1. Explain the basic data types available in python with examples.

Following are the standard or built-in data type of Python:

Numeric, Sequence Type, **Boolean**, **Set**, and **Dictionary**

Numeric:

- In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.
- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Sequence:

- Sequence data type is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion.
- In Python, **Strings** are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.
- **Lists** are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.
- **Tuple** is also an ordered collection of Python objects. The only difference between type and list is that tuples are immutable i.e. tuples cannot be modified after it is created

Boolean:

- The truth values of an expression is stored as a python data type called bool. There are only two such values in this data type. True and False.

Set:

- **Set** is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Dictionary:

- **Dictionary** in Python is an unordered collection of data values, used to store data values like a **map**, which unlike other Data Types that hold only single value as an element.

Example:

```
>>> a=5
>>> type(a)
<class 'int'>
>>> a=5.0
>>> type(a)
<class 'float'>
>>> a=2+5j
>>> type(a)
<class 'complex'>
>>> a="python"
>>> type(a)
<class 'str'>
>>> a=[1,2,3]
>>> type(a)
<class 'list'>
>>> a=(1,2,3)
>>> type(a)
<class 'tuple'>
>>> a=True
>>> type(a)
<class 'bool'>
>>> a={1,2,3}
>>> type(a)
<class 'set'>
>>> a={1:2}
>>> type(a)
<class 'dict'>
>>> D
```

2. Explain the Identifiers, Keywords, Statements, Expressions, and Variables in Python programming language with examples.

Ans- **Identifiers -** A Python identifier is a name used to identify a variable, function, class, module or other
object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
Python does not allow punctuation characters such as @, $, and % within identifiers.
Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.
Here are naming conventions for Python identifiers-
 Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
 Starting an identifier with a single leading underscore indicates that the identifier is private.

 Starting an identifier with two leading underscores indicates a strong private identifier.
 If the identifier also ends with two trailing underscores, the identifier is a language defined special name.

Keywords- These are reserved words and we cannot use them as constants or variables or any other identifier names. All the Python keywords contain lowercase letters only.
Eg- and, as, finally, if, else, def, etc…..

Expression- Expressions are representations of value. They are different from statement in the fact that statements do something while expressions are representation of value. For example, any string is also an expression since it represents the value of the string as well.
Python expressions only contain identifiers, literals, and operators.

Variables- Variables are nothing but reserved memory locations to store values. It means that when we create a variable, we reserve some space in the memory.

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Eg-
```
x = 5
y = "John"
print(x)
print(y)
```

Output- 5
Jhon

3. Describe arithmetic operators, Assignment operators , comparison operators, Logical operators ,and Bitwise operators in detail with example:-

Ans: operators are special symbols that designated that

Some sort of computation should be performed.

- **Arithmetic operators:**

```
X=15
y = 4
print('x + y =',x+y)
# Output: x + y = 19
print('x - y =',x-y)
# Output: x - y = 11
print('x * y =',x*y)
# Output: x * y = 60
print('x / y =',x/y)
# Output: x / y = 3.7
print('x // y =',x//y)
# Output: x // y = 3
```

```
print('x ** y =',x**y)
# Output: x ** y = 50625
```

- **Assignment Operator**

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

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable *a* on the left.

- **Comparison Operator**

```
x = 10
y = 12
print('x > y is',x>y)
# Output: x > y is False
print('x < y is',x<y)
# Output: x < y is True
print('x == y is',x==y)
# Output: x == y is False
print('x != y is',x!=y)
# Output: x != y is True
```

- **Logical Operator**

```
x = True
y = False
print('x and y is',x and y)
print('x or y is',x or y)
print('not x is',not x)
```

- **Bitwise Operator**

  Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

  For example, 2 is 10 in binary and 7 is 111.

4. Describe the **is** and **is not** operators and type() function. Also, discuss why python is called as dynamic and strongly typed language.

-‘is’ operator – Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.

Example:

```
x = 5
```

```
if (type(x) is int):
   print("true")
else:
   print("false")
```

Output: true

-'is not' operator – Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example:

```
x = 5.2
```

-type()

type() method returns class type of the argument(object) passed as parameter. type() function is mostly used for debugging purposes.

example:

```
v=1
print (type(v))
```

output: int

Python is strongly typed as the interpreter keeps track of all variables types. It's also very dynamic as it rarely uses what it knows to limit variable usage

5. Illustrate the different types of control flow statements available in python.

Control flow statement defines the flow of the execution of the program.

Control flow statements available in python are **conditional statements** (if,elif and nested-if), **Looping statements**(for,while and nested loop),**looping condition statements(**break,continue,pass**).**

Conditional Statements

Conditional statements allow us to execute particular block of code depending upon condition evaluates to True or False.

1.If statement

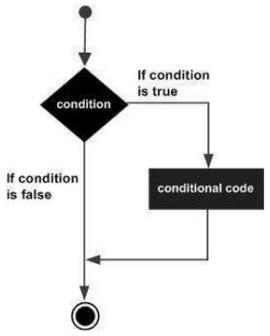
Syntax-

if expression:

  statement(s)

If statement is used to check a condition.If the condition is True, then the block of statement inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of block is executed.

Flow diagram-



2.If else..Elif statement

Syntax-

if expression:

  statement(s)

else:

  statement(s)

An else statement can be combined with an if statement. An else statement contains a block of code that executes if the conditional expression in the if statement resolves to a FALSE value.

The else statement is an optional statement and there could be at the most only one else statement following if.

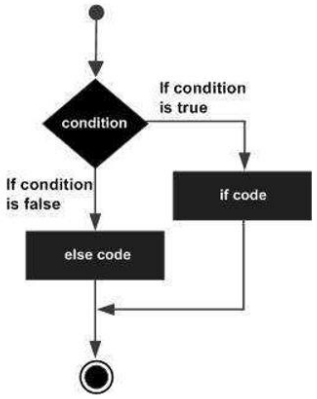
If we want to combine more if statement we use Elif statement.

Syntax-

```
if expression1:
  statement(s)
elif expression2:
  statement(s)
elif expression3:
  statement(s)
else:
  statement(s)
```

The elif statement allows you to check multiple expressions for TRUE. Elif statement is optional, there can be an arbitrary number of elif statements following an if.

Flow diagram-



3.Nested if statement

Syntax-

```
if expression1:
  statement(s)
  if expression2:
    statement(s)
  elif expression3:
    statement(s)
```

else

statement(s)

elif expression4:

statement(s)

else:

statement(s)

If statement within a if statement is termed as nested if statement.

In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct.

Flow diagram-

Looping statemts.

A loop statement allows us to execute a statement or group of statements multiple times.

1.While loop

Syntax-

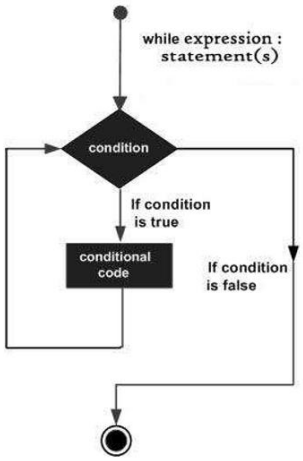
while expression:

statement(s)

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.It executes a target statement as long as a given condition is true.

Statement(s) may be a single statement or a block of statements with uniform indent. When the condition becomes false, program control passes to the line immediately following the loop.

Flow diagram-



2.For loop

Syntax-

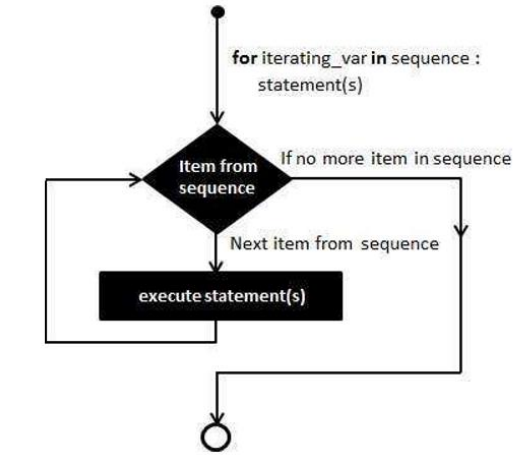
for iterating_var in sequence:

statements(s)

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.

Flow diagram-



3.Nested loop

Syntax-

for iterating_var in sequence:

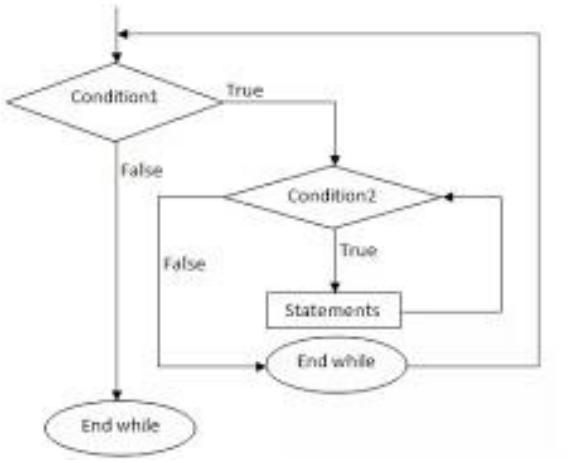
  for iterating_var in sequence:

    statements(s)

  statements(s)

In python we can use of one loop inside another loop.We can put any type of loop inside any other type of loop.

Flow diagram-



Loop control statements.

The Loop control statements change the execution from its normal sequence. When the execution leaves a scope, all automatic objects that were created in that scope are destroyed.

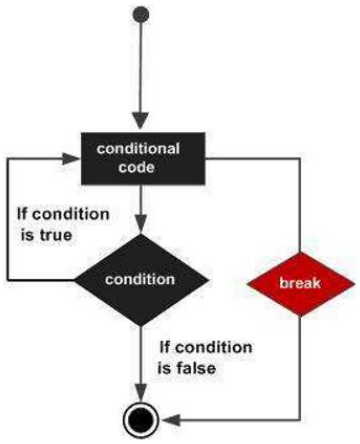
1.Break statement

Syntax-

break

Terminates the loop statement and transfers execution to the statement immediately following the loop. The break statement is used for premature termination of the current loop. The break statement can be used in both while and for loops.

Flow diagram-



2.Continue statement

Syntax-

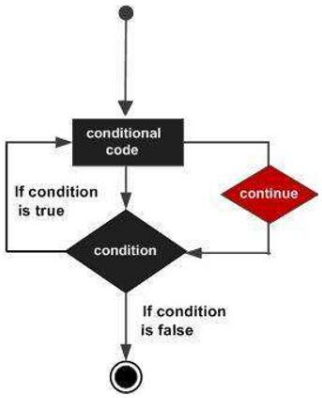
Continue

Continue causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

The continue statement can be used in both while and for loops.

Flow diagram-



3.Pass statement

Syntax-

Pass

The pass statement in Python 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 statement is also useful in places where your code will eventually go, but has not been written,

6. Explain the salient features of Python.

a) Python is a dynamic, high level, free open source and interpreted programming language.

b) Python language is easy to code.

c) Python is object oriented programming language.

d) Python is a high-level and portable language.

e) Python provides a wide range of libraries for the various fields such as Machine learning, Scripting and Web developing.

f) Python supports Graphical User Interface (GUI).

g) Python can be easily integrated with languages such as C, C++ and Java, it makes easy to debug the code.

h) Python has a powerful feature of automatically managing the memory.

i) Python has built in data structure which includes lists, numbers and dictionaries.

7. Write Python expressions corresponding to these statements:

(a) The sum of the first seven positive integers
Ans: sum =(7*(7 + 1))/2

(b) The average age of Sara (age 65), Fatima (57), and Mark (age 45)
Ans: S_age=65. F_age=57. M_age=45
Average= (S_age+F_age+M_age)/3

(c) 2 to the 20th power
Ans:power=:2**20

(d) The number of times 61 goes into 4356
Ans:no_of_time=4356/61

(e) The remainder when 4365 is divided by 61
Ans:remainder=4365%61

8. Mention the advantages of Continue statement. Write a program to compute only even numbers sum within the given number using continue statement.

Ans: advantages of 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 moves the control back to the top of the loop.
. The continue statement can be used in both while and for loops.

Program:

```
n=[1,2,3,4,5,6,7,8,9,10]
sum=0
for i in n:
  if( i%2==1 ) :
     continue

   sum = sum + i
print (sum )

Output
30
```

9. Write the Python programs to

- Find the largest of three numbers

```
def largestOfThree (a, b, c) :
    return max (max(a,b), c)

def max (a, b) :
    if a>b:
        return a
    return b
```

- Check whether the given year is a leap year or not

```
def checkLeapYear (year) :
    if year%4==0 :
        return True
    return False
```

10. Explain the following functions with examples i)input() ii)range() iii)print() iv)eval()

i) The input() method reads a line from the input (usually from the user), converts the line into a string by removing the trailing newline, and returns it.

Example:

```
inputString = input()
print('The inputted string is:', inputString)
```

Output:

```
Python is interesting.
```

The inputted string is: Python is interesting

ii) The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Example:

```
x = range(6)
for n in x:
print(n)
```

Output :

```
0 1 2 3 4 5
```

iii) The print() function prints the specified message to the screen, or other standard output device.

Example:

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

Output:

```
Hello World
```

iv) The eval() method parses the expression passed to it and runs python expression(code) within the program.

The syntax of eval is:

```
eval(expression, globals=None, locals=None)
```

Example:

```
x = 'print(55)'
eval(x)
```

Output:

```
55
```

11. What are user defined Functions ? How can we pass parameters in user defined functions ? Explain with suitable examples.

All the functions that are written by us comes under the category of user defined functions.

Syntax:

```
def function_name():
   statements
   .
   .
```

Example:

```
def fun():
   print("Inside function")
# Calling function
fun()
```

Output:

```
Inside function
```

The function may take arguments(s) also called parameters as input within the opening and closing parentheses, just after the function name followed by a colon.

Syntax:

```
def function_name(argument1, argument2, ...):
   statements
```

.
.

Example:

```
def evenOdd( x ):
   if (x % 2 == 0):
      print("even")
   else:
      print("odd")
evenOdd(2)
evenOdd(3)
```

Output:

```
even
odd
```

12. Write a Python Program to create a user defined function to find maximum and minimum letters in a String. Also find the length of the String without using inbuilt function.

a)

```
ASCII_SIZE = 256

def getMaxOccuringChar(str):

   # Create array to keep the count of individual characters

   # Initialize the count array to zero

   count = [0] * ASCII_SIZE


   # Utility variables

   max = -1

   c = ''


   # Traversing through the string and maintaining the count of

   # each character

   for i in str:

      count[ord(i)]+=1;


   for i in str:
```

```
        if max < count[ord(i)]:
            max = count[ord(i)]
            c = i
    return c
# Driver program to test the above function
str = "sample string"
print "Max occurring character is " + getMaxOccuringChar(str)
```

b)

```
def findLength(string):

    # Initialize count to zero
    count = 0

    # Counting character in a string
    for i in string:
        count+= 1
    # Returning count
    return count

# Driver code
string = "geeksforgeeks"
print(findLength(string))
```

13. What is Parameter Passing ? Explain Immutable and Mutable Parameter passing with examples.

Parameter passing is the mechanism used to pass parameters to procedure or function. Functions are either called from within the interactive shell or by another program. We refer to either as the *calling program*. The input arguments in a function call are *names* of objects created in the calling program. These names may refer to objects that are mutable or immutable.

Immutable parameter passing –

We use the function g() to discuss the effect of a passing a reference to an immutable object in a function call.

```
def g(x):
x = 5
```

Let's start by assigning integer 3 to variable name a:

```
>>> a = 3
```

In this assignment statement, integer object 3 is created and given name a. Now let's call function g() with name a as the input argument: >>> g(a)

When this function call is made, the argument a is evaluated first. It evaluates to integer object 3. Now, recall that function g() was defined as:

```
def g(x): x = 5
```

The name x in def g(x): is now set to refer to the input integer object 3. In effect, it is as if we have executed the assignment x = a: Now, there are two variables that refer to the single object 3: variable a defined in the interactive shell and variable x defined in function g().

Mutable parameter passing-

We use the next function to see what happens when the name of a mutable object is passed as the argument of a function call.

```
def h(lst):

lst[0] = 5
```

Consider what happens when we execute:

```
>>> myList = [3, 6, 9, 12]
>>> h(myList)
```

In the assignment statement, a list object is created and assigned name myList. Then the function call h(myList) is made. When function h() starts executing, the list referred to by myList will be assigned to variable name lst defined in the function definition of h().

This example illustrate that when a mutable object, like object [3,6,9,12], is passed as an argument in a function call, it may be modified by the function.

14. The probability of getting n heads in a row when tossing a fair coin n times is 2n.

Implement function prob() that takes a nonnegative integer n as input and returns the

probability of n heads in a row when tossing a fair coin n times .

```
>>> prob(1)
0.5
```

```
>>> prob(2)
0.25
```

```
def prob(n):
  return 1/(2**n)
```

15. Write a note on Python Standard Library?

Ans: We know that a module is a file with some Python code, and a package is a directory for sub packages and modules. But the line between a package and a Python library is quite blurred.

A Python library is a reusable chunk of code that you may want to include in your programs/ projects. Compared to languages like C++ or C, a Python libraries do not pertain to any specific context in Python. Here, a 'library' loosely describes a collection of core modules. Essentially, then, a library is a collection of modules. A package is a library that can be installed using a package manager like rubygems or npm.

Learn: A Comprehensive Guide on Python Packages

3. Python Standard Library
The Python Standard Library is a collection of exact syntax, token, and semantics of Python. It comes bundled with core Python distribution. We mentioned this when we began with an introduction.

It is written in C, and handles functionality like I/O and other core modules. All this functionality together makes Python the language it is. More than 200 core modules sit at the heart of the standard library. This library ships with Python. But in addition to this library, you can also access a growing collection of several thousand components from the Python Package Index (PyPI).

16. What is a string? Write a Python Program to demonstrate traversal through a string with a loop. Also explain the concepts of String Slicing

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.

Python Code to demonstrate traversal through a string with a loop

```
for i in range(len(string)):
   print(string[0:i+1])
```

String slicing

In Python, indexing syntax can be used as a substitute for the slice object. This is an easy and convenient way to slice a string both syntax wise and execution wise.

```
string[start:end:step]
```

Example-

```
var1 = 'Hello World!'
var2 = "Python Programming"
print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
print ("var2[1:9:2]: ", var2[1:9:2])
```

When the above code is executed, it produces the following result

var1[0]: H

var2[1:5]: ytho

var3[1:7:2] : yhnP

17. Write a Python Program to concatenate and Compare two strings and read the Strings from the user.

```
#reading input
str1 = input("Please Enter the First String: ")
str2 = input("Please Enter the Second String: ")
#concatenation
concat1 = str1 + ' ' + str2
print("The Final after String Concatenation = ", concat1)
#comparison
 if(str1==str2):
                print("The two strings are equal")
 else:
      print (“The two strings are not equal”)
```

18. What is list? Explain the concept of list Slicing and list traversing with example.

A list is a sequence of objects. The objects can be of any type: numbers, strings, even other lists. For example, here is how we would assign to the variable pets the list of strings representing several pets: >>> pets = ['goldfish', 'cat', 'dog','tiger','lion']

List Slicing:-

If L is a list, the expression L [start : stop : step] returns the portion of the list from index start to index stop, at a step size step.

Syntax:

```
L[start:stop:step]
```

Start position End position The increment

Example

```
pets=['Goldfish','cat','dog','tiger',lion']
print(pets[1:3])
#prints['cat','dog','tiger']
```

List traversing:-

In Python, list is a type of container in Data Structures, which is used to store multiple data at the same time. Unlike Sets, the lists in Python are ordered and have a definite count. There are multiple ways to iterate over a list in Python. Let's see all different ways to iterate over a list in Python, and a performance comparison between them.

Method #1: Using For loop

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
for i in list:     # Using for loop
    print(i)
```

Output: 1 3 5 7 9

Method #2: For loop and range()

In case we want to use the traditional for loop which iterates from number x to number y.

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
length = len(list)          # getting length of
list # Iterating the index
# same as 'for i in range(len(list))'
for i in range(length):
    print(list[i])
```

Output: 1 3 5 7 9

Method #3: Using while loop

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
length = len(list) # Getting length of list
i = 0
while i < length: # Iterating using while loop
        print(list[i])
        i += 1
```

Output: 1 3 5 7 9

Method #4: Using list comprehension (Possibly the most concrete way).

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
 [print(i) for i in list]       # Using list comprehension
```

Output: 1 3 5 7 9

Method #5: Using enumerate()

If we want to convert the list into an iterable list of tuples (or get the index based on a condition check, for example in linear search you might need to save the index of minimum element), you can use the enumerate() function.

```
# Python3 code to iterate over a list
list = [1, 3, 5, 7, 9]
for i, val in enumerate(list):      # Using enumerate()
        print (i, ",",val)
```

Output:

```
0,1
1,3
2,5
3,7
4,9
```

Method #6: Using Numpy
For very large n-dimensional lists (for example an image array), it is sometimes better to use an external library such as numpy.

```
# Python program
for # iterating over
array import numpy
as geek

# creating an array
using # arrange
method
a = geek.arange(9)

# shape array with 3
rows # and 4 columns
a = a.reshape(3, 3)

# iterating an array
for x in
        geek.nditer(
        a): print(x)
```

Output: 0 1 2 3 4 5 6 7 8

We can use np.ndenumerate() to mimic the behavior of enumerate. The extra power of numpy comes from the fact that we can even control the way to visit the elements but the one caveat is that the np.nditer treats the array as read-only by default, so one must pass extra flags such
as op_flags=['readwrite'] for it to be able to modify elements.

19. Explain how to access values in lists, update lists, delete elements in list and also explain basic list operations.

Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and Array List in Java. A single list may contain Data types like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Accessing Values in a List.

- In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. The index must be an integer.
- In Python, negative sequence indexes represent positions from the end of the array. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

Updating a list/Adding elements

- Elements can be added to the List by using built-in append function. Only one element at a time can be added to the list by using append() method.
- For addition of element at the desired position, insert() method is used.
  Unlike append which takes only one argument, insert() method requires two arguments(position, value).
- The extend() method is used to add multiple elements at the same time at the end of the list.

Deleting elements from a list

- Elements can be removed from the List by using built-in remove() function but an Error arises if element doesn't exist in the set. Remove() method only removes one element at a time.
- Pop() function can also be used to remove and return an element from the set, but by default it removes only the last element of the set, to remove element from a specific position of the List, index of the element is passed as an argument to the pop() method.

Example:

```
>>> list=[1,2,3]
>>> list[0]
1
>>> list[-1]
3
>>> list.append(4)
>>> list
[1, 2, 3, 4]
>>> list.insert(4,5)
>>> list
[1, 2, 3, 4, 5]
>>> list.extend([6])
>>> list
[1, 2, 3, 4, 5, 6]
>>> list.remove(6)
>>> list
[1, 2, 3, 4, 5]
>>> list.pop()
5
>>> list
[1, 2, 3, 4]
```

20. Compare and Contrast Tuples and Lists.

Tuples	Lists
● Tuples are immutable.	● Lists are mutable.
● Implication of iterations is comparatively faster.	● Implication of iterations is time consuming.
● Tuple datatype is appropriate for accessing the elements.	● The list is better for performing operations, such as insertion and deletion.
● Tuple consumes less memory as compared to the list.	● List consume more memory.
● Tuples have less in-built methods.	● List provides many in-built methods.
● Tuples are less error prone compared to list.	● List operations are more error prone.

21.) What is tuple i)how to access the values in tuple ii)built in tuple functions?

A tuple is a collection which is ordered and unchangeable.In python tuples are written with round brackets.A tuple is similar to a list except that the objects in tuple is similar to the list That the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned.

Accessing values in tuples
To access values in tuple use the sequence branches for slicing along the index or indices to obtain with the index or indices to obtain value available at the index
For eg:

```
Tup 1=('physics','chemistry');
 Tup 2=(1,2,3,4,5,6,7);
print("tup 1[0]:",tup 1[0])
print (tup2[1:5]:",tup 2[1:5])
```

Built in tuple functions:-

Python includes the following functions

Sr no.	function and description
1)	Emp(tup1 ,tup2) Compares elements of both tuples
2)	Len(tuple) Gives the total length of the tuple
3)	Max(tuple) Gives max value of tuple
4)	Min(tup) Gives min value of the tuple

22)

The split() method splits a string into a list. You can specify the separator, default separator is any whitespace.

Example:

1) txt = "hello, my name is Peter, I am 26 years old"
   x = txt.split(", ")
   print(x)

   Output: ['hello',' my name is Peter', 'I am 26 years old']

The join() string method returns a string by joining all the elements of an iterable, separated by a string separator.

Example:

1) myTuple = ("John", "Peter", "Vicky")

   x = "#".join(myTuple)

   print(x)

Immutablity of Strings: In python, the string data types are immutable. Which means a string value cannot be updated. We can verify this by trying to update a part of the string which will led us to an error.

Example:

1) t = "UVCE"
   print(t)
   t[0] = "V"

Output: t[0] = "V"
TypeError: 'str' object doesnot support item assignment

23. Discuss the relation between tuples and lists , tuples and dictionaries in detail.

A: TUPLES and LISTS

1. A list is created using square brackets [] whereas the tuple is ceated using parenthesis ().
2. Lists are mutable while tuples are immutable
3. we can change/modify the values of a list but we cannot change/modify the values of a tuple.
4. Tuples cannot copy data but lists can copy all data to new list
5. list has a larger size than the tuple
6. Tuples are used to store heterogeneous elements, which are of different data types but Lists are used to store homogenous elements, which are of same data type.

TUPLES and DICTIONARIES

1. In tuple,Order is maintained but in dictionary order is not guaranteed
2. In tuples, Elements are accessed via numeric (zero based) indices but in dictionary Elements are accessed using key's values
3. Tuple items in round brackets or parentheses (), and dictionary items in curly brackets {}
4. Dictionary objects are mutable i.e. it is possible to add new item or delete an item from it. Tuple is an immutable object. Addition or deletion operations are not possible on tuple object.

24.Explain the concept of Type conversion functions and math functions in python with examples.

Ans :<u>Type conversion functions</u>:

Python defines type conversion functions to directly convert one data type to another which is useful in day to day and competitive programming.These built-in functions return a new object representing the converted value.

1.int(x,[base]): Converts x to an integer.The base specifies the base if x is a string.

Example: s="10010"

```
c=int(s,2)
print(c)
```

2.float(x): Converts x to a floating-point number.

Example: s="10010"

```
c=float(s)
print(c)
```

3.complex(real,[imag]): Creates a complex number.

Example: c=complex(1,2)

```
print(c)
```

4.str(x): Converts object x to a string representation.

Example: c=str(1)

```
print(c)
```

5.repr(x): Converts object x to an expression string.

6.eval(str): Evaluates a string and returns an object.

7.tuple(s): Converts s to a tuple.

Example: s='geeks'

```
c=tuple(s)
print(c)
```

8.list(s): Converts s to a list.

Example: s='geeks'

```
c=list(s)
print(c)
```

9.set(s): Converts s to a set.

Example: s='geeks'

```
c=set(s)
```

print(c)

10.dict(d): Creates a dictionary ,d must be a sequence of (key,value) tuples.

Example: tup=((‘a’,1),(‘f’,2),(‘g’,3))

c=dict(tup)

print(c)

11.frozenset(s): Converts s to a frozen set.

Example:s=frozenset([1,2,3])

print(s)

12.chr(x): Converts an integer to a character.

Example: a=chr(76)

print(a)

13.unichr(x): Converts an integer to a Unicode character.

Example: c=unichr(6)

print(c)

14.ord(x): Converts a single character to its integer value.

Example: s=”4”

c=ord(s)

print(c)

15.hex(x): Converts an integer to a hexadecimal string.

Example: c=hex(56)

print(c)

16.oct(x): Converts an integer to an octal string.

Example: c=oct(56)

print(c)

Math functions:

Python includes the following functions that perform mathematical calculations.

abs(x): The absolute value of x: the (positive) distance between x and zero.

Example: #!/usr/bin/python3

print ("abs(-45) : ", abs(-45))

print ("abs(100.12) : ", abs(100.12))

Output: abs(-45) : 45

abs(100.12) : 100.1

ceil(x): The ceiling of x: the smallest integer not less than x.

Example: #!/usr/bin/python3

import math   # This will import math module

print ("math.ceil(-45.17) : ", math.ceil(-45.17))

Output: math.ceil(-45.17) : -45

cmp(x, y): -1 if x < y, 0 if x == y, or 1 if x > y. Deprecated in Python 3; Instead use return (x>y)-(x<y).

exp(x): The exponential of x: ex

Example: import math   # This will import math module

print ("math.exp(-45.17) : ", math.exp(-45.17))

Output: math.exp(-45.17) : 2.4150062132629406e-20

fabs(x): The absolute value of x.

Example: import math   # This will import math module

print ("math.fabs(-45.17) : ", math.fabs(-45.17))

Output: math.fabs(-45.17) : 45.17

floor(x): The floor of x: the largest integer not greater than x.

Example: import math   # This will import math module

print ("math.floor(-45.17) : ", math.floor(-45.17))

Output: math.floor(-45.17) : -46

log(x): The natural logarithm of x, for x> 0.

Example: import math   # This will import math module

print ("math.log(100.12) : ", math.log(100.12))

Output: math.log(100.12) : 4.6063694665635735

log10(x): The base-10 logarithm of x for x> 0.

Example:import math   # This will import math module

print ("math.log10(100.12) : ", math.log10(100.12))

Output: math.log10(100.12) : 2.0005208409361854

max(x1, x2,...): The largest of its arguments: the value closest to positive infinity.

Example: print ("max(80, 100, 1000) : ", max(80, 100, 1000)

Output: max(80, 100, 1000) : 1000

min(x1, x2,...): The smallest of its arguments: the value closest to negative infinity.

Example: print ("min(80, 100, 1000) : ", min(80, 100, 1000))

Output: min(80, 100, 1000) : 80

modf(x): The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.

Example: import math # This will import math module

print ("math.modf(100.12) : ", math.modf(100.12))

Output: math.modf(100.12) : (0.12000000000000455, 100.0)

pow(x, y): The value of x**y.

Example: import math # This will import math module

print ("math.pow(100, 2) : ", math.pow(100, 2))

Output: math.pow(100, 2) : 10000.0

round(x [,n]): x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is 1.0.

Example: print ("round(70.23456) : ", round(70.23456))

Output: round(70.23456) : 70

sqrt(x) The square root of x for x >0.

Example: import math # This will import math module

print ("math.sqrt(100) : ", math.sqrt(100))

Output: math.sqrt(100) : 10.0

25)List and explain 4 built in sting manipulation functions in python.

a) String capitalize() :

It returns a copy of the string with only its first character capitalized.

Syntax

```
str.capitalize()
```

Ex: m = ” rcb will win this year’s IPL cup”

print(“capitalised sentence is : ”, m.capitalize())

o/p: Rcb will win this year’s IPL cup

b) The find ()

This method determines if the string str2 occurs in str1, or in a substring of string if the starting index beg and ending index end are given.

Syntax

```
str.find(str, beg=0 end=len(string))
```

Parameters

- **str** - This specifies the string to be searched.
- **beg** - This is the starting index, by default its 0.
- **end** - This is the ending index, by default its equal to the lenght of the string.

Return Value

Index if found and -1 otherwise.

Ex: str1= ”ABD is love, superman for a reason”

Str2= ”super”

Print(str1.find(str2))

O/p: 13

c) String isalnum()

isalnum() method checks whether the string consists of alphanumeric characters.

Syntax

Following is the syntax for isalnum() method-

```
str.isa1num()
```

Ex: ABD= ”Mr360”

Print(ABD.isalnum())

O/p: True

d) String islower()

The islower() method checks whether all the case-based characters (letters) of the string are lowercase.

Syntax

Following is the syntax for islower() method-

```
str.islower()
```

Parameters

NA

Return Value

This method returns true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

Ex: abd= ”most dangerous batsman”

Vk= ”MOST dangerous batsman”

Print(abd.islower())

Print(Vk.islower())

O/p: True

False

26)What are Lists ? Lists are mutable. Justify the statement with examples.

Sol:

In Python, lists are usually stored in a type of object called a list. A list is a sequence of objects(ordered and have a definite count). The objects can be of any type: numbers, strings, even other lists.

An important property of lists is that they are *mutable*. What that means is that the content of a list can be changed (they can be altered even after their creation).

For example, suppose that we want to be more specific about the type of cat in list pets.

We would like pets[1] to evaluate to 'cymric cat' instead of just plain 'cat'. To do this, we assign 'cymric cat' to pets[1]:

```
>>> pets[1] = 'cymric cat'
>>> pets
```

Output: ['goldfish', 'cymric cat', 'dog']

So, the list no longer contains the string 'cat' at index 1; instead, it contains the string 'cymric cat'.

27. Explain the working of while loop in python with suitable examples.

Ans: A while loop statement in python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax :

While expression:

    Statement(s)

Here, statement(s) may be single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

While the condition becomes false, program control passes to the line immediately following the loop.

In python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Example :

```
#!/user /bin/python3

Count =0

While (count<9):

    Print('the count is :', count)

    Count = count +1

Print ("good bye!")
```

Output :

The count is : 0 The count is : 1 The count is: 2

The count is : 3 The count is : 4 The count is : 5

The count is : 6 The count is : 7 The count is : 8

Good bye!

28.

```
# function to print occurrences of each vowel in the given
string

def vowelCount(str):
    str = str.lower()
    ans=[0,0,0,0,0]

    for c in str:
        if c =='a':
            ans[0]+=1
        elif c=='e':
            ans[1]+=1
        elif c=='i':
            ans[2]+=1
        elif c=='o':
            ans[3]+=1
        elif c=='u':
            ans[4]+=1

    return ans
```

29) Write a python program to demonstrate Counting, Summing and average of elements using loops.

a)Counting elements

```
list=['a','b',1,2]
count=0
for i in list:
        count=count+1

print(count)
```

b) sum

```
tupl=(1,2,3,4)
sum=0
for i in tupl:
        sum=sum+i

print(sum)
```

c) average

```
lst=[1,2,3,4,5]
sum=0
count=0
for i in lst:
        count=count+1
        sum=sum+i

print(sum/count)
```

30. Explain file open, file close, file read and file write concepts in Python with example.

Solution :

open(): Before we can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it.

Syntax:

file object = open(file_name [, access_mode][, buffering])

where**,**

file_name: The file_name argument is a string value that contains the name of the file that you want to access.

access_mode**:** The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc.

buffering: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file.

The different modes of opening a file are r, rb, r+, rb+, w, wb, w+, wb+, a, ab, a+, ab+.

close(): The close() method of a file object flushes any unwritten information and closes the fileobject, after which no more writing can be done.

Syntax:

fileObject.close()

read(): The read() method reads a string from an open file.

Syntax:

fileObject.read([count])

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

write(): The write() method writes any string to an open file.

Syntax:

fileObject.write(string)

Here, passed parameter is the content to be written into the opened file.

Example:

```
 # Open a file
f = open("file1.txt", "w")
f.write( "Python is a great language.\nYeah its great!!\n")
# Close opend file
f.close()
#open and read the file after the writing
f = open("file1.txt", "r+")
```

```
str = f.read(10)
print ("Read String is : ", str)
# Close opened file
f.close()
```

Output:

```
Read String is : Python is
```

31) Write a python program to accept a file name from user:

a) Display the first N-lines of the file.
b) Find the frequency of occurrence of the word accepted from the user in the file.

```
def file_read_from_head(fname, nlines):
    from itertools import islice
        with open(fname) as f:
        for line in islice(f, nlines):
            print(line)
    file_read_from_head('test.txt',2)
```

```
from collections import Counter
    def word_count(fname):
        with open(fname) as f:
            return Counter(f.read().split())

    print("Number of words in the file :",word_count("test.txt"))
```

32. Differentiate Pop and Remove methods on the lists. How to delete more than one element from a list.

Pop(): Removes the element at the specified index from the list. Index is an optional parameter. If no index is specified, then removes the last object(or element) from the list.

Syntax: list.pop(|index|)

Example: >>>num_list = [6, 3 , 7, 0, 1, 2, 4, 9]

>>>print(num_list.pop())

>>>print(num_list)

Output: 9

[6, 3, 7, 0, 1, 2, 4]

Remove(): Removes or deletes object(or element) from the list . ValueError is generated if obj is not present in the list. If multiple copies of obj exists in the list, then the first value is deleted.

Syntax: list.remove(obj)

Example: >>> num_list=[6, 3, 7, 0, 1, 2, 9]

>>>num_list.remove(0)

>>>print(num_list)

Output: [6, 3, 7, 1, 2, 9]

Deletion of more than one element from a list: We have the del operator . This is a Python built-in keyword, instead of a list method. The del operator allows us to remove elements from a list by indice(s). This approach is useful because it can remove both single elements, by providing a single index, as well as multiple elements, by providing a range of indices.

Example: >>> num_list = [2, 3, 4, 5, 6]

>>>del num_list[0:2]

>>>print(num_list)

Output: [4, 5, 6]

Praveen v naik 18GAEI6036

33.

Convert dictionary to list of tuple

Using `items()`

```
# Python code to convert dictionary into list of tuples


# Initialization of dictionary

dict = { 'Geeks': 10, 'for': 12, 'Geek': 31 }


# Converting into list of tuple

list = list(dict.items())


# Printing list of tuple

print(list)
```

Output:

```
[('for', 12), ('Geeks', 10), ('Geek', 31)]
```

2.

Get all tuple keys from dictionary

Sometimes, while working with Python dictionaries, we can have it's keys in form of tuples. A tuple can have many elements in it and sometimes, it can be essential to get them. If they are a part of a dictionary keys and we desire to get all the tuple key elements, we need to perform certain functionalities to achieve this. Let's discuss certain ways in which this task can be performed.

Method #1 : Using list comprehension
In this method, we just iterate through the each dictionary item and get it's key's elements into a list.

```
# Python3 code to demonstrate working of
```

```
# Get all tuple keys from dictionary
# Using list comprehension

# Initializing dict
test_dict = {(5, 6) : 'gfg', (1, 2, 8) : 'is', (9, 10) : 'best'}

# printing original dict
print("The original dict is : " + str(test_dict))

# Get all tuple keys from dictionary
# Using list comprehension
res = [ele for key in test_dict for ele in key]

# printing result
print("The dictionary tuple key elements are : " + str(res))
```

Output :

```
The original dict is : {(5, 6): 'gfg', (9, 10): 'best', (1, 2, 8):
'is'}
The dictionary tuple key elements are : [5, 6, 9, 10, 1, 2, 8]
```

34. Explain with syntax about Decision Control Statements in Python with examples.

Ans: Decision making is anticipation of condtions occurring while execution of the program and specifiying actions taken according to the conditions.

- If Statement:

An if Statement consists of a Boolean expression followed by one or more statements.

Syntax:

```
If ( condition) :
        Statements
```

Example:

```
Var1 = 100
if ( Var1 == 100) :
        print("100")
print("not 100)
```

- If Else Statement:

An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.

Syntax:

```
If ( condition) :
        Statements
else:
        Statements
```

Example:

```
Var1 = 100
if ( Var1 == 100) :
        print("100")
```

```
else:
        print("not 100")
```

- Nested If Statement:

You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Syntax:

```
If ( condition) :
        If ( condition) :
                Statements
        else:
                Statements
else:
        Statements
```

Example:

```
Var1 = 100
if ( Var1 >= 100) :
        if ( Var1 == 100) :
                print ("100")
        else :
                print ("greater than 100")
else:
        print("less than 100")
```

35) Write a note on Two-Dimensional Lists.

Sol: There can be more than one additional dimension to lists in Python. Keeping in mind that a list can hold other lists, that basic principle can be applied over and over. Two-dimensional lists are the list within lists (a list, where each element is in turn a list).

Example:

```
# Python program to demonstrate that we

# can access multidimensional list using
# square brackets
a = [ [2, 4, 6, 8 ],
      [ 1, 3, 5, 7 ],
      [ 8, 6, 4, 2 ],
      [ 7, 5, 3, 1 ] ]

for i in range(len(a)) :
      for j in range(len(a[i])) :
            print(a[i][j], end=" ")
      print()
```

Output:

```
2  4  6  8
1  3  5   7
8  6  4  2
7  5  3  1
```

36.Finite loop:

A loop that we explicitly know, in advance, which values the control variables will have during the loop execution. The simplest case of a finite loop is the **for loop** within a range, as in the example appearing here. This loop goes through all the values from 0 to 9 (one before 10) for the control variable *i*:

```
1. for i in range(0, 10):
2.     print(i)
```

Example:

```
1. i = 0
2. while i < 10:
3.     print(i)
4.     i += 1
```

Infinite loop:

An **infinite loop**, on the other hand, is characterized by not having an explicit end, unlike the finite ones exemplified previously, where the control variable *i* clearly went from 0 to 9 (note that at the end *i = 10*, but that value of *i* wasn't printed). In an infinite loop the control is not explicitly clear.

```
1. i = 0
2. while True:
3.     print(i)
4.     if i == 9:
5.         break
6.     else:
7.         i += 1
8.
```

Example:

```
1. from random import randint
2. num_acc = 0
3. num_rand = 0
4. acc = 0
5. while (true):
6.     x = randint(0, 10)
7.     num_rand += 1
8.     if acc + x > 63:
9.         print(' Discarding: ', x)
10.        continue
11.    else:
12.        print('Considering: ', x)
13.        num_acc += 1
14.        acc += x
15.        if acc == 63:
16.            break
```

```
17. print('%s numbers were generated, but only %s numbers were considered to reach
    the %s threshold' % (num_rand, num_acc, acc))
```

37) What is the need for break and continue statements ? and also write a note on pass statement.

The ***break*** statement is used to terminate the execution of the nearest enclosing loop in which it appears.

When the compiler encounters a **continue** statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

Python provides **break** and **continue** statements to handle such situations wherein you need to exit a loop completely when an external condition is triggered or there may also be a situation when you want to skip a part of the loop and start next execution and to have good control on your loop.

The Pass Statement

- It used when a statement is required syntactically but you do not want any command or code to execute.
- It specifies a *null* operation or simply No Operation (NOP) statement.
- Nothing happens when the pass statement is executed.
- It is used for writing empty loops, control statement, function and classes, where your code will eventually go, but has not been written yet.

```
for letter in "HELLO":
        pass     #The statement is doing nothing
        print("Pass : ", letter)
print("Done")
```

OUTPUT

```
Pass :  H
Pass :  E
Pass :  L
Pass :  L
Pass :  O
Done
```

Name : Rohit Bhandari

Reg.no : 18GAEI6041

Subject : Python Programming

College : UVCE

ASSIGNMENT

38. Implement function fib() that takes a non-negative integer n as input and returns the nth Fibonacci number.

```
# Function for nth Fibonacci number

def Fibonacci(n):
    if n<0:
        print("Incorrect input")

    # First Fibonacci number is 0
    elif n==0:
        return 0

    # Second Fibonacci number is 1
    elif n==1:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

print(Fibonacci(9))
```

OUTPUT : 21

PYTHON PROGRAMMING ASSIGNMENT

39. What is Exception? How to handle an Exception in Python?

Sol : Even if a statement is syntactically correct, it may still cause an error when executed. Such errors that occur at run-time (or during execution) are known as *exceptions.* An exception is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions. When a program encounters a situation which it cannot deal with, it raises an exception. Therefore, we can say that an exception is a Python object that represents an error.

When a program raises an exception, it must handle the exception or the program will be immediately terminated.

Handling Exceptions

We can handle exceptions in our program by using try block and except block. A critical operation which can raise exception is placed inside the try block and the code that handles exception is written in except block. The syntax for try–except block can be given as,

Syntax :

```
try :
    statements
except ExceptionName :
    statements
```

Example :

```
num = int(input("Enter the numerator : "))
deno = int(input("Enter the denominator : "))
try :
    quo = num/deno
    print("Quotient : ",quo)
except ZeroDivisionError :
    print("Denominator cannot be zero")
```

Output :

```
Enter the numerator : 10
Enter the denominator : 0
Denominator cannot be zero
```

40) Explain the following-

(i) except: Block without Exception

You can even specify an except block without mentioning any exception (i.e., except:). This type of except block if present should be the last one that can serve as a wildcard (when multiple except blocks are present). But use it with extreme caution, since it may mask a real programming error.

In large software programs, may a times, it is difficult to anticipate all types of possible exceptional conditions.

Therefore, the programmer may not be able to write a different handler (except block) for every individual type of exception. In such situations, a better idea is to write a handler that would catch all types of exceptions. The syntax to define a handler that would catch every possible exception from the try block is,

Example-

```
try:
   Write the operations here
   ....................
except:
   If there is any exception, then execute this block.
   ....................
else:
   If there is no exception then execute this block.
```

```
try:
    file = pen('File1.txt')
    str = f.readline()
    print(str)
except IOError:
    print("Error occured during Input ..... Program Terminating...")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error.... Program Terminating...")
```

Programming Tip: When an exception occurs, it may have an associated value, also known as the exception's *argument*.

OUTPUT

```
Unexpected error.... Program Terminating...
```

(ii) Multiple Except Block

Python allows you to have multiple except blocks for a single try block. The block which matches with the exception generated will get executed. A try block can be associated with more than one except block to specify handlers for different exceptions.

However, only one handler will be executed. Exception handlers only handle exceptions that occur in the corresponding try block. We can write our programs that handle selected exceptions. The syntax for specifying multiple except blocks for a single try block can be given as,

```
try:
   operations are done in this block
   ......................
except Exception1:
   If there is Exception1, then execute this block.
except Exception2:
   If there is Exception2, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

```
   try:
     num = int(input("Enter the number : "))
     print(num**2)
except (KeyboardInterrupt, ValueError, TypeError):
     print("Please check before you enter..... Program Terminating...")
print("Bye")
```

Programming Tip: No code should be present between a list of `except` blocks.

OUTPUT

```
Enter the number : abc
Please check before you enter..... Program Terminating...
Bye
```

(iii) try and finally clause

The try block has another optional block called finally which is used to define clean-up actions that must be executed under all circumstances. The finally block is always executed before leaving the try block. This means that the statements written in finally block are executed irrespective of whether an exception has occurred or not. The syntax of finally block can be given as,

try:

Write your operations here

.....

Due to any exception, operations written here will be skipped

finally:

This would always be executed.

.....

Example -

```
try:
    print("Raising Exception.....")
    raise ValueError
finally:
    print("Performing clean up in Finally......")
```

OUTPUT

```
Raising Exception.....
Performing clean up in Finally......
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 4, in <module>
    raise ValueError
ValueError
```

41. Write Python program to check for the presence of a key in the dictionary and find the sum all its values.

Ans:

Program to check whether a given key already exists in a dictionary.

```
def checkKey(dict, key):
if key in dict.keys():
print("Present, ", end =" ")
print("value =", dict[key])
else:
print("Not present")
dict = {'a': 100, 'b':200, 'c':300}
key = 'b'
checkKey(dict, key)
key = 'w'
checkKey(dict, key)
# Python3 Program to find sum of all items in a Dictionary
def returnSum(myDict):
sum = 0
for i in myDict:
sum = sum + myDict[i]
return sum
dict = {'a': 100, 'b':200, 'c':300}
print("Sum :", returnSum(dict))
```

42. Pythonic program to sort a sequence of names according to their alphabetical order without usig Sort() function.

Ans:

```
names=input("Enter the names")
split_names=names.split(',')
empty_list=[]
while split_names:
empty_list+=[split_names.pop(split_names.index(min(split_names)))]
print("output is:")
print(','.join(empty_list))
```

43. Write Python program to count the number of times an item appears in the list.

Ans:

Examples:

Input: 1st = [15, 6, 7, 10, 12, 20, 10, 28, 10]
x = 10
Output: 3
10 appears three times in given list.
Input: 1st = [8, 6, 8, 10, 8, 20, 10, 8, 8]
x = 16
Output: 0

Method 1 (Simple approach)

We keep a counter that keeps on increasing if the required element is found in the list. **I**

code to count the number of occurrences

```
def countX(lst, x):
count = 0
for ele in lst:
if (ele == x):
count = count + 1
return count
lst = [8, 6, 8, 10, 8, 20, 10, 8, 8]
x = 8
print('{} has occurred {} times'.format(x, countX(lst, x)))
```

Output:

8 has occurred 5 times

Method 2 (Using count())

The idea is to use list method count() to count number of occurrences. **Python code t**

count the number of occurrences

```
def countX(lst, x):
return lst.count(x)
lst = [8, 6, 8, 10, 8, 20, 10, 8, 8]
x = 8
print('{} has occurred {} times'.format(x, countX(lst, x)))
```

Output:

8 has occurred 5 times

Method 3 (Using Counter()) Counter method returns a dictionary with occurrences elements as a key-value pair, where key is the element and value is the number of t that element has occurred. **from** collections **import** Counter

```
# declaring the list
l = [1, 1, 2, 2, 3, 3, 4, 4, 5, 5]
x = 3
d = Counter(l)
print('{} has occurred {} times'.format(x, d[x]))
```

Output:

3 has occurred 2 times

44. Write Python program to perform a linear search for a given Key number in the list and report Success or Failure.

Ans:

```
def search(l, x):
    for i in range(len(l)):
      if l[i] == x:
          return i
  return -1
n = int(raw_input())
arr = raw_input()
l = list(map(int,arr.split(' ')))
```

```
If(Search (l,key)==-1)
    Print("key not found")
Else
    Print("key found")
```

45. Write a Python program to remove duplicates fr
list.

```
# Python code to remove duplicate elements
def Remove(duplicate):
final_list = [ ]
for num in duplicate:
if num not in final_list:
final_list.append(num)
return final_list
duplicate = [2, 4, 10, 20, 5, 2, 20, 4]
print(Remove(duplicate))
```

OUTPUT:
[2, 4, 10, 20, 5]

46. Write a python program to get the frequency of the elements in the list
Ans:

```
import collections
my_list=[10,10,30,10,20,30,10,50,40,50,10,30,20]
print("original list:",my_list)
ctr=collections.Counter(my_list)
print("Frequency of the elements in the list:",ctr)
```

47. Write a Python program to print a nested list (each list on a new line) using the print() function.
Ans:

```
Matrix = [ [1, 2, 3, 4], [7, 8, 4, 3], [3, 7, 3, 8], [3, 5, 7, 3,] ]
for item in Matrix:
for x in item:
print(x, end=" "),
print('\n')
```

Output:
1 2 3 4
7 8 4 3
3 7 3 8
3 5 7 3

48. Write a Python program to replace the last element in a list with another list.
Ans:

```
list1=[1,2,3,4,5,6,7]
list2=[3,6,9,12]
```

```
new_list = list1[:-1]+list2
print(new_list)
```
Sample Output:
[1,2,3,4,5,6,3,6,9,12]

49.Write a python program to iterate over two lists simultaneously.
Ans:
Code-
```
import itertools
#input two lists a and b
a=list(map(int,input().split()))
b=list(map(int,input().split()))
#printing elements of two lists simultaneously
print("Elements of lists a and b respectively:")
for i in itertools.zip_longest(a,b):
print(list(i))
```

50. Write a Python program to remove the Kth element from a given list, print the new list.
Ans:
#the aim of the program is to develop a python program to pop out the kth (the last) element from the list and print out the new list.
#the code goes this way.
```
X = [1,2,3,4,5,6,7,8,9]   #the list is pre-defined as shown
X.pop(len(x)-1)           #the Kth element (last element) i.e. 9 is popped from the
                                  List using the pop() function of python
print(X)                  #the updated list is then displayed
```
OUTPUT:
[1,2,3,4,5,6,7,8]

#If the list was supposed to be the user input,
```
My_list=[]       #declare an Empty List with a suitable name
My_list.append(int(input()))      #input given by the user is appended to the List using
                                                    The append() function
My_list.pop(len(My_list)-1)       #the Kth element in which the user has given
                                                    Will be popped out
print(My_list)                          #the list will be printed
```

INPUT:
[12,4,34,64,67,78,87,76,56]
OUTPUT:
[12,4,34,64,67,78,87,76]

51. Write a Python program to read a file line by line and store it into a list.
Ans:
```
with open("test.txt") as file_in:
```

```
lines = []
for line in file_in:
lines.append(line)
print(lines)
```

Output:
['Hai\n', 'Welcome to\n', 'Python programming!\n']

52. Writeapythonprogram tofindthelongestwords.
Ans:

```
def longest_word(filename):
with open(filename,'r') as in file:
words=infile.read().split()
max_len=len(max(words,key=len))
#OR
max_len=max(len(w) for w in words)
return [word for word in words if len(word)==max_len]
print (longest_word('test.txt'))
```

Example:

```
>>>s="aaaaaaaa"
>>>max(s.split(),key=len)
'aaa'
```

split() splits the string into words (separated by white space); max() finds the largest element using the builtin len() function, i.e.the string length, as the key to find out what "largest" means.

53. Write a Python program to count the number of lines in a text file.
Ans:
Code-

```
#opening and reading the file
f_name = input(" Enter Text file name to open : (eg : name.txt)")
file = open(f_name,'r')
#Reading the contents and printing number of lines
contents = file.read()
print ("\n\n Number of lines in file '{}' is : {}" .format ( f_name, len(contents.split('\n') ) ) )
#Closing of file
file.close()
```

54. Write a Python program that takes a text file as input and returns the number of words of a
given text file.
Ans:

```
Fname = input(“Enter file name: “)
 Num_words = 0
 With open(fname, ‘r’) as f:
    For line in f:
        Words = line.split()
```

```
        Num_words += len(words)
Print(“Number of words:”)
Print(num_words)
```

Runtime Test Cases:

Case 1:

Contents of file:

Hello world

Output:

Enter file name: data1.txt

Number of words: 2

Case 2:

Contents of file:

This programming language is

Python

Output:

Enter file name: data2.txt

Number of words: 5

55. Write a Python program to create a file where all letters of English alphabet are listed by specified number of letters in each line.

Ans:

```
import string
def letters_file_line(n):
   with open("words1.txt", "w") as f:
      alphabet = string.ascii_uppercase
      letters = [alphabet[i:i + n] + "\n" for i in range(0, len(alphabet), n)]
      f.writelines(letters)
letters_file_line(3)
```

Output:

words1.txt

ABC

DEF

GHI

JKL

MNO

PQR

STU

VWX

YZ

56. Write a Python program to find the repeated items of a tuple.

Ans:

```
 tuple=(1,2,3,2,1,3,4,2,5,3,2)
list=[]
```

```
for i in tuple:
    if tuple.count(i)>1:
        list.append(i)
set=set(list)
print(set)
```

57. **Write a Python Program to check whether an element exists within a tuple.**

Ans:

Code:

```
tuple = ( “ a “, “ b ” , 1 ,  ” c ” , “ d ” , 2  , “ e “ , “ f “, “ g ” )
Print ( “ c “  in tuple )
Print (  4   in  tuple )
```

Output:

True
False

58. Can we convert a list into tuple? If so , write a python program to convert a list into tuple.

Ans:

Yes a list can be converted to a tuple , the program for
Which is given below :
Example ->

```
#Convert list to tuple
listx = [5, 10, 7, 4, 15, 3]
print(listx)
#use the tuple() function built-in Python, passing as parameter the list
tuplex = tuple(listx)
print(tuplex)
```

59. Write a Python program to slice a tuple

Ans:

Python Code:

```
#create a tuple
tuplex=(2,4,3,5,4,6,7,8,6,1)
#used tuple[start:stop] the start index is inclusive and the stop index
_slice =tuplex[3:5]
#is exclusive
print(_slice)
#if the start index isn't defined, is taken from the beg inning of the tuple
_slice =tuplex[:6]
print(_slice)
#if the end index isn't defined, is taken until the end of the tuple
_slice =tuplex[5:]
print(_slice)
#if neither is defined, returns the full tuple
_slice =tuplex[:]
print(_slice)
```

```
#The indexes can be defined with negative values
_slice =tuplex[-8:-4]
print(_slice)
#create another tuple
tuplex=tuple("HELLO WORLD")
print(tuplex)
#step specify an increment between the elements to cut of the tuple
#tuple[start:stop:step]
_slice =tuplex[2:9:2]
print(_slice)
#returns a tuple with a jump every 3 items
_slice =tuplex[::4]
print(_slice)
#when step is negative the jump is made back
_slice =tuplex[9:2:-4]
print(_slice)
```

Copy
Sample Output:

```
(5, 4)
(2, 4, 3, 5, 4, 6)
(6, 7, 8, 6, 1)
(2, 4, 3, 5, 4, 6, 7, 8, 6, 1)
(3, 5, 4, 6)
('H', 'E', 'L', 'L', 'O', ' ', 'W', 'O', 'R', 'L', 'D')
('L', 'O', 'W', 'R')
('H', 'O', 'R')
('L', ' ')
```

60. Write a Python program to reverse a tuple.

Ans:

```
#create a tuple
x = ("PYTHON")
# Reversed the tuple
y = reversed(x)
print(tuple(y))
#create another tuple
x = (2,4,6,8)
# Reversed the tuple
y = reversed(x)
print(tuple(y))
```

Output:

```
('N','O','H','T','Y','P')
(8,6,4,2)
```