UNIT-IV

LIST,TUPLES,DICTIONARIES

TWO MARKS:

1. Define List with example?

- A List is an **ordered set of values**, where each value is identified by an index.
- The Values that make up a list are called its **elements or items**.

Example 1:[10, 20, 30, 40] *# list of four integers*
Example 2:['frog', 'Dog', 'Cow'] *# list of three strings*

2. What are the list operations?

- Lists respond to the + and * operators much like strings; they mean **concatenation and repetition** here too, except that the result is a new list, not a string.
- **The + operator concatenates lists**:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

- **Similarly, the * operator repeats a list a given number of times:**

```
>>> [0] * 4
[0, 0, 0, 0]# It repeats [0] four times
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3] # It repeats the list [1, 2, 3] three times.
```

3. Describe list slicing with examples.

- ➢ A **sub-sequence of a sequence** is called a slice and the operation that performs on subsequence is called slicing.

The slice operator also works on lists:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
Output : ['b', 'c']

>>> t[:4]
Output:['a', 'b', 'c', 'd']

>>> t[3:]
Output:['d', 'e', 'f']

>>> t[:]
Output : ['a', 'b', 'c', 'd', 'e', 'f']
```

4. What is Cloning of List?

- ➢ Cloning is **the process of modifying a list** and also keeps a copy of the original. The entire list is copied not just the reference.
- ➢ This process is sometimes called cloning, to **avoid the ambiguity of the copy**.
- ➢ The easiest way to clone a list is to use the slice operator.

Example:

```
>>>a=[1,2,3]
>>>b=a[:]
>>>print b
[1,2,3].
```

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

- **Index can have negative value, it counts backward from the list.**

Example:

```
>>> Fruit  = ['Apple', 'Grapes', 'Orange']
>>>Fruit(0) =  'pear'
>>>Fruit(-1) =  'Jackfruit'
>>>print (fruit)
['pear', 'grape', 'Jackfruit']
```

6. **What is Aliasing?**
   - More than one list variable can point to the same data. This is called an alias.
   - If 'A' refers to an object and you assign ' B = A', then both variables refer to the same object:

```
>>> A = [1, 2, 3]
>>> B= A
>>> B is A
True
```

7. **What is the purpose of list methods? Types of List method?**
   - Python has a number of built-in data structures, including lists.
   - Data structures provides us with a way to organize and store data.
   - We can use built-in methods to retrieve or manipulate that data.

1.List.append( )	5.List.insert ( )	9.List.reverse( )
2.List.remove( )	6.List.pop ( )	10.List.clear ( )
3.List.copy ( )	7.List.sort ( )	11.List.index ( )
4.List. Count ( )	8.List.extend ( )	

8. **What is the List Parameter?**
   - Passing a list as an argument actually passes a reference to the list, not a copy of the list. Since list mutable changes made to the parameter changes the argument as well.
   - When you pass a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change.

For example, **delete_head** removes the first element from a list:

```
def delete_head(t):
del t[0]
```

Here's how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> print letters
['b', 'c'].
```

9. Define Tuples?

- A tuple is a **sequence of values**.
- The values can be any type, and they are indexed by integers, so in that respect tuples are a lot like lists. The important difference is that **tuples are immutable.**

Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

Although it is not necessary, it is common to enclose tuples in parentheses:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

To create a tuple with a single element, you have to include the final comma:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

10. Give Example for Tuple Assignment?

- One of the unique features of the python language is the ability to have a tuple on the left hand side of an assignment statement.
- This allows you to assign more than one variable at a time when the left hand side is a sequence.

Example:

Two element in list (which is a sequence) and assign the first and second elements of the variables x and y in a single statement.

```
>>> m = ( ‘have’, ‘fun’)
>>> x,y= m
>>> x
‘have’
>>>y
‘fun’
```

11. Give Example for Tuple as Return Values?

- A function can only return one value, but if the value is a tuple, the effect is the same as **returning multiple values.**

Example:

```
 >>> t = divmod(7, 3)
>>> print t
(2, 1)
```

Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> print quot
2
>>> print rem
1
```

12. What is List comprehension?Give Eg.

- A list Comprehension is a convenient way **to produce a list from an iterable** (a sequence or other object that can be iterated over).
- In the simplest form, a list comprehension resembles the header line of a “for Statement” inside square brackets.

```
Syntax : [Expr(x) for x in iterable]
```

Example:

```
>>> a = [11,22,33,44]
>>> b =[x*2 for x in a]
>>>b
[22,44,66,88]
```

13. List the function of Tuple Data Types?

S.No	Function	Description
1	Cmp(tup1,tup2)	Compare elements of both tuples
2	Len(tuple)	Gives the total length of the tuple
3	Max(tuple)	Returns item from the tuple with max value
4	Min(tuple)	Returns item from the tuple with min value
5	Sorted(tuple)	Sort the tuple in ascending or descending

14. What is the difference between Tuples and List in Python?

- The main difference between list and tuples are:
- List are enclose in square bracket [ ] and their elements size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated.

15. Define Mutable and Immutable data type?

- **Immutable data value:** A data value which cannot be modified. Assignments to elements or slices of immutable values causes a runtime error
- **Mutable data value:** A data value which can be modified .The Types of all mutable value are compound types.List and dictionaries are Mutable; Strings and Tuple are not.

16. Differentiate between append( ) and extend ( ) methods?

- Both append ( ) and extend ( ) methods belong to the list method.These methods are used to add the elements to the list.
- **Append(element)** – adds the given element at the end of the list or at the given index postion which has called this method
- **Extend (another_list)-** adds the element of another-list at the end of the list which is called the extend method.

17. When is a dictionary used instead of a list?

- **Dictionaries** are best suited when the data is labelled ie. The data is a record with field names.
- **List** are better option to store collection of un-labelled items say all the files and sub-directories in a folder.
- Generally search operation on dictionary object is faster than searching a list objects

.

18. Differentiate between tuples and dictionaries.

Tuples	Dictionaries
• A tuple is a sequence of values. • The values can be any type, and they are indexed by integers, so in that respect tuples are a lot like lists. The important difference is that tuples are immutable.	• Python dictionary are kind of hash table type. • Dictionary work like associative arrays or hashes found in perl and consist of key value pairs.

19. Define dictionary with an example.

- Dictionary is one of the compound data type like strings, list and tuple. Every element in a dictionary is the **key-value pair.**
- An empty dictionary without any items is written with just two curly braces, like this: {}.

Example:

```
>>> eng2sp = {}
>>> eng2sp["one"] = "uno"
>>> eng2sp["two"] = "dos"
>>> print(eng2sp)
```

Output:

```
{"two": "dos", "one": "uno"}
```

20. What is the output of Print List + tinylist*2?
If List = [‘abcd’, ‘786’, ‘2.23’, ‘Nita’, ‘70.2’] and tinylist = [‘123’, ‘Nita’]?

- It will print the concatenated list since it has the (+) operator.
- Ouput will be [‘abcd’, ‘786’, ‘2.23’, ‘Nita’, ‘70.2’, ‘123’, ‘Nita’, ‘123’, ‘Nita’]

21. How will u create a Dictionary in Python?

- A dictionary can be Created by specifying the key and value separated by colon(:) and the elements are separated by comma (,).The entire set of elements must be enclosed by curly braces {}.

Syntax:

```
#To Create a Dictionary with Key-Value Pairs:

dict={Key 1:Value 1, Key 2:Value 2, Key 3:Value 3, Key 4:Value 4}

#To create an empty Dictionary

Dict={ }
```

22. List out the methods on dictionaries?

S.No	Method	Description
1	dict.clear()	Removes all elements of dictionary dict
2	dict.copy()	Removes all elements of dictionary dict
3	dict.items()	Returns a list of dict's (key, value) tuple pairs
4	dict.keys()	Returns list of dictionary dict's keys
5	dict.values()	Returns list of dictionary dict's values

23. List out the operations on dictionaries?

Operation	Description
cmp(dict1, dict2)	Compares elements of both dict.
len(dict)	Gives the total length of the dictionary.
str(dict)	Produces a printable string representation of a dictionary
type (variable)	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

16 MARKS

1. **What is List Values? Describe about creating a list, accessing the values in list, deleting a list, updating a list.**

LIST VALUES:

- A List is an **ordered set of values**, where each value is **identified by an index**.
- The Values that make up a list are called its **elements or items**.
- Lists are similar to strings, which are **ordered set of characters**, except that the element of a list can have any type.

CREATING A LIST:

There are several ways to create a new list, the simplest is to enclose the elements in square Bracket [ ]

[10, 20, 30, 40] *# list of four integers*
['frog', 'Dog', 'Cow'] *# list of three strings*

The elements of a list don't have to be the same type.
['spam', 2.0, 5, [10, 20]] ***#*** list contains a string, a float, an integer, and another list:

A list within another list is **nested.**
A list that contains no elements is called an **empty list**;
We can create one with empty brackets [].

ACCESSING VALUES IN LIST:

We can assign list values to variables:

```
>>> Fruits = ['Apple', 'Watermelon', 'Banana']
>>> numbers = [17, 12.3]
>>> empty = [ ]
>>> print ( Fruits)
>>>print( numbers)
>>>print(empty)
```

Output:

```
['Apple', 'Watermelon', 'Banana']

 [17, 12.3]

 [ ]
```

DELETING A LIST:

- ➢ Any element in the list can be deleted, del removes an element from a list.

Example:

```
>>> a=('one','two','three')
>>>del a(1)
>>>a
```

Output:
(‘one’,’three’)

UPDATING A LIST:

A slice operator on the left side of an assignment can update multiple elements:
```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print t
Output: ['a', 'x', 'y', 'd', 'e', 'f'].
```

2. **Explain the basic List Operations and list slices in details with necessary programs.**

LIST OPERATIONS:

24. Lists respond to the + and * operators much like strings; they mean **concatenation and repetition** here too, except that the result is a new list, not a string.

Python Expression	Description	Results
len([1, 2, 3])	Length	3
[1, 2, 3] + [4, 5, 6]	Concatenation	[1, 2, 3, 4, 5, 6]
['Hi!'] * 4	Repetition	['Hi!', 'Hi!', 'Hi!', 'Hi!']
3 in [1, 2, 3]	Membership	True

The + operator concatenates lists:
```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

Similarly, the * operator repeats a list a given number of times:
```
>>> [0] * 4
[0, 0, 0, 0]# It repeats [0] four times
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3] # It repeats the list [1, 2, 3] three times.
```

LIST SLICES:

- A **sub-sequence of a sequence** is called a slice and the operation that performs on subsequence is called slicing.

The slice operator also works on lists:
```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
Output : ['b', 'c']

>>> t[:4]
Output:['a', 'b', 'c', 'd']

>>> t[3:]
Output:['d', 'e', 'f']
```

```
>>> t[:]
Output : ['a', 'b', 'c', 'd', 'e', 'f']
```

Since **lists are mutable**, it is often useful to make a copy before performing operations..
A slice operator on the left side of an assignment can update multiple elements:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']
>>> print t
Output: ['a', 'x', 'y', 'd', 'e', 'f'].
```

3. Discuss the Python List Methods with examples.(16 mark)

Python provides methods that operate on lists.

The various methods used in list are:

1.List.append( )	5.List.pop ( )	9. List.clear ( )
2.List.remove( )	6.List.sort ( )	10. List.index ( )
3. List. Count ( )	7.List.extend ( )	
4. List.insert ( )	8.List.reverse( )	

1.List.append( ):	**2. List.remove ( )**

➢ The method **append ()** appends a passed *obj* into the existing list.
 	
Example
Eg 1:
x = [123, 'xyz', 'zara', 'abc'];
x.append (2009);
print ("Updated List : ", x)
Output: Updated List: [123, 'xyz', 'zara', 'abc', 2009]
 	
Eg 2:
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print (t)
Output:['a', 'b', 'c', 'd']
	➢ List.remove() method is to remove the object from the list.
	➢ List.remove() method does not return any value but removes the given object from the list.
	Example
	y = [123, 'xyz', 'zara', 'abc', 'xyz'];
	y.remove('xyz');
	print("List 1 : ", y);
	y.remove('abc');
	print("List 2: ", y)
	Output:
	List 1: [123, 'zara', 'abc', 'xyz']
	List 2: [123, 'zara', 'xyz'

3. List.count ( )	4. **List.insert ( )**
➢ The method returns the count of how many times an element occurs in specified list. **Example** X = [123, 'xyz', 'zara', 'abc', 'xyz',123, ‘xyz’]; Print (“Count for 123:”,X.count(123)) Print (“Count for Zara:”,X.count(‘zara’)) Print (“Count for xyz:”,X.count(‘xyz’)) **Output:** Count for 123: 2 Count for Zara: 1 Count for xyz: 3	➢ **List.insert ( )** method inserts an object into the list at the offset of an index. ➢ Syntax : List.insert (index,object) #index position starts from 0. **Example** X = [123, 'xyz', 'zara', 'abc']; X.insert (3,2009) Print(“Final List :”,X) **Output:** Final List: [123, 'xyz', 'zara', 2009,'abc'];

5. **List.pop ( )**	6. **List.Sort ( )**
➢ **List.pop ( )** method is used to return the item at the given index position from the list and the removes that item. **Example** X = [123, 'xyz', 'zara', 'abc']; Print("List 1 :",X.pop(0)) Print ("List 2: ", X.pop(3)) **Output:** List 1: ['xyz', 'zara', 2009,'abc']; List 2: [123, 'xyz', 'zara',]	➢ **List.Sort ( )** method is used to sort the items in the list. **Example** X = [54, 12, 85, 65, 74, 90]; Print("sorted list:", X.sort ( )) **Output:** Sorted list= [12,54 ,65,74,85,90]
7.**List.reverse ( )**	**8. List.clear ( )**
➢ **List.reverse ( )**can reverse the order of items in a list by using this method. **Example** X = [123, 'xyz', 'zara', 'abc','xyz']; X.reverse( ) Print ("List 1 :",X) **Output:** List 1: `['xyz', 'abc', 'zara', 'xyz', 123]`	➢ List.clear() can remove all values contained in it by using this method. **Example** X = [123, 'xyz', 'zara', 'abc','xyz']; X..clear( ) Print (X) **Output:** **[ ]** ➢ We get square bracket as our output after using list.clear ( ) method , it implies that the list is now clear of all items.
9.**List.index ( )**	**10.List.extend ( )**

➢ When list starts to get long, it becomes more difficult for us to count out the items in the list.
➢ So we determine at what index position the items are located.
 	
Example
x = [123, 'xyz', 'zara', 'abc'];
Print("Index for xyz:", x.index('xyz'))
Print("Index for 123:",x.index(123))
Print("Index for zara:",x.index('Zara')
 	
Output:
Index for xyz: 1
Index for 123: 0
Index for zara: 2
	➢ If we want to combine more than one list , we use extend method .
	
	Example
	Eg 1:
	X = [123, 'xyz', 'zara', 'abc', 123];
	Y = [2009, 'manni'];
	X.extend(Y)
	Print(X)
	
	Output:
	Extended List= [123, 'xyz', 'zara', 'abc', 123,2009, 'manni'];

4. Discuss about the list loop, list mutability with examples.(8 mark)

- In List loop, we use a loop to access all the element in a list. A loop is a block of code that repeats itself until it run out of items to work with or until a certain condition is met.
- Our loop will run once for every item in our list,

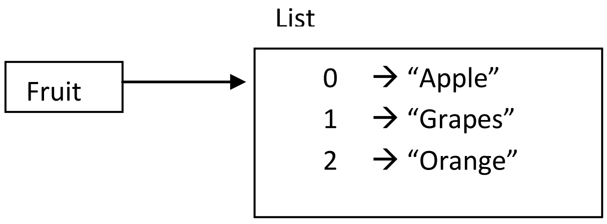
```
#LIST LOOPING EXAMPLE 1:
Colours_list=["red", "green", "yellow", "blue", "purple"]
for i in colours_list:
Print(i)
Output:
Colours in the list
Red
Green
Yellow
Blue
Purple.
```

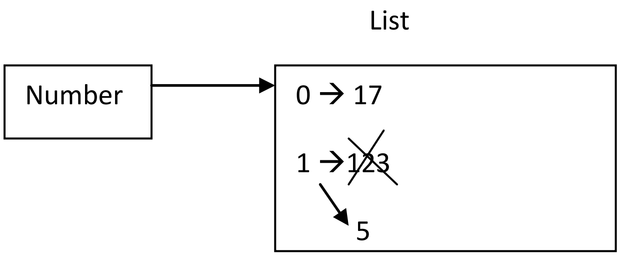
```
EXAMPLE 2:
Month_list = ["Jan", "Feb", "March", "April", "May"]
Print("months in the list")
for month in month_list:
Print (month)
OUTPUT:
Months in the list
Jan
Feb
March
April
May
```

List Mutability:

- Mutability is the ability for certain types of data to be changed without entirely recreating it.
- List is a mutable data type; which mean we can change their element.
- The syntax for accessing the elements of a list is the same as for accessing the characters of a string—the bracket operator.
- The expression inside the brackets specifies the index. Remember that the indices start at 0.

```
>>> Fruit = ['Apple', 'Grapes', 'Orange']
>>>print (fruit(0))
Apple
>>> numbers = [17, 123]
>>> numbers[1] = 5
>>> print numbers
[17, 5]
```





- Lists are represented by boxes with the word “list” outside and the elements of the list inside.
- Fruits refer to a list with three elements indexed 0, 1 and 2.
- Numbers contains two elements; the diagram shows that the value of the second element has been reassigned from 123 to 5.

.

The in operator also works on lists.

```
>>> Fruit  = ['Apple', 'Grapes', 'Orange']
>>> ‘Apple’ in Fruit
True
>>> ‘Watermelon’ in Fruit
False
```

Index can have negative value, it counts backward from the list.
Example:

```
>>> Fruit  = ['Apple', 'Grapes', 'Orange']
>>>Fruit(0) =  ‘pear’
>>>Fruit(-1) =  ‘Jackfruit’
>>>print (fruit)
[‘pear’, ‘grape’, ‘Jackfruit’]
```

```
#List Mutable Example
Colours_list=[“red”, “green”, “blue”, “purple”]
Print (“original list:”,colour_list)
#red is changed to pink
Colour_list[0]= “pink”

#blue is change to orange
Colour_list [-2] = “orange”
Print(colour_list)
```

Output:

```
Original List =[“red”, “green”, “blue”, “purple”]
[“pink”, “green”, “orange”, “purple”].
```

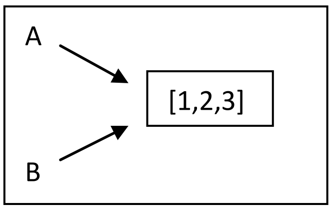
5. **Discuss about the list aliasing, cloning list and list parameter with examples.**

<u>LIST ALIASING</u>:

- Since variables refer to object, If ‘A’ refers to an object and you assign ‘ B = A’, then both variables refer to the same object:

```
>>> A = [1, 2, 3]
>>> B= A
>>> B is A
True
```

- In this case, Diagram looks like this



- The association of a variable with an object is called a reference. In this example, there are two references to the same object.
- An object with more than one reference has more than one name, then the object is said to be aliased.

If the aliased object is mutable, changes made with one alias affect the other:

```
>>> B[0] = 9
>>> print A
[9, 2, 3]
```

- Although this behaviour can be useful, it is error-prone. In general, it is safer to avoid aliasing when you are working with mutable objects.

- For immutable objects like strings, aliasing is not as much of a problem.

In this example:

```
A = ('banana')
B=( 'banana')
```

It almost never makes a difference whether A and B refer to the same string or not.

CLONING LIST:

- If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference.
- This process is sometimes called cloning, to avoid the ambiguity of the copy.
- The easiest way to clone a list is to use the slice operator

```
>>>a=[1,2,3]
>>>b=a[:]
>>>print b
[1,2,3]
```

- Taking any slice, creates a new list. In this case the slice happens to consists of the whole list.
- Now we are free to make changes to b without worrying about list ‘a’.

```
>>>b[0]=5
>>>print a
>>>print b
[1,2,3]
[5,2,3]
```

LIST PARAMETERS:

- Passing a list to a function, the function gets a reference to the list. If the function modifies a list parameter, the caller sees the change.

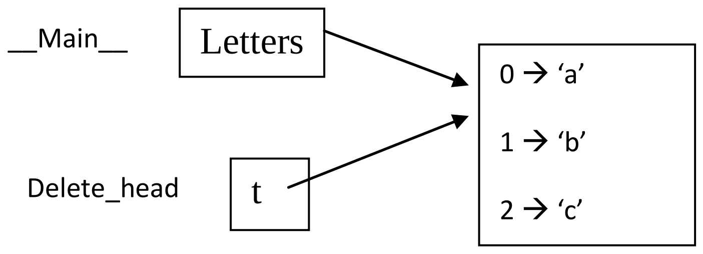
For example, delete_head removes the first element from a list:

```
def delete_head(t):
del t[0]
```

Here’s how it is used:

```
>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> print letters
['b', 'c']
```

- The parameter t and the variable letters are aliases for the same object. The stack diagram looks like this:



- Since the list is shared by two frames, I drew it between them.
- It is important to distinguish between operations that modify lists and operations that create new lists.

For example, the append method modifies a list, but the + operator creates a new list:

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> print t1
[1, 2, 3]
>>> print t2
None
>>> t3 = t1 + [3]
>>> print t3
[1, 2, 3]
>>> t2 is t3
False.
```

6. **Explain about tuples and also the concept of tuple assignment and tuples as return value with example.**
   - A tuple is a sequence of values.
   - The values can be any type, and they are indexed by integers, so in that respect tuples a like lists. The important difference is that tuples are immutable.

➢ Syntactically, a tuple is a comma-separated list of values:

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

➢ Although it is not necessary, it is common to enclose tuples in parentheses:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
```

➢ To create a tuple with a single element, you have to include the final comma:

```
>>> t1 = ('a',)
>>> type(t1)
<type 'tuple'>
```

➢ Without the comma, Python treats ('a') as a string in parentheses:

```
>>> t2 = ('a')
>>> type(t2)
<type 'str'>
```

➢ Another way to create a tuple is the built-in function tuple. With no argument, it creates an empty tuple:

```
>>> t = tuple()
>>> print (t)
( )
```

- If the argument is a sequence (string, list or tuple), the result is a tuple with the elements of the sequence:

```
>>> t = tuple('lupins')
>>> print (t)
('l', 'u', 'p', 'i', 'n', 's')
```

- Because tuple is the name of a built-in function, avoid using it as a variable name.

- Most list operators also work on tuples. The bracket operator indexes an element:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print (t[0])
'a'
```

And the slice operator selects a range of elements.

```
>>> print t[1:3]
('b', 'c')
```

But if you try to modify one of the elements of the tuple, you get an error:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

- can’t modify the elements of a tuple, but you can replace one tuple with another:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>>t = ('A',) + t[1:]
>>> print (t)
('A', 'b', 'c', 'd', 'e')
```

TUPLE ASSIGNMENT:

- One of the unique features of the python language is the ability to have a tuple on the left hand side of an assignment statement.
- This allows you to assign more than one variable at a time when the left hand side is a sequence.

In the below example , we have two element list (which is a sequence) and assign the first and second elements of the variables x and y in a single statement.

```
>>> m = ( ‘have’, ‘fun’)
>>> x,y= m
>>> x
‘have’
>>>y
‘fun’
```

Python roughly translates the tuple assignment syntax to be the following;

```
>>>m = (‘have’, ‘fun’)
>>>x = m(0)
>>>y = m(1)
>>>x
‘have’
>>>y
‘fun’
```

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

```
>>> temp = a
>>> a = b
>>> b = temp
```

Tuple assignment is more elegant:

```
>>> a, b = b, a
```

- The left side is a tuple of variables; the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments.
- The number of variables on the left and the number of values on the right have to be the same:

```
>>> a, b = 1, 2, 3

ValueError: too many values to unpack.
```

<u>TUPLES AS RETURN VALUES</u>:

- A function can only return one value, but if the value is a tuple, the effect is the same as returning multiple values.
- For example, if you want to divide two integers and compute the quotient and remainder, it is inefficient to compute x/y and then x%y. It is better to compute them both at the same time.
- The built-in function divmod takes two arguments and returns a tuple of two values, the quotient and remainder.

You can store the result as a tuple:

```
>>> t = divmod(7, 3)
>>> print t
(2, 1)
```

Or use tuple assignment to store the elements separately:

```
>>> quot, rem = divmod(7, 3)
>>> print quot
2
>>> print rem
1
```

Example of a function that returns a tuple:

```
def min_max(t):
return min(t), max(t)
```

max and min are built-in functions that find the largest and smallest elements of a sequence.
min_max computes both and returns a tuple of two values.

7. What is dictionary in python? List out the operations and methods with example.

- Keys are unique within a dictionary while values may not be.
- The values of a dictionary can be of any type, but the keys must be of an **immutable data type such as strings, numbers, or tuples.**
- Dictionary is one of the compound data type like strings, list and tuple. Every element in a dictionary is the **key-value pair.**
- An empty dictionary without any items is written with just two curly braces, like this: {}.

Creating a Dictionary:

- A dictionary can be Created by specifying the key and value separated by colon(:) and the elements are separated by comma (,).The entire set of elements must be enclosed by curly braces {}.

Syntax:

```
#To Create a Dictionary with Key-Value Pairs:

dict={Key 1:Value 1, Key 2:Value 2, Key 3:Value 3, Key 4:Value 4}

#To create an empty Dictionary

Dict={ }
```

Example:
#Keys-Value can have mixed datatype

Dict1 = {1: “Fruit”, 2: “Vegetabe”,3: “Fish”}
Dict2={“Name”: “Zara”, “Subject”:[“Maths”, “Phy”, “Chemistry”], “Marks”:[198,192,193], “Avg”:92}

Accessing Elements in Dictionary:

- To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.
- Since Dictionary is an unordered Data type. Hence the indexing operator cannot be used to access the Values.
- To access the data, we have to use key which is associated with the Value.

Example:

```
>>Dict2={“Name”: “Zara”, “Subject”:[“Maths”, “Phy”, “Chemistry”],
“Marks”:[198,192,193], “Avg”:92}
>>Dict2[“Name”]
>>Dict2[“Avg”]
>>Dict2[“Subject”]
```

Output:

```
Zara
20.1
[“Maths”, “Phy”, “Chemistry”]
```

Deleting Element in Dictionary:

- The element in the dictionary can be deleted by using **Del statement.**
- The entire dictionary can be deleted by specifying the dictionary variable name.

Syntax:

```
Del (dictionary_Name)<Key-Value>
```

Example:

```
>>> dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
>>> del dict['Name']
>>> dict
{'Age': 7, 'Class': 'First'
```

Updating Element in Dictionary:

- In Dictionary the Keys are immutable, however the values are mutable.
- Hence only the Value pair can be Updated

Example

```
>>> My_dict={'Name':'Nivetha','rank':5,'Average':78.9}
>>> My_dict['rank']=3
>>> My_dict
{'Name': 'abi', 'rank': 3, 'Average': 78.9}
```

DICTIONARY METHODS:

S.No	Method	Description
1	dict.clear()	Removes all elements of dictionary dict
2	dict.copy()	Removes all elements of dictionary dict
3	dict.items()	Returns a list of dict's (key, value) tuple pairs
4	dict.keys()	Returns list of dictionary dict's keys
5	dict.values()	Returns list of dictionary dict's values
6	dict.update(dict2)	Adds dictionary dict2's key-values pairs to dict
7	dict.fromkeys()	Create a new dictionary with keys from seq and values set to value.
8	dict.get(key, default=None)	For key, returns value or default if key not in dictionary
9	dict.has_key(key)	Returns true if key in dictionary dict, false otherwise

DICTIONARY OPERATION:

Operation	Description	Input	Function	Output

cmp(dict1, dict2)	Compares elements of both dict.	Dict1={“Name”:”zara”,”Age”:14,”sex”:”M”}
		Dict2={“Name”:”zara”,”Age”:14}
		Dict3={“Name”:”zara”,”Age”:14}
			Cmp(dict1,dict2)
			Cmp(dict2,dict3)
				1
				0

len(dict)	Gives the total length of the dictionary.	Dict1={“Name”:”zara”,”Age”:14,”sex”:”M”}
		Dict2={“Name”:”zara”,”Age”:14}
			Len(dict1)
			Len(dict2)
				2
				3

str(dict)	Produces a printable string representation of a dictionary	Dict1={“Name”:”zara”,”Age”:14,”sex”:”M”}	Str(dict1)	{“Name”:”zara”,”Age”:14, ,”sex”:”M”}
type (variable)	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.	Dict1={“Name”:”zara”,”Age”:14,”sex”:”M”}	Type(dict1)	<type ‘dict’>

Example:

```
>>>dict1 = {1: “Fruit”, 2: “Vegetabe”,3: “Fish”}
>>>Print(dict1)
>>>{1: “Fruit”, 2: “Vegetabe”,3: “Fish”}
>>> del dict[1]
>>>print(dict1)
>>>{ 2: “Vegetabe”,3: “Fish”}
```

The len function also works on dictionaries; it returns the number of key:value pairs:

```
>>> len(dict1)
3
```

8. Illustrate List Comprehension with suitable examples.

(or)

Explain about the advanced list processing.

- A list Comprehension is a convenient way to produce a list from an iterable (a sequence or other object that can be iterated over).
- In the simplest form, a list comprehension resembles the header line of a “for Statement” inside square brackets.
- However,in a list Comprehension, the for statement header is prefixed with an expression and surrounded by square bracket.

```
Syntax : [Expr(x) for x in iterable]
```

Where:

Expr(x) is an expression,usually but not always containing X.

Iterable is some iterable.An itrable may be a sequence or an unordered collection a list, string or tuple.

Example 1:

```
>>> a = [11,22,33,44]
>>> b =[x*2 for x in a]
>>>b
[22,44,66,88]
```

Example 2:

Given the following list of strings:

Names= [‘alice’, ‘ramesh’, ‘nitya’]

A list of all upper case Names

A List of Capitalized (first letter upper case)

```
>>>[x.upper() for x in names]
[‘ALICE’, ‘RAMESH’, ‘NITA’]
>>>[x.capitalize() for x in names]
[‘Alice’ , ‘Ramesh’ , ‘Nita’]
```

Example 3:

```
>>> fish_tuple=('blowfish','clowfish','catfish','octopus')
>>> fish_list=[fish for fish in fish_tuple if fish != 'octopus']
>>> print(fish_list)
['blowfish', 'clowfish', 'catfish']
```

Example 4:

```
My_list=[]
for x in [20,40,60]:
        for y in [2,4,6]:
          My_list.append(x*y)
print (My_list)
Output: [40, 80, 120, 80, 160, 240, 120, 240, 360]
```

Note: This code multiplies the items in the first list (x) by the items in the second list (y) over each iteration

Program For Example 4 using List Comprehension

```
My_list=[x*y for x in [20,40,60] for y in [2,4,6]]
print(My_list)
Output: [40, 80, 120, 80, 160, 240, 120, 240, 360]
```

List Comprehensions allows us to transform one list or other sequence into a new list. They Provide a concise syntax for completing the task and limiting the lines of code.

Example 5:

```
#To create a simple list
x=[ i for i in range (10)]
print (x)
Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

ILLUSTRATIVE PROGRAMS:

1. **SELECTION SORT:**

```
def  selectionSort(alist):
for fill slot in range(len(alist) - 1, 0, -1):
positionOfMax = 0
for location in range(1, fillslot + 1):
if alist[location] >alist[positionOfMax]:
positionOfMax = location
temp = alist[fillslot]
alist[fillslot] = alist[positionOfMax]
alist[positionOfMax] = temp
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
selectionSort(alist)
print(alist)
```

OUTPUT:

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

2. **INSERTION SORT:**

```
def  insertionSort(alist):
for index in range(1,len(alist)):
currentvalue = alist[index]
position = index
while position > 0 and alist[position - 1] >currentvalue:
alist[position] = alist[position - 1]
position = position - 1
alist[position] = currentvalue
alist = [54, 26, 93, 17, 77, 31, 44, 55, 20]
insertionSort(alist)
print(alist)
```

OUTPUT:

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

3. **MERGE SORT:**

```
def mergeSort(alist):
print("Splitting ",alist)
if len(alist)>1:
mid = len(alist)//2
lefthalf = alist[:mid]
righthalf = alist[mid:]
mergeSort(lefthalf)
mergeSort(righthalf)
i=0
j=0
k=0
while i <len(lefthalf) and j <len(righthalf):
if lefthalf[i] <righthalf[j]:
alist[k]=lefthalf[i]
i=i+1
else:
alist[k]=righthalf[j]
j=j+1
k=k+1
while i <len(lefthalf):
alist[k]=lefthalf[i]
i=i+1
k=k+1
while j <len(righthalf):
alist[k]=righthalf[j]
j=j+1
k=k+1
print("Merging ",alist)
alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)
print(alist)
```

OUTPUT:

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

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

PROBLEM SOLVING & PYTHON PROGRAMMING
UNIT – I ALGORITHMIC PROBLEM SOLVING

PART-A

1. **What is an algorithm?**
   Algorithm is defined as a step by step procedure for solving any problem. The sequence of steps to be performed in order to solve a problem by the computer is known as an algorithm.

2. **What are the characteristics of an algorithm?**
   - Algorithm has a finite number of inputs.
   - Every instruction should be precise and unambiguous.
   - Ensure that the algorithm has proper termination.
   - Effectiveness of each step is very important.
   - The desired output must be obtained only after the algorithm terminates.
   - The algorithm should be in sequence.

3. **How can measure the quality of algorithm?**
   - Time - Lesser the time taken better the quality.
   - Memory - Require minimum computer memory.
   - Accuracy – Should provide accurate result.
   - Sequence - Procedure of an algorithm in a sequential form.
   - Generality – fit to all type of inputs.

4. **What are the kinds of statement available in algorithm?**
   - Simple statement
   - Compound Statement.

5. **What are the representations of algorithm?**
   - Normal English
   - Program
   - Flowchart
   - Pseudo code
   - Decision table

6. **What is Flowchart? Why it is required?**
   Flowchart is a diagrammatic representation (or) graphical representation of an algorithm, often used in the design phase of programming to work out the logic flow of a program.

7. **What is Pseudocode?**
   Pseudocode came from two words. Pseudo and Code Pseudo means imitation and Code
   refer to instructions written in a programming language.

8. What are the rules for drawing a flowchart?

- The standard symbols should only be used.
- The arrowheads in the flowchart represent the direction of flow in the problem.
- The direction of the flow from top to bottom (or) left to right.
- The flow lines should not cross each other.
- Keep flowchart as simple as possible.
- Words in the flowchart should be common and easy to understand.
- More than one flowcharts connected by using connectors.

9. What are the rules for writing Pseudocode?

- Write one statement per line.
- Capitalize initial keywords.
- Ident to show hierarchy.
- End multi line structure.
- Keep statement language independent.

10. List out the basic design structure (or) basic logic structure.

- Sequence structure.
- Selection structure.
- Loop structure.

11. List out the advantages of flowchart.

- Communication.
- Effective analysis.
- Proper documentation.
- Efficient coding.
- Proper Testing and Debugging.
- Efficient Program Maintenance.

12. What are the limitations (or) disadvantages of flowchart?

- Complex Logic
- Alteration or Modification
- No update.

13. List the advantages of Pseudocode.

- Can be done easily on a word processor
- Easily modified
- No symbols are used
- It is simple because it uses English-like statements.
- No specific syntax is used.

14. List out the Disadvantages of Flowchart.

- It's not visual
- There is no accepted standard.
- Cannot be compiled not executed.

15. Define Functions.

Functions are "self-contained" modules of code that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result.

16. What are the elements of functions?

- Function declaration
- Function call
- Function definition

17. Define State

- Sate is the result of a test (or) condition.
- If the result is "TRUE", take a certain course of action and if the result is "FALSE", take another course of action.

18. Give the types of programming languages used in computer programming.

- Machine language
- Assembler language
- High level programming language.

19. What are the steps required to solve the problem using algorithm?

- Understanding the problem.
- Ascertaining the capabilities of the computational device.
- Choosing between exact and approximate problem solving.
- Algorithm design techniques.
- Method of specifying an algorithm
- Proving an algorithm's correctness
- Analysing an algorithm.
- Coding an algorithm.

20. Differentiate Iteration and Recursion.

Iteration	**Recursion**
An iterative function is one that loops to repeat some part of the code.	A recursive is a function calls itself repeatedly until some specified condition has been satisfied.
Iterative approach involves four steps, initialization, condition, execution and updation.	In recursive function, only base condition (terminate condition) is specified.
Whereas iterative approach makes your code longer.	Recursion keeps your code **short and simple**
Iteration is faster.	Recursion is slower than iteration due to overhead of maintaining stack

21. List the advantages of Recursive functions.

- Recursion can produce simpler, more natural solutions to a problem.
- It is written with less number of statements.
- Recursive functions are effective.
- It requires few variables which makes program clean.
- It is useful for branching.

22. List the applications of iteration algorithm.

- Factorial of a given number.
- Reverse of a number.
- Fibonacci series.
- Convert the string into lowercase.
- Sum of digits.
- Armstrong number.

23. List the applications of recursion algorithm.

- Factorial of a given number using recursion.
- GCD using recursion
- Towers of Hanoi.
- 8 Queen Problem.

24. Differentiate testing and debugging

Testing	**Debugging**
Finding and locating the defect.	Fixing the defect.
Done by testing team.	Done by development team.
Testing is to find the defects	Debugging is to remove the defects

PART – B

1. What is algorithm? Explain characteristics, quality and representation.

Algorithm is defined as a step by step procedure for solving any problem. The sequence of steps to be performed in order to solve a problem by the computer is known as an algorithm.

Characteristics of an algorithm:

- Algorithm has a finite number of inputs.
- Every instruction should be precise and unambiguous.
- Ensure that the algorithm has proper termination.
- Effectiveness of each step is very important.
- The desired output must be obtained only after the algorithm terminates.
- The algorithm should be in sequence.

Quality of algorithm:

- Time - Lesser the time taken better the quality.
- Memory - Require minimum computer memory.
- Accuracy – Should provide accurate result.
- Sequence - Procedure of an algorithm in a sequential form.
- Generality – fit to all type of inputs.

Representation of algorithm:

- Algorithm has a starting point and a final point. Between these two points are the instructions that solve the problem.
- Algorithms often have steps that repeat or require decisions.
- Algorithm can be expressed in any language from natural languages to programming languages.

Example:

Algorithm (find the area)

Step 1: start
Step 2: Read the value of radius r
Step 3: calculate area=3.14*r*r
Step 4: Print the area of circle.
Step 5: Stop

2. Explain in detail about building blocks of algorithms.

- Instruction/Statements
- State/Selection
- Control flow
- Functions

(i) <u>Instruction/Statements:</u>

- A statement is the smallest standalone element of a programming language that expresses some action to be carried out.

- An instruction written in a high-level language that commands the computer to perform a specified action.

- A program written in a language is formed by a sequence of one or more statements. A statement may have internal components like expressions.

Kinds of statements:

i. Simple statements
   - Assignment:A=A+2
   - Goto:goto next;
   - Return: return 10;

ii. Compound statements
   Block:begin------------end
   Do-loop:do-------------while(i!=10)
   For-loop:for(…)----------------

- A statement is executed, with an expression is evaluated.

(ii) <u>State:</u>

- An algorithm is deterministic automation for accomplishing a goal which, given an initial state, will terminate in a defined end-state.
- When an algorithm is associated with processing information, data is read from an input source or device, written to an output device, and/or stored for further processing.
- Stored data is regarded as part of the internal state of the algorithm.
- The state is stored in a data structure.

(iii) Control flow :(Explain the control structures in detail)

- Flow of control (or) control flow is the order function calls, instructions, and statements are executed or evaluated when a program is running.
- Program control structures are defined as the program statements that specify the order in which statements are executed.

(i)Sequence Control Structures

- Sequential control structure is used to perform the actions one after another.
- This structure is represented by writing one process after another.

Example

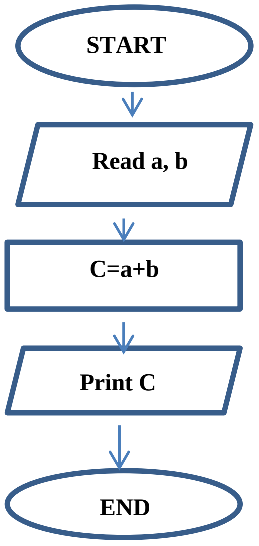
Algorithm (Add two numbers)

Step 1: Start
Step 2: Get the value of a and b.
Step 3: c=a+b
Step 4: print value c.
Step 5: End.

Pseudocode

```
READ a, b
C=a+b
WRITE C
```

Flowchart



(ii)Selection Control Structures

- Selection control structures (or) Decision structures allows the program to make a choice between two alternate paths whether it is true or false.
- IF…THEN…ELSE or a case structures are the selection structures.
- This logic is used for making decisions.

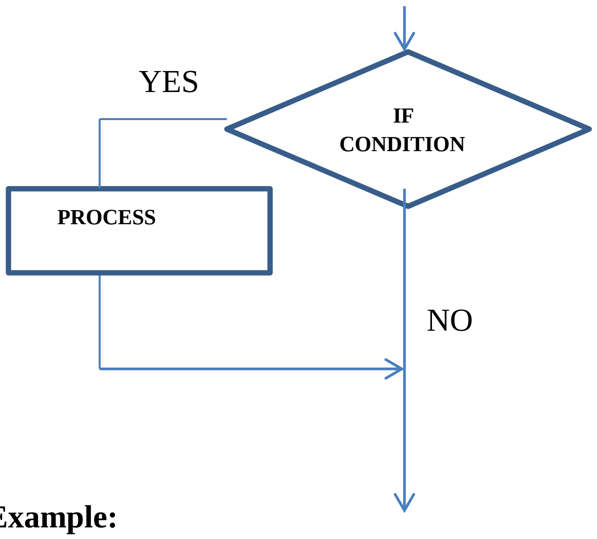
(a)IF…THEN Structures

- This makes a choice between two processes. If the condition is true it performs the process. If the condition in false it skips over the process.

Pseudocode

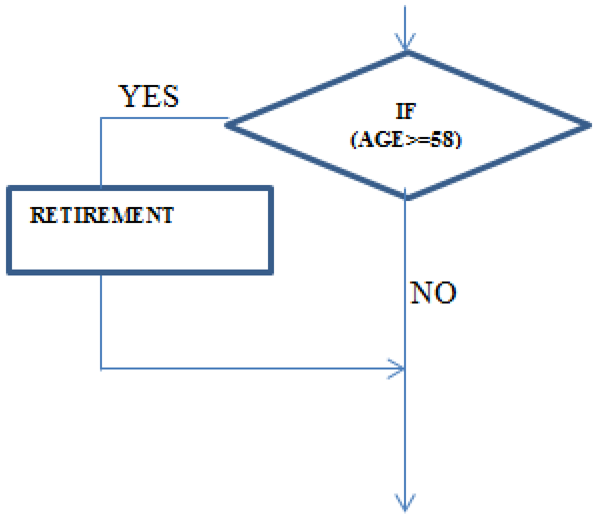
```
IF condition THEN
Process 1
……..
……..
ENDIF
```

Flowchart



Example:

```
IF age>=58 THEN
WRITE person gets retirement
ENDIF
```

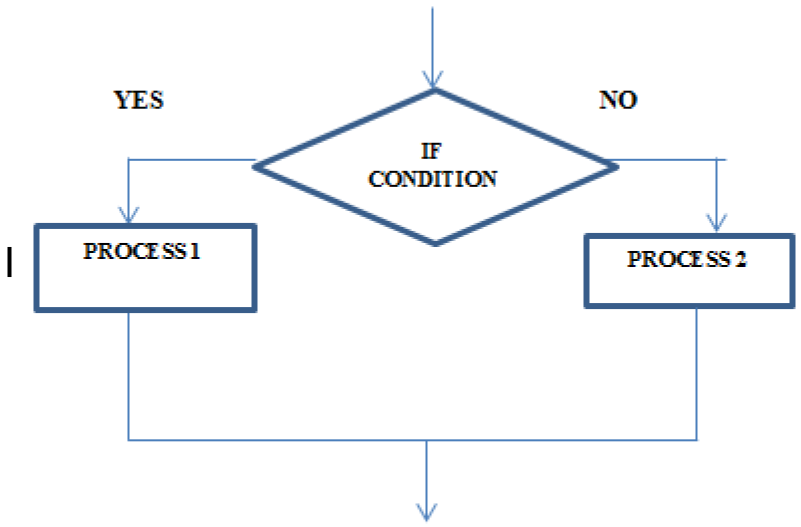


(b) IF…THEN…ELSE Structures

Pseudocode

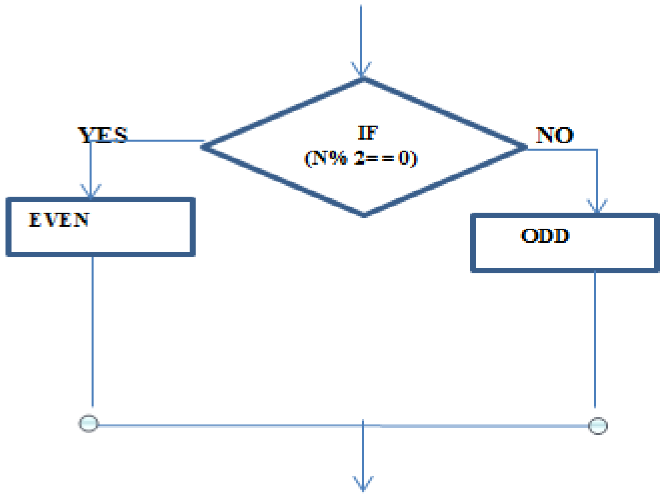
```
IF condition THEN
  Process 1
  …..
.
  ELSE
  Process 2
  …..
  …..
  ENDIF
```

Flowchart



Example

```
IF n%2= = 0 THEN
WRITE n is even
ELSE
WRITE n is odd.
```



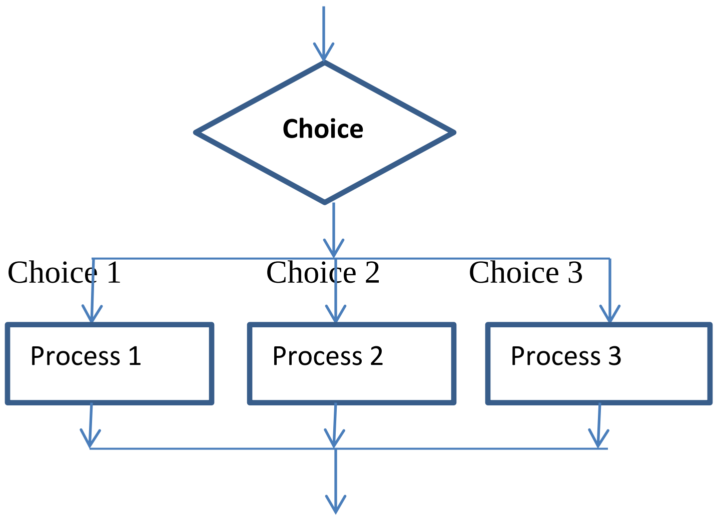
(iii) Case Structure

- This is a multiway selection structures that is used to choose one option from many options.
- ENDCASE is used to indicate the end of the CASE structure.

Pseudocode

```
   CASE TYPE
Case choice – 1:
Process 1
break;
Case choice – 2:
Process 2
….
….
Case choice-n:
Process n
….
END CASE
```

Flowchart



(iv) <u>Functions:</u>

- ➢ Functions are self-contained modules of code that accomplish a specific task.
- ➢ Functions usually take in data, process it, and return a result. Once a function is written, it can be used again and again.
- ➢ Functions can be from inside of other functions.

Importance of function(Need of function)

- It allows to divide the larger programs into sub-programs (or) sub-modules.
- It allows to reuse code instead of rewriting it.
- Functions allow to keep variable namespace clean.
- Functions allow to test small parts of our program in isolation from the rest.

Steps to Writing a function

- Understand the purpose of the function.
- Define the data that comes into the function from the caller.
- Define data variables are needed inside the function to accomplish a goal.
- Decide on the steps that the program will use to accomplish this goal.

Parts of a function

- Function declaration – int add(int,int);
- Function call – add(a,b);
- Function definition – int add(int x, int y){ z=x+y,return z}

3. Explain in detail about notations.

<u>(i) Pseudocode:</u>

- ➢ Pseudocode is a kind of structure English for designing algorithm.
- ➢ Pseudo means false and code refers to instructions written in programming languages.
- ➢ Pseudocode cannot be compiled or executed and there are no real formatting or syntax rules.
- ➢ The benefit of pseudocode is that it enables the programmer to concentrate on algorithms without worrying about all syntactic details of a particular language.

Example:

```
READ num1,num 2
Result=num1 +num 2
WRITE result
```

Guidelines for writing pseudocode:

- **Only one statement per line:** Readability improves if just one action for the computer is written in one statement.
- **Capitalized initial keyword:** Keyword like READ,WRITE etc.
  Example: IF,ELSE,ENFIF,WHILE,ENDWHILE etc.
- **Indent to show hierarchy:** In loop, states and iterations the logically dependent statements must be indent.
  Example:
  ```
  If a>b then
  Print a
  ELSE
  Print b
  ```
- **End multi-line structures:** To improve readability the initial state and end of the several line must be specified properly.
  Example: ENDIF for IF statement.
- **Keep statement language independent:** The programmer must never use the syntax of any programming language.

Advantages of pseudocode:

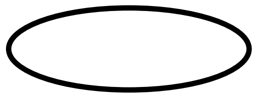
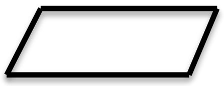
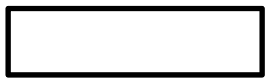
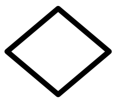
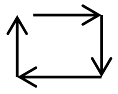
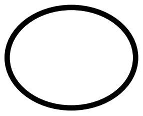
- Can be done easily on a word processor.
- Easily modified.
- Implements structured concepts.
- No special symbols are used.
- No specific syntax is used

Disadvantages of pseudocode:

- It’s not visual.
- Cannot be compiled not executed.
- There is no accepted standard, so it varies widely from company to company.

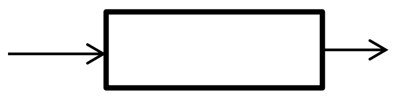
(ii) Flowchart:

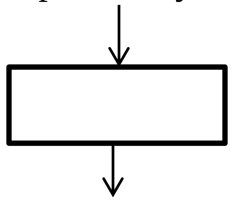
- A flow chart is a diagrammatic representation that illustrates the sequence of operations to be performed to arrive at the solutions.
- Each step in the process is represented by a different symbol and contains a of the process step.
- The flowchart symbols are linked together with arrows showing the flow of directions.

S.NO	NAME OF SYMBOL	SYMBOL	DESCRIPTION
1.	Terminal symbols		Represent the start and stop of the program.
2.	Input/Output		Denoted either an input or output operation.
3.	Process symbol		Denotes the process to be carried out
4.	Decision		Represent decision making and branching
5.	Flow lines		Represent the sequence of steps and direction of flow .Used to connect symbols.
6.	Connectors		A connector symbol is represented by a circle. The symbol are used to conect the flowchart

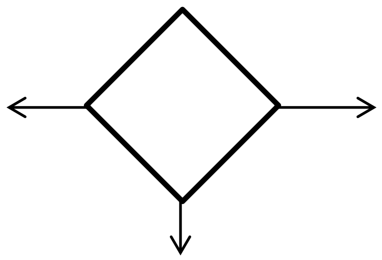
Guidelines For Preparing Flowchart:

- In drawing a proper flow chart all necessary requirements should be listed out in logical order.
- The flowchart should be clear and easy to follow.
- Only one flow line should come out from process symboL

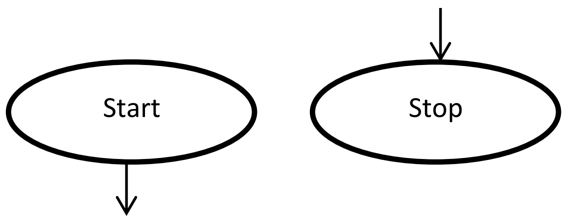




- Only one flow line should enter a decision symbol, but two or more flow lines, can leave the decision symbol

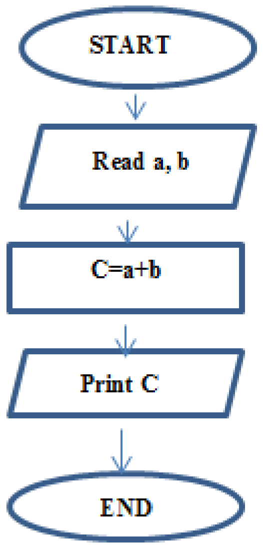


- Only one flow line is used in conjunction with terminal symbol



- If the flow chart becomes complex, it is better to use connector symbol to reduce the number of flow lines.

Example: Adding two numbers



<u>**Advantages of flow chart:**</u>

- **Communication:** Flowcharts are better way of communicating the logic of a system.
- **Effective analysis:** Problem can be analyzed in effective way.
- **Proper documentation:** Programs flowcharts serve as a good proper documentation, which is need for various purposes.
- **Efficient coding:** The flowchart acts as a guide or blueprint during the system analysis.
- **Proper Testing and debugging:** Flowchart helps in debugging process.

<u>Disadvantages of flowcharts:</u>

- Complex logic.
- Alterations and modifications.
- No update.

(iii) Programming Language:

In computer technology, a set of conventions in which instructions for the machine are written. There are many languages that allow humans to communicate with computers.

(a)Machine Language

- The machine language consists of binary numbers that encode instructions for the computer.
- Every computer has its own machine language.

Example: 10110101-B5

(b)Assembler Language

- An assembler language consists of mnemonics
- There is one mnemonic for each machine instruction of the computer.
- Each assembler instruction is a mnemonic that corresponds to a unique machine instruction.

Example:

```
Start
Add x,y
Sub x,y
…………….
……………..
END
```

(c)High level language

- A high level programming language allows the programmer to write sentences in this language which can be easily translated into machine instructions.
- The sentences written in a high level programming language are called statements.

Example:

```
main()
{
if(x<y)
min=x
}
```

4. Explain about algorithmic problem solving in detail.

Or

Explain about the steps needed to solve a problem using algorithm in detail.

- Algorithms are procedural solutions to problems.
- These solutions are not answers but specific instructions for getting answers.
- The sequence of steps for designing and analyzing algorithm follows,
    1. Understanding the problem.
    2. Ascertaining the capabilities of the computational device.
    3. Choosing between exact and approximate problem solving.
    4. Algorithm design techniques.
    5. Method of specifying an algorithm
    6. Proving an algorithm's correctness
    7. Analyzing an algorithm.
    8. Coding an algorithm.

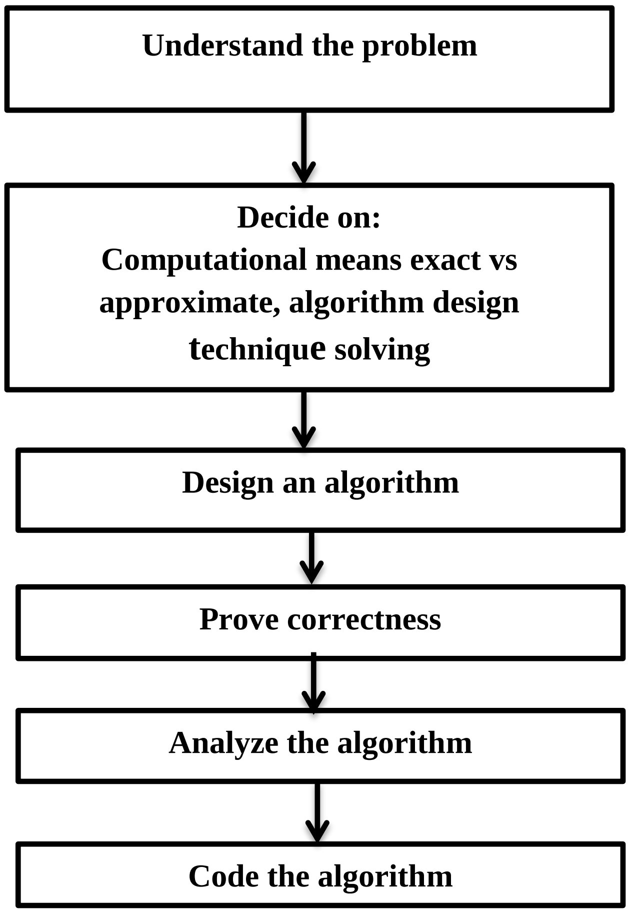


Fig: Algorithm design and analysis process

(i) Understanding The Problem:

- The given problem must be understood completely.
- An input to an algorithm specifies an instance of the problem the algorithm solves.

(ii) Ascertaining The Capabilities Of The Computational Device:

- After understanding a problem, ascertain the capabilities of the computational device.
- **Sequential algorithm:** Instructions are executed one after another, one operation at a time.
- **Parallel algorithms:** The central assumption of the RAM model does not hold for some newer computers that can execute operations concurrently, (i.e) in parallel.

(iii) Choosing between exact and approximate problem solving:

- Solving the problem exactly is called an exact algorithm.
- Solving it approximately is called an approximation algorithm.

(iv) Algorithm design techniques:

- Algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
- It provides guidance for designing algorithms for new problem.

(v) Methods of specifying an algorithm:

- Pseudo code
- Flowchart

Pseudo code:

- It is a mixture of a natural language and programming language like constructs.
- Pseudo code is usually more precise than programming language.

Flowchart:

- Flow chart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm steps.

(vi) Proving an algorithm correctness:

- After specifying the algorithm, it should be proved for its correctness.
- A common technique for proving correctness is to use mathematical induction.

- Tracing the algorithm correctness for specific inputs can be very worthwhile activity, it cannot prove the algorithm's correctness conclusively.
- The notion of correctness for approximation algorithms is less straight forward than it is for exact algorithms.

(vii) Analysing an algorithm

Three Characteristics of algorithm

- Efficiency
- Simplicity
- Generality.

Efficiency:

- There are two kinds of algorithm efficiency:
  - **Time efficiency**-Indicating how fast the algorithm runs
  - **Space efficiency**-Indicating how much extra memory it uses.

Simplicity:

- Simpler algorithms are easier to understand and easier to program**.**
- The resulting programs usually contain fewer bugs.

Generality:

- Generality of the problem the algorithm solves.
- The set of input it accepts.

(viii) Coding an Algorithm:

- Coding or programming is the process of translating the algorithm into the syntax of a given programming language.
- Convert each step in the algorithm into one or more statements in a programming language

5. Explain about Simple Strategies for Defining An Algorithm (Iteration and recursion).

Iteration and recursion are key computer science techniques used in creating algorithms and developing software.

ITERATION

- Iterative are programs that follow a path from the starting instruction till the end of the algorithm.
- Iterative functions are on that loops to repeat some part of the code.
- Using a simple loop to display the numbers from one to ten is iterative process.

Steps to develop an iterative algorithm:

- **Define problem**: The problem that needs iteration has to be identified and defined.
- **Initial conditions**: The condition that has to be satisfied, to start the iteration.
- **Define loop variants**: The variable that controls the number of iteration has to be defined.
- **Define step**: The steps that are to be repeated must be defined.
- **Define measure of progress**: The loop invariant that have been defined would be changed when algorithm progress.
- **Define Exit condition**: When the iteration would be stopped has to be identified.
- **Make progress**: Move forward after executing an instruction.
- **Maintain Loop variants**: In order to repeat the steps the loop invariant must be maintained in range.
- **Ending**: When the iteration has stop.

Example algorithm for iteration process:

1.Factorial of a given number.
2.Reverse of a number.
3.Sum of digits.

RECURSION:

- Recursive function is one that calls itself again to repeat the code.
- Recursion is a problem, solving approach by which a function calls itself repeatedly until some specified condition has been satisfied.
- Recursion splits a problem into one or more simpler versions of itself.
- In a recursive algorithm the algorithm calls itself with smaller input values.
- The recursive programs require more memory and computation compared with iterative algorithms.

Example: factorial of a given number.
n!=n x (n-1)!
4!=4 x(4-1)!
=4 x (3!)
=4 x 3 x 2 x1
=24

Advantages of recursive functions:

- Recursion can produce simpler, more natural solutions to a problem.
- It is written with less number of statements.
- Recursive functions are effective where the terms are generated successively to compute a value.
- It requires few variables.
- It is useful for branching process.

Example algorithm for recursion:

1. GCD using recursion.
2. Factorial given number using recursion.

6. ILLUSTRATIVE PROBLEMS:

1. To find a minimum in a list:

Problem statement: Find in a minimum in a list.

Problem description:

- Minimum in a list of elements can be achieved in different ways.
- One way is to sort the list of element in ascending order and get the first element as minimum.
- Another method is to compare each element with other.
- Assume the first element as minimum element and start comparing with the next element.
- If the next element is smaller than assume the second the minimum and keep repeating the procedure till the last element.

Algorithm:

Step1: Get the list of elements.
Step2: Assume first element as Min.
Step 3: Compare Min with next element.
Step 4: If MIN is greater than next element; Set MIN=next element
Step 5: Repeat step 3 and 4 till the last element.
Step 6: Display MIN as the minimum element of the list.

UNIT-III
CONTROL FLOW, FUNCTIONS

SYLLABUS:

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, local and global scope, function composition, recursion; Strings: string slices,immutability, string functions and methods, string module; Lists as arrays. Illustrative programs:square root, gcd, exponentiation, sum an array of numbers, linear search, binary search.

TWO MARKS

1. **Define Boolean values**

   The Boolean data type have 2 values (usually denoted by True or False), When converting a **Boolean to an integer**, the integer value is always **0 or 1**, but when converting an **integer to a Boolean,** the Boolean value is true for all integers **except 0**.
2. **List out the Boolean operators.**

   - Or operator.
   - And operator.
   - Not operator.
   -
3. **Define conditional statements.**

- The execution of the program acts according the **conditions**. This concept is known as **Conditional statements or Branching Statements.**

- Conditional Statements are used to perform different computation or action depending on whether a condition evaluates to **TRUE or FALSE.**

   1. **If Statement**
   2. **If...else Statement**
   3. **If..elif..else Statement**
   4. **Nested if...elif..Else Statement.**

4. **Write the syntax of if and if-else statements.**

- **Condition if:**The boolean expression after the if statement is called the **condition**. If it is true, then the indentedstatement gets executed.
  Syntax:

if (Condition):

True Statement Block

- **Alternative(IF....Else Statement):**A second form of the if statement is alternative execution, in which there are **two possibilities** and the condition determines which one gets executed.

Syntax:

```
if (Condition):
        True Statement Block
    else:
        False Statement Block
```

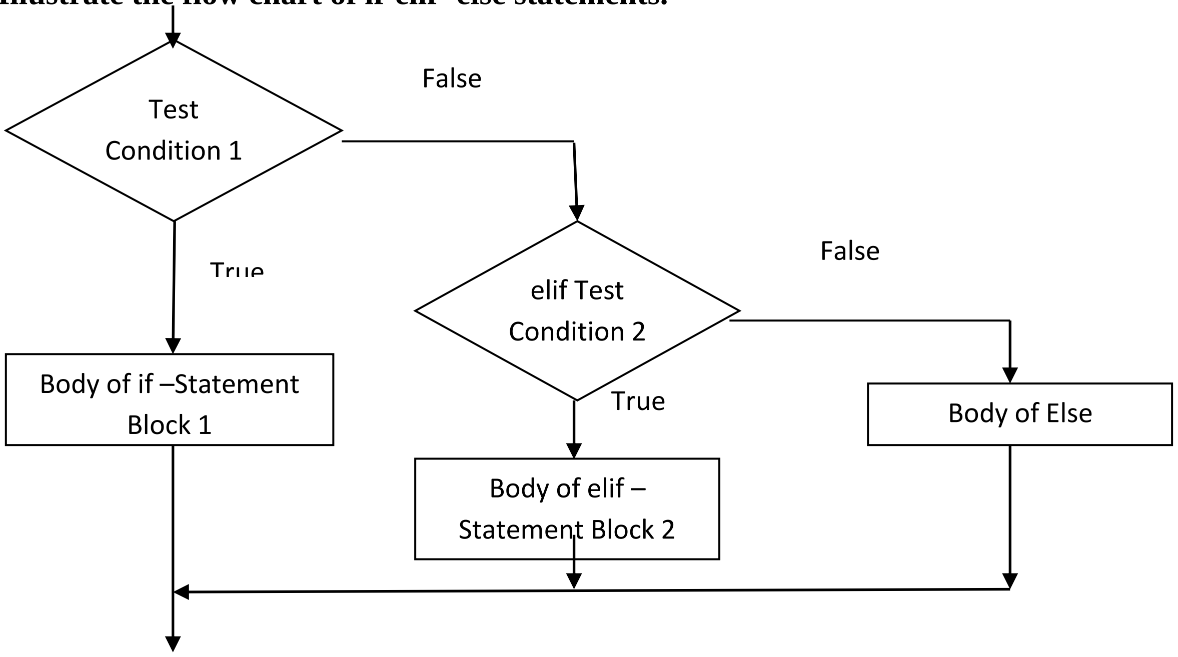
5. **What is the purpose of iteration? Give example.**

   - A loop statement allows us to execute a statement or **group of statements multiple times.**
   - **Repeated execution of a set of statements is called iteration.**

   Example:

   ```
   Count=0
   While(count<9):
   Print'the count is:',count
   Count=count+1
   ```

6. **Illustrate the flow chart of if-elif- else statements.**



7. **What are unconditional looping statements?**
   - Loop that needs to be executed compulsorily **without any condition** is called an **unconditional loop**.
   - The loop will be executed in theprogram without any conditional checks.

   Example:
   - **Break statement.**
   - **Continue statement.**
   - **Pass statement**.

8. **Explain about break statement with an example.**
   - It **terminates the current loop** and resumes execution at the next statement.
   - The most common use for break is when some external condition is triggered requiring a hasty exit from a loop.

- The **break** statement can be used in both *while* and *for* loops.

Syntax:
Break

Example :

```
forletter in 'Python':
if ( letter == 'h'):
break
print("Current Letter :", letter)
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
```

9. Discuss about continue and pass statements.

Continue Statement:

- It returns the control to the **beginning of the while loop**.
- The continue statementrejects all the remaining statements in the current iteration of the loop and movesthe control back to the top of the loop.
- The continue statement can be used in both **while and for loops.**

Syntax:
Continue

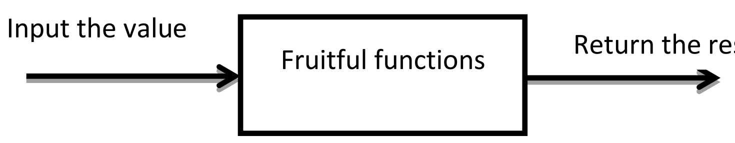
Pass Statement:

- It is used when a **statement is required syntactically** but you do not want any command or code to execute.
- The pass statement is a **null operation**; nothing happens when it executes.

Syntax:
Pass

10. What are function and fruitful function?

- **Function:** It is a block of statement that will execute for a specific task.
- **Fruitful function:**Functions that **return values** are called **fruitful function.**



- **Example:** The square function will take one number as parameter and return the result of squaring that number

11. Define parameter. List out it types.

- Parameter is the input data that is sent from **one function to another.**
- Parameter is of two types,
  - ✓ **Actual parameter.**
  - ✓ **Formal parameter**.

Actual parameter:

- Parameter is **defined in the function call**.

Formal parameter:

- Parameter is defined as **part of function definition**.
- Actual parameter value is received by formal parameter

12. Classify global variable with local variable.

- A variable in the program can be either **local variable or global variable.**
- A **global variable** is a variable that is declared in the **main program** while a **local variable** is a variable declared **within the function**.

Example:

```
S=10 # s is the global variable
def f1()
S=55# s is the local variable
Print s
```

Output:

```
55
```

13. Describe various methods used on a string. (Any Four)

- **is digit()-**returns true if string contains only digits and false otherwise.
- **islower()-**returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
- **isnumeric()-**returns true id a Unicode string contains only numeric characters and false otherwise.
- **isupper()-**returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

14. What are the advantages and disadvantages of recursion function?

Advantages:

- Reduces **time complexity**.
- Performs better in solving problems based on **tree structures**.

Disadvantages:

- It is usually slower due to the **overhead of maintain stack**.
- It usually uses **more memory of the stack.**

15. What is string?Giveexample.

- A string is a **sequence of characters**.(i.e) it can be a letter , a number, or a backslash.
- Python strings **are immutable**, which means they cannot be changed after they are created.

Example:

Var1='hello world!

16. How will you slice the given string in python?

A segment of a string is called a slice. Selecting a slice is similar to selecting a character:

<u>Example</u>

```
>>> s = 'Monty Python'
>>> print s[0:5]
Monty
>>> print s[6:13]
Python
```

17. What will be the output of print str[2:5] if str='helloworld!'?

H	E	L	L	O	W	O	R	L	D	!
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Str="HELLOWORLD!"
Str[2:5]
O/P = LLO

18. List out the applications of arrays.

Applications - Arrays are used to implement in data structures, such as lists, heaps, hash tables, deques, queues, stacks and strings.

19. Write a program to iterate a range using continue statement.

```
for   letter  in 'Python':
if ( letter == 'h'):
continue
print("Current Letter :", letter)
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

20. Define array with an example.

An array is a collection of data that holds fixed number of values of same type.
Forexample: if you want to store marks of 100 students, you can create an array for it.

21.Differentiate for loop and while loop. ?

For Loop	**While Loop**
1.Executes the sequence of statements multiple times and abbreviates the code that manages the loop **2.**Eg. for i in '123': Print ("Welcome",i, "Times") **Output:** Welcome 1 Times Welcome 2 Times Welcome 3 Times	**1.**Repeats a statement until a give condition is TRUE. It test the While condition before executing the loop body **2**.Eg.Counter = 0 While (counter < 3): Print('countis:',counter) Counter=Counter+1 **Output:** Count is: 0 Count is: 1 Count is: 2

16 MARKS

1. Boolean Value and operators(8m)

- The Boolean data type have 2 values (usually denoted by True or False), When converting a Boolean to an integer, the integer value is always 0 or 1, but when converting an integer to a Boolean, the Boolean value is true for all integers except 0.

Boolean and logical operators:

Boolean values respond to logical operators (AND / OR)

True AND False = False
True AND True = True
False AND True= False
False OR True= True
False Or False= False

- A boolean expression is an expression that is either true or false.

BOOLEAN OPERATORS(and,or,not)
These are the Boolean operation, ordered by ascending priority:

Operation	Result
X or Y	If X is false then y, else X
X and Y	If X is false, then x else Y
Not X	If x is false,then true else False

The following examples usethe operator ==, which compares two operands and produces True if they are equal and False
Otherwise:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

True and False are special values that belong to the type bool; they are not strings:

```
>>>type (True)
<type 'bool'>
>>>type(False)
<type 'bool'>
```

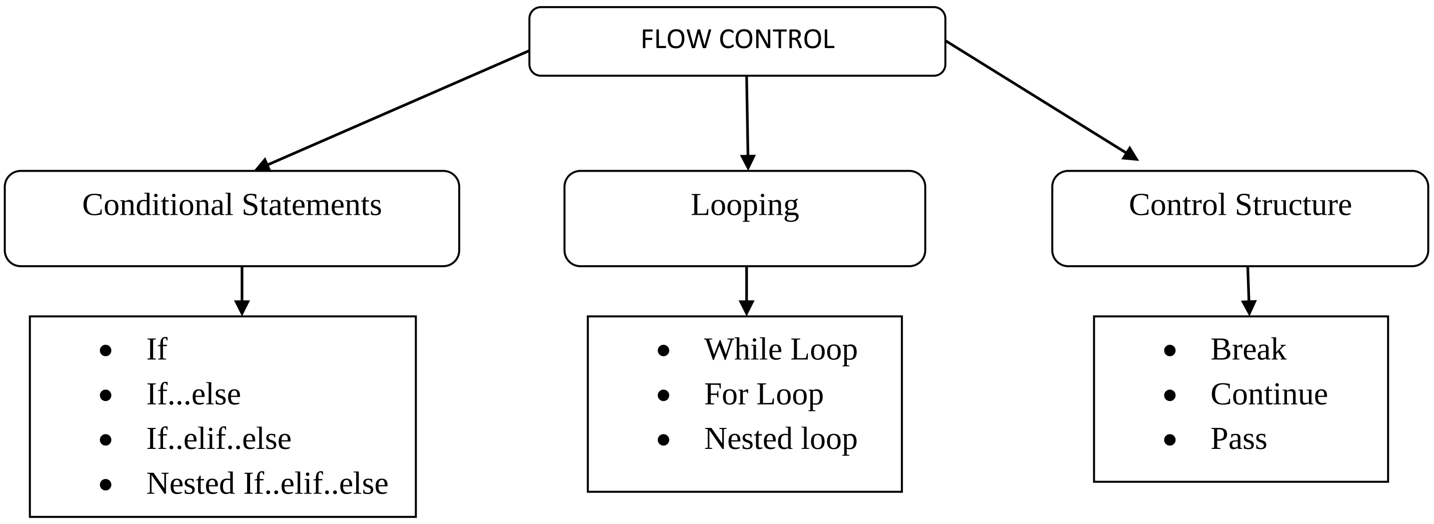
Boolean and Comparison operators:

The == operator is one of the comparison operators; the others are:
x!= y # x is not equal to y

x > y # x is greater than y
x < y # x is less than y
x >= y # x is greater than or equal to y
x <= y # x is less than or equal to y
Although these operations are probably familiar to you, the Python symbols are different from themathematical symbols. A common error is to use a single equal sign (=) instead of a double equalsign (==). Remember that = is an assignment operator and == is a comparison operator.

2. Explain the branching statements in python with relevant syntax and example.(16m)

- Control Flow is the order in which **individual Statements, instructions are executed** or evaluated.
- Flow is just a way or **sequence of program Execution**.
- Every Statement of a program is **executed one by one.** Python provides various tools for flow Control.
- Some of them are if, if..else, if..elif..else, Nestedif..elif..else, For, While, Nested, Break, Continue etc



Conditional Statements:

- The execution of the program acts **according the conditions.** This concept is known as **Conditional statements or Branching Statements**.

- Conditional Statements are used to perform different computation or action depending on whether a condition evaluates to **TRUE or FALSE.**

,
1. **If Statement**
2. **If...else Statement**
3. **If..elif..else Statement**
4. **Nested if...elif..Else Statement**

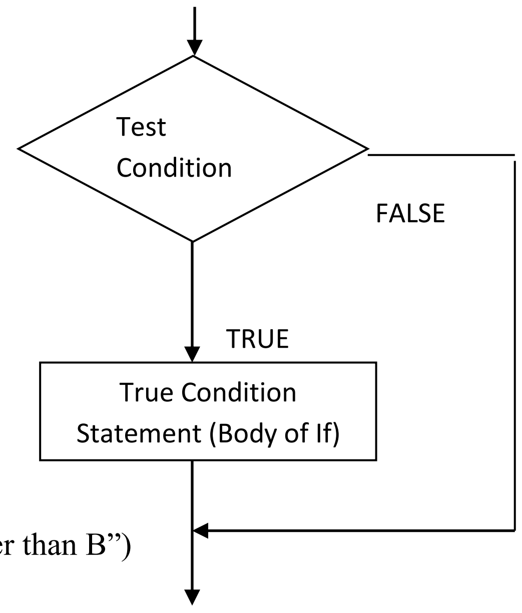
1. Conditional (If Statement):

➢ The Boolean expression after the if statement is called the **condition**. If it is true, then the indentedstatement gets executed.

- If the text condition is **FALSE**, the Statements are **not executed.**
  If statements have the same structure as function definitions: **a header followed by an indented block**. Statements like this are called **compound statements**.

Syntax:
if (Condition):
    True Statement Block

Flowchart for if statement:



Example 1:

```
A=500
B=200
If (A>B):
    Print ("A is greater than B")
```

Output:

```
A is Greater than B
```

Explanation:

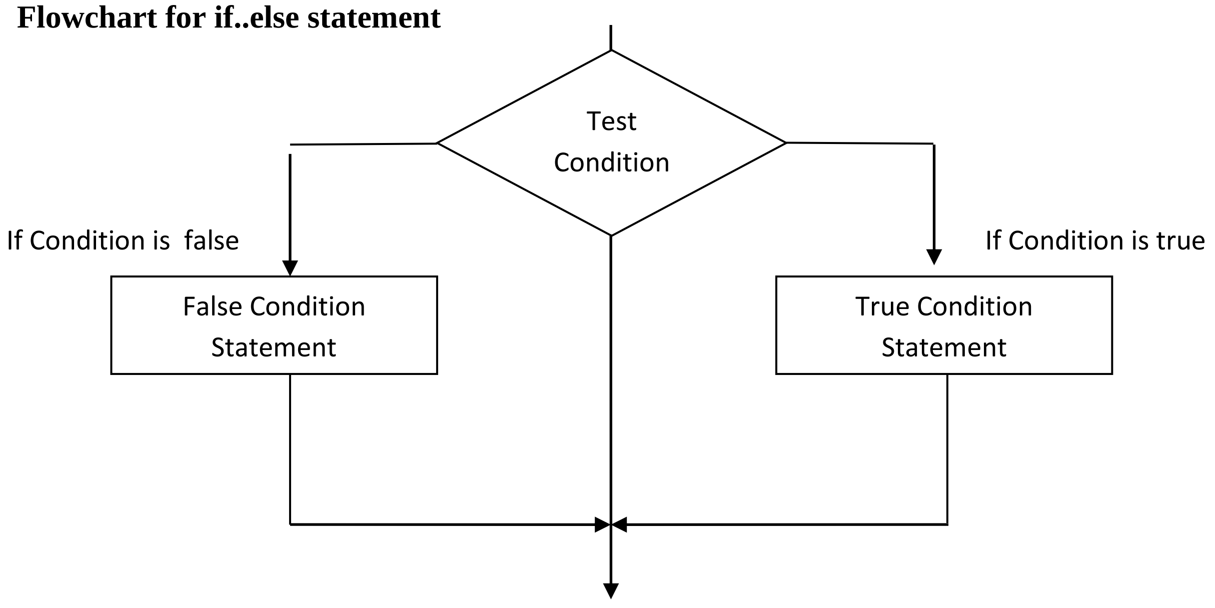
A>B is a test condition, namely 500>200, if it returns TRUE, it will execute the code "print ( )", if it returns FALSE, it will not execute the code " print( )".

2.Alternative(IF....Else Statement):

- A second form of the if statement is **alternative execution**, in which there are **two possibilities** and the condition determines which one gets executed.

Syntax:
if (Condition):
    True Statement Block
else:
    False Statement Block

Flowchart for if..else statement



Example 1:

```
X=4
if x%2 == 0:
print 'x is even'
else:
print 'x is odd'
```

Output:

```
X is even
```

Explanation:

- If the remainder when x is divided by 2 is 0, x is even
- If the condition is false, the second set of statements is executed.(i.e) x is odd

3.Chained condidtional (If...elif...else Statement):

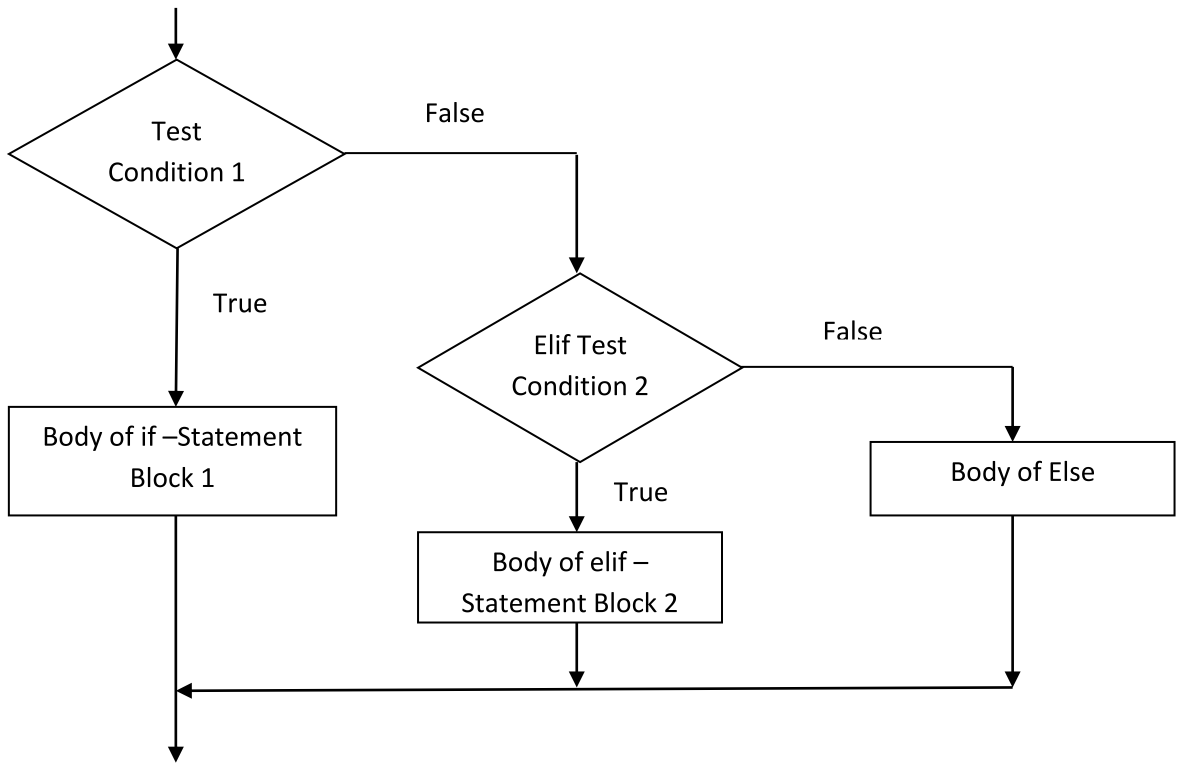
- If there are more than two possibilities and **need more than two branches**.
- One way to express a computation like that is a **chained conditional**.

Syntax:

```
if (Condition 1):
            Statement Block1
   elif(Condition 2):
             Statement Block 2
elif(Condition 3):
             Statement Block 3
   else:
           Statement(s)
```

- elif is an abbreviation of "else if."
- There is no limit onthe number of elif statements.

Flowchart for if..elif...else statement:



Example:
```
X=10
Y=5
if x < y:
print 'x is less than y'
elif x > y:
print 'x is greater than y'
else:
print 'x and y are equal'
```

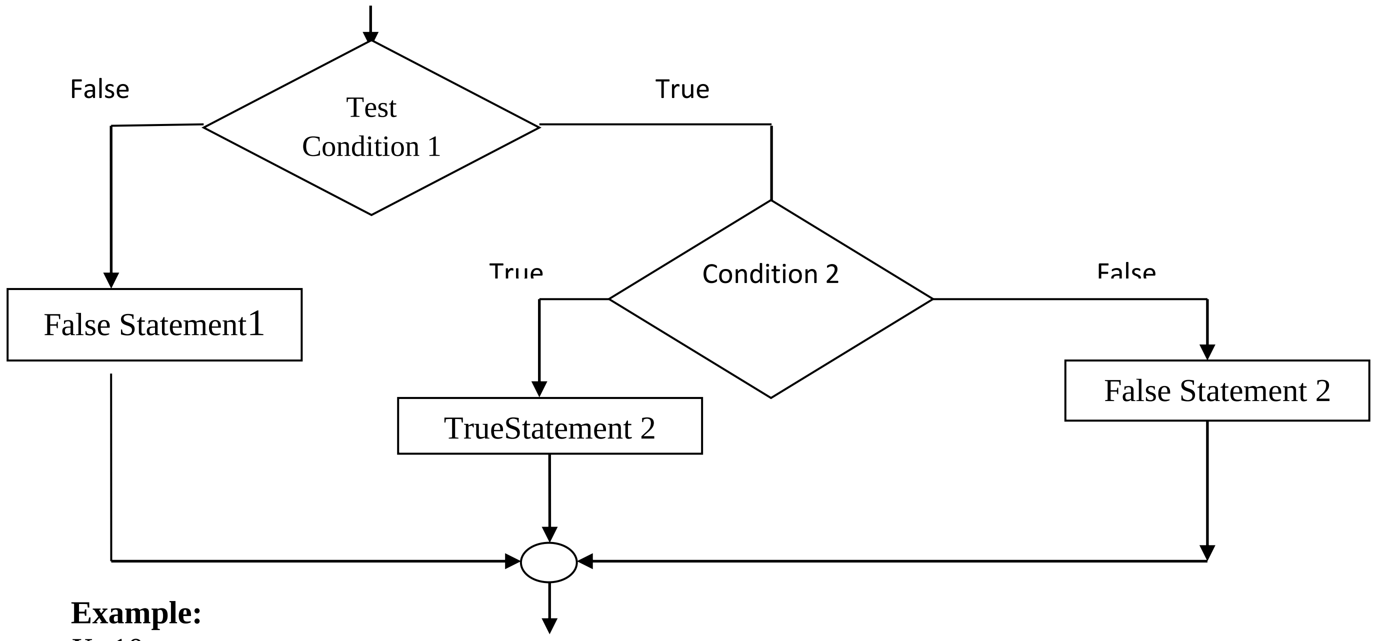
Output:
```
x is greater than y
```

4.Nested if...elif..Else Statement:

- If..elif..else statement can be used **inside another if..elif..else statement**.This is called **nesting. One condition can also be nested within another.**

Syntax:
```
        if (Condition 1):
            if (Condition 2):
                  True Statement 2
    else:
                False Statement 2
    else:
              False  statement 1
```

Flowchart for Nested if..elif...else statement:



Example:
```
X=10
Y=5
if x == y:
print 'x and y are equal'
else:
```

```
if x < y:
print 'x is less than y'
else:
print 'x is greater than y'
```

OUTPUT:

```
X is greater than y.
```

3. Explain the iteration in python with relevant syntax and example.(16m)

- Computers often do repetitive task. There are situation when programmers used to execute a block of code several number of times.
- **Repeated execution of a set of statements is called Iteration/Looping.**

Python Programming Language provide following types of Iterative Statements.

1. **For Loop**
2. **While Loop**
3. **Nested Loop**

Loop Type	Description
For Loop	Executes the sequence of statements multiple times and abbreviates the code that manages the loop
While Loop	Repeats a statement until a give condition is TRUE.It test the While condition before executing the loop body
Nested Loop	You can use one or more loop inside any other While or For Loop

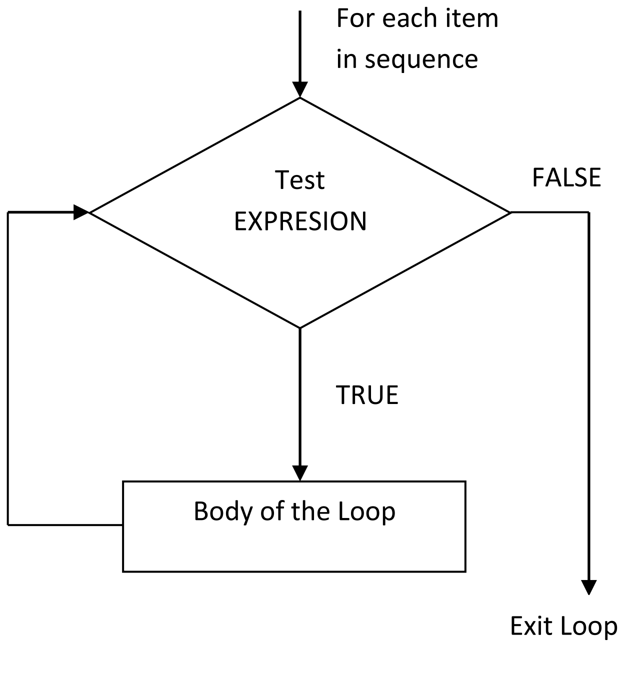
1. For Loop:

- It executes the sequence of statements multiple times and abbreviates the code that manages the loop.
- The for – loop repeats a given block of codes by specified number of times.

Flowchart:

Syntax*:*

```
For <variable> in <sequence>:
<statement 1>
   <statement 2>
   <statement 3>
   .
   .
   .
   <statement n>
```



Example 1 – For Loop for i in '123': Print ("Welcome",i, "Times") **Output:** Welcome 1 Times Welcome 2 Times Welcome 3 Times	**Example 2 – For Loop** Pets_List=['Parrot', 'Rabbit', 'Pigeon', 'Dog'] For mypets in Pets_List: Print(mypets) **Output:** Parrot Rabbit Pigeon Dog
Example 3 – For Loop Message = "PROGRAM" For i in message: Print(i) **Output:** P R O G R A M	**Example 4 – For Loop** Language= ("c", "c++", "Java", "Python") for i in Lanuage: print(i) **Output:** C C++ Java Python

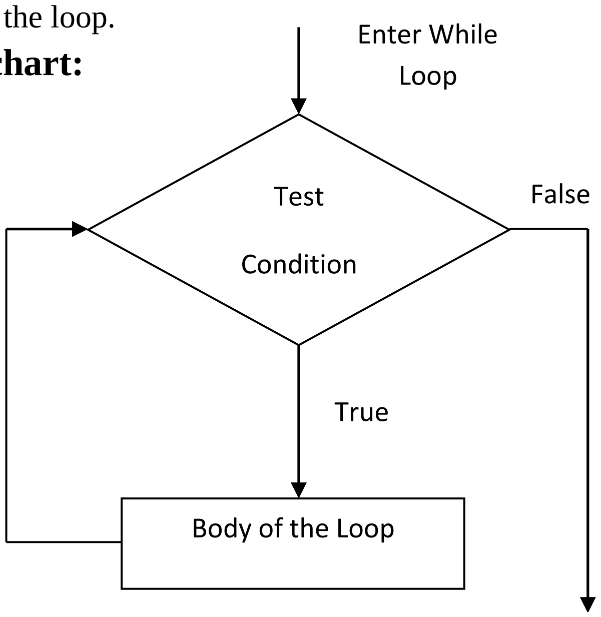
2. While Loop:

- While loop is used, when we need to repeatedly execute a statement or group of statements until the condition is true.
- It tests the condition before executing the loop body so this technique is known as Entry Controlled Loop.

Syntax:
```
While expression:
      Statement(s)
```

- Here, statements may be a single statement or a block of statements.
- The condition may be any expression, the loop iterates while the condition is true, when the condition becomes false, program control passes to the line immediately following the loop.

Flowchart:



Example 1 – While Loop	**Example 2 – While Loop**
#Program to print the number from 0 to 4 Counter = 0 While (counter < 4): Print('countis:',counter) Counter=Counter+1 Print("Exit the loop") **Output:** Count is: 0 Count is: 1 Count is: 2 Count is: 3 Exit the Loop	**#Program to print the number from 0 to 5** a=0 While (a<5): a=a+1 Print (a) **Output:** 0 1 2 3 4

3. Nested Loop:

- Nesting is defined as placing of **one loop inside the body of another loop**.
- It can use **one or more loop inside any another while, for..loopetc**

Syntax:

```
For <variable> in <sequence>:
    For<variable> in <sequence>:
    Statement(s)
Statement(s)
```

4. STATE:

- An algorithm is deterministic automation for accomplishing a goal which, given an initial state, will terminate in a defined end-state.
- When an algorithm is associated with processing information, data is read from an input source or device, written to an output device, and/or stored for further processing.
- Stored data is regarded as part of the internal state of the algorithm.
- The state is stored in a data structure.

5. Explain about unconditional looping statements with relevant example.

- It controls the **flow of program execution** to get desired result.

Python supports following three control statements:

1. **Break.**
2. **Continue.**
3. **Pass.**

Control Statements	Explanation
Break	It terminates the loop statement and transfers execution to the statement immediately following loop.
Continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating
Pass	The pass statement in python is used when a statement is required syntactically but we do not want any command or code to execute.

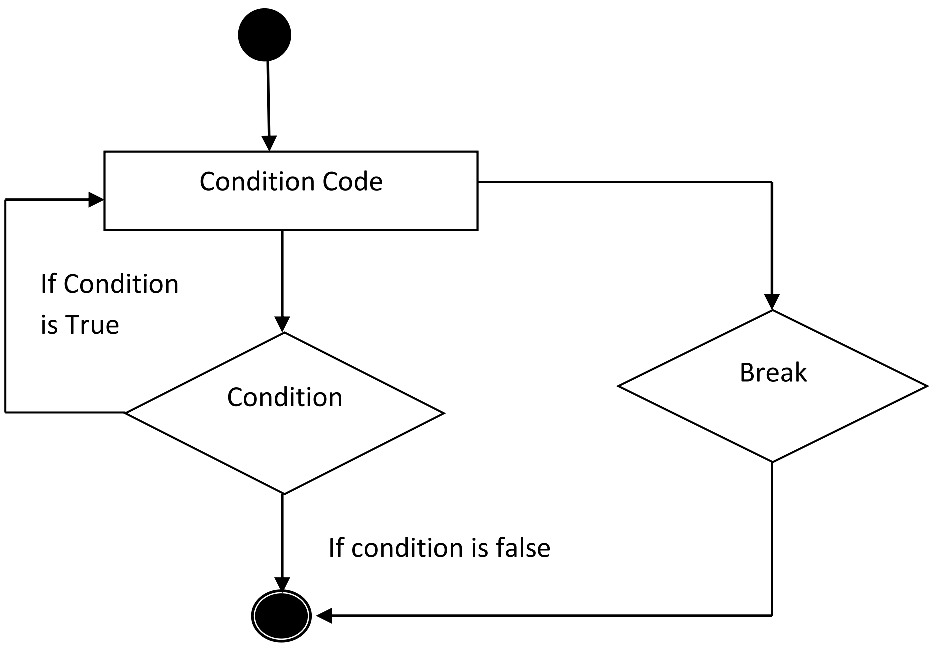
Break Statement:

- It terminates the current loop and **resumes execution at the next statement.**
- The most common use for break is when some external condition is triggered requiring a hasty exit from a loop.
- The **break** statement can be used in both **while and for loops**.
- If you are using nested loops, the break statement **stops the execution of the innermost loop** and start executing the next line of code after the block.

Syntax:

Break

Flow Diagram for Break Statement:



Example 1:	**Example 2:**

for letter in 'Python':
if (letter == 'h'):
break
print(“Current Letter :”, letter)
 	
Output:
Current Letter : P
Current Letter : y
Current Letter : t
	var = 10
	while var> 0:
	print (“Current variable value :”, var)
	var = var -1
	if (var == 5):
	break
	print("Good bye!")
	
	Output:
	Current variable value : 9
	Current variable value : 8
	Current variable value : 7
	Current variable value : 6
	Good bye!

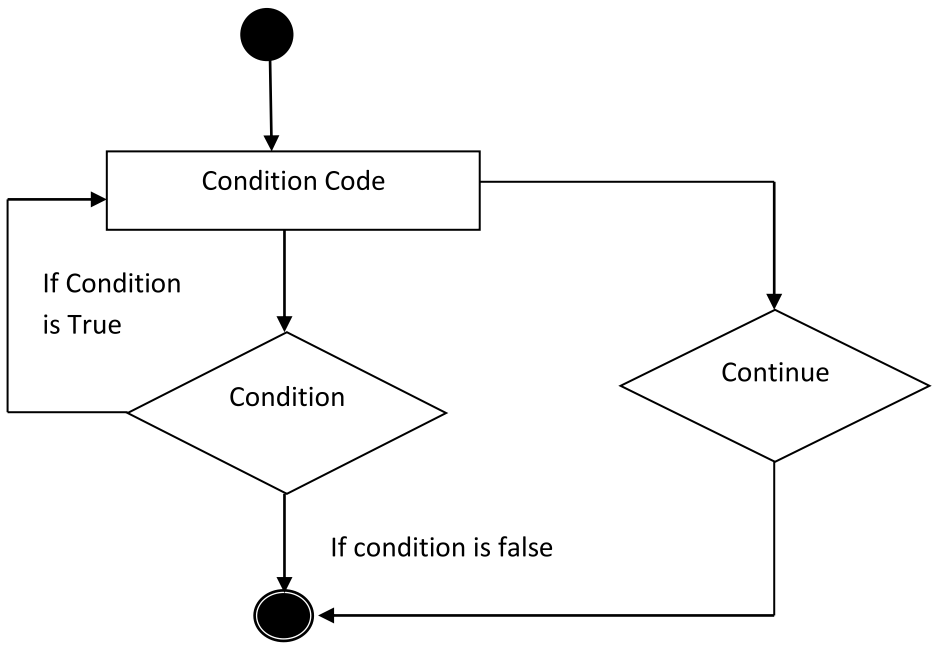
2.Continue Statement:

- It returns the control to the **beginning of the while loop**.
- The continue statement**rejects all the remaining statements in the current iteration** of the loop and movesthe control back to the top of the loop.
- The continue statement can be used in **both while and for loops.**

Syntax:

Continue

Flow Diagram for Break Statement:



Example 1:	**Example 2:**

for letter in 'Python':
if (letter == 'h'):
continue
print(“Current Letter :”, letter)
 	
Output:
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
	var = 10
	while (var> 0):
	var = var -1
	print (“Current variable value :”, var)
	if (var == 5):
	Coninue
	Print("Good bye!")
	
	Output:
	Current variable value : 9
	Current variable value : 8
	Current variable value : 7
	Current variable value : 6
	Current variable value : 4
	Current variable value : 3
	Current variable value : 2
	Current variable value : 1
	Current variable value : 0
	Good bye!

3.Pass Statement:

- It is used when **a statement is required syntactically** but you do not want any command or code to execute.
- The pass statement is a **null operation**; nothing happens when it executes.
- The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example)

Syntax:

Pass

Example 1:

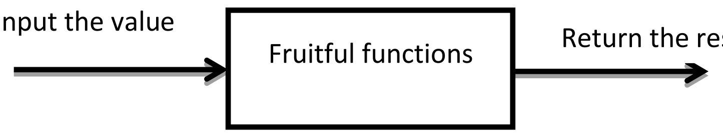
```
forletter in 'Python':
if ( letter == 'h'):
Pass
Print("This is pass block")
print("Current Letter :", letter)
print("Good bye")
```

Output:

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good Bye
```

6. Briefly explain about fruitful functions with example.

- Functions that **return values** are called **fruitful function.**



- **Example:** The square function will take one number as parameter and return the result of squaring that number.

RETURN VALUES:

- The Built-in functions such as the **math functions, POW, etc. produce results.**
- **Calling the function generates a value**, which we usually assign to a variable or use as part of an expression.

- The first example is area, which returns the area of a circle with the given radius:

  def area(radius):
  temp = math.pi * radius**2
  return temp

- We have seen the return statement before, but in a fruitful function the return statement includes an expression.

  def area(radius):
  returnmath.pi * radius**2

PARAMETERS:

A function in python,

- **Takes input data,** called parameters or arguments.
- **Perform computation**
- **Returns the result**.

Def func(param1,param2);
#computations
Return result

- Parameter is the input data that is sent **from one function to another.**

- Parameter is of two types,
  - ✓ **Actual parameter.**
  - ✓ **Formal parameter**.

Actual parameter:

- Parameter is defined in the **function call**.

Formal parameter:

- Parameter is defined as **part of function definition.**
- Actual parameter value is received by formal parameter.

Example:

```
# Function With Actual And Formal Parameters
def cube(x)
return x*x*x
a=input("enter the number:")
b=cube(a)
Print("cube of the given number:",b)
```

Output:

```
Enter the number:2
Cube of the given number:8
```

Parameter passing techniques:

- Call by value
- Call by reference

Call by value:

- A copy of actual parameter is **passed to formal arguments** and any changes made to the formal arguments have **no effect on the actual arguments.**

Call by reference:

- A copy of actual parameter is passed to formal arguments and any changes made to the formal arguments **will affect the actual arguments.**

SCOPE OF THE VARIABLE:

- A variable in the program can be either **local variable or global variable.**
- A global variable is a variable that is declared in the **main program** while a local variable is a variable declared **within the function**.

<u>Example:</u>
```
S=10 # s is the global variable
def f1()
S=55# s is the local variable
Print s
```
Output:
```
55
```

COMPOSITION:

- When a **function is called from within another function**, it is called composition.
- A function that **takes two points, the center of the circle and a point onthe perimeter**, and computes the area of the circle.
- Assume that the center point is stored in the variables xc and yc, and the perimeter point is in xp and yp.
- The first step is to find the radius of the circle, which is the distance between the two points.

  ```
  radius = distance(xc, yc, xp, yp)
  result = area(radius)
  ```

  Encapsulating these steps in a function, we get:

  ```
  defcircle_area(xc, yc, xp, yp):
  radius = distance(xc, yc, xp, yp)
  result = area(radius)
  return result
  ```

  we can make it more concise by composing the function calls:

  ```
  defcircle_area(xc, yc, xp, yp):
  return area(distance(xc, yc, xp, yp))
  ```

RECURSION:

- A function is **recursive if it calls itself and has a termination condition**.
- Termination condition **stops the function from calling itself**.

  <u>Example:</u>
  ```
  def factorial(n):
  If n==0:
  Return 1
  else:
  recurse =factorial(n-1)
  Result =n* recurse
  Return result
  ```

7. Explain in detail about strings with examples.

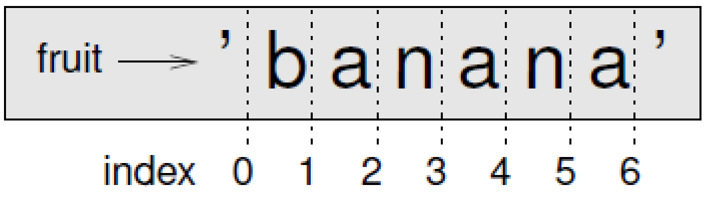
- A string is a **sequence of characters**.(i.e) it can be a letter , a number, or a backslash.
- Python strings are **immutable**, which means they cannot be changed after they are created.

STRING SLICES:

- A **segment of a string** is called a slice. Selecting a slice is similar to selecting a character.

Example:
```
>>> s = 'MontyPython'
>>> print s[0:5]
Monty
>>> print s[6:13]
Python
```

- The operator [n:m] returns the part of the string from the “n-eth” character to the “m-eth” character,including the first but excluding the last.



- If you omit the first index (before the colon), the slice starts at the beginning of the string. If youomit the second index, the slice goes to the end of the string.

Example:
```
>>> fruit = 'banana'
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
```

- If the first indexes is greater than or equal to the second the result is an empty string, represented by two quotation marks.

Example:
```
>>> fruit = 'banana'
>>> fruit[3:3]
```

- An empty string contains no characters and has length 0, but other than that, it is the same as anyother string.

STRINGS ARE IMMUTABLE:

- It is tempting to use the [] operator on the left side of an assignment, with the intention of changing a character in a string.

For example:

```
>>> greeting = 'Hello, world!'
>>>greeting[0] = 'J'

TypeError: object does not support item assignment.
```

- The "object" in this case is the string and the "item" is the character you tried to assign.
- An object is the same thing as a value, but we will refine that definition later. An item is one of the values in a sequence.
- The reason for the error is that strings are immutable, which means you can't change an existing string.
- The best is to create a new string that is a variation on the original:

```
>>> greeting = 'Hello, world!'
>>>new_greeting = 'J' + greeting[1:]
>>> print new_greeting
```

Output:

```
Jello, world!
```

- This example concatenates a new first letter onto a slice of greeting. It has no effect on the original string.

STRING FUNCTIONS AND METHODS:

- A **method** is similar to a function—it takes arguments and returns a value—but the syntax is different.
- For example, the method upper takes a string and returns a new string with all uppercase letters:
- Instead of the function syntax upper(word), it uses the method syntax word.upper().

```
>>>word = 'banana'
>>>new_word = word.upper()
>>> print new_word
BANANA
```

- This form of dot notation specifies the name of the method, upper, and the name of the string to apply the method to, word. The empty parentheses indicate that this method takes no argument.
- A method call is called an **invocation**; in this case, we would say that we are invoking upper on the word.

```
>>>word = 'banana'
>>>index = word.find('a')
>>>print index
   1
```

- The find method is more general than our function; it can find substrings, not just characters:

```
>>>word.find('na')
```

2

- ➢ It can take as a second argument the index where it should start:

>>>**word.find('na', 3)**

4

- ➢ And as a third argument the index where it should stop:

>>>**name = 'bob'**
>>>**name.find('b', 1, 2)**

-1

METHODS:

METHODS	DESCRIPTION
Capitalize()	Capitalizes first for letter of string
is digit()	returns true if string contains only digits and false otherwise.
islower()	returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric()-	returns true id a Unicode string contains only numeric characters and false otherwise
isupper()	returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

STRING MODULE

- The string module provides tools to **manipulate strings**. Some methods available in the standard data structure are not available in the string module (e.g. isalpha).

Example:

```
>>>import string
>>>string.digits
'0123456789'
>>>string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>>string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
>>>string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>>string.hexdigits
'0123456789abcdefABCDEF'
>>>string.octdigits
'01234567'
>>>string.punctuation
'!"@#$%^&*()":><{}[]'
>>>string.whitespace
'\t\n\r\x0b\x0c'
>>>string.Capwords("python")
'PYTHON'
```

8.Why we need list instead of array in Python?Explain in detail about lists with example?

LIST AS ARRAY

- A list in Python is just an **ordered collection of items** which can be of any type.
- By comparison an array is an ordered collection of items of a single type - so in principle a list is more flexible than an array but it is this flexibility that makes things slightly harder when you want to work with a regular structure.
- A list is also a **dynamic mutable type** and this means you can add and delete elements from the list at any time.
- To define a list you simply write a comma separated list of items in square brackets,

  myList=[1,2,3,4,5,6]

- This looks like an array because you can use "slicing" notation to pick out an individual element - **indexes start from 0**.

  For example print myList[2]
- It will display the third element, i.e. the value 3 in this case. Similarly to change the third element you can assign directly to it:

  myList[2]=100

- The slicing notation looks like array indexing but it is a lot more flexible.
  For example myList[2:5]

- It is a sublist from the third element to the fifth i.e. from myList[2] to myList[4].

- The final element specified i.e. [5] is not included in the slice.

- Also notice that you can leave out either of the start and end indexes and they will be assumed to have their maximum possible value.

  For example myList[5:] is the list from List[5] to the end of the list andmyList[:5] is the list up to and not including myList[5] andmyList[:] is the entire list.
- List slicing is more or less the same as string slicing except that you can modify a slice.

  For example:
  myList[0:2]=[0,1] has the same effect as
  myList[0]=0
  myList[1]=1
- List slicing is more or less the same as string slicing except that it can modify a slice.

Basic array operations

- For example, to find the maximum value (forgetting for a moment that there is a built-in max function) you could use:

```
m=0
for e in myList:
if m<e:
m=e
```

- This uses the for..in construct to scan through each item in the list. This is a very useful way to access the elements of an array but it isn't the one that most programmers will be familiar with. In most cases arrays are accessed by index and you can do this in Python:

```
m=0
for i in range(len(myList)):
 if m<myList[i]:
  m=myList[i]
```

- Notice that we are now using range to generate the sequence 0, 1, and so on up to the length of myList.
- You have to admit that there is a lot to be said for the simplicity of the non-index version of the for loop but in many cases the index of the entry is needed. There is the option of using the index method to discover the index of an element but this has its problems.

ILLUSTRATIVE PROGRAMS:

1. **Find the square root of number.**

```
n = int(input("Enter a number"))
howmany = int(input("Enter another number"))
approx = 0.5 * n
for i in range(howmany):
betterapprox = 0.5 *(approx + n/approx)
approx = betterapprox
print("The square root of a number is:", betterapprox)
```

OUTPUT:

```
Enter a number25
Enter another number36
The square root of a number is: 5.0
```

2. **GCD of a given number.**

```
d1 = int(input("Enter a number:"))
d2 = int(input("Enter another number"))
rem = d1 % d2
while rem != 0 :
   d1 = d2
   d2 = rem
rem=d1 % d2
print ("gcd of given numbers is :", d2)
```

OUTPUT:

```
Enter a number:2
Enter another number4
gcd of given numbers is : 2
```

3. **Exponentiation**

```
n = input ("Enter a number : ")
n = int(n)
e = input ("Enter an exponent : ")
e = int(e)
r = n
for i in range (1,e):
r = n * r
print(r)
```

OUTPUT:

```
Enter a number : 2
Enter an exponent : 2
4
```

4. **Sum an array of elements**.

```
a = []
n = int(input("Enter number of elements:"))
for i in range(1, n+1):
b = int(input("Enter element:"))
a.append(b)
a.sort()
print("Largest element is:",a[n-1])
```

OUTPUT:

```
Enter number of elements:5
Enter element:5
Enter element:54
Enter element:24
Enter element:58
Enter element:1
Largest element is: 58
```

5. **Linear search**

```
list = [4,1,2,5,3] #Set up array
search = int(input("Enter search number")) # Ask for a number
for i in range(0,len(list)): # Repeat for each item in list
if search==list[i]: #if item at position i is search time
print(str(search)+"found at position " + str(i)) #Report found
```

OUTPUT:

```
Enter search number 4
4   found at position 0
```

6. **Binary search**

```
defbinary_search(item_list,item):
first = 0
```

```
last = len(item_list)-1
found = False
while( first<=last and not found):
mid = (first + last)//2
ifitem_list[mid] == item :
found = True
else:
if item <item_list[mid]:
last = mid - 1
else:
first = mid + 1
return found
print(binary_search([1,2,3,5,8], 6))
print(binary_search([1,2,3,5,8], 5))
```

OUTPUT:

```
False
True
```

PART-B

1. Explain the concepts Python interpreter and interactive mode in detail.

- A Python is a high-level programming language. Python is an easy to learn, powerful programming language.
- Python is programming language as well as scripting language. Python is also called as Interpreted language.

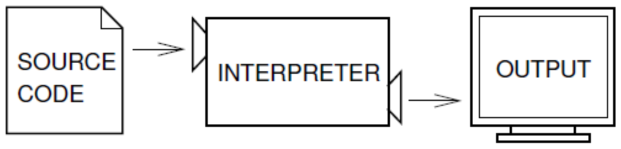
Advantages of High-level language:

- It is much easier to program in a high-level language
- Programs written in a high-level language take less time to write
- They are shorter and easier to read
- High-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications.

Two kinds of programs process high-level languages into low-level languages: interpreters and Compilers.

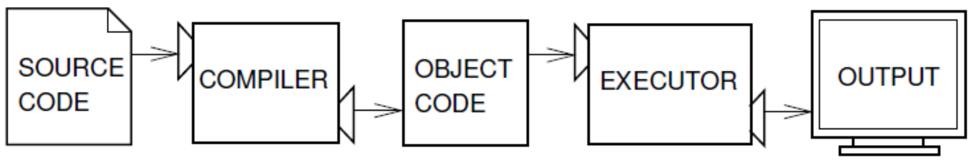
Interpreter:

- Translates program one statement at a time.
- It takes less amount of time to analyze the source code but the overall execution time is slower.
- No intermediate object code is generated, hence are memory efficient.
- Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.
- Programming language like Python, Ruby use interpreters.



Compiler:

- Scans the entire program and translates it as a whole into machine code.
- It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
- Generates intermediate object code which further requires linking, hence requires more memory.
- It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
- Programming language like C, C++ uses compilers.



There are two ways to use the interpreter: interactive mode and script mode.

Interactive mode:

- Working in interactive mode is convenient for testing small pieces of code because you can type and execute them immediately.
- Interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.
- As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.
- Interactive mode is a good way to play around and try variations on syntax.

Example:

```
>>> 1 + 1
Output:2

>>> print (5*7)
Output: 35
```

- >>>, is the prompt the interpreter uses to indicate that it is ready.

Script mode:

- Store code in a file and use the interpreter to execute the contents of the file, which is called a script.
- Python scripts have names that end with .py.
- To execute the script, you have to tell the interpreter the name of the file.
- Save code as a script which can modify and execute it in the future.
- In script mode, however, Python doesn't automatically display results.
- Explicitly from the command line. In this case running Python file and providing the name of the script as an argument.

EX:

```
Sample.py
a=10
b=20
print (a+b)

Output : 3
```

Program:

A computer program is a collection of instructions that performs a specific task when executed by a computer. A computer requires programs to function and typically executes the program's instructions in a central processing unit. A computer program is usually written by a computer programmer in a high-level programming language.

Elements of program:

- **input:** Get data from the keyboard, a file, or some other device.
- **output:** Display data on the screen or send data to a file or other device.
- **math:** Perform basic mathematical operations like addition and multiplication.
- **conditional execution:** Check for certain conditions and execute the appropriate sequence of statements.
- **repetition:** Perform some action repeatedly, usually with some variation.

Debugging:

- Programming errors are called bugs and the process of tracking them down is called debugging.
- Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors.

Syntax Errors:

- Python can only execute a program if the syntax is correct; otherwise, the interpreter displays an error message.
- Syntax refers to the structure of a program and the rules about that structure.

Runtime Errors:

- The second type of error is a runtime error, so called because the error does not appear until after the program has started running.
- These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.
- Runtime errors are rare in the simple programs.

Semantic errors:

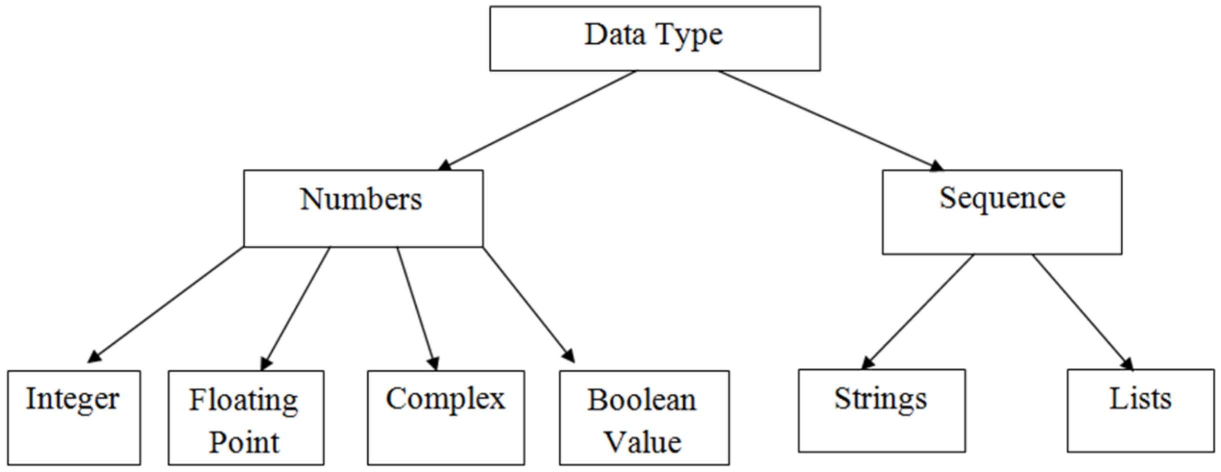
- The third type of error is the semantic error. If there is a semantic error in your program, it will run successfully in the sense that the computer will not generate any error messages, but it will not do the right thing.

Scope of Python:

- Web Development, Internet scripting, Test scripts, Embedded Scripting
- Graphical user Interface [GUI],
- Game Programming, Distributed Programming

2. Explain in detail about values and data types available in python with examples.

- A Value is one of the fundamental things, like a letter or a number that a program manipulates. They are grouped into different data types.
- Data type is a set of values and allowable operations on those values. Data types allow programming languages to organize different kinds of data.



Numbers:

A Numerical literals Containing only the digits (0-9), Optional sign character (+ or -) and a decimal point.

The types of Numerical Data types are:

- Integer (signed integer and long int)
- Floating Point
- Boolean Value
- Complex integer

Syntax:

Variable_name = initial_value

Where,

Variable_name is user defined name

Initial_name is the value to initialized (Integer, Floating Point, Complex, Boolean)

Integer	:	a = 10	Complex Integer	:	Z = 1+2j
Floating Point	:	b = 5.6	Boolean Value	:	Flag = True / False

1.Integers:

- Integers are whole Numbers with no Fractional parts and Decimal Point. They can have Positive, Negative or zero Values.

Data Type	Example
Integer type	-55,-45,-28,-2,0,5,19,56

Integers belong to the type **int**.

Example for integers:

```
>>>1+1
2
>>> a = 4
>>>type(a)
<type ‘int’>
```

2.Floating Point:

- Numbers with fractions or decimal points are called floating point numbers. A floating point number will consists of signs (+,-) sequence of decimal digits.

Data Type	Example
Floating Point numbers	-1.25,-1.0,0.0,0.5,12.6,15.2

Floating Point belongs to the type **Float.**

Example for Float:

```
>>> c=2.5
>>>type(c)
<type ‘Float’
```

3.Boolean Value:

- Boolean is a data type named after George Boole. Boolean often called bools, which are either TRUE or FALSE condition.It is the Simplest Built-in type in python.

- A Boolean variable can take only two values True (1) and False (0)

Data Type	Example
Boolean Value	True / false

- Boolean Value belongs to the type **Bool.**

Example for Float:

```
>>> c=True
>>>type(c)
<type ‘bool’>
```

- A Boolean true value is always considered to be a Non-zero, Non-null and Non-empty value.

Sequence:

A sequence is an ordered collection of items, indexed by positive integers. It is a combination of mutable and non mutable data types.

The types of sequence data types are

- String
- lists

1.Strings:

- Strings are literal or it is a sequence of characters which may consists of letters, number, special symbols or a combination of these types represented within the pair of double or single quotation marks.

Creating string is as Simple as assigning a value to a variable.

```
>>> a = “Hello”
>>>type(a)
<type ‘str’>
```

Operations on Strings:

1. + Operator
2. * Operator

- The + operator concatenate strings

```
>>> ‘Horse’+ ‘and’+ ‘dog’
Output : ‘ Horse and dog’
```

- The * operator creates a multiple Concatenated Copies of a string

```
>>> A * 5
```

Output: AAAAA

2. Lists:

- Lists in Python are used to store a list of values.
- It is an ordered sequence of values which may consists of any type of data (Intger, Float,String etc)
- Values in the list are called element/items.
- They are Mutable and indexed ordered.

To create a list, we define a variable to contain an ordered series of items separated by a comma. A square bracket is used to enclose the items.

Syntax:

To create an empty list : **My_list=[ ]**

To create a list of items : **My_list=[item1, item2,item3….]**

Example:

#List with Integer

Num_list=[0,5,4,7,8]

#List with String

String_list=[‘cat’, ‘rat’, ‘dog’, ‘lion’]

3. Define variable. Explain how to declare variable and explain about the scope of variable.

- A variable is basically a name that represent some value, Variable are reserved memory location to store values.
- Every value in Python has a datatype.Diffferent data types in python are Number, List, Tuple, Strings etc.

Syntax : **Variable_name = Value**

<u>Rules for Naming a variable:</u>

- A variable name can contain both **Upper case** (A-Z) and **Lower case** characters(a-z) and **Underscore character** (_).
- Numbers are allowed, but they should not be used at the beginning of the variable name. Eg: **1firstname** is not valid, **firstname1** is valid.
- No Special symbol other than **underscore** is allowed.
- There are some Keywords that cannot be used as a variable name because they already have pre-assigned meaning , some of the reserved word are ***Print,Input,if,while*** etc

<u>Assigning Values to the variable:</u>

The declaration happens automatically while assigning values to a variable. The equal sign (=) is used to assign values to variables.

Eg: Name = “Dog”

Int_num= 500

Float_num=45.5

List_num= [“Dog”, 50,25, 0.5]

Re-declare a Variable:

In this we can re-declare the variable even after you have declared it once.

```
#Declare a variable and Initialize it:
C = 0
Print c
#re-declare the variable
C= “Welcome”
Print c
```

- Here variable initialized to c=0.
- If we re-assign the assign the Variable c to value “ welcome”

Output:

```
0
Welcome
```

4. Write short on Expressions.

An Expression is combination of values, variables and operators.

If we type an expression on the command line, the interpreter evaluates it and displays the results.

```
>>>2+2

4
```

The evaluation of an expression produces a value. Some legal expression are a follows:

```
>>> a= 15

>>> a

Output: 15

>>> a + 15

Output: 30
```

When we type an expression at the prompt, the interpreter evaluates it, which means that it finds the value of the expression. In the above example " a" has a value 15 and "a+15" has a value 15.

Evaluating an expression is not the same as printing a value

```
>>> Line = "God is Great"

>>>Line

Output: "God is Great"

>>> Print  Line

Output: God is Great.
```

5. Explain about Statements.

- A statement is an **instruction** that python interpreter can execute.
- There are two kinds of statements.
  - **Print statement**
  - **Assignment statement**.
- When a statement is typed in interactive mode, the interpreter executes it and displays the result.
- A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script

```
print 1
```

```
x = 2
print x
```

output:

```
1
2
```

The assignment statement produces no output.

6. Explain about the concept of tuple assignment.

- ➢ The tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.
- ➢ It is useful to swap the value of two variables.

Example: To swap a, b

```
Temp=a
a=b
b=temp
```

Tuple assignment solves this problem neatly

(a,b)=(b,a)

- ➢ The number of variables on the left and the number of values on the right have to be the same:

```
>>> a, b = 1, 2, 3
```

ValueError: too many values to unpack.

7. What are the types of operators supported by python language? List the operators and explain them.

Operator:

- An operator is a special symbol that represents a simple computation like addition, subtraction,…
- The values (or) the variables are applied on the operator is called as operands.

Types of Operator:

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators

- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Assume variable a =10,b=20

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 – that divides two numbers and chops off the fraction part.	9//2 = 4

Python Comparison or relational Operators:

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators. Assume a=15, b=23

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) False(0)
!=	If values of two operands are not equal, then condition becomes 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

Python Assignment Operators:
Assume variable a holds 10 and variable b holds 20, then

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 / 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

Python Bitwise Operators:

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

----

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

There are following Bitwise operators supported by Python language

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Identity Operators

Identity operators compare the memory locations of two objects.

	Description	**Example**
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

8. Briefly explain about the concept of precedence of operators.

- The list below will show which operator has more precedence over the operators.
- The first operator in the list will run before the second operator below.

OPERATORS	**MEANING**
()	Parentheses
**	Exponent
*, /, //, %	Multiplication, Division, Floor division, Modulus
+,-	Addition,subtraction
<<,>>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==,!=,>,>=,<,<=,is, is not,in not, not in	Comparisons ,Identity, Membership operators

not	Logical NOT
and	Logical AND
or	Logical OR

Operator precedence examples:

Exponent and Multiplication:

- Exponent will always run before the multiplication equation.

Example:

```
#Exponent and multiplication
#Exponent runs first
>>>2**4+2
>>>16+2
18
```

Exponent And Division:

- Exponent will always run before a division equation.

Example:

```
#Exponent and Division
#Exponent runs first
>>4/2**6
>>4/64
0.0625
```

Mutiplication and Division:

- Python will run the equation from left to right since multiplication and division has same precedence.

Example:

```
#Multiplication and Division
#In this case division ran first the multiplied by 3.
>>>5/4*3
3.75
#In this case 3 is multiplied by 4 then divide by 5
>>>3*4/5
2.4
```

Multiplication and addition:

- Multiplication will run before an addition equation since multiplication has more precedence over addition.

Example:

```
#Multiplication and addition
>>>2=4*4
18
```

Addition and subtraction:

- The equation will run left to right since addition and subtraction are on the same level.

Example:

```
#Addition and subtraction
>>>2=3-5+8-4+2-9
-3
```

EXAMPLE PROBLEMS:

1. **X=(15=6)-10*4**
   Print (x)
   In the expression to evaluate the value of x, the brackets have highest of all precedence.
   So this is evaluated before anything else, then * is done and lastly the subtraction.
   15+6 is 21
   1084 is 40
   21-40 is -19
   Ans: -19

2. **X=17/2%2*3**3**
   Print(x)
   In the expression to evaluate the value of x, the exponentiation is done first and then division operator followed by modulo and multiplication.
   3**3 is 27
   17/2 is 8.5
   8.5%2 is 0.5
   0.5*27 is 13.5
   Ans: 13.5

9. Illustrate the concept of comments.

- Comments statements are used to provide a summary of the code in plain English to help future developer for better understand of the code.
- There are two types of comments
    - Single line comment.
    - Multiline comment.

Single Line comment:

- A single line comment starts with the number sign (#) character:

Example:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

In this case, the comment appears on a line by itself. You can also put comments at the end of a line:

```
percentage = (minute * 100) / 60 # percentage of an hour.
```

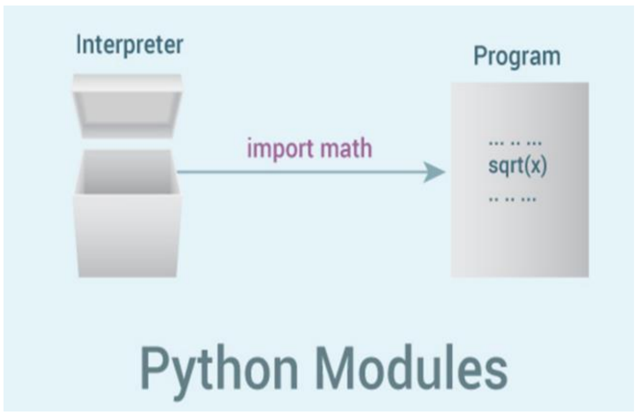
Multiline comment:

- Multiple lines can be created by repeating the number sign several times:

```
#This is a comment
#second line
X=4
```

10. Explain about the features of module.

- Modules refer to a file containing Python statements and definitions.



module and its

- We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

- We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as example.py

# Python Module example

```
def add(a, b):
  """This program adds two
  numbers and return the result"""
  result = a + b
  return result
```

Here, we have defined a function add ( ) inside a module named example. The function takes in two numbers and returns their sum.

How to import modules in Python?

- We can import the definitions inside a module to another module or the interactive interpreter in Python.
- We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

```
>>> import example
```

- This does not enter the names of the functions defined in example directly in the current symbol table. It only enters the module name example there.
- Using the module name we can access the function using dot (.) operation. For example:

```
>>> example.add(4,5.5)
9.
```

<u>Python import statement:</u>

We can import a module using import statement and access the definitions inside it using the dot operator as described above. Here is an example.

```
# Import statement example
# To import standard module math
import math
print("The value of pi is", math.pi)
```

When you run the program, the output will be:

```
The value of pi is 3.141592653589793
```

Python from...import statement:

We can import specific names form a module without importing the module as a whole. Here is an example.

```
# import only pi from math module

from math import pi

print("The value of pi is", pi)

The value of pi is 3.141592653589793
```

The from...import * Statement:

It is also possible to import all names from a module into the current namespace by using the following import statement.

from modname import *

This provides an easy way to import all the items from a module into the current namespace.

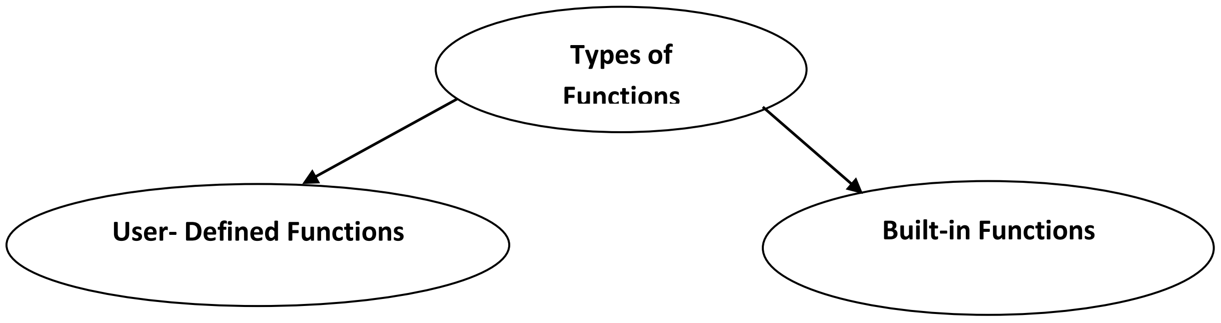
11. Explain about the features of function.

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Types of Functions:

Basically, we can divide functions into the following two types:

1. **Built-in functions -** Functions that are built into Python.
2. **User-defined functions -** Functions defined by the users themselves.



Function calls

- A function is a named sequence of statements that performs a computation.
- When you define a function, you specify the name and the sequence of statements.
- Later,you can “call” the function by name. We have already seen one example of a function call:

```
>>> type(32)
<type 'int'>
```

- The name of the function is type.
- The expression in parentheses is called the argument of the function.
- The result, for this function, is the type of the argument. It is common to say that a function “takes” an argument and “returns” a result. The result is called the return value.

Type conversion functions

- Python provides built-in functions that convert values from one type to another.

The int function takes any value and converts it to an integer, if it can, or complains otherwise:

```
>>> int('32')
32

>>> int('Hello')
ValueError: invalid literal for int(): Hello
```

The int can convert floating-point values to integers, but it doesn’t round off; it chops off the fraction part:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Float converts integers and strings to floating-point numbers:

```
 >>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Finally, str converts its argument to a string:

```
>>> str(32)
```

```
'32'
>>> str(3.14159)
'3.14159'
```

Math functions

- Python has a math module that provides most of the familiar mathematical functions.
- A module is a file that contains a collection of related functions.

Before we can use the module, we have to import it:

```
>>> import math
```

- This statement creates a module object named math. If you print the module object, you get some information about it:

```
>>> print math
<module 'math' from '/usr/lib/python2.5/lib-dynload/math.so'>
```

- The module object contains the functions and variables defined in the module.
- To access one of the functions, you have to specify the name of the module and the name of the function, separated by a dot (also known as a period). This format is called dot notation.

  Example:

  ```
  >>> ratio = signal_power / noise_power
  >>> decibels = 10 * math.log10(ratio)
  ```

The example computes the logarithm base 10 of the signal-to-noise ratio.
The math module also provides a function called log that computes logarithms base e.

12. Explain about flow of execution.

- The orders in which statements are executed are called 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.
- A function call is like a detour in the flow of execution.
- Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

- That sounds simple enough, until you remember that one function can call another. While in the middle of one function, the program might have to execute the statements in another function.

Example:

```
Def pow(5,2)
Y=b**p
Return y

def square(x)
a=pow(x,2)
return a

X=5
Result=square(x)
Print(result)
```

Output:

25

13. Explain the concept of parameters and arguments

- The arguments are assigned to variables called **parameters**.

  EXAMPLE:

  ```
  def print_twice(Flower):
  Print flower
  Print flower
  ```

- This function assigns the argument to a parameter named flower.
- When the function is called, it prints the value of the parameter twice.

  EXAMPLE:

  ```
  >>>print_twice('world')
  World
  World
  >>>print_twice(56)
  56
  56
  >>>print_twice(math.pi)
  ```

```
3.14
3.14
```

The same rule of composition that apply to built in functions also apply to user defined functions, so we can use any kind of expression as an argument for print_twice:

```
>>>print_twice('spam'*4)
Spam spam spam spam
Spam spam spam spam
>>>print_twice(math.cos(math.pi))
-1.0
-1.0
```

- The argument is evaluated before the function is called,so in the example the expression 'spam'*4 and math.cos (math.pi)are only evaluated once.
- The variable can also be used as an argument:

EXAMPLE:

```
>>>book='python'
>>>print_twice(book)
Python
python
```

UNIT-V

FILES, MODULES, PACKAGES

PART-A

1. What is file? List out its types.

- FILES refer to a location with Filename that stores information.
- File must be opened to read or Write. Moreover it must be closed after read/write operations to avoid data damages.
- Almost all the information Stored in a computer must be in a file.

  Types:
  - Data files
  - Text files
  - Program files.

2. List out the types of file operations.

- Open the file.
- Read or write
- Close the file

3. Give the syntax and example of file open.

Syntax:
File_VariableName=Open ("Filename", 'mode')

Example:
f = open("text.txt") #open file in current directory
f = open ("text.txt", 'w') #write in text mode
f = open("text.txt", 'r+b') # read in binary mode

4. What is error? List out the basic categories of error.

- Errors or mistakes in a program are often referred to as bugs.
- The process of finding and eliminating error is called debugging.
- Errors are classified into three groups.

  1. Syntax errors.
  2. Runtime errors.
  3. Logical errors.

5. List out the common python syntax errors.

- Leaving out a keyword.
- Leaving a symbols such as colon, comma or brackets.
- Misspelling a keyword.

6. Give some example of runtime errors.

Division by zero.
Trying to access a file which doesn't exist.
Accessing a list element or dictionary values which doesn't exist.

7. Give some examples of logical errors.

- ➢ Using integer division instead of floating point division.
- ➢ Wrong operator precedence.
- ➢ Mistakes in Boolean expression.
- ➢ Indenting a block to the wrong level.
- ➢ Error in Boolean Expression.

8. What is exception in python?

- ➢ Exception is an event, which occurs during the execution of a program.
- ➢ Exception disrupts the normal flow of the programs instructions.
- ➢ Exception is a python object that represents an error.

9. How to handle exception in python.

- ➢ Python provides two very important features to handle any unexpected error in the python programs and to add debugging capabilities in item.
  - Exception handling.
  - Assertions.

10. List any four standard exceptions in python.

Standard Exceptions:

1. **Floating Point Error**-Raised when floating point calculation fails.
2. **Import error**-Raised when an import statement fails.
3. **Syntax error**-Raised when there is an error in python syntax.
4. **Indentation error**-Raised when indentation is not specified properly

11. Give the syntax for raise exception.

12. We can raise exceptions in several ways by using the raise statement.

Syntax:

Raise(Exception(,args(,traceback)))

Example:

```
 def get_age():
age = int(input("Please enter your age: "))
 if age < 0:
 my_error = ValueError("{0} is not a valid age".format(age))
raise my_error
```

```
    return age
```

Output:

```
raise ValueError("{0} is not a valid age".format(age))
```

12. What are the advantages of exception handling?

- It separates normal code from code that handles errors.
- Exception can easily be passed along functions in the stack until they reach a function.

13. Differentiate module and package in python.

Module	Package
1. A module is a single file that are imported under one import. 2. **Example:** import my_module	1. A package is a collection of modules in directories that give a package hierarchy. 2. **Example:** from pots import pots

PART-B

1. Discuss with suitable example (i) Read a File (ii) Write a file (iii) Close a file (or)

Explain the file concepts in detail with example.

FILES

- FILES refer to a location with Filename that stores information. Like a book a File must be opened to read or Write. Moreover it must be closed after read/write operations to avoid data damages.
- Almost all the information Stored in a computer must be in a file.
- A File is a named location on disk to store related information.

CREATE A TEXT FILE:

- Text file in Python, Lets Create a new text file:

Example:

```
file = open("sample1.txt", "w")
file.write("Hello World")
file.write("God is  great")
file.write("Think Positive")
file.close()
```

- If you Open the text file, Python interpreter will add the text into the file.

Output:

```
Hello World
God is great
Think Positive.
```

OPENING A FILE:

- Python has a built-in function Open( ) to open a file.
- The contents of a file can be read by opening the file in read mode. There are various modes to open a file. They are listed below:

Syntax:

File_VariableName=Open (“Filename”, ‘mode’)

Where

- “File_VariableName” is the variable to add the file object.
- “Mode” tells the interpreter and developer which way the file will be used.

Example:

```
f = open(“text.txt”) #open file in current directory
f = open (“text.txt”, ‘w’) #write in text mode
f = open(“text.txt”, ‘r+b’) # read in binary mode
```

READING AND WRITING FILES:

- In General,there are two methods to manipulate the files, They are given below:
  - Write( )
  - Read ()

Write ( ) Method:

- This method writes any string to an open file.

Syntax:

File_Variablename.Write(string);

Example

```
#open a file
f = open(“sample.txt”, ‘w’)
f.write(“ God is Great.\n God is always with me!\n”);
#close the opened file
f.close( )
```

In the above example the methods would create sample.txt file and would write the given content in that file, finally it would close that file. (\n) adds a newline character.

Sample.txt
God is Great God is always with me!

Read ( ) Method:

- This Method read a string from a file. It is important to note that Python strings can have binary data apart from text data.

Syntax

File_variablename.read(count);

Example:

```
#opens a file
f = open (“sample.txt”, “r+”)
str=f.read(10);
print(“Read the string:”,str)
f.close()
Output: Read the string: God is great.
```

2. Explain about the concepts of format operator.

- Format operator is used to print the output in a desired format.
- The % operator is the format operator.
- It is also called as interpolation operator.
- The % operator is also used for modulus.
- The usage varies according to the operands of the operator.

Syntax:

<format operator>% values.

Example:

```
>>>camels=42
>>>’%d’% camels
‘42’
>>>camels=42
>>>’I have spotted %d camels’% camels
‘I have spotted 42 camels’
>>>In %d years I have spotted %f %s’(3,0.1,’camels’)
‘In 3 years I have spotted 0.1 camels’
```

3. Illustrate about the concept of errors and exception. Explain the types of error.

- Errors or mistakes in a program are often referred to as bugs.
- The process of finding and eliminating error is called debugging.
- Errors are classified into three groups.

4. Syntax errors.
5. Runtime errors.
6. Logical errors.

SYNTAX ERRORS:

- Python can only execute a program, if the syntax is correct, otherwise interpreter displays error message.

- Python will find these kinds of error when it tries to parse a program and exit with an error message without running anything.
- Syntax errors are mistakes in the use of the python language or grammar mistakes.

Common python syntax errors include:

- Leaving out a keyword.
- Leaving a symbols such as colon, comma or brackets.
- Misspelling a keyword
- Incorrect indentation.

RUNTIME ERROR:

- The second type of error is a runtime error. It does not appear until after the program has started running.
- The program may exit unexpectedly during execution.
- Runtime error was not detected when the program was parsed.

Common python runtime errors include:

- Division by zero.
  Ex:
  ```
  >>> print(55/0)
  ZeroDivisionError: integer division or modulo by zero.
  ```
- Trying to access a file which doesn't exist.
- Accessing a list element or dictionary values which doesn't exist.
  Ex:
  ```
  >>> a = []
  >>> print(a[5])
  IndexError: list index out of range
  ```
- Using an identifier which has not be defined.

LOGICAL ERRORS:

- Logical errors are more difficult to fix.
- The program runs without crashing, but produces an incorrect result.
- The error is caused by a mistake in the programs logic.
- Error message will not be appeared, because no syntax or runtime error has occurred.

Common python logical errors include:

- Using integer division instead of floating point division.
- Wrong operator precedence.
- Mistakes in Boolean expression.
- Indenting a block to the wrong level.
- Error in Boolean Expression.

4. Describe in detail how exceptions are handled in python. Give relevant examples.

- Exception is an event, which occurs during the execution of a program.
- Exception disrupts the normal flow of the programs instructions.
- Exception is a python object that represents an error.

Standard Exceptions:

1. **Floating Point Error**-Raised when floating point calculation fails.
2. **Import error**-Raised when an import statement fails.
3. **Syntax error**-Raised when there is an error in python syntax.
4. **Indentation error**-Raised when indentation is not specified properly.
5. **EOF error**-Raised when there is no input until the end of while.
6. **IOError-** open()function when trying to open a file that does not exist.
7. **ArithmeticError-**Base class for all errors that occur for numeric calculation.
8. **AssertionError**-Raised in case of failure of the assert statement.
9. **RuntimeError-**Raised when a generated error does not fall into any category.
10. **ValueError-**Raised when the built in functions for a data type has the valid type of arguments,but the have invalid values specified.

Syntax:

```
Try:
Operator
……
Except exception:
If there is exception, execute this block.
Except Exception
If there is exception, execute this block.
…..
else
If there is no exception, then execute this block.
```

Example:

It tries to open a file where do not have write permission, so it raises exception.

```
Try:
fh=open("testfile",'r')
fh.write("This is my test file for exception handling!")
Except IOerror
Print("error:can't find file or read data")
else:
Print("written content in the file successfully")
```

Output:

```
Error: can't find file or read data.
```

Raising An Exceptions:

- We can raise exceptions in several ways by using the raise statement.

Syntax:
Raise(Exception(,args(,traceback)))

Example:

```
 def get_age():
 age = int(input("Please enter your age: "))
  if age < 0:

 # Create a new instance of an exception
  my_error = ValueError("{0} is not a valid age".format(age))
 raise my_error
 return age
```

Output:

```
raise ValueError("{0} is not a valid age".format(age))
```

5. Illustrate the important of modules with examples.

- Modules refer to a file containing Python statements and definitions.
- Modules to break down large programs into small manageable and organized files.

# Python Module example

```
def add(a, b):
   """This program adds two
   numbers and return the result"""
   result = a + b
   return result
```

How to import modules in Python?

- We can import the definitions inside a module to another module or the interactive interpreter in Python.
- >>> import example
- Using the module name we can access the function using dot (.) operation.
- For example:
  >>> **example.add(4,5.5)**

<u>Python import statement:</u>

We can import a module using import statement and access the definitions inside it using the dot operator.

```
# Import statement example
import math
print("The value of pi is", math.pi)

OUTPUT:
 The value of pi is 3.14
```

Python from...import statement:

- Import specific names form a module without importing the module as a whole. Here is an example.

```
# import only pi from math module

from math import pi

print("The value of pi is", pi)

The value of pi is 3.14
```

The from...import * Statement:

- It is also possible to import all names from a module into the current namespace by using the following import statement.

from modname import *

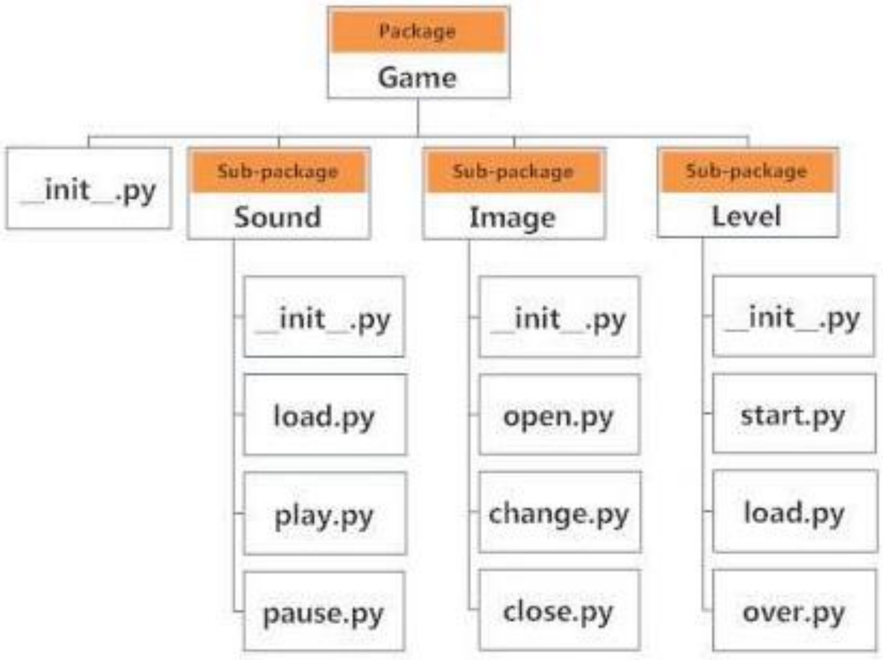
This provides an easy way to import all the items from a module into the current namespace.

6. Discuss Python Package in detail.

- A package is a collection of modules in directories that give a package hierarchy.
- The package gives the hierarchical file director structure of the python application environment that consists of modules and package.
- A package is a directory which contains a special file called __init__.py

Example:

- If the project team has decided to maintain all the modules in “MyProject” directory, a folder in the name of “MyProject” is creacted and the file __init__.py is placed in that folder.
- The package has two normal programs and two subpackages.



Importing module from a package:

The modules in package can be used by importing then in python program.

Syntax:

Import<packageName>.{<subpackageName>.}ModuleName>

- The {} braces specify that there can be any number of subpackages.

Example:

```
Import MyProject.subproject1.prog1_1

MyProject.subProject1.Prog1_1.start()
```