
BEGIN  
GET n  
INITIALIZE i=1  
WHILE(i<=n) DO  
    PRINT i*i  
    i=i+1  
ENDWHILE  
END
```



Write an algorithm to print to print cubes of a number

Step 1: start
step 2: get n value
step 3: set initial value i=1
step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: print i*i *i value
step 6: increment i value by 1
step 7: goto step 4
step 8: stop

```
BEGIN
GET n
INITIALIZE i=1
WHILE(i<=n) DO
    PRINT i*i*i
    i=i+2
ENDWHILE
END
```



Write an algorithm to find sum of a given number

Step 1: start

step 2: get n value

step 3: set initial value $i=1$, $sum=0$

Step 4: check i value if($i \leq n$) goto step 5 else goto step8

step 5: calculate $sum=sum+i$

step 6: increment i value by 1

step 7: goto step 4

step 8: print sum value

step 9: stop

BEGIN

GET n

INITIALIZE $i=1, sum=0$

WHILE($i \leq n$) DO

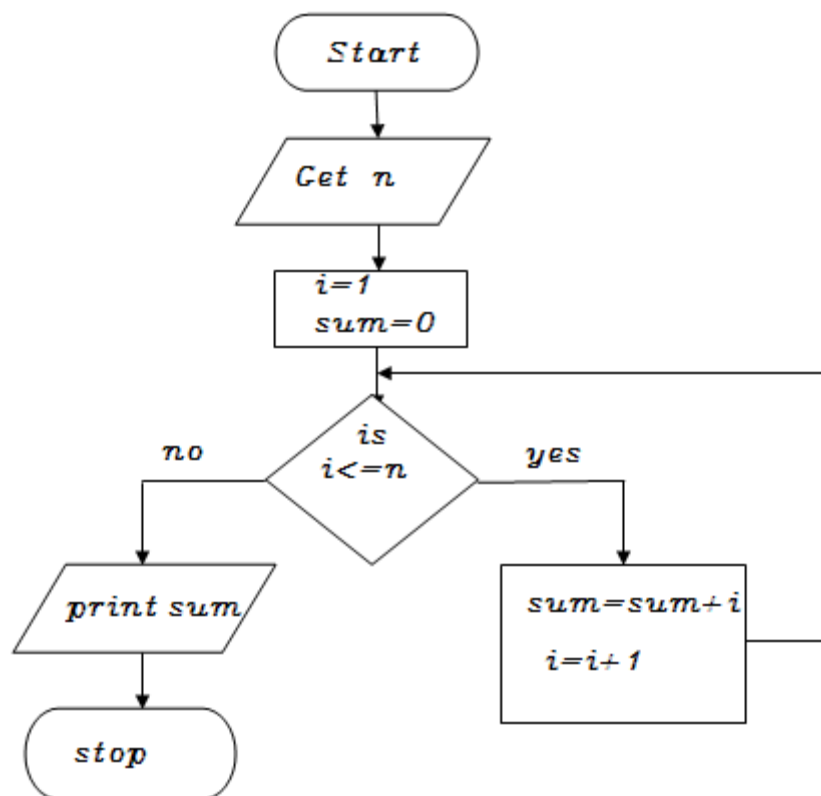
$sum=sum+i$

$i=i+1$

ENDWHILE

PRINT sum

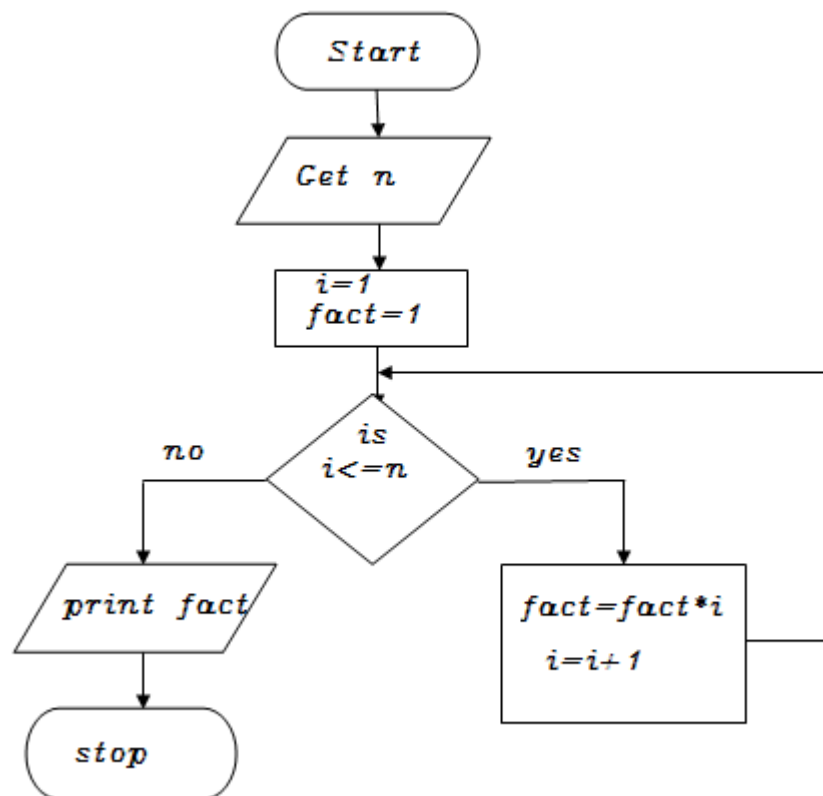
END



Write an algorithm to find factorial of a given number

- Step 1: start
- step 2: get n value
- step 3: set initial value i=1, fact=1
- Step 4: check i value if(i<=n) goto step 5 else goto step8
- step 5: calculate fact=fact*i
- step 6: increment i value by 1
- step 7: goto step 4
- step 8: print fact value
- step 9: stop

```
BEGIN
GET n
INITIALIZE i=1,fact=1
WHILE(i<=n) DO
    fact=fact*i
    i=i+1
ENDWHILE
PRINT fact
END
```



Basic python programs:

Addition of two numbers	Output
<pre>a=eval(input("enter first no")) b=eval(input("enter second no")) c=a+b print("the sum is ",c)</pre>	<pre>enter first no 5 enter second no 6 the sum is 11</pre>
Area of rectangle	Output
<pre>l=eval(input("enter the length of rectangle")) b=eval(input("enter the breath of rectangle")) a=l*b print(a)</pre>	<pre>enter the length of rectangle 5 enter the breath of rectangle 6 30</pre>
Area & circumference of circle	output
<pre>r=eval(input("enter the radius of circle")) a=3.14*r*r c=2*3.14*r print("the area of circle",a) print("the circumference of circle",c)</pre>	<pre>enter the radius of circle4 the area of circle 50.24 the circumference of circle 25.12</pre>
Calculate simple interest	Output
<pre>p=eval(input("enter principle amount")) n=eval(input("enter no of years")) r=eval(input("enter rate of interest")) si=p*n*r/100 print("simple interest is",si)</pre>	<pre>enter principle amount 5000 enter no of years 4 enter rate of interest6 simple interest is 1200.0</pre>
Calculate engineering cutoff	Output
<pre>p=eval(input("enter physics marks")) c=eval(input("enter chemistry marks")) m=eval(input("enter maths marks")) cutoff=(p/4+c/4+m/2) print("cutoff =",cutoff)</pre>	<pre>enter physics marks 100 enter chemistry marks 99 enter maths marks 96 cutoff = 97.75</pre>
Check voting eligibility	output
<pre>age=eval(input("enter ur age")) If(age>=18): print("eligible for voting") else: print("not eligible for voting")</pre>	<pre>Enter ur age 19 Eligible for voting</pre>

Find greatest of three numbers	output
<pre>a=eval(input("enter the value of a")) b=eval(input("enter the value of b")) c=eval(input("enter the value of c")) if(a>b): if(a>c): print("the greatest no is",a) else: print("the greatest no is",c) else: if(b>c): print("the greatest no is",b) else: print("the greatest no is",c)</pre>	<pre>enter the value of a 9 enter the value of a 1 enter the value of a 8 the greatest no is 9</pre>

Programs on for loop

Print n natural numbers	Output
<pre>for i in range(1,5,1): print(i)</pre>	<pre>1 2 3 4</pre>
Print n odd numbers	Output
<pre>for i in range(1,10,2): print(i)</pre>	<pre>1 3 5 7 9</pre>
Print n even numbers	Output
<pre>for i in range(2,10,2): print(i)</pre>	<pre>2 4 6 8</pre>
Print squares of numbers	Output
<pre>for i in range(1,5,1): print(i*i)</pre>	<pre>1 4 9 16</pre>
Print squares of numbers	Output
<pre>for i in range(1,5,1): print(i*i*i)</pre>	<pre>1 8 27 64</pre>

Programs on while loop

Print n natural numbers	Output
<pre>i=1 while(i<=5): print(i) i=i+1</pre>	<pre>1 2 3 4 5</pre>
Print n odd numbers	Output
<pre>i=2 while(i<=10): print(i) i=i+2</pre>	<pre>2 4 6 8 10</pre>
Print n even numbers	Output
<pre>i=1 while(i<=10): print(i) i=i+2</pre>	<pre>1 3 5 7 9</pre>
Print n squares of numbers	Output
<pre>i=1 while(i<=5): print(i*i) i=i+1</pre>	<pre>1 4 9 16 25</pre>
Print n cubes numbers	Output
<pre>i=1 while(i<=3): print(i*i*i) i=i+1</pre>	<pre>1 8 27</pre>
find sum of n numbers	Output
<pre>i=1 sum=0 while(i<=10): sum=sum+i i=i+1 print(sum)</pre>	<pre>55</pre>

factorial of n numbers/product of n numbers	Output
<pre>i=1 product=1 while(i<=10): product=product*i i=i+1 print(product)</pre>	3628800
sum of n numbers	Output
<pre>def add(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a+b print("the sum is",c) add()</pre>	enter a value 6 enter b value 4 the sum is 10
area of rectangle using function	Output
<pre>def area(): l=eval(input("enter the length of rectangle")) b=eval(input("enter the breath of rectangle")) a=l*b print("the area of rectangle is",a) area()</pre>	enter the length of rectangle 20 enter the breath of rectangle 5 the area of rectangle is 100
swap two values of variables	Output
<pre>def swap(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a a=b b=c print("a=",a,"b=",b) swap()</pre>	enter a value3 enter b value5 a= 5 b= 3

check the no divisible by 5 or not	Output
<pre>def div(): n=eval(input("enter n value")) if(n%5==0): print("the number is divisible by 5") else: print("the number not divisible by 5") div()</pre>	<pre>enter n value10 the number is divisible by 5</pre>
find remainder and quotient of given no	Output
<pre>def remainder(): a=eval(input("enter a")) b=eval(input("enter b")) R=a%b print("the remainder is",R) def quotient(): a=eval(input("enter a")) b=eval(input("enter b")) Q=a/b print("the remainder is",Q) remainder() quotient()</pre>	<pre>enter a 6 enter b 3 the remainder is 0 enter a 8 enter b 4 the remainder is 2.0</pre>
convert the temperature	Output
<pre>def ctof(): c=eval(input("enter temperature in centigrade")) f=(1.8*c)+32 print("the temperature in Fahrenheit is",f) def ftoc(): f=eval(input("enter temp in Fahrenheit")) c=(f-32)/1.8 print("the temperature in centigrade is",c) ctof() ftoc()</pre>	<pre>enter temperature in centigrade 37 the temperature in Fahrenheit is 98.6 enter temp in Fahrenheit 100 the temperature in centigrade is 37.77</pre>

program for basic calculator	Output
<pre> def add(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a+b print("the sum is",c) def sub(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a-b print("the diff is",c) def mul(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a*b print("the mul is",c) def div(): a=eval(input("enter a value")) b=eval(input("enter b value")) c=a/b print("the div is",c) add() sub() mul() div() </pre>	<pre> enter a value 10 enter b value 10 the sum is 20 enter a value 10 enter b value 10 the diff is 0 enter a value 10 enter b value 10 the mul is 100 enter a value 10 enter b value 10 the div is 1 </pre>

Part A:

1. What is mean by problem solving?
2. List down the problem solving techniques?
3. Define algorithm?
4. What are the properties of algorithm?
5. List down the equalities of good algorithm?
6. Define statements?
7. Define state?
8. What is called control flow?
9. What is called sequence execution?
10. Define iteration?
11. What is mean by flow chart?
12. List down the basic symbols for drawing flowchart?
13. List down the rules for drawing the flowchart?
14. What are the advantages of flowchart?
15. What are the disadvantages of flowchart?
16. Define pseudo code?
17. List down the keywords used in writing pseudo code?
18. Mention the advantages of using pseudo code?
19. Mention the disadvantages of using pseudo code?
20. What are the ways available to represent algorithm?
21. Differentiate flowchart and pseudo code?
22. Differentiate algorithm and pseudo code?
23. What is programming language?
24. Mention the types of programming language?
25. What is mean by machine level language?
26. What are the advantages and disadvantages of machine level language?
27. What is high level programming language and mention its advantages?
28. What are the steps in algorithmic problem solving?
29. Write the algorithm for any example?
30. Draw the flow chart for any example?
31. Write pseudo code for any example?

Part B:

1. Explain in detail about problem solving techniques?
2. Explain in detail about building blocks of algorithm?
3. Discuss the symbols and rules for drawing flowchart with the example?
4. Explain in detail about programming language?
5. Discuss briefly about algorithmic problem solving?
6. Write algorithm, pseudo code and flow chart for any example?
7. Explain in detail about simple strategies for developing algorithms?

UNIT II

DATA, EXPRESSIONS, STATEMENTS

Python interpreter and interactive mode; values and types: int, float, boolean, string, and list; variables, expressions, statements, tuple assignment, precedence of operators, comments; Modules and functions, function definition and use, flow of execution, parameters and arguments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

1. INTRODUCTION TO PYTHON:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

It was created by Guido van Rossum during 1985- 1990.

Python got its name from “Monty Python’s flying circus”. Python was released in the year 2000.

- ❖ **Python is interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- ❖ **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- ❖ **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- ❖ **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications.

1.1. Python Features:

- ❖ **Easy-to-learn:** Python is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords.
- ❖ **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- ❖ **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- ❖ **Interpreted:** Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- ❖ **Extensible:** Programmers can embed python within their C,C++,Java script ,ActiveX, etc.
- ❖ **Free and Open Source:** Anyone can freely distribute it, read the source code, and edit it.
- ❖ **High Level Language:** When writing programs, programmers concentrate on solutions of the current problem, no need to worry about the low level details.
- ❖ **Scalable:** Python provides a better structure and support for large programs than shell scripting.

1.2. Applications:

- ❖ Bit Torrent file sharing
- ❖ Google search engine, Youtube
- ❖ Intel, Cisco, HP, IBM
- ❖ i-Robot
- ❖ NASA

❖ Facebook, Drop box

1.3. Python interpreter:

Interpreter: To execute a program in a high-level language by translating it one line at a time.

Compiler: To translate a program written in a high-level language into a low-level language all at once, in preparation for later execution.

Compiler	Interpreter
Compiler Takes Entire program as input	Interpreter Takes Single instruction as input
Intermediate Object Code is Generated	No Intermediate Object Code is Generated
Conditional Control Statements are Executes faster	Conditional Control Statements are Executes slower
Memory Requirement is More (Since Object Code is Generated)	Memory Requirement is Less
Program need not be compiled every time	Every time higher level program is converted into lower level program
Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted (if any)
Example : C Compiler	Example : PYTHON

1.4 MODES OF PYTHON INTERPRETER:

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

1. Interactive mode
2. Script mode

Interactive mode:

- ❖ Interactive Mode, as the name suggests, allows us to interact with OS.
- ❖ When we type Python statement, **interpreter displays the result(s) immediately.**

Advantages:

- ❖ Python, in interactive mode, is good enough to learn, experiment or explore.
- ❖ Working in interactive mode is convenient for beginners and for testing small pieces of code.

Drawback:

- ❖ We cannot save the statements and have to retype all the statements once again to re-run them.

In interactive mode, you type Python programs and the interpreter displays the result:

```
>>> 1 + 1
2
```

The chevron, >>>, is the prompt the interpreter uses to indicate that it is ready for you to enter code. If you type 1 + 1, the interpreter replies 2.

```
>>> print ('Hello, World!')
Hello, World!
```

This is an example of a print statement. It displays a result on the screen. In this case, the result is the words.

```

Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (
AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> 2**6
64
>>> a=3
>>> b=6
>>> c=a-b
>>> print(c)
-3
>>> print("good morning")
good morning
>>> |
    
```

Script mode:

- ❖ In script mode, we type python program in a file and then use interpreter to execute the content of the file.
- ❖ Scripts can be saved to disk for future use. **Python scripts have the extension .py**, meaning that the filename ends with .py
- ❖ Save the code with **filename.py** and run the interpreter in script mode to execute the script.

Example:

```

print(1)
x = 2
print(x)
    
```

Output:

```

.....
>>>1
2
    
```

Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

Integrated Development Learning Environment (IDLE):

- ❖ Is a **graphical user interface** which is completely written in Python.
- ❖ It is bundled with the default implementation of the python language and also comes with optional part of the Python packaging.

Features of IDLE:

- ❖ Multi-window text editor with syntax highlighting.

- ❖ Auto completion with **smart indentation**.
- ❖ **Python shell** to display output with syntax highlighting.

2. VALUES AND DATA TYPES

Value:

Value can be any letter ,number or string.

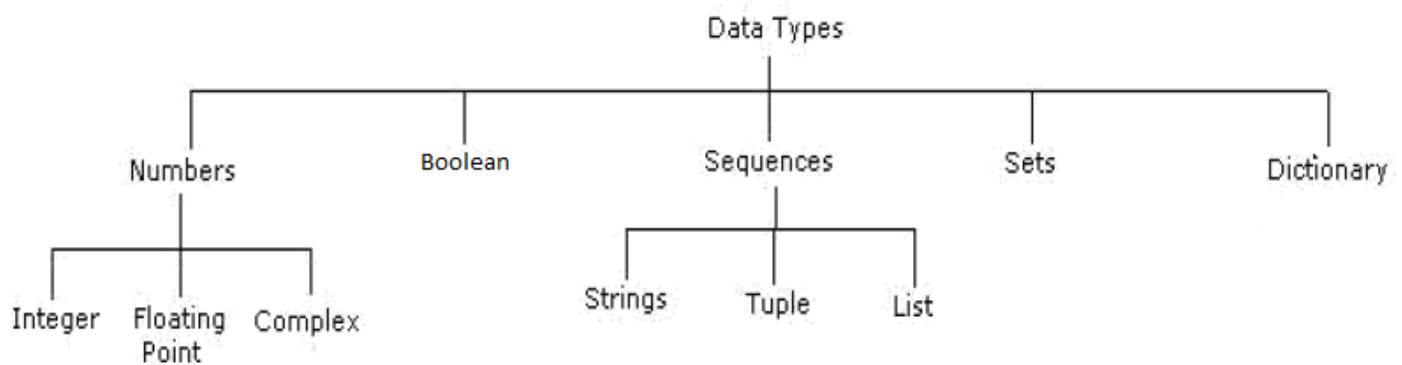
Eg, Values are 2, 42.0, and 'Hello, World!'. (These values belong to different datatypes.)

Data type:

Every value in Python has a data type.

It is a set of values, and the allowable operations on those values.

Python has four standard data types:



2.1 Numbers:

- ❖ Number data type stores **Numerical Values**.
- ❖ This data type is immutable [i.e. values/items cannot be changed].
- ❖ Python supports integers, floating point numbers and complex numbers. They are defined as,

Integers	Long	Float	Complex
- They are often called just integers or int . - They are positive or negative whole numbers with no decimal point.	-They are long integers. -They can also be represented in <u>octal</u> and hexadecimal representation.	-They are written with a decimal point dividing the integer and the fractional parts.	-They are of the form $a + bj$, where a and b are floats and <u>j</u> represents the square root of -1 (which is an imaginary number). -The real part of the number is a, and the imaginary part is b.
Eg, 56	Eg, 5692431L	Eg, 56.778	Eg, square root of -1 is a complex number

2.2 Sequence:

- ❖ A sequence is an **ordered collection of items**, indexed by positive integers.
- ❖ It is a combination **of mutable** (value can be changed) **and immutable** (values cannot be changed) data types.

❖ There are three types of sequence data type available in Python, they are

1. **Strings**
2. **Lists**
3. **Tuples**

2.2.1 Strings:

- A String in Python consists of a series or sequence of characters - letters, numbers, and special characters.
 - Strings are marked by quotes:
 - single quotes (' ') Eg, 'This a string in single quotes'
 - double quotes (" ") Eg, "This a string in double quotes"
 - triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""
 - Individual character in a string is accessed using a subscript (index).
 - Characters can be accessed using indexing and slicing operations
- Strings are immutable i.e. the contents of the string cannot be changed after it is created.

Indexing:

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

- Positive indexing helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or -ve n (where n is length of the string) displays the first element.
Example: A[0] or A[-5] will display “H”
- Subscript 1 or -ve (n-1) displays the second element.
Example: A[1] or A[-4] will display “E”

Operations on string:

- i. Indexing
- ii. Slicing
- iii. Concatenation
- iv. Repetitions
- v. Member ship

Creating a string	>>> s="good morning"	Creating the list with elements of different data types.
Indexing	>>> print(s[2]) 0 >>> print(s[6]) 0	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1)	>>> print(s[2:]) od morning	- Displaying items from 2 nd till last.

<u>Slice operator is used to extract part of a data type</u>	>>> print(s[:4]) Good	- Displaying items from 1 st position till 3 rd .
Concatenation	>>> print(s+"friends") good morningfriends	-Adding and printing the characters of two strings.
Repetition	>>> print(s*2) good morninggood morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" >>>"m" in s True >>> "a" not in s True	Using membership operators to check a particular character is in string or not. Returns true if present.

2.2.2 Lists

- ❖ List is an ordered sequence of items. Values in the list are called elements / items.
- ❖ It can be written as a list of comma-separated items (values) between **square brackets []**.
- ❖ Items in the lists can be of different data types.

Operations on list:

Indexing
Slicing
Concatenation
Repetitions
Updation, Insertion, Deletion

Creating a list	>>>list1=["python", 7.79, 101, "hello"] >>>list2=["god",6.78,9]	Creating the list with elements of different data types.
Indexing	>>>print(list1[0]) python >>> list1[2] 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1) <u>Slice operator is used to extract part of a string, or some part of a list</u> <u>Python</u>	>>> print(list1[1:3]) [7.79, 101] >>>print(list1[1:]) [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1 st position till last.
Concatenation	>>>print(list1+list2) ['python', 7.79, 101, 'hello', 'god',	-Adding and printing the items of two lists.

	6.78, 9]	
Repetition	>>> list2*3 ['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]	Creates new strings, concatenating multiple copies of the same string
Updating the list	>>> list1[2]=45 >>>print(list1) ['python', 7.79, 45, 'hello']	Updating the list using index value
Inserting an element	>>> list1.insert(2,"program") >>> print(list1) ['python', 7.79, 'program', 45, 'hello']	Inserting an element in 2 nd position
Removing an element	>>> list1.remove(45) >>> print(list1) ['python', 7.79, 'program', 'hello']	Removing an element by giving the element directly

2.2.4 Tuple:

- ❖ A tuple is same as list, except that the set of elements is **enclosed in parentheses** instead of square brackets.
- ❖ **A tuple is an immutable list.** i.e. once a tuple has been created, you can't add elements to a tuple or remove elements from the tuple.
- ❖ Benefit of Tuple:
- ❖ Tuples are faster than lists.
- ❖ If the user wants to protect the data from accidental changes, tuple can be used.
- ❖ Tuples can be used as keys in dictionaries, while lists can't.

Basic Operations:

Creating a tuple	>>>t=('python', 7.79, 101, "hello")	Creating the tuple with elements of different data types.
Indexing	>>>print(t[0]) python >>> t[2] 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1)	>>>print(t[1:3]) (7.79, 101)	❖ Displaying items from 1st till 2nd.
Concatenation	>>> t+("ram", 67) (python', 7.79, 101, 'hello', 'ram', 67)	❖ Adding tuple elements at the end of another tuple elements
Repetition	>>>print(t*2) (python', 7.79, 101, 'hello', 'python', 7.79, 101, 'hello')	❖ Creates new strings, concatenating multiple copies of the same string

Altering the tuple data type leads to error. Following error occurs when user tries to do.


```
>>> t[0]="a"
Trace back (most recent call last):
  File "<stdin>", line 1, in <module>
Type Error: 'tuple' object does not support item assignment
```

2.3 Mapping

-This data type is unordered and mutable.

-Dictionaries fall under Mappings.

2.3.1 Dictionaries:

- ❖ Lists are ordered sets of objects, whereas **dictionaries are unordered sets**.
- ❖ Dictionary is created by using **curly brackets**. i.e. {}
- ❖ Dictionaries **are accessed via keys** and not via their position.
- ❖ A dictionary is an associative array (also known as hashes). Any key of the dictionary is associated (or mapped) to a value.
- ❖ The values of a dictionary can be any Python data type. So dictionaries are **unordered key-value-pairs**(The association of a key and a value is called a key-value pair)

Dictionaries don't support the sequence operation of the sequence data types like strings, tuples and lists.

Creating a dictionary	<pre>>>> food = {"ham": "yes", "egg" : "yes", "rate": 450 } >>> print(food) {'rate': 450, 'egg': 'yes', 'ham': 'yes'}</pre>	Creating the dictionary with elements of different data types.
Indexing	<pre>>>>> print(food["rate"]) 450</pre>	Accessing the item with keys.
Slicing(ending position -1)	<pre>>>> print(t[1:3]) (7.79, 101)</pre>	Displaying items from 1st till 2nd.

If you try to access a key which doesn't exist, you will get an error message:

```
>>> words = {"house" : "Haus", "cat": "Katze"}
>>> words["car"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'car'
```

Data type	Compile time	Run time
int	a=10	a=int(input("enter a"))
float	a=10.5	a=float(input("enter a"))
string	a="panimalar"	a=input("enter a string")
list	a=[20,30,40,50]	a=list(input("enter a list"))
tuple	a=(20,30,40,50)	a=tuple(input("enter a tuple"))

3.Variables,Keywords Expressions, Statements, Comments, Docstring ,Lines And Indentation, Quotation In Python, Tuple Assignment:

3.1VARIABLES:

- ❖ A variable allows us to store a value by assigning it to a name, which can be used later.
- ❖ Named memory locations to store values.
- ❖ Programmers generally choose names for their variables that are meaningful.
- ❖ It can be of any length. No space is allowed.
- ❖ We don't need to declare a variable before using it. In Python, we simply assign a value to a variable and it will exist.

Assigning value to variable:

Value should be given on the right side of assignment operator(=) and variable on left side.

```
>>>counter =45
print(counter)
```

Assigning a single value to several variables simultaneously:

```
>>> a=b=c=100
```

Assigning multiple values to multiple variables:

```
>>> a,b,c=2,4,"ram"
```

3.2KEYWORDS:

- ❖ Keywords are the reserved words in Python.
- ❖ We cannot use a keyword as variable name, function name or any other identifier.
- ❖ They are used to define the syntax and structure of the Python language.
- ❖ Keywords are case sensitive.

<i>False</i>	<i>class</i>	<i>finally</i>	<i>is</i>	<i>return</i>
<i>None</i>	<i>continue</i>	<i>for</i>	<i>lambda</i>	<i>try</i>
<i>True</i>	<i>def</i>	<i>from</i>	<i>nonlocal</i>	<i>while</i>
<i>and</i>	<i>del</i>	<i>global</i>	<i>not</i>	<i>with</i>
<i>as</i>	<i>elif</i>	<i>if</i>	<i>or</i>	<i>yield</i>
<i>assert</i>	<i>else</i>	<i>import</i>	<i>pass</i>	
<i>break</i>	<i>except</i>	<i>in</i>	<i>raise</i>	

3.3IDENTIFIERS:

Identifier is the name given to entities like class, functions, variables etc. in Python.

- ❖ Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).

- ❖ all are valid example.
- ❖ An identifier cannot start with a digit.
- ❖ Keywords cannot be used as identifiers.
- ❖ Cannot use special symbols like !, @, #, \$, % etc. in our identifier.
- ❖ Identifier can be of any length.

Example:

Names like myClass, var_1, and **this_is_a_long_variable**

Valid declarations	Invalid declarations
Num	Number 1
Num	num 1
Num1	addition of program
_NUM	1Num
NUM_temp2	Num.no
IF	if
Else	else

3.4 STATEMENTS AND EXPRESSIONS:**3.4.1 Statements:**

- Instructions that a Python interpreter can executes are called statements.
- A statement is a unit of code like creating a variable or displaying a value.

```
>>> n = 17
>>> print(n)
```

Here, The first line is an assignment statement that gives a value to n.
The second line is a print statement that displays the value of n.

3.4.2 Expressions:

- An expression is a **combination of values, variables, and operators.**
- A value all by itself is considered an expression, and also a variable.
- So the following are all legal expressions:

```
>>> 42
42
>>> a=2
>>> a+3+2
7
>>> z=("hi"+"friend")
>>> print(z)
hifriend
```

3.5 INPUT AND OUTPUT

INPUT: Input is data entered by user (end user) in the program.

In python, **input () function** is available for input.

Syntax for input() is:
variable = input ("data")

Example:

```
>>> x=input("enter the name:")
enter the name: george
```

```
>>>y=int(input("enter the number"))
enter the number 3
```

#python accepts string as default data type. conversion is required for type.

OUTPUT: Output can be displayed to the user using Print statement .

Syntax:

```
print (expression/constant/variable)
```

Example:

```
>>> print ("Hello")
Hello
```

3.6 COMMENTS:

- ❖ A **hash sign (#)** is the beginning of a comment.
- ❖ Anything written after # in a line is ignored by interpreter.
Eg:percentage = (minute * 100) / 60 # **calculating percentage of an hour**
- ❖ Python **does not have multiple-line commenting feature.** You have to comment each line individually as follows :

Example:

```
# This is a comment.
# This is a comment, too.
# I said that already.
```

3.7 DOCSTRING:

- ❖ Docstring is short for documentation string.
- ❖ It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
- ❖ **Triple quotes** are used while writing docstrings.

Syntax:

```
functionname __doc__
```

Example:

```
def double(num):
    """Function to double the value"""
    return 2*num
>>> print(double.__doc__)
Function to double the value
```

3.8 LINES AND INDENTATION:

- ❖ Most of the programming languages like C, C++, Java use braces { } to define a block of code. But, python uses indentation.
- ❖ Blocks of code are denoted by line indentation.
- ❖ It is a space given to the block of codes for class and function definitions or flow control.

Example:

```

a=3
b=1
if a>b:
    print("a is greater")
else:
    print("b is greater")

```

3.9 QUOTATION IN PYTHON:

Python accepts single ('), double (") and triple (""" or """) quotes to denote string literals. Anything that is represented using quotations are considered as string.

- ❖ single quotes (' ') Eg, 'This a string in single quotes'
- ❖ double quotes (" ") Eg, "This a string in double quotes"
- ❖ triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences."""

3.10 TUPLE ASSIGNMENT

- ❖ An assignment to all of the elements in a tuple using a single assignment statement.
- ❖ Python has a very powerful **tuple assignment** feature that allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- ❖ The left side is a tuple of variables; the right side is a tuple of values.
- ❖ Each value is assigned to its respective variable.
- ❖ All the expressions on the right side are evaluated before any of the assignments. This feature makes tuple assignment quite versatile.
- ❖ Naturally, the number of variables on the left and the number of values on the right have to be the same.

```

>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack

```

Example:

-It is useful to swap the values of two variables. With **conventional assignment statements**, we have to use a temporary variable. For example, to swap a and b:

Swap two numbers	Output:
<pre> a=2;b=3 print(a,b) temp = a a = b b = temp print(a,b) </pre>	<pre> (2, 3) (3, 2) >>> </pre>

-Tuple assignment solves this problem neatly:

```
(a, b) = (b, a)
```

-One way to think of tuple assignment is as tuple packing/unpacking.

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

```
>>> b = ("George", 25, "20000") # tuple packing
```

-In tuple unpacking, **the values in a tuple on the right are 'unpacked' into the variables/names on the right:**

```
>>> b = ("George", 25, "20000") # tuple packing
>>> (name, age, salary) = b # tuple unpacking
>>> name
'George'
>>> age
25
>>> salary
'20000'
```

-The right side can be any kind of sequence (string,list,tuple)

Example:

-To split an email address in to user name and a domain

```
>>> mailid='god@abc.org'
>>> name, domain=mailid.split('@')
>>> print name
god
>>> print (domain)
abc.org
```

4.OPERATORS:

- ❖ Operators are the constructs which can manipulate the value of operands.
- ❖ Consider the **expression $4 + 5 = 9$** . Here, **4 and 5 are called operands** and **+ is called operator**
- ❖ Types of Operators:
 - Python language supports the following types of operators
 - Arithmetic Operators
 - Comparison (Relational) Operators
 - Assignment Operators
 - Logical Operators
 - Bitwise Operators
 - Membership Operators
 - Identity Operators

4.1 Arithmetic operators:

They are used to perform **mathematical operations** like addition, subtraction, multiplication etc. **Assume, a=10 and b=5**

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
* Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
** Exponent	Performs exponential (power) calculation on operators	$a^{**}b = 10$ to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	$5 // 2 = 2$

Examples

```
a=10
b=5
print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b=",a/b)
print("a%b=",a%b)
print("a//b=",a//b)
print("a**b=",a**b)
```

Output:

```
a+b= 15
a-b= 5
a*b= 50
a/b= 2.0
a%b= 0
a//b= 2
a**b= 100000
```

4.2 Comparison (Relational) Operators:

- Comparison operators are used to compare values.
- It either returns True or False according to the condition. **Assume, a=10 and b=5**

Operator	Description	Example
==	If the values of two operands are equal, then the condition	(a == b) is

	becomes true.	not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

a=10

b=5

print("a>b=>",a>b)

print("a>b=>",a<b)

print("a==b=>",a==b)

print("a!=b=>",a!=b)

print("a>=b=>",a<=b)

print("a>=b=>",a>=b)

Output:

a>b=> True

a>b=> False

a==b=> False

a!=b=> True

a>=b=> False

a>=b=> True

4.3 Assignment Operators:

-Assignment operators are used in Python to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a

<code>*=</code> AND	Multiply	It multiplies right operand with the left operand and assign the result to left operand	<code>c *= a</code> is equivalent to <code>c = c * a</code>
<code>/=</code> AND	Divide	It divides left operand with the right operand and assign the result to left operand	<code>c /= a</code> is equivalent to <code>c = c / a</code> <code>c /= a</code> is equivalent to <code>c = c / a</code>
<code>%=</code> AND	Modulus	It takes modulus using two operands and assign the result to left operand	<code>c %= a</code> is equivalent to <code>c = c % a</code>
<code>**=</code> AND	Exponent	Performs exponential (power) calculation on operators and assign value to the left operand	<code>c **= a</code> is equivalent to <code>c = c ** a</code>
<code>//=</code> Division	Floor	It performs floor division on operators and assign value to the left operand	<code>c //= a</code> is equivalent to <code>c = c // a</code>

Example

```

a = 21
b = 10
c = 0
c = a + b
print("Line 1 - Value of c is ", c)
c += a
print("Line 2 - Value of c is ", c)
c *= a
print("Line 3 - Value of c is ", c)
c /= a
print("Line 4 - Value of c is ", c)
c = 2
c %= a
print("Line 5 - Value of c is ", c)
c **= a
print("Line 6 - Value of c is ", c)
c //= a
print("Line 7 - Value of c is ", c)

```

Output

```

Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52.0
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864

```

4.4 Logical Operators:

-Logical operators are the and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Example

```
a = True
b = False
print('a and b is',a and b)
print('a or b is',a or b)
print('not a is',not a)
```

Output

```
x and y is False
x or y is True
not x is False
```

4.5 Bitwise Operators:

- A **bitwise operation** operates on one or more **bit** patterns at the level of individual bits

Example: Let x = 10 (0000 1010 in binary) and
y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
	Bitwise OR	x y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

Example

```
a = 60      # 60 = 0011 1100
b = 13     # 13 = 0000 1101
c = 0
c = a & b;  # 12 = 0000 1100
print "Line 1 - Value of c is ", c
c = a | b;  # 61 = 0011 1101
print "Line 2 - Value of c is ", c
c = a ^ b;  # 49 = 0011 0001
print "Line 3 - Value of c is ", c
c = ~a;    # -61 = 1100 0011
```

Output

```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

```
print "Line 4 - Value of c is ", c
c = a << 2;    # 240 = 1111 0000
print "Line 5 - Value of c is ", c
c = a >> 2;    # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

4.6 Membership Operators:

- ❖ Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- ❖ Let, **x=[5,3,6,4,1]**. To check particular item in list or not, **in and not in** operators are used.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

```
x=[5,3,6,4,1]
>>> 5 in x
True
>>> 5 not in x
False
```

4.7 Identity Operators:

- ❖ They are used to check if two values (or variables) are located on the same part of the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
x2 = 'Hello'
y2 = 'Hello'
print(x1 is not y1)
print(x2 is y2)
```

Output
False
True

5. OPERATOR PRECEDENCE:

When an expression contains **more than one operator**, the order of evaluation depends on the order of operations.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^	Bitwise exclusive 'OR' and regular 'OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //=- = += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

-For mathematical operators, Python follows mathematical convention.

-The acronym **PEMDAS** (Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction) is a useful way to remember the rules:

- ❖ Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, $2 * (3-1)$ is 4, and $(1+1)**(5-2)$ is 8.
- ❖ You can also use parentheses to make an expression easier to read, as in $(minute * 100) / 60$, even if it doesn't change the result.
- ❖ Exponentiation has the next highest precedence, so $1 + 2**3$ is 9, not 27, and $2 * 3**2$ is 18, not 36.
- ❖ Multiplication and Division have higher precedence than Addition and Subtraction. So $2*3-1$ is 5, not 4, and $6+4/2$ is 8, not 5.
- ❖ Operators with the same precedence are evaluated from left to right (except exponentiation).

Example:

$a=9-12/3+3*2-1$ $a=?$ $a=9-4+3*2-1$ $a=9-4+6-1$ $a=5+6-1$ $a=11-1$ a=10	$A=2*3+4\%5-3/2+6$ $A=6+4\%5-3/2+6$ $A=6+4-3/2+6$ $A=6+4-1+6$ $A=10-1+6$ $A=9+6$ A=15	find $m=?$ $m=-43\ 8\&0\ -2$ $m=-43\ 0\ -2$ $m=1\ -2$ m=1
$a=2,b=12,c=1$ $d=ac$ $d=2<12>1$ $d=1>1$ d=0	$a=2,b=12,c=1$ $d=ac-1$ $d=2<12>1-1$ $d=2<12>0$ $d=1>0$ d=1	$a=2*3+4\%5-3//2+6$ $a=6+4-1+6$ $a=10-1+6$ a=15

6.Functions, Function Definition And Use, Function call, Flow Of Execution, Function Prototypes, Parameters And Arguments, Return statement, Argumentstypes,Modules

6.1 FUNCTIONS:

➤ **Function is a sub program which consists of set of instructions used to perform a specific task. A large program is divided into basic building blocks called function.**

Need For Function:

- ❖ When the program is too complex and large they are divided into parts. Each part is separately coded and combined into single program. Each subprogram is called as function.
- ❖ Debugging, Testing and maintenance becomes easy when the program is divided into subprograms.
- ❖ Functions are used to avoid rewriting same code again and again in a program.
- ❖ Function provides code re-usability
- ❖ The length of the program is reduced.

Types of function:

Functions can be classified into two categories:

- i) user defined function
- ii) Built in function

i) Built in functions

- ❖ Built in functions are the functions that are already created and stored in python.
- ❖ These built in functions are always available for usage and accessed by a programmer. It cannot be modified.

Built in function	Description
-------------------	-------------

>>>max(3,4) 4	# returns largest element
>>>min(3,4) 3	# returns smallest element
>>>len("hello") 5	#returns length of an object
>>>range(2,8,1) [2, 3, 4, 5, 6, 7]	#returns range of given values
>>>round(7.8) 8.0	#returns rounded integer of the given number
>>>chr(5) \x05'	#returns a character (a string) from an integer
>>>float(5) 5.0	#returns float number from string or integer
>>>int(5.0) 5	# returns integer from string or float
>>>pow(3,5) 243	#returns power of given number
>>>type(5.6) <type 'float'>	#returns data type of object to which it belongs
>>>t=tuple([4,6.0,7]) (4, 6.0, 7)	# to create tuple of items from list
>>>print("good morning") Good morning	# displays the given object
>>>input("enter name: ") enter name : George	# reads and returns the given string

ii) User Defined Functions:

- ❖ User defined functions are the functions that programmers create for their requirement and use.
- ❖ These functions can then be **combined to form module** which can be used in other programs by importing them.
- ❖ Advantages of user defined functions:
 - Programmers working on large project can divide the workload by making different functions.
 - If repeated code occurs in a program, function can be used to include those codes and execute when needed by calling that function.

6.2 Function definition: (Sub program)

- ❖ def keyword is used to define a function.
- ❖ Give the function name after def keyword followed by parentheses in which arguments are given.
- ❖ End with colon (:)
- ❖ Inside the function add the program statements to be executed
- ❖ End with or without return statement

Syntax:

```
def fun_name(Parameter1,Parameter2...Parameter n):
    statement1
    statement2...
    statement n
    return[expression]
```

Example:

```
def my_add(a,b):
    c=a+b
    return c
```

6.3 Function Calling: (Main Function)

- Once we have defined a function, we can call it from another function, program or even the Python prompt.
- To call a function we **simply type the function name with appropriate arguments.**

Example:

```
x=5
y=4
my_add(x,y)
```

6.4 Flow of Execution:

- ❖ The order in which statements are executed is called the **flow of execution**
- ❖ Execution always begins at the first statement of the program.
- ❖ Statements are executed one at a time, in order, from top to bottom.
- ❖ Function definitions do not alter the flow of execution of the program, but remember that statements inside the function are not executed until the function is called.
- ❖ Function calls are like a bypass in the flow of execution. Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

Note: When you read a program, don't read from top to bottom. Instead, follow the flow of execution. This means that you will read the **def** statements as you are scanning from top to bottom, but you should skip the statements of the function definition until you reach a point where that function is called.

6.5 Function Prototypes:

- i. Function without arguments and without return type
- ii. Function with arguments and without return type
- iii. Function without arguments and with return type
- iv. Function with arguments and with return type

i) Function without arguments and without return type

- In this type no argument is passed through the function call and no output is return to main function
- The sub function will read the input values perform the operation and print the result in the same block

ii) Function with arguments and without return type

- Arguments are passed through the function call but output is not return to the main function

iii) Function without arguments and with return type

- In this type no argument is passed through the function call but output is return to the main function.

iv)Function with arguments and with return type

- In this type arguments are passed through the function call and output is return to the main function

Without Return Type	
Without argument	With argument
<pre>def add(): a=int(input("enter a")) b=int(input("enter b")) c=a+b print(c) add()</pre>	<pre>def add(a,b): c=a+b print(c) a=int(input("enter a")) b=int(input("enter b")) add(a,b)</pre>
<p>OUTPUT: enter a 5 enter b 10 15</p>	<p>OUTPUT: enter a 5 enter b 10 15</p>

With return type	
Without argument	With argument
<pre>def add(): a=int(input("enter a")) b=int(input("enter b")) c=a+b return c c=add() print(c)</pre>	<pre>def add(a,b): c=a+b return c a=int(input("enter a")) b=int(input("enter b")) c=add(a,b) print(c)</pre>
<p>OUTPUT: enter a 5 enter b 10 15</p>	<p>OUTPUT: enter a 5 enter b 10 15</p>

6.6 Parameters And Arguments:

Parameters:

- Parameters are the value(s) provided in the parenthesis when we write function header.
- These are the values required by function to work.
- If there is more than one value required, all of them will be listed in parameter list separated by **comma**.
- Example: `def my_add(a,b):`

Arguments :

- Arguments are the value(s) provided in function call/invoke statement.
- List of arguments should be supplied in same way as parameters are listed.
- Bounding of parameters to arguments is done 1:1, and so there should be same number and type of arguments as mentioned in parameter list.
- Example: `my_add(x,y)`

6.7 RETURN STATEMENT:

- The **return statement is used to exit a function** and go back to the place from where it was called.
- If the return statement has no arguments, then it will not return any values. But exits from function.

Syntax:

```
return[expression]
```

Example:

```
def my_add(a,b):
    c=a+b
    return c
x=5
y=4
print(my_add(x,y))
```

Output:

9

6.8 ARGUMENTS TYPES:

1. Required Arguments
2. Keyword Arguments
3. Default Arguments
4. Variable length Arguments

- ❖ **Required Arguments:** The number of arguments in the function call should match exactly with the function definition.

```
def my_details( name, age ):
    print("Name: ", name)
    print("Age ", age)
    return
my_details("george",56)
```

Output:

```
Name: george
Age 56
```

❖ **Keyword Arguments:**

Python interpreter is able to use the keywords provided to match the values with parameters even though if they are arranged in out of order.

```
def my_details( name, age ):
    print("Name: ", name)
    print("Age ", age)
    return
my_details(age=56,name="george")
```

Output:

```
Name: george
Age 56
```

❖ **Default Arguments:**

Assumes a default value if a value is not provided in the function call for that argument.

```
def my_details( name, age=40 ):
    print("Name: ", name)
    print("Age ", age)
    return
my_details(name="george")
```

Output:

```
Name: george
Age 40
```

❖ **Variable length Arguments**

If we want to specify more arguments than specified while defining the function, variable length arguments are used. It is denoted by * symbol before parameter.

```
def my_details(*name ):
    print(*name)
my_details("rajan","rahul","micheal",
ärjun")
```

Output:

```
rajan rahul micheal ärjun
```

6.9 MODULES:

- **A module is a file containing Python definitions ,functions, statements and instructions.**
- Standard library of Python is extended as modules.
- **To use these modules in a program, programmer needs to import the module.**

- Once we import a module, we can reference or use to any of its functions or variables in our code.
 - There is large number of standard modules also available in python.
 - Standard modules can be imported the same way as we import our user-defined modules.
 - Every module contains many function.
 - To access one of the function, you have to specify the name of the module and the name of the function separated by dot. This format is called dot notation.

Syntax:

```
import module_name
module_name.function_name(variable)
```

Importing Builtin Module:	Importing User Defined Module:
<pre>import math x=math.sqrt(25) print(x)</pre>	<pre>import cal x=cal.add(5,4) print(x)</pre>

There are **four ways to import a module** in our program, they are

<p><u>Import:</u> It is simplest and most common way to use modules in our code.</p> <p>Example:</p> <pre>import math x=math.pi print("The value of pi is", x)</pre> <p>Output: The value of pi is 3.141592653589793</p>	<p><u>from import :</u> It is used to get a specific function in the code instead of complete file.</p> <p>Example:</p> <pre>from math import pi x=pi print("The value of pi is", x)</pre> <p>Output: The value of pi is 3.141592653589793</p>
<p><u>import with renaming:</u></p> <p>We can import a module by renaming the module as our wish.</p> <p>Example:</p> <pre>import math as m x=m.pi print("The value of pi is", x)</pre> <p>Output: The value of pi is 3.141592653589793</p>	<p><u>import all:</u></p> <p>We can import all names(definitions) form a module using *</p> <p>Example:</p> <pre>from math import * x=pi print("The value of pi is", x)</pre> <p>Output: The value of pi is 3.141592653589793</p>

Built-in python modules are,

1.math – mathematical functions:

some of the functions in math module is,

- ✚ math.ceil(x) - Return the ceiling of x, the smallest integer greater

than or equal to x

- + `math.floor(x)` - Return the floor of x , the largest integer less than or equal to x .
- + `math.factorial(x)` -Return x factorial. `math.gcd(x,y)`- Return the greatest common divisor of the integers a and b
- + `math.sqrt(x)`- Return the square root of x
- + `math.log(x)`- return the natural logarithm of x
- + `math.log10(x)` - returns the base-10 logarithms
- + `math.log2(x)` - Return the base-2 logarithm of x .
- + `math.sin(x)` - returns sin of x radians
- + `math.cos(x)`- returns cosine of x radians
- + `math.tan(x)`-returns tangent of x radians
- + `math.pi` - The mathematical constant $\pi = 3.141592$
- + `math.e` - returns The mathematical constant $e = 2.718281$

2 .random-Generate pseudo-random numbers

- + `random.randrange(stop)`
- + `random.randrange(start, stop[, step])`
- + `random.uniform(a, b)`
- + -Return a random floating point number

ILLUSTRATIVE PROGRAMS

<u>Program for SWAPPING(Exchanging)of values</u>	<u>Output</u>
<pre>a = int(input("Enter a value ")) b = int(input("Enter b value ")) c = a a = b b = c print("a=",a,"b=",b,)</pre>	<pre>Enter a value 5 Enter b value 8 a=8 b=5</pre>
<u>Program to find distance between two points</u>	<u>Output</u>
<pre>import math x1=int(input("enter x1")) y1=int(input("enter y1")) x2=int(input("enter x2")) y2=int(input("enter y2")) distance =math.sqrt((x2-x1)**2)+((y2-y1)**2) print(distance)</pre>	<pre>enter x1 7 enter y1 6 enter x2 5 enter y2 7 2.5</pre>
<u>Program to circulate n numbers</u>	<u>Output:</u>
<pre>a=list(input("enter the list"))</pre>	<pre>enter the list '1234'</pre>

<pre>print(a) for i in range(1,len(a),1): print(a[i:]+a[:i])</pre>	<pre>['1', '2', '3', '4'] ['2', '3', '4', '1'] ['3', '4', '1', '2'] ['4', '1', '2', '3']</pre>
--	--

Part A:

1. What is interpreter?
2. What are the two modes of python?
3. List the features of python.
4. List the applications of python
5. List the difference between interactive and script mode
6. What is value in python?
7. What is identifier? and list the rules to name identifier.
8. What is keyword?
9. How to get data types in compile time and runtime?
10. What is indexing and types of indexing?
11. List out the operations on strings.
12. Explain slicing?
13. Explain below operations with the example
(i)Concatenation (ii)Repetition
14. Give the difference between list and tuple
15. Differentiate Membership and Identity operators.
16. Compose the importance of indentation in python.
17. Evaluate the expression and find the result

$$(a+b)*c/d$$

$$a+b*c/d$$
18. Write a python program to print 'n' numbers.
19. Define function and its uses
20. Give the various data types in Python
21. Assess a program to assign and access variables.
22. Select and assign how an input operation was done in python.
23. Discover the difference between logical and bitwise operator.
24. Give the reserved words in Python.
25. Give the operator precedence in python.
26. Define the scope and lifetime of a variable in python.
27. Point out the uses of default arguments in python
28. Generalize the uses of python module.
29. Demonstrate how a function calls another function. Justify your answer.
30. List the syntax for function call with and without arguments.
31. Define recursive function.
32. What are the two parts of function definition? give the syntax.
33. Point out the difference between recursive and iterative technique.
34. Give the syntax for variable length arguments.

Part B

1. Explain in detail about various data types in Python with an example?
2. Explain the different types of operators in python with an example.
3. Discuss the need and importance of function in python.
4. Explain in details about function prototypes in python.
5. Discuss about the various type of arguments in python.
6. Explain the flow of execution in user defined function with example.
7. Illustrate a program to display different data types using variables and literal constants.
8. Show how an input and output function is performed in python with an example.
9. Explain in detail about the various operators in python with suitable examples.
10. Discuss the difference between tuples and list
11. Discuss the various operation that can be performed on a tuple and Lists (minimum 5)with an example program
12. What is membership and identity operators.
13. Write a program to perform addition, subtraction, multiplication, integer division, floor division and modulo division on two integer and float.
14. Write a program to convert degree Fahrenheit to Celsius
15. Discuss the need and importance of function in python.
16. Illustrate a program to exchange the value of two variables with temporary variables
17. Briefly discuss in detail about function prototyping in python. With suitable example program
18. Analyze the difference between local and global variables.
19. Explain with an example program to circulate the values of n variables
20. Analyze with a program to find out the distance between two points using python.
21. Do the Case study and perform the following operation in tuples i) Maxima minima iii)sum of two tuples iv) duplicate a tuple v)slicing operator vi) obtaining a list from a tuple vii) Compare two tuples viii)printing two tuples of different data types
22. Write a program to find out the square root of two numbers.

UNIT III

CONTROL FLOW, FUNCTIONS

Conditionals: Boolean values and operators, conditional (if), alternative (if-else), chained conditional (if-elif-else); Iteration: state, while, for, break, continue, pass; Fruitful functions: return values, parameters, scope: local and global, composition, recursion; Strings: string slices, immutability, string functions and methods, string module; Lists as arrays. Illustrative programs: square root, gcd, exponentiation, sum the array of numbers, linear search, binary search.

BOOLEAN VALUES:

Boolean:

- ❖ Boolean data type have two values. They are 0 and 1.
- ❖ 0 represents False
- ❖ 1 represents True
- ❖ True and False are keyword.

Example:

```
>>> 3==5
False
>>> 6==6
True
>>> True+True
2
>>> False+True
1
>>> False*True
0
```

OPERATORS:

- ❖ Operators are the constructs which can manipulate the value of operands.
- ❖ Consider the expression $4 + 5 = 9$. Here, 4 and 5 are called operands and + is called operator.

Types of Operators:

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operators
4. Logical Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

Arithmetic operators:

They are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Description	Example
		a=10,b=20
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a - b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed	5//2=2

Comparison (Relational) Operators:

- ❖ Comparison operators are used to compare values.
- ❖ It either returns True or False according to the condition.

Operator	Description	Example
		a=10,b=20
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a!=b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Assignment Operators:

Assignment operators are used in Python to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a

*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Logical Operators:

Logical operators are and, or, not operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise Operators:

Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
	Bitwise OR	x y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

Membership Operators:

- ❖ Evaluates to find a value or a variable is in the specified sequence of string, list, tuple, dictionary or not.
- ❖ To check particular element is available in the list or not.
- ❖ Operators are in and not in.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

```
x=[5,3,6,4,1]
>>> 5 in x
True
>>> 5 not in x
False
```

Identity Operators:

They are used to check if two values (or variables) are located on the same part of the memory.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Example

```
x = 5
y = 5
a = 'Hello'
b = 'Hello'
print(x is not y) // False
print(a is b)//True
```

CONDITIONALS

- ❖ Conditional if
- ❖ Alternative if... else
- ❖ Chained if...elif...else
- ❖ Nested if....else

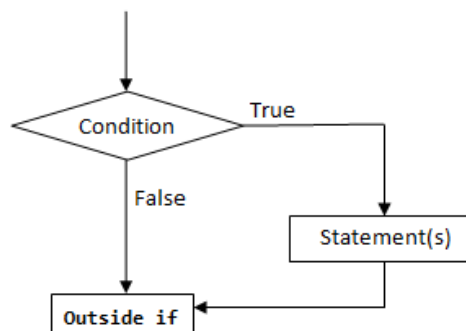
Conditional (if):

conditional (if) is used to test a condition, if the condition is true the statements inside if will be executed.

syntax:

```
if(condition 1):
    Statement 1
```

Flowchart:



Example:

1. Program to provide flat rs 500, if the purchase amount is greater than 2000.
2. Program to provide bonus mark if the category is sports.

Program to provide flat rs 500, if the purchase amount is greater than 2000.	output
<pre>purchase=eval(input("enter your purchase amount")) if(purchase>=2000): purchase=purchase-500 print("amount to pay",purchase)</pre>	<pre>enter your purchase amount 2500 amount to pay 2000</pre>
Program to provide bonus mark if the category is sports	output
<pre>m=eval(input("enter ur mark out of 100")) c=input("enter ur category G/S") if(c=="S"): m=m+5 print("mark is",m)</pre>	<pre>enter ur mark out of 100 85 enter ur category G/S S mark is 90</pre>

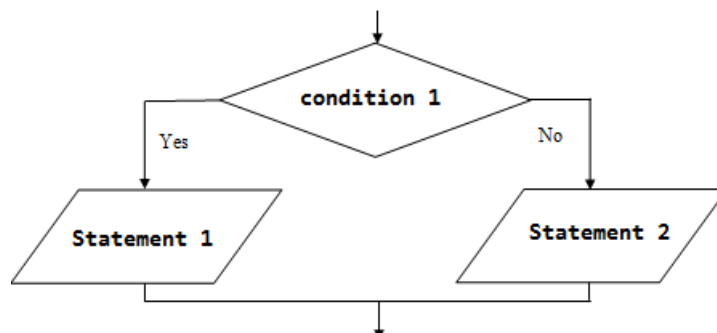
alternative (if-else)

In the alternative the condition must be true or false. In this **else** statement can be combined with **if** statement. The **else** statement contains the block of code that executes when the condition is false. If the condition is true statements inside the if get executed otherwise else part gets executed. The alternatives are called branches, because they are branches in the flow of execution.

syntax:

```
if(condition 1):
    Statement 1
else:
    Statement 2
```

Flowchart:



Examples:

1. odd or even number
2. positive or negative number
3. leap year or not

4. greatest of two numbers

5. eligibility for voting

Odd or even number	Output
<pre>n=eval(input("enter a number")) if(n%2==0): print("even number") else: print("odd number")</pre>	<pre>enter a number4 even number</pre>
positive or negative number	Output
<pre>n=eval(input("enter a number")) if(n>=0): print("positive number") else: print("negative number")</pre>	<pre>enter a number8 positive number</pre>
leap year or not	Output
<pre>y=eval(input("enter a yaer")) if(y%4==0): print("leap year") else: print("not leap year")</pre>	<pre>enter a yaer2000 leap year</pre>
greatest of two numbers	Output
<pre>a=eval(input("enter a value:")) b=eval(input("enter b value:")) if(a>b): print("greatest:",a) else: print("greatest:",b)</pre>	<pre>enter a value:4 enter b value:7 greatest: 7</pre>
eligibility for voting	Output
<pre>age=eval(input("enter ur age:")) if(age>=18): print("you are eligible for vote") else: print("you are eligible for vote")</pre>	<pre>enter ur age:78 you are eligible for vote</pre>

Chained conditionals(if-elif-else)

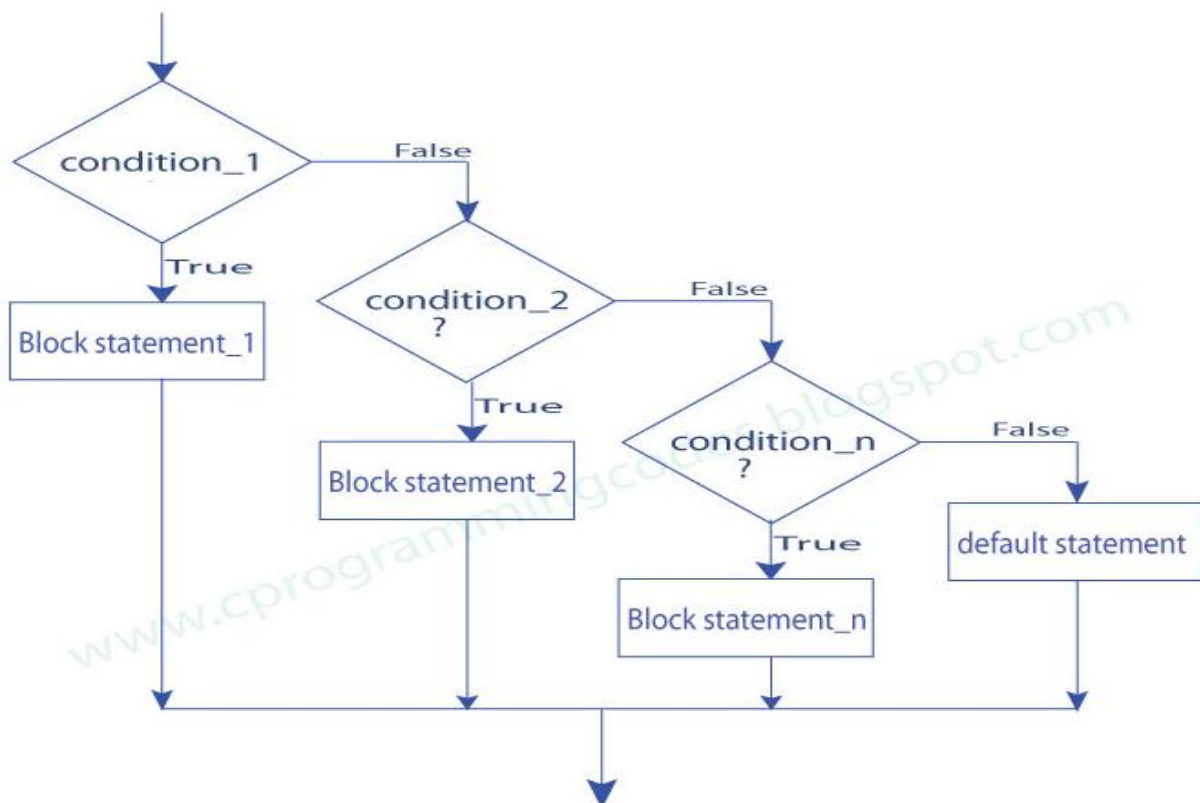
- The elif is short for else if.
- This is used to check more than one condition.
- If the condition1 is False, it checks the condition2 of the elif block. If all the conditions are False, then the else part is executed.
- Among the several if...elif...else part, only one part is executed according to the condition.

- The if block can have only one else block. But it can have multiple elif blocks.
- The way to express a computation like that is a chained conditional.

syntax:

```
if(condition 1):  
    statement 1  
elif(condition 2):  
    statement 2  
elif(condition 3):  
    statement 3  
else:  
    default statement
```

Flowchart:



Example:

1. student mark system
2. traffic light system
3. compare two numbers
4. roots of quadratic equation

student mark system	Output
<pre>mark=eval(input("enter ur mark:")) if(mark>=90): print("grade:S") elif(mark>=80): print("grade:A") elif(mark>=70): print("grade:B") elif(mark>=50): print("grade:C") else: print("fail")</pre>	<pre>enter ur mark:78 grade:B</pre>
traffic light system	Output
<pre>colour=input("enter colour of light:") if(colour=="green"): print("GO") elif(colour=="yellow"): print("GET READY") else: print("STOP")</pre>	<pre>enter colour of light:green GO</pre>
compare two numbers	Output
<pre>x=eval(input("enter x value:")) y=eval(input("enter y value:")) if(x == y): print("x and y are equal") elif(x < y): print("x is less than y") else: print("x is greater than y")</pre>	<pre>enter x value:5 enter y value:7 x is less than y</pre>
Roots of quadratic equation	output
<pre>a=eval(input("enter a value:")) b=eval(input("enter b value:")) c=eval(input("enter c value:")) d=(b*b-4*a*c) if(d==0): print("same and real roots") elif(d>0): print("diffrent real roots") else: print("imaginagry roots")</pre>	<pre>enter a value:1 enter b value:0 enter c value:0 same and real roots</pre>

Nested conditionals

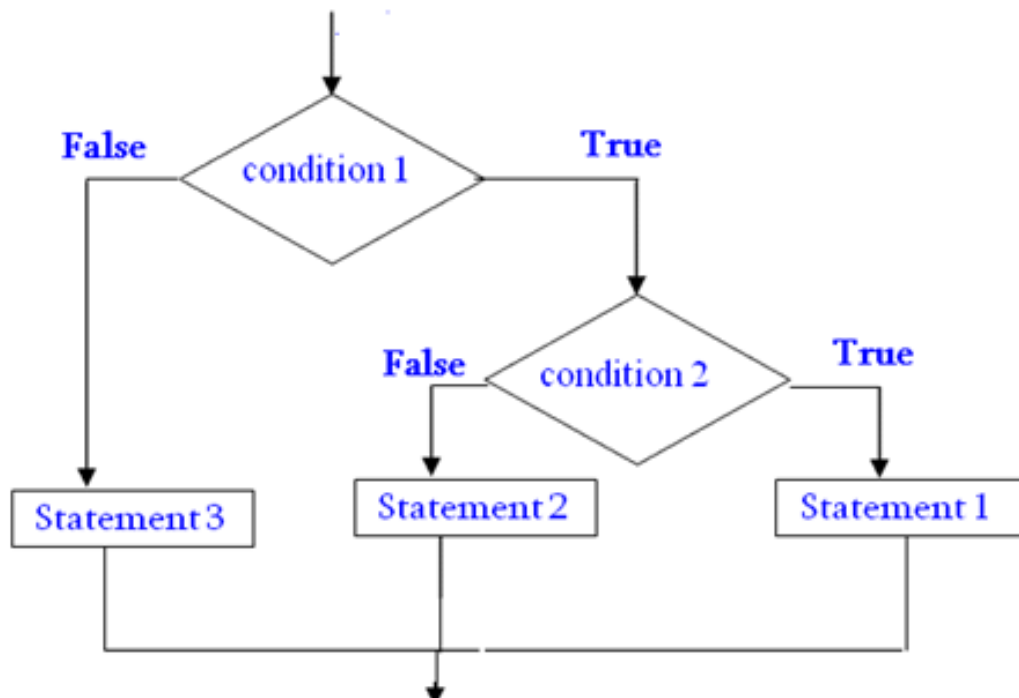
One conditional can also be nested within another. Any number of condition can be nested inside one another. In this, if the condition is true it checks another if condition1. If both the conditions are true statement1 get executed otherwise statement2 get execute. if the condition is false statement3 gets executed

Syntax:

```

if (condition):
    if(condition 1):
        statement 1
    else:
        statement 2
else:
    statement 3
    
```

Flowchart:



Example:

1. greatest of three numbers
2. positive negative or zero

greatest of three numbers	output
<pre> a=eval(input("enter the value of a")) b=eval(input("enter the value of b")) c=eval(input("enter the value of c")) if(a>b): if(a>c): print("the greatest no is",a) else: print("the greatest no is",c) </pre>	<pre> enter the value of a 9 enter the value of a 1 enter the value of a 8 the greatest no is 9 </pre>

```

else:
    if(b>c):
        print("the greatest no is",b)
    else:
        print("the greatest no is",c)
    
```

positive negative or zero	output
---------------------------	--------

<pre> n=eval(input("enter the value of n:")) if(n==0): print("the number is zero") else: if(n>0): print("the number is positive") else: print("the number is negative") </pre>	<pre> enter the value of n:-9 the number is negative </pre>
---	---

ITERATION/CONTROL STATEMENTS:

- | |
|--|
| <ul style="list-style-type: none"> ❖ state ❖ while ❖ for ❖ break ❖ continue ❖ pass |
|--|

State:

Transition from one process to another process under specified condition with in a time is called state.

While loop:

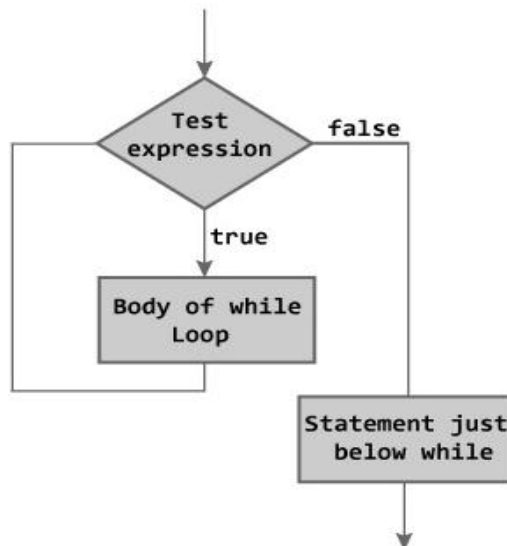
- While loop statement in Python is used to repeatedly executes set of statement as long as a given condition is true.
- In while loop, test expression is checked first. The body of the loop is entered only if the test_expression is True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.
- In Python, the body of the while loop is determined through indentation.
- The statements inside the while starts with indentation and the first unindented line marks the end.

Syntax:

```

inital value
while(condition):
    body of while loop
increment
    
```


Flowchart:



Examples:

1. program to find sum of n numbers:
2. program to find factorial of a number
3. program to find sum of digits of a number:
4. Program to Reverse the given number:
5. Program to find number is Armstrong number or not
6. Program to check the number is palindrome or not

Sum of n numbers:	output
<pre>n=eval(input("enter n")) i=1 sum=0 while(i<=n): sum=sum+i i=i+1 print(sum)</pre>	<pre>enter n 10 55</pre>
Factorial of a numbers:	output
<pre>n=eval(input("enter n")) i=1 fact=1 while(i<=n): fact=fact*i i=i+1 print(fact)</pre>	<pre>enter n 5 120</pre>
Sum of digits of a number:	output
<pre>n=eval(input("enter a number")) sum=0 while(n>0): a=n%10</pre>	<pre>enter a number 123 6</pre>

<pre> sum=sum+a n=n//10 print(sum) </pre>	
Reverse the given number:	output
<pre> n=eval(input("enter a number")) sum=0 while(n>0): a=n%10 sum=sum*10+a n=n//10 print(sum) </pre>	<pre> enter a number 123 321 </pre>
Armstrong number or not	output
<pre> n=eval(input("enter a number")) org=n sum=0 while(n>0): a=n%10 sum=sum+a*a*a n=n//10 if(sum==org): print("The given number is Armstrong number") else: print("The given number is not Armstrong number") </pre>	<pre> enter a number153 The given number is Armstrong number </pre>
Palindrome or not	output
<pre> n=eval(input("enter a number")) org=n sum=0 while(n>0): a=n%10 sum=sum*10+a n=n//10 if(sum==org): print("The given no is palindrome") else: print("The given no is not palindrome") </pre>	<pre> enter a number121 The given no is palindrome </pre>

For loop:

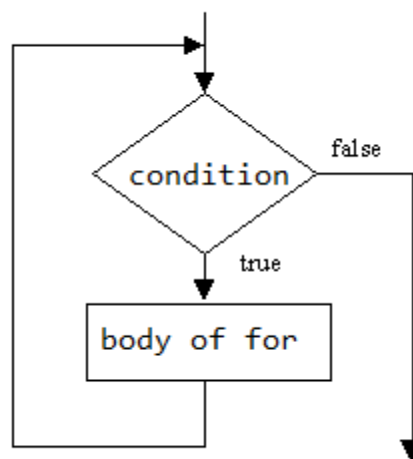
❖ ***for in range:***

- ❖ We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
- ❖ In range function have to define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

syntax

```
for i in range(start,stop,steps):  
    body of for loop
```

Flowchart:



For in sequence

- ❖ The for loop in Python is used to iterate over a sequence (list, tuple, string). Iterating over a sequence is called traversal. Loop continues until we reach the last element in the sequence.
- ❖ The body of for loop is separated from the rest of the code using indentation.

```
for i in sequence:  
    print(i)
```

Sequence can be a list, strings or tuples

s.no	sequences	example	output
1.	For loop in string	for i in "Ramu": print(i)	R A M U

2.	For loop in list	for i in [2,3,5,6,9]: print(i)	2 3 5 6 9
3.	For loop in tuple	for i in (2,3,1): print(i)	2 3 1

Examples:

1. print nos divisible by 5 not by 10:
2. Program to print fibonacci series.
3. Program to find factors of a given number
4. check the given number is perfect number or not
5. check the no is prime or not
6. Print first n prime numbers
7. Program to print prime numbers in range

print nos divisible by 5 not by 10	output
n=eval(input("enter a")) for i in range(1,n,1): if(i%5==0 and i%10!=0): print(i)	enter a:30 5 15 25
Fibonacci series	output
a=0 b=1 n=eval(input("Enter the number of terms: ")) print("Fibonacci Series: ") print(a,b) for i in range(1,n,1): c=a+b print(c) a=b b=c	Enter the number of terms: 6 Fibonacci Series: 0 1 1 2 3 5 8
find factors of a number	Output
n=eval(input("enter a number:")) for i in range(1,n+1,1): if(n%i==0): print(i)	enter a number:10 1 2 5 10

check the no is prime or not	output
<pre>n=eval(input("enter a number")) for i in range(2,n): if(n%i==0): print("The num is not a prime") break else: print("The num is a prime number.")</pre>	<pre>enter a no:7 The num is a prime number.</pre>
check a number is perfect number or not	Output
<pre>n=eval(input("enter a number:")) sum=0 for i in range(1,n,1): if(n%i==0): sum=sum+i if(sum==n): print("the number is perfect number") else: print("the number is not perfect number")</pre>	<pre>enter a number:6 the number is perfect number</pre>
Program to print first n prime numbers	Output
<pre>number=int(input("enter no of prime numbers to be displayed:")) count=1 n=2 while(count<=number): for i in range(2,n): if(n%i==0): break else: print(n) count=count+1 n=n+1</pre>	<pre>enter no of prime numbers to be displayed:5 2 3 5 7 11</pre>
Program to print prime numbers in range	output:
<pre>lower=eval(input("enter a lower range")) upper=eval(input("enter a upper range")) for n in range(lower,upper + 1): if n > 1: for i in range(2,n): if (n % i) == 0: break else: print(n)</pre>	<pre>enter a lower range50 enter a upper range100 53 59 61 67 71 73 79 83 89 97</pre>

Loop Control Structures

BREAK

- ❖ Break statements can alter the flow of a loop.
- ❖ It terminates the current loop and executes the remaining statement outside the loop.
- ❖ If the loop has else statement, that will also gets terminated and come out of the loop completely.

Syntax:

break

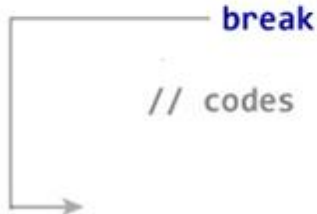
```
while (test Expression):
```

```
    // codes
```

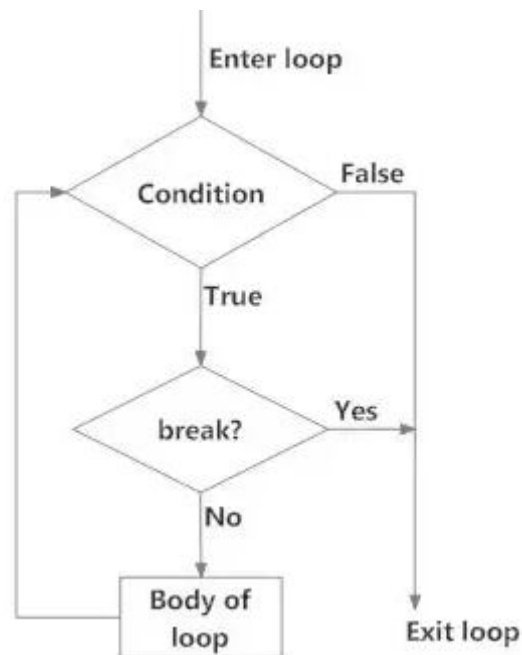
```
    if (condition for break):
```

```
        break
```

```
    // codes
```



Flowchart



example	Output
<pre>for i in "welcome": if(i=="c"): break print(i)</pre>	<pre>w e l</pre>

CONTINUE

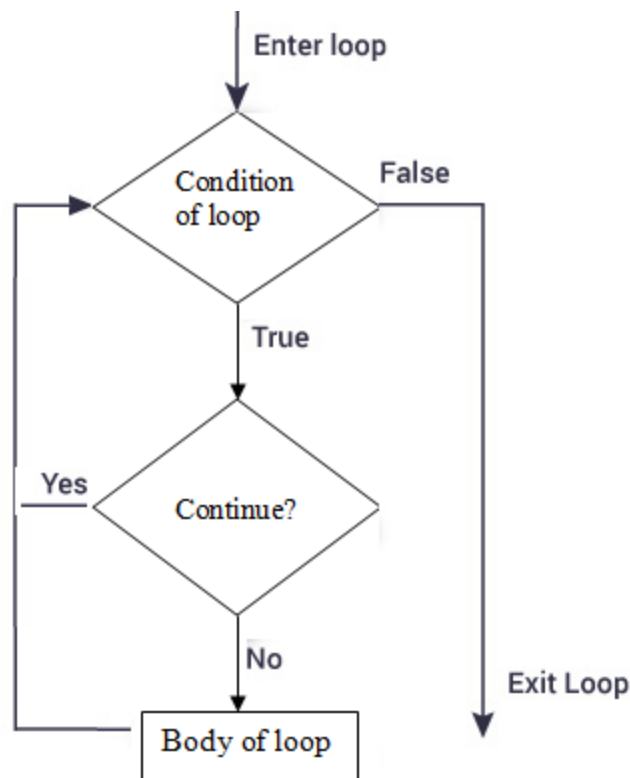
It terminates the current iteration and transfer the control to the next iteration in the loop.

Syntax: Continue

```

→ while (test Expression):
    // codes
    if (condition for continue):
        continue
    // codes

```

Flowchart

Example:	Output
<pre> for i in "welcome": if(i=="c"): continue print(i) </pre>	<pre> w e l o m e </pre>

PASS

- ❖ It is used when a statement is required syntactically but you don't want any code to execute.
- ❖ It is a null statement, nothing happens when it is executed.

Syntax:

pass

break

Example	Output
<pre>for i in "welcome": if (i == "c"): pass print(i)</pre>	<pre>w e l c o m e</pre>

Difference between break and continue

<u>break</u>	<u>continue</u>
It terminates the current loop and executes the remaining statement outside the loop.	It terminates the current iteration and transfer the control to the next iteration in the loop.
syntax: break	syntax: continue
<pre>for i in "welcome": if(i=="c"): break print(i)</pre>	<pre>for i in "welcome": if(i=="c"): continue print(i)</pre>
<pre>w e l</pre>	<pre>w e l o m e</pre>

else statement in loops:

else in for loop:

- ❖ If else statement is used in for loop, the else statement is executed when the loop has reached the limit.
- ❖ The statements inside for loop and statements inside else will also execute.

example	output
<pre>for i in range(1,6): print(i) else: print("the number greater than 6")</pre>	<pre>1 2 3 4 5 the number greater than 6</pre>

else in while loop:

- ❖ If else statement is used within while loop , the else part will be executed when the condition become false.
- ❖ The statements inside for loop and statements inside else will also execute.

Program	output
<pre>i=1 while(i<=5): print(i) i=i+1 else: print("the number greater than 5")</pre>	<pre>1 2 3 4 5 the number greater than 5</pre>

Fruitful Function

- ❖ Fruitful function
- ❖ Void function
- ❖ Return values
- ❖ Parameters
- ❖ Local and global scope
- ❖ Function composition
- ❖ Recursion

Fruitful function:

A function that returns a value is called fruitful function.

Example:

```
Root=sqrt(25)
```

Example:

```
def add():
    a=10
    b=20
    c=a+b
    return c

c=add()
print(c)
```

Void Function

A function that perform action but don't return any value.

Example:

```
print("Hello")
```

Example:

```
def add():
    a=10
    b=20
```

```
c=a+b
print(c)
add()
```

Return values:

return keywords are used to return the values from the function.

example:

```
return a – return 1 variable
return a,b– return 2 variables
return a,b,c– return 3 variables
return a+b– return expression
return 8– return value
```

PARAMETERS / ARGUMENTS:

- ❖ Parameters are the variables which used in the function definition. Parameters are inputs to functions. Parameter receives the input from the function call.
- ❖ It is possible to define more than one parameter in the function definition.

Types of parameters/Arguments:

1. Required/Positional parameters
2. Keyword parameters
3. Default parameters
4. Variable length parameters

Required/ Positional Parameter:

The number of parameter in the function definition should match exactly with number of arguments in the function call.

Example	Output:
<pre>def student(name, roll): print(name,roll) student("George",98)</pre>	George 98

Keyword parameter:

When we call a function with some values, these values get assigned to the parameter according to their position. When we call functions in keyword parameter, the order of the arguments can be changed.

Example	Output:
<pre>def student(name,roll,mark): print(name,roll,mark) student(90,102,"bala")</pre>	90 102 bala

Default parameter:

Python allows function parameter to have default values; if the function is called without the argument, the argument gets its default value in function definition.

Example	Output:
<pre>def student(name, age=17): print (name, age) student("kumar"): student("ajay"):</pre>	<pre>Kumar 17 Ajay 17</pre>

Variable length parameter

- ❖ Sometimes, we do not know in advance the number of arguments that will be passed into a function.
- ❖ Python allows us to handle this kind of situation through function calls with number of arguments.
- ❖ In the function definition we use an asterisk (*) before the parameter name to denote this is variable length of parameter.

Example	Output:
<pre>def student(name,*mark): print(name,mark) student ("bala",102,90)</pre>	<pre>bala (102 ,90)</pre>

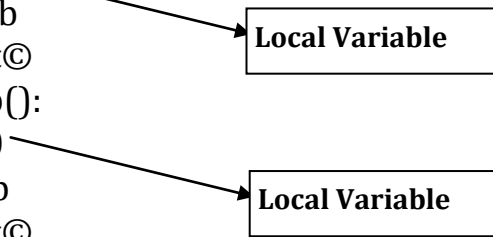
Local and Global Scope

Global Scope

- ❖ The *scope* of a variable refers to the places that you can see or access a variable.
- ❖ A variable with global scope can be used anywhere in the program.
- ❖ It can be created by defining a variable outside the function.

Example	output
<pre>a=50 def add(): b=20 c=a+b print© def sub(): b=30 c=a-b print© print(a)</pre>	<pre>70 20 50</pre>

Local Scope A variable with local scope can be used only within the function .

Example	output
<pre>def add(): b=20 c=a+b print(c) def sub(): b=30 c=a-b print(c) print(a) print(b)</pre> 	<p>70</p> <p>20</p> <p>error</p> <p>error</p>

Function Composition:

- ❖ Function Composition is the ability to call one function from within another function
- ❖ It is a way of combining functions such that the result of each function is passed as the argument of the next function.
- ❖ In other words the output of one function is given as the input of another function is known as function composition.

Example:	Output:
<pre>math.sqrt(math.log(10))</pre>	
<pre>def add(a,b): c=a+b return c def mul(c,d): e=c*d return e c=add(10,20) e=mul(c,30) print(e)</pre>	<p>900</p>
find sum and average using function composition	output
<pre>def sum(a,b): sum=a+b return sum def avg(sum): avg=sum/2 return avg a=eval(input("enter a:")) b=eval(input("enter b:")) sum=sum(a,b) avg=avg(sum)</pre>	<p>enter a:4</p> <p>enter b:8</p> <p>the avg is 6.0</p>

```
print("the avg is",avg)
```

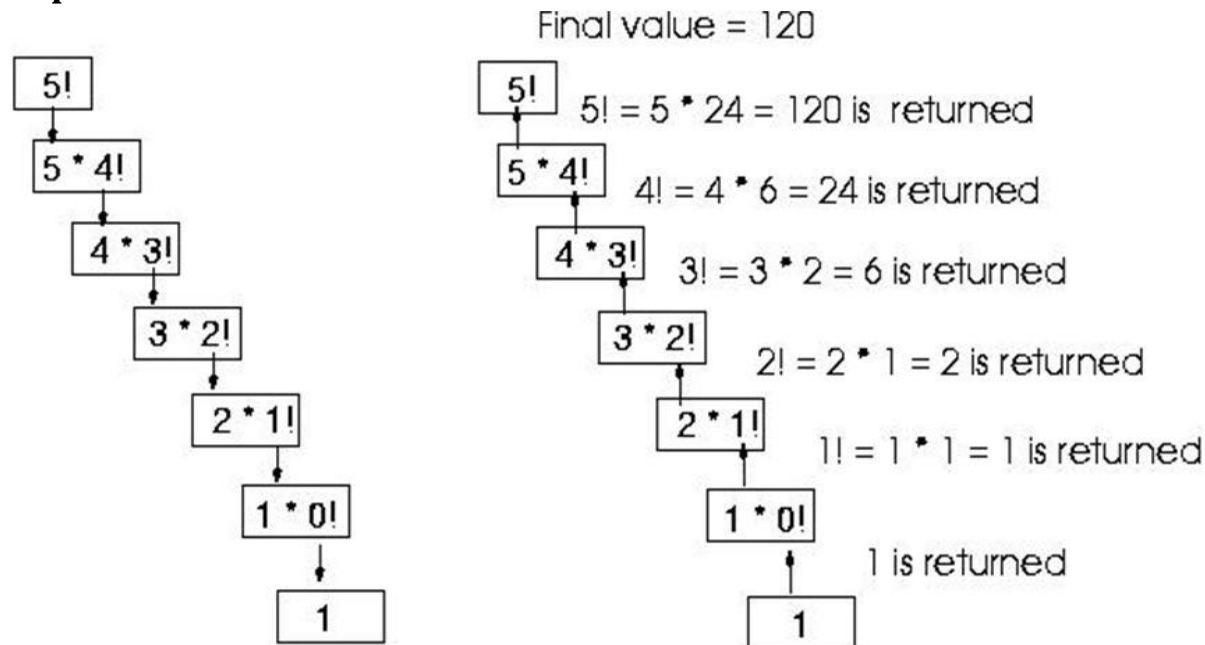
Recursion

A function calling itself till it reaches the base value - stop point of function call.

Example: factorial of a given number using recursion

Factorial of n	Output
<pre>def fact(n): if(n==1): return 1 else: return n*fact(n-1) n=eval(input("enter no. to find fact:")) fact=fact(n) print("Fact is",fact)</pre>	<pre>enter no. to find fact:5 Fact is 120</pre>

Explanation



Examples:

1. sum of n numbers using recursion
2. exponential of a number using recursion

Sum of n numbers	Output
<pre>def sum(n): if(n==1): return 1 else: return n*sum(n-1) n=eval(input("enter no. to find sum:")) sum=sum(n) print("Fact is",sum)</pre>	<pre>enter no. to find sum:10 Fact is 55</pre>

Strings:

- ❖ Strings
- ❖ String slices
- ❖ Immutability
- ❖ String functions and methods
- ❖ String module

Strings:

- ❖ String is defined as sequence of characters represented in quotation marks (either single quotes (') or double quotes (").
 - ❖ An individual character in a string is accessed using a index.
 - ❖ The index should always be an integer (positive or negative).
 - ❖ A index starts from 0 to n-1.
 - ❖ Strings are immutable i.e. the contents of the string cannot be changed after it is created.
 - ❖ Python will get the input at run time by default as a string.
 - ❖ Python does not support character data type. A string of size 1 can be treated as characters.
1. single quotes (' ')
 2. double quotes (" ")
 3. triple quotes("" "" "")

Operations on string:

1. Indexing
2. Slicing
3. Concatenation
4. Repetitions
5. Member ship

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

indexing

```
>>>a="HELLO"
>>>print(a[0])
>>>H
>>>print(a[-1])
>>>O
```

- ❖ Positive indexing helps in accessing the string from the beginning
- ❖ Negative subscript helps in accessing the string from the end.

Slicing:	Print[0:4] – HELL Print[:3] – HEL Print[0:]- HELLO	The Slice[start : stop] operator extracts sub string from the strings. A segment of a string is called a slice.
Concatenation	a="save" b="earth" >>>print(a+b) saveearth	The + operator joins the text on both sides of the operator.
Repetitions:	a="panimalar " >>>print(3*a) panimalarpanimalar panimalar	The * operator repeats the string on the left hand side times the value on right hand side.
Membership:	>>> s="good morning" >>>"m" in s True >>> "a" not in s True	Using membership operators to check a particular character is in string or not. Returns true if present

String slices:

- ❖ A part of a string is called string slices.
- ❖ **The process of extracting a sub string from a string is called slicing.**

Slicing: a="HELLO"	Print[0:4] – HELL Print[:3] – HEL Print[0:]- HELLO	The Slice[n : m] operator extracts sub string from the strings. A segment of a string is called a slice.
-------------------------------------	--	---

Immutability:

- ❖ Python strings are “immutable” as they cannot be changed after they are created.
- ❖ Therefore [] operator cannot be used on the left side of an assignment.

operations	Example	output
element assignment	a="PYTHON" a[0]='x'	TypeError: 'str' object does not support element assignment
element deletion	a="PYTHON" del a[0]	TypeError: 'str' object doesn't support element deletion
delete a string	a="PYTHON" del a	NameError: name 'my_string' is not defined

```
print(a)
```

string built in functions and methods:

A **method** is a function that “belongs to” an object.

Syntax to access the method

Stringname.method()

a="happy birthday"

here, a is the string name.

	syntax	example	description
1	a.capitalize()	>>> a.capitalize() ' Happy birthday'	capitalize only the first letter in a string
2	a.upper()	>>> a.upper() 'HAPPY BIRTHDAY'	change string to upper case
3	a.lower()	>>> a.lower() ' happy birthday'	change string to lower case
4	a.title()	>>> a.title() ' Happy Birthday '	change string to title case i.e. first characters of all the words are capitalized.
5	a.swapcase()	>>> a.swapcase() 'HAPPY BIRTHDAY'	change lowercase characters to uppercase and vice versa
6	a.split()	>>> a.split() ['happy', 'birthday']	returns a list of words separated by space
7	a.center(width,"fillchar")	>>>a.center(19,"*") '***happy birthday***'	pads the string with the specified “fillchar” till the length is equal to “width”
8	a.count(substring)	>>> a.count('happy') 1	returns the number of occurrences of substring
9	a.replace(old,new)	>>>a.replace('happy', 'wishyou happy') 'wishyou happy birthday'	replace all old substrings with new substrings
10	a.join(b)	>>> b="happy" >>> a="-" >>> a.join(b) 'h-a-p-p-y'	returns a string concatenated with the elements of an iterable. (Here “a” is the iterable)
11	a.isupper()	>>> a.isupper() False	checks whether all the case-based characters (letters) of the string are uppercase.
12	a.islower()	>>> a.islower() True	checks whether all the case-based characters (letters) of the string are lowercase.
13	a.isalpha()	>>> a.isalpha() False	checks whether the string consists of alphabetic characters only.

14	a.isalnum()	>>> a.isalnum() False	checks whether the string consists of alphanumeric characters.
15	a.isdigit()	>>> a.isdigit() False	checks whether the string consists of digits only.
16	a.isspace()	>>> a.isspace() False	checks whether the string consists of whitespace only.
17	a.istitle()	>>> a.istitle() False	checks whether string is title cased.
18	a.startswith(substring)	>>> a.startswith("h") True	checks whether string starts with substring
19	a.endswith(substring)	>>> a.endswith("y") True	checks whether the string ends with the substring
20	a.find(substring)	>>> a.find("happy") 0	returns index of substring, if it is found. Otherwise -1 is returned.
21	len(a)	>>>len(a) >>>14	Return the length of the string
22	min(a)	>>>min(a) >>>' '	Return the minimum character in the string
23	max(a)	max(a) >>>'y'	Return the maximum character in the string

String modules:

- ❖ A module is a file containing Python definitions, functions, statements.
- ❖ Standard library of Python is extended as modules.
- ❖ To use these modules in a program, programmer needs to import the module.
- ❖ Once we import a module, we can reference or use to any of its functions or variables in our code.
- ❖ There is large number of standard modules also available in python.
- ❖ Standard modules can be imported the same way as we import our user-defined modules.

Syntax:

import module_name

Example	output
import string print(string.punctuation) print(string.digits) print(string.printable) print(string.capwords("happy birthday")) print(string.hexdigits) print(string.octdigits)	!"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~ 0123456789 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJ KLMNOPQRSTUVWXYZ!"#\$%&'()*+,- ./:;<=>?@[\\]^_`{ }~ Happy Birthday 0123456789abcdefABCDEF 01234567

Escape sequences in string

Escape Sequence	Description	example
\n	new line	>>> print("hai \nhello") hai hello
\\	prints Backslash (\)	>>> print("hai\\hello") hai\hello
\'	prints Single quote (')	>>> print('') '
\"	prints Double quote (")	>>>print("\") "
\t	prints tab sapace	>>>print("hai\thello") hai hello
\a	ASCII Bell (BEL)	>>>print("\a")

List as array:**Array:**

Array is a collection of similar elements. Elements in the array can be accessed by index. Index starts with 0. Array can be handled in python by module named array.

To create array have to import array module in the program.

Syntax :

```
import array
```

Syntax to create array:

```
Array_name = module_name.function_name('datatype',[elements])
```

example:

```
a=array.array('i',[1,2,3,4])
```

a- array name

array- module name

i- integer datatype

Example

Program to find sum of array elements	Output
<pre>import array sum=0 a=array.array('i',[1,2,3,4]) for i in a: sum=sum+i print(sum)</pre>	10

Convert list into array:

fromlist() function is used to append list to array. Here the list is act like a array.

Syntax:

arrayname.fromlist(list_name)

Example

program to convert list into array	Output
<pre>import array sum=0 l=[6,7,8,9,5] a=array.array('i',[]) a.fromlist(l) for i in a: sum=sum+i print(sum)</pre>	35

Methods in array

a=[2,3,4,5]

	Syntax	example	Description
1	array(data type, value list)	array('i',[2,3,4,5])	This function is used to create an array with data type and value list specified in its arguments.
2	append()	>>>a.append(6) [2,3,4,5,6]	This method is used to add the at the end of the array.
3	insert(index,element)	>>>a.insert(2,10) [2,3,10,5,6]	This method is used to add the value at the position specified in its argument.
4	pop(index)	>>>a.pop(1) [2,10,5,6]	This function removes the element at the position mentioned in its argument, and returns it.
5	index(element)	>>>a.index(2) 0	This function returns the index of value
6	reverse()	>>>a.reverse() [6,5,10,2]	This function reverses the array.
7	count()	a.count()	This is used to count number of

	4	elements in an array
--	---	----------------------

ILLUSTRATIVE PROGRAMS:

Square root using newtons method:	Output:
<pre>def newtonsqrt(n): root=n/2 for i in range(10): root=(root+n/root)/2 print(root) n=eval(input("enter number to find Sqrt: ")) newtonsqrt(n)</pre>	<pre>enter number to find Sqrt: 9 3.0</pre>
GCD of two numbers	output
<pre>n1=int(input("Enter a number1:")) n2=int(input("Enter a number2:")) for i in range(1,n1+1): if(n1%i==0 and n2%i==0): gcd=i print(gcd)</pre>	<pre>Enter a number1:8 Enter a number2:24 8</pre>
Exponent of number	Output:
<pre>def power(base,exp): if(exp==1): return(base) else: return(base*power(base,exp-1)) base=int(input("Enter base: ")) exp=int(input("Enter exponential value:")) result=power(base,exp) print("Result:",result)</pre>	<pre>Enter base: 2 Enter exponential value:3 Result: 8</pre>
sum of array elements:	output:
<pre>a=[2,3,4,5,6,7,8] sum=0 for i in a: sum=sum+i print("the sum is",sum)</pre>	<pre>the sum is 35</pre>
Linear search	output
<pre>a=[20,30,40,50,60,70,89] print(a) search=eval(input("enter a element to search:")) for i in range(0,len(a),1): if(search==a[i]): print("element found at",i+1) break else: print("not found")</pre>	<pre>[20, 30, 40, 50, 60, 70, 89] enter a element to search:30 element found at 2</pre>

Binary search	output
<pre> a=[20, 30, 40, 50, 60, 70, 89] print(a) search=eval(input("enter a element to search:")) start=0 stop=len(a)-1 while(start<=stop): mid=(start+stop)//2 if(search==a[mid]): print("elemrnt found at",mid+1) break elif(search<a[mid]): stop=mid-1 else: start=mid+1 else: print("not found") </pre>	<pre> [20, 30, 40, 50, 60, 70, 89] enter a element to search:30 element found at 2 </pre>

Part A:

1. What are Boolean values?
2. Define operator and operand?
3. Write the syntax for if with example?
4. Write the syntax and flowchart for if else.
5. Write the syntax and flowchart for chained if.
6. define state
7. Write the syntax for while loop with flowchart.
8. Write the syntax for for loopwith flowchart.
9. Differentiate break and continue.
10. mention the use of pass
11. what is fruitful function
12. what is void function
13. mention the different ways of writing return statement
14. What is parameter and list down its type?
15. What is local and global scope?
16. Differentiate local and global variable?
17. What is function composition, give an example?
18. Define recursion.
19. Differentiate iteration and recursion.
20. Define string. How to get a string at run time.

21. What is slicing? Give an example.
22. What is immutability of string?
23. List out some string built in function with example?
24. Define string module?
25. How can list act as array?
26. write a program to check the number is odd or even.
27. write a program to check the number positive or negative
28. write a program to check the year is leap year or not
29. write a program to find greatest of two numbers
30. write a program for checking eligibility for vote
31. write a program to find sum of n numbers
32. write a program to find factorial of given numbers
33. write a program to find sum of digits of a number
34. Write a program to reverse the given number.
35. Write a program to check the given number is palindrome or not.
36. write a program to check the given number is Armstrong or not
37. how can you use for loop in sequence.
38. how can you use else statement if loops.
- 39. What is the use of map() function?**

Part B:

1. Explain conditional statements in detail with example(if, if..else, if..elif..else)
2. explain in detail about operators in detail
3. Explain in detail about iterations with example.(for, while)
4. Explain the usage of else statements in loops
5. Explain in detail about using for loop in sequence.
6. Explain in detail about string built in function with suitable examples?
7. Explain about loop control statement(break, continue, pass)
8. Briefly discuss about fruitful function.
9. Discuss with an example about local and global variable
10. Discuss with an example about function composition
11. Explain in detail about recursion with example.
12. Explain in detail about strings and its operations(slicing,immutability)
13. Program to find square root of a given number using newtons method
14. program to find gcd of given nnumber
15. program to find exponentiation of given number using recursion
16. program to find sum of array elements.
17. program to search an element using linear search.
18. program to search an element using binary element.
19. program to find factorial of a given number using recursion