UNIT I

Questions carrying 2 Marks

1. What is Python Virtual Machine?

Python Virtual Machine (PVM) is a program that provides a programming environment. The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.

2. How to view the Python byte code?

To view the byte code instructions that were created internally by the Python compiler, we should specify the dis module using Python command as:

C:\>python -m dis first.py

3. How to write a comment line in Python? mention two types.

Comments in Python are the lines in the code that the compiler ignores during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully. There are three types of comments in Python –
Single line Comments
Multiline Comments
Docstring Comments

4. What is docstring?

Python docstring is the string literals with triple quotes that are appeared right after the function. It is used to associate documentation that has been written with Python modules, functions, classes, and methods. It is added right below the functions, modules, or classes to describe what they do.

5. What is meant by Garbage Collection in Python?

Garbage collection is to release memory when the object is no longer in use. This system destroys the unused object and reuses its memory slot for new objects. You can imagine this as a recycling system in computers. Python has an automated garbage collection.

6. List any four differences between C and Python

Variables are declared in C.	Python has no declaration.
C doesn't have native OOP.	Python has OOP which is a part of the language.
Pointers are available in C language.	No pointers functionality is available in Python.
C is a compiled language.	Python is an interpreted language.

7. List any four differences between Java and Python.

JAVA	PYTHON
Java programs contain more number of lines	Python program are compact ,containing less number of lines.
It is compulsory to declare the datatypes of variables, arrays etc in Java.	Type declaration is not required in Python.
Java has switch statement	Python doesn't have switch statement
Indentations are not necessary	Indentations are required to represent a block of statements.

8. List any four standard data types supported by Python.

- NoneType
- Numeric types
- Sequences
- Sets
- Mappings

9. Differentiate bytes and byte array data type with example

The primary difference is that a bytes object is immutable, meaning that once created, you cannot modify its elements. By contrast, a bytearray object allows you to modify its elements.

bytes Datatype

The bytes datatype represents a group of byte numbers just like an array does. A byte number is any positive integer from 0 to 255 (inclusive). bytes array can store numbers in the range from 0 to 255 and it cannot even store negative numbers. For example,

```
elements = [10, 20, 0, 40, 15]  # this is a list of byte numbers
x = bytes(elements)  # convert the list into bytes array
print(x[0])   # display 0th element, i.e 10
```

bytearray Datatype

The bytearray datatype is similar to bytes datatype. The difference is that the bytes type array cannot be modified but the bytearray type array can be modified. It means any element or all the elements of the bytearray type can be modified. To create a bytearray type array, we can use the function bytearray as:

```
elements = [10, 20, 0, 40, 15]  # this is a list of byte numbers
x = bytearray(elements)  # convert the list into bytearray type array
print(x[0])   # display 0th element, i.e 10
```

We can modify or edit the elements of the bytearray. For example, we can write:

```
x[0] = 88  # replace 0th element by 88
x[1] = 99  # replace 1st element by 99
```

10. List any four types of sequences.

- str
- bytes
- bytearray
- list
- tuple
- range

11. Differentiate tuple and list with example

SR.NO.	LIST	TUPLE
1	Lists are mutable	Tuples are immutable
2	Implication of iterations is Time-consuming	The implication of iterations is comparatively Faster
3	The list is better for performing operations, such as insertion and deletion.	Tuple data type is appropriate for accessing the elements
4	Lists consume more memory	Tuple consume less memory as compared to the list
5	Lists have several built-in methods Ex: `list = [10, -20, 15.5, 'vijay', "Mary"]`	Tuple does not have many built-in methods. Ex: `tpl = (10, -20, 15.5, 'vijay', "Mary")`

12. Differentiate set and frozenset it with example.

Set is a most basic level datatype, It supports all the method operations of the set such as add(), remove(), and so on.

```
A = {1,2,3,4}
# Perform add() operation
A.add(8)
print(A)
#Output
>>>{1,2,3,4,8}
```

The Frozen set is immutable, it does not support any operations like add(), remove(), and so on.

```
A = frozenset([1,2,3,4])
# Perform add() operation
A.add(8)
print(A)
#output
Traceback (most recent call last):
File "<string>", line 9, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

13. How to retrieve the key and values of a dictionary? Give example.

14. How to determine the data type of a variable? Give the syntax and example

To know the datatype of a variable or object, we can use the type() function. For example, type(a) displays the datatype of the variable 'a'.

```
a=15
print(type(a))
<class 'int'>
```

15. What is the purpose of membership operators? Give example

The membership operators are useful to test for membership in a sequence such as strings, lists, tuples, or dictionaries. For example, if an element is found in this sequence or not, can be asserted using these operators. There are 2 membership operators:

- in
- not in

example:

```
names = ['Rani','Yamini','Ram']
for name in names:
    print (name)
```

16. What is the purpose of the pass statement? Give its syntax

In Python, pass is a null statement. The interpreter does not ignore a pass statement, but nothing happens and the statement results into no operation. The pass statement is useful when you don't write the implementation of a function but you want to implement it in the future.

Syntax: pass

17. What is the purpose of the assert statement? Give its syntax.

assert statement is used to check types, values of argument, and the function's output. assert statement is used as a debugging tool as it halts the program when an error occurs. Its syntax is: assert expression, message

18. What is the purpose of else suit in Python loops? Give example

Python allows the else keyword to be used with the for and while loops too. The else block appears after the body of the loop. The statements in the else block will be executed after all iterations are completed. The program exits the loop only after the else block is executed.

```
for i in range(5):
        print ("yes")
else:
        print ("no")
```

19. Give two methods of importing array modules.

- Importing entire array module using import statement:
  import array
- Importing array module by giving alias name as :
  import array as arr

20. What do you mean by array slicing? Give example

Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: [start:end] . We can also define the step, like this: [start:end:step] .

Ex: arr[1:4]

21. What is array indexing? Give example

Python arrays are variables that consist of more than one element. In order to access specific elements from an array, we use the method of array indexing. The first element starts with index 0 and followed by the second element which has index 1 and so on.

```
from array import *
x=array('i',[10,20,30,40])
n=len(x)
i=0
while i<n:
        print(x[i])
        i++
```

22. What is the purpose of reshape() and flatten() methods?

The reshape() method is useful to change the shape of an array. The new array should have the same number of elements as in the original array.

The flatten() method is useful to return a copy of the array collapsed into one dimension.

5 marks Questions

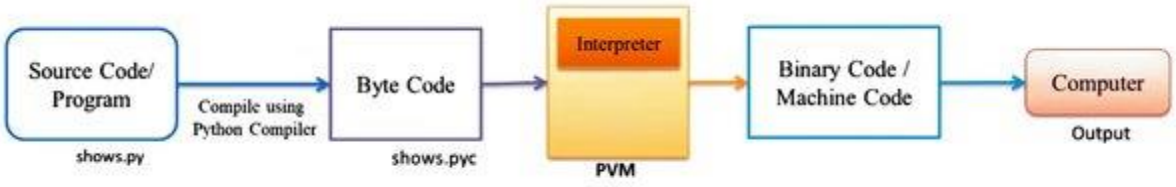
1. Explain any five features of Python

- Simple: Python is a simple programming language. When we read a Python program, we feel like reading English sentences. Hence, developing an understanding programs will become easy.
- Easy to learn: Python uses very few keywords. It's program use very simple structure. So developing programs in Python becomes easy.
- Open source: there is no need to pay for Python software. Python can be freely downloaded from its website. Its source code can be read. Modified and can be used in programs as desired by the programmers.
- High level language: Programming languages are of 2 types: low level and high level. High level languages use English words to develop program. These are easy to learn and use. Example: COBOL., PHP., Java. Python.
- Dynamically typed: in Python, we need not declare anything. An assignment statement binds a name to an object. And the object can be of any type.

2. Explain the execution of a Python program by PVM

Python Virtual Machine (PVM) is a program which provides programming environment. The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.

An interpreter which is present inside the PVM, converts the byte code(.pyc or .pyo) into machine code(0&1) and sends that machine code to the computer processor for execution.



The Python interpreter performs following tasks to execute a Python program :

- The interpreter reads a python code or instruction. Then it verifies that the instruction is well formatted, i.e. it checks the syntax of each line.If it encounters any error, it immediately halts the translation and shows an error message.
- If there is no error, i.e. if the python instruction or code is well formatted then the interpreter translates it into its equivalent form in intermediate language called "Byte code".Thus, after successful execution of Python script or code, it is completely translated into Byte code.
- Byte code is sent to the Python Virtual Machine(PVM).Here again the byte code is executed on PVM.If an error occurs during this execution then the execution is halted with an error message.

3. Explain any five flavors of Python

CPython:

This is the standard Python compiler implemented in C language. This is the Python software being downloaded and used by programmers directly from CPython . In this, any Python program is internally converted into byte code using C language functions. This byte code is run on the interpreter available in Python Virtual Machine (PVM) created in C language. The advantage is that it is possible to execute C and C++ functions and programs in CPython.

Jython:

This is earlier known as JPython. This is the implementation of Python programming language which is designed to run on Java platform. Jython compiler first compiles the Python program into Java byte code. This byte code is executed by Java Virtual Machine (JVM) to produce the output. Jython contains libraries which are useful for both Python and Java programmers.

RubyPython:

This is a bridge between the Ruby and Python interpreters. It encloses a Python interpreter inside Ruby applications.

Pythonxy:

This is pronounced as Python xy and written as Python(X,Y). This is the Python implementation that we get after adding scientific and engineering related packages.

AnacondaPython:

When Python is redeveloped for handling large-scale data processing, predictive analytics and scientific computing, it is called Anaconda Python. This implementation mainly focuses on large scale of data.

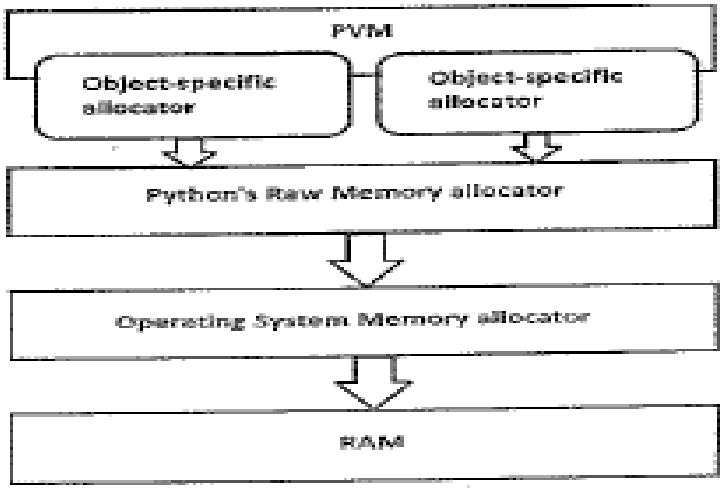
4. Explain the concept of Python Virtual Machine [refer long ans 2]

5. Write a note on memory management in Python

In Python, memory allocation and deallocation are done during runtime automatically. The programmer need not allocate memory while creating objects or deallocate memory while deleting the objects. Pythons PVM will take care of such issues.

In Python, strings list functions everything are considered as objects. Even modules are objects. For every object, memory should be allocated. Memory manager inside the PVM, allocates memory required for the objects created in a Python program. All these objects are stored on a separate memory called heap. Heap is the memory which is allocated during runtime. Its size depends on RAM of our computer and it can increase or decrease its size depending and it can increase or decrease its size depending on the requirement of the program.

The actual memory(RAM) is allocated by the underlying operating system. On top of the operating system, rock memory allocator overseas whether enough memory is available for storing objects. On the top of the raw memory allocator, there are several objects specific allocators operate on the same heap. These memory allocators will implement different types of management policies depending on the type of the objects.



6. Write a note on the purpose of Docstring in Python with an example

In Python, if we write strings inside “”” or ‘’’ and if these things are written as first statements in a module, class, function, or a method, then these strings are called documentation strings or docstrings. These docstrings are useful to create an API documentation file from a Python program. An API documentation file is a text file or HTML file that contains description of all the features of software, language, or the product.

When a new software is created, it is the duty of the developers to describe all the classes, modules, functions etc which are written in that software so that the user will be able to understand the software and use it in a proper way. These descriptions are provided in a separate file, either as a text file or an HTML file. So we can understand that the API documentation file is like a help file for the end user.

7. Explain numeric data type and bool data type in Python .

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. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.

Complex Numbers – Complex number is represented by complex class. It is specified as (real part) + (imaginary part)j. For example – 2+3j

Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool. True and False with capital ‘T’ and ‘F’ are valid booleans otherwise python will throw an error.

8. Explain the various sequence data types in Python with Syntax and example

- **str** : In Python, str represents string data type. A string is represented by a group of characters. Strange are enclosed in single or double-quotes. Both are valid.

  Example: str=”welcome”
  str=’welcome’

- **bytes** : the bytes datatype represents a group of byte numbers just like an array does. A byte number is any positive integer from 0 to 255. It cannot store negative numbers. We cannot modify or edit any element in the bytes array.

  ```
  Example: elements=[10,20,30,40]
  x=bytes(elements)
  print(x[0])
  ```

- **bytearray**: the bytearray datatype is similar to bytes datatype. The difference is that the bytes type array cannot be modified but the bytearray type can be modified. It means any elements of the bytearray type can be modified. To create a bytearray type array, we can use the function bytearray as:

  ```
  elements=[10,20,0,50]
  x=bytearray(elements)
  print(x[0])
  ```

  We can modify or edit the elements of the btearray as follows:

  ```
  x[0]=88
  x[1]=77
  ```

- **list**: A list represents a group of elements. The main difference between a list and an array is that a list can store different types of elements but an array can store only one type of elements. List can grow dynamically in memory. Buy the size of arrays is fixed and they cannot grow at runtime. They are represented using square brackets [ ] and the elements are separated using commas.

  List = [10, -20, 15.5, ‘vijay’ ,”ram”]

- **tuple:** A tuple is similar to a list. A tuple contains a group of elements which can be of different types. Elements in the tuple are separated by commas and enclosed in parentheses{} . Whereas the list elements can be modified, it is not possible to modify the tuple elements. That means a tuple can be treated as a read-only list. Let’s Create a tuple as;

  tp1 = (10,-20,10.5,’vijay’,”mary”)

  The individual elements of the tuple can be referenced using braces as tp1[1], tp1[2]…., Now,if we try to modify the 0^{th} element as:

  ```
  tp1[0]=99
  ```

- **range:** The range datatype represents a sequence of numbers. The numbers in the range are not modifiable. Generally, range is used for repeating a for loop for a specific number of times. To create a range of numbers, we can simply write:

  ```
  r=range(10)
  ```

9. List any five naming conventions in Python

- **Modules**: Modules names should be written in all lower case letters. When multuple words are used for a name, we should separate them using an underscore (_) .

- **Classes**: Each word of a class name should start with a capital letter. This rule is applicable for the classes created by us. Python's built-in class names use all lowercase words. When a class represents exception, then its name should end With a word 'Error'

- **Global variables or Module-level variables:** Global variables names should be all lower case letters. When multiple words are used for a name, we should separate them using an underscore (_)

- **Instance variables:** Instance variables names should be all lower case letters. When multiple words are used for a name, we should separate them using an underscore (_). Non-public instance variable name should begin with an underscore.

- **Functions:** Function names should be all lower case letters. When multiple words are used for a name, we should separate them using an underscore (_).

- **Methods**: Method names should be all lower case letters. When multiple words are used for a name, we should separate them using an underscore (_)

- **Method arguments**: In case of instance methods, their first argument name should be 'Self'. In case of class methods, their first argument name should be 'cls'.

- **Constants**: Constants names should be written in all capital letters. If a constant has several words, then each word should be separated by an underscore (_).

10. Write a note on assignment, unary minus operator in Python.

- **Assignment Operators**
  These operators are useful to store the right side value into a left side variable. They can also be used to simple arithmetic operations like addition,subtraction. etc., and then Store the result into a variable.

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b

>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

- **Unary Minus operator**
  The unary minus operator is denoted by the symbol minus (-). When this operator is used before a variable, its value is negated. That means if variable value is positive, it will be converted into negative For example,

```
n=10
print(-n)    #displays -10

num=-10
num = -num
Print(num)     #displays 10
```

11. Explain the Arithmetic operators and membership operators available in Python with example[refer notes for example]

- **Arithmetic operators:** These operators are used to perform basic arithmetic operations like addition, subtraction, division, etc.

Operator	Meaning
+	Addition operator. Adds two values.
-	Subtracts one value from another.
*	Multiplies values on either side of the operator.
/	Divides left operand by right operand.
%	Modulus gives the remainder of division.
**	Exponent operator calculates exponential power value.
//	Integer division performs division and gives only integer quotient.

- **Membership operators**
  in and not in are the membership operators; used to test whether a value or variable is in a sequence.

  in True if value is found in the sequence
  not in True if value is not found in the sequence

12. Explain the logical and relational operators with an example

- **Relational operators**

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y

- **Logical operators**

Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

13. Explain the bitwise operators

Bitwise operators:

These operators act on individual bits (O and I) of the operands. We can use bitwise operators directly on binary numbers or on integers also. When we use these operators on integers, these numbers are converted into bits (binary number system) and then bitwise operators act upon those bits. The results given by these operators are always in the form of integers.

We use decimal number system in our daily life. This number system consists Of 10 digits from O to 9. We count all numbers using these 10 digits only, But in case Of binary system that is used computers internally, there only 2 digits, i.e. 0 and 1 which are called bits[binary digits]. All values are represented only using these two bits. It is possible to convert a number into binary number and Vice versa.

Bitwise operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

14. Explain the following mathematical functions with an example:

a) ceil(x) : Return the Ceiling value. It is the smallest integer, greater or equal to the number x.

b) floor(x) : Return the Floor value. It is the largest integer, less or equal to the number x.

c) fabs(x) : Returns the absolute value of x.

d) fmod(x) : fmod() function is one of the Standard math library function in Python, which is used to calculate the Module of the specified given arguments.

e) pow(x,y): Return the x to the power y value.

15. How to read different types of input from the keyboard. Give example

To accept input from keyboard, Python provides the input() function. This function takes a value from the keyboard and returns it as a string. It is better to display a message to the user so that the user understands what to enter. This can be done by writing a message inside the input() function as:

```
str=input("Enter your name")
Enter your name: Anu

print(str)
Anu
```

We can use the int() function before the input() function to accept an integer from the keyboard as:

```
a=int(input("Enter the age: "))
Enter the age: 22

print(a)
22
```

Similarly, to accept a float value from the keyboard, we can use the float() function along with the input() function as:

```
m1=float(input("Enter the marks: "))
Enter the age: 15.5

print(m1)
```

15.5

16. How to read multiple values from the keyboard using the split() method. Give example.

Using split() method :

This function helps in getting multiple inputs from users. It breaks the given input by the specified separator. If a separator is not provided then any white space is a separator. Generally, users use a split() method to split a Python string but one can use it in taking multiple inputs.

Syntax : input().split(separator, maxsplit)

Example:
```
x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
print()
```

Output:
Enter a tree value: 30 10 20
Total number of students: 30
Number of boys: 10
Number of girls: 20

17. Write a note on different formats of print() function

18. Explain passing command-line arguments with a program example

19. Explain the different functions used to create an Array using the numpy package.

Numpy is a package that contains several classes,functions,variables etc. to deal with a scientific calculations in Python. Numpy is useful to create abd also process single and multidimensional arrays. Numoy also contains a huge library of mathematical functions. Creating an array in numpy can be done in the following ways:

- **Using linspace() function:** It is used to create an array with evenly spaced points between a starting point and ending point.

  linspace(start,stop,n)

  ex: a=linspace(0,10,5)

- **Using logspace():** this function produces the evenly spaced points on a logarithmically spaced scale.

  logspace(start,stop,n)

  ex: a=logspace(1,4,5)

- **Using arange() function:** this is same as range() function in python.

  arange(start,stop,stepsize)

  ex: arange(10)

  arange(1,10,3)

- **Using zeros() and ones() function:** we can use zeros() function to create an array with all zeros. The ones() function is used to creater an array with all 1s. they are written in the following format:

zeros(n,datatype)
ones(n,datatype)

20. Explain any five attributes of numpy array

(1) ndarray.ndim: ndim represents the number of dimensions (axes) of the ndarray.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of ndim will be 2. This ndarray has two dimensions (axes) - rows (axis=0) and columns (axis=1)

(2) ndarray.shape : shape is a tuple of integers representing the size of the ndarray in each dimension.

e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], value of shape will be (2,3) because this ndarray has two dimensions - rows and columns - and the number of rows is 2 and the number of columns is 3

(3) ndarray.size: size is the total number of elements in the ndarray. It is equal to the product of elements of the shape. e.g. for this 2-dimensional array [[3,4,6], [0,8,1]], shape is (2,3), size will be product (multiplication) of 2 and 3 i.e. (2*3) = 6. Hence, the size is 6.

(4) ndarray.dtype: dtype tells the data type of the elements of a NumPy array. In NumPy array, all the elements have the same data type.

e.g. for this NumPy array [[3,4,6], [0,8,1]], dtype will be int64

(5) ndarray.itemsize : itemsize returns the size (in bytes) of each element of a NumPy array.

e.g. for this NumPy array [[3,4,6], [0,8,1]], itemsize will be 8, because this array consists of integers and size of integer (in bytes) is 8 bytes.

UNIT II

Questions carrying 2 Marks

1. How to create multiline strings in Python? Give example.

Use triple quotes to create a multiline string. It is the simplest method to let a long string split into different lines. You will need to enclose it with a pair of Triple quotes, one at the start and second in the end. Anything inside the enclosing Triple quotes will become part of one multiline string.

Ex: """Learn Python
Programming"""

2. Give the general format and example of slicing strings.

Slicing the Strings :A slice represents a part or piece Of a string. The format of slicing is:
Stringname[start: Stop: stepsize]
Ex: str='Core Python'
str[0:9:1]
Core pyth

3. How to compare and Concatenate two strings in Python? Give one example for each.

Comparing Strings

We can use the relational operators like > ,<,<=,>=,= Or != operators to compare two strings. They return Boolean value, i.e. either True or False depending on the strings being compared

```
S1='Box
S2='Boy'
if(s1==s2):
    print('Both are same')
else:
    print('Not same')
```

You can concatenate string literals ('...' or "...") and string variables with the + operator.

```
s = 'aaa' + 'bbb' + 'ccc'
print(s)
# aaabbbccc

s1 = 'aaa'
s2 = 'bbb'
s3 = 'ccc'

s = s1 + s2 + s3
print(s)
# aaabbbccc

s = s1 + s2 + s3 + 'ddd'
print(s)
# aaabbbcccddd
```

4. Give the output of the following Python code:

```
str1 = 'This is Pyhton'
print "Slice of String : ", str1[1 : 4 : 1]
print "Slice of String : ", str1[0 : -1 : 2]
```

output:
Slice of String : his
Slice of String : Ti sPho

5. Give the syntax and purpose of the find() method.

The find() method returns -1 if the sub string is not found in the main string. The format of find() method is:
mainstring.find(substring, beginning, ending)

6. Differentiate count and len methods of string.

length method= len() => It's return number of element from value of variable.

count method = count() =>It's return how many times appeared from value of variable which you are specified value.

7. List 4 types of actual arguments use in Python function calls.

The actual arguments used in a function call are of 4 types:

- Positional arguments.
- Keyword arguments.
- Default arguments.
- Variable length arguments.

8. What is the purpose of the global keyword?

In Python, global keyword allows you to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

9. What is Anonymous / Lambda function? How it is defined in Python?

In Python, an anonymous function is a function that is defined without a name. While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

10. What is a function decorator?

Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it. But before diving deep into decorators let us understand some concepts that will come in handy in learning the decorators.

11. What is generator function?

Python generators are a simple way of creating iterators. All the work we mentioned above are automatically handled by generators in Python. Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

12. Write a python code to create a list using the range() function

```
# Create a list in a range of 10-20
My_list = [*range(10, 21, 1)]

# Print the list
print(My_list)
```

13. How to add and remove list elements? Give example

Appending an element means adding an element at the end of the list. To append a new element to the list, we should use the append() method.

```
lst=list(range(1,5))
print(lst)
```

We can delete an element using the remove() method.

```
lst.remove(11)
print(lst)
```

14. What are indexing and negative indexing in Tuple?

Indexing represents the position number of the element in the tuple. Now, tup[0], represents 0th element,tup[1] represents 1st element and so on.

Similarly, negative indexing is also possible. Like, tup[-1] represents last element, tup[-2] represents the second element from the end and so on.

15. List two advantages of Tuple over List

The advantages of tuples over the lists are as follows:
1. Tuples are faster than lists.
2. Tuples make the code safe from any accidental modification.
If a data is needed in a program which is not supposed to be changed, then it is better to put it in 'tuples' than in 'list'.

16. Write the output of the given python code :
aList = [123, 'xyz', 'zara', 'abc'];
aList.insert (3,2009)
print ("Final List:", aList)

output:
Final List: [123, 'xyz', 'zara', 2009, 'abc']

17. What is the output of

print (tuple[1:3])
if tuple = ('abcd', 786 , 2.23, 'john', 70.2)?

Output: (786, 2.23)

18. What is a dictionary? Give example

A dictionary represents a group of elements arranged in the form of key-value pairs. In the dictionary, the first element is considered as 'key' and the immediate next element is taken as its 'value'.

Ex: dict = {'Name': 'Chandra', 'Id': 200, 'Salary': 9080.50}

19. What is meant by key-value pairs in a dictionary?

Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable

20. Write a Python code to print key-value pairs from a dictionary using for loop

```
colors = {'r': "Red", 'g': "Green", 'b': "Blue", 'w': "White"}
for k in colors:
  print (k)
for k in colors:
  print (colors[k])
for k, v in colors.items():
  print('Key= {} Value= {}'. format(k, v))
```

21. Write a Python code to convert a string into Dictionary.

```
str = "Vijay=23,Ganesh=20,Lakshmi=19,Nikhil=22"
lst=[]
for x in str.split(','):
```

```
    y= x.split('=')
    lst.append(y)
d = dict(lst)
d1={}
for k, v in d.items():
    d1[k] = int(v)
print(d1)
```

22. How does del operation work on dictionaries? Give an example.

We can use del on a dictionary. We pass the key we want to remove. It removes both the key and its associated value. Only one entry can be removed at a time.

Ex:

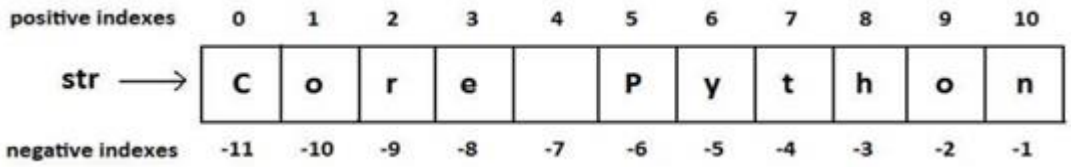
```
colors = {"red" : 100, "blue" : 50, "purple" : 75}
# Delete the pair with a key of "red."
del colors["red"]
print(colors)

output: {'blue': 50, 'purple': 75}
```

5 marks Questions

1. Explain different ways of slicing strings with examples

A slice represents a part or piece of a string. The format of slicing is:
stringname[start: stop: stepsize]
If 'start' and 'stop' are not specified, then slicing is done from 0th to n-1th elements. If 'stepsize' is not written, then it is taken to be 1.
str = 'Core Python'
str = 'Core Python'




```
print( str[0:9:1] )   #'Core Pyth'
print( str[0:9:2] )   # 'Cr yh'
print( str[::] )   # 'Core Python'
print( str[2:4:1] )   # 're'
print( str[::2] )   #'Cr yhn'
print( str[2::] )   # 're Python'
print( str[:4:] )   # 'Core'
print( str[-4:-1] )   # 'tho'
print( str[-6::] )   # 'Python'
print( str[-1:-4:-1] )   # 'noh'
print( str[-1::-1] )   # 'nohtyPeroC'
print( str[::-1] )   # 'nohtyPeroC'
```

2. With an example explain find() and index() methods of the string object.

Python find() method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match.

find(sub[, start[, end]])

Parameters

- **sub** : substring
- **start** : start index a range
- **end** : last index of the range

ex:

```
 # Python find() function example
# Variable declaration
str = "Welcome to the Javatpoint."
# Calling function
str2 = str.find("the")
# Displaying result
print(str2)
```

Python string method index() determines if string str occurs in string or in a substring of string if starting index beg and ending index end are given. This method is same as find(), but raises an exception if sub is not found.

Syntax: str.index(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 length of the string.

Ex:

```
str1 = "this is string example....wow!!!";
str2 = "exam";

print str1.index(str2)
print str1.index(str2, 10)
print str1.index(str2, 40)
```

3. Explain string splitting and string joining with examples.

split() method in Python split a string into a list of strings after breaking the given string by the specified separator.

Syntax : str.split(separator, maxsplit)

Parameters :

separator : This is a delimiter. The string splits at this specified separator. If is not provided then any white space is a separator.

maxsplit : It is a number, which tells us to split the string into maximum of provided number of times. If it is not provided then the default is -1 that means there is no limit.

Ex:

```
text = 'geeks for geeks'
  # Splits at space
print(text.split())
```

```
word = 'geeks, for, geeks'
# Splits at ','
print(word.split(','))
word = 'geeks:for:geeks'
# Splitting at ':'
print(word.split(':'))
```

When a group of strings are given. it is possible to join them all and make a single string. For this purpose, we can use join() method as: separator .join(str)
where, the 'separator' represents the character to be used between the strings in the output. 'str' represents a tuple Or list of strings. In the following example, are taking a tuple that contains 3 strings as:

```
('one', 'two', 'three')
```

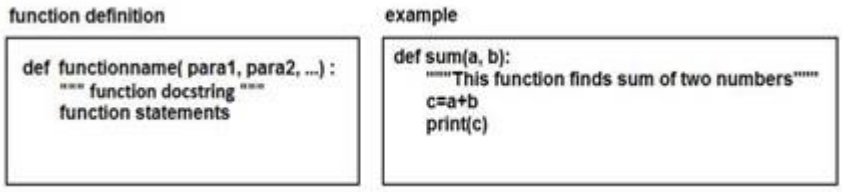
Wc want to join the three strings and form a single string. Also, we want to use hyphen (-)between the three strings in the output. The join() method can be written as:

```
str="-".join(str)
Print(str1)
```

4. Explain with syntax and example defining, calling function in Python.

Defining a Function

We can define a function using the keyword def followed by function name. After the function name, we should write parentheses () which may contain parameters. Consider the syntax of function, as shown below



we can write a function to add two values as:
def sum(a, b):
Here, **'def'** represents the starting of function definition. 'sum' is the name of the function. After this name, **parentheses ()** are compulsory as they denote that it is a function and not a variable or something else In the parentheses, we wrote two **variables 'a' and 'b'**. These variables are called **'parameters'**. A parameter is a variable that receives data from outside into a function. So, this function can receive two values from outside and those values are stored in the variables 'a' and 'b'.
After parentheses, we put a colon (:) that represents the beginning of the function body. The function body contains a group of statements called 'suite'.

Calling a Function

A function cannot run on its own. It runs only when we call it. So, the next step is to call the function using its name. While calling the function, we should pass the necessary values to the function in the parentheses as:
sum(10, 15)
Here, we are calling the 'sum' function and passing two values 10 and 15 to that function. When this statement is executed, the Python interpreter jumps to the function definition and copies the values 10 and 15 into the parameters 'a' and 'b' respectively.
A function that accepts two values and finds their sum.

```
def sum(a, b):
  c = a+b
```

```
print('Sum=', c)
sum(10, 15)
sum(1.5, 10.75)
```

5. With an example explain returning multiple values from a function.

A function returns a single value in the programming languages like C or Java. But in Python, a function can return multiple values. When a function calculates multiple results and wants to return the results, we can use the return statement as: return a, b, c

Here, three values which are in 'a', 'b' and 'c' are returned. These values are returned by the function as a tuple. Please remember a tuple is like a list that contains a group of elements. To grab these values, we can use three variables at the time of calling the function as:

```
x, y, z = function()
def sum_sub(a, b):
c = a + b
d = a - b
d = a - b
return c, d
```

A function that returns the results of addition, subtraction, multiplication and division.

```
def sum_sub_mul_div(a, b):
  c = a + b
  d = a - b
  e = a * b
  f = a/ b
  return c, d, e, f
t = sum_sub_mul_div(10, 5)
print('The results are:')
for i in t:
print(i, end='\t')
```

6. Explain different types of actual arguments used in the function call with examples.

The actual arguments used in a function call are of 4 types:

1. Positional arguments
2. Keyword arguments
3. Default arguments
4. Variable length arguments

1) Positional Arguments
These are the arguments passed to a function in correct positional order. Here, the number of arguments and their positions in the function definition should match exactly with the number and position of the argument in the function call. For example, take a function definition with two arguments as:

```
def attach(s1, s2)
attach('New', 'York')
```

Also, if we try to pass more than or less than 2 strings, there will be an error. For example, if we call the function by passing 3 strings as:

```
attach('New', 'York', 'City') #results error
```

2) Keyword Arguments

Keyword arguments are arguments that identify the parameters by their names. For example, the definition of a function that displays grocery item and its price can be written as:
def grocery(item, price):
At the time of calling this function, we have to pass two values and we can mention which value is for what. For example,
grocery(item='Sugar', price=50.75)
Here, we are mentioning a keyword 'item' and its value and then another keyword 'price' and its value. Please observe these keywords are nothing but the parameter names which receive these values. We can change the order of the arguments as:
grocery(price=88.00, item='Oil')

3.Default Arguments
We can mention some default value for the function parameters in the definition. Let's take the definition of grocery() function as:
def grocery(item, price=40.00):
Here, the first argument is 'item' whose default value is not mentioned. But the second argument is 'price' and its default value is mentioned to be 40.00.
At the time of calling this function, if we do not pass 'price' value, then the default value of 40.00 is taken. If we mention the 'price' value, then that mentioned value is utilized. So, a default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

4.Variable Length Arguments
Sometimes, the programmer does not know how many values a function may receive. In that case, the programmer cannot decide how many arguments to be given in the function definition.
If the programmer wants to develop a function that can accept 'n' arguments, that is also possible in Python. For this purpose, a variable length argument is used in the function definition. A variable length argument is an argument that can accept any number of values.
The variable length argument is written with a' * ' symbol before it in the function definition as:
def add(farg, *args):
Here, 'farg' is the formal argument and '*args' represents variable length argument. We can pass 1 or more values to this '*args' and it will store them all in a tuple.

7. Explain How to define lambdas in Python with an example program.

A function without a name is called 'anonymous function'. So far, the functions we wrote were defined using the keyword 'def'.But anonymous functions are not defined using 'def'. They are defined using the keyword lambda and hence they are also called 'Lambda functions'. Let's take a normal function that returns square of a given value.

```
def square(x):
return x*x
```

The same function can be written as anonymous function as: lambda x: x*x
The format of lambda functions is: lambda argument_list: expression
Normally, if a function returns some value, we assign that value to a variable as:

```
y = square(5)
```

But, lambda functions return a function and hence they should be assigned to a function as:

```
f = lambda x: x*x
```

Here, 'f' is the function name to which the lambda expression is assigned. Now, if we call the function f() as:

```
value = f(5)
```

Now, 'value' contains the square value of 5, i.e. 25
Ex:

```
f = lambda x: x*x
```

```
value = f(5)
print('Square of 5 =', value)
```

8. Explain use of filter() , map() and reduce() methods with suitable examples

Using Lambdas with filter() Function

The filter() function is useful to filter out the elements of a sequence depending on the result of a function. We should supply a function and a sequence to the filter() function as:

filter(function, sequence)

Here, the 'function' represents a function name that may return either True or False; and 'sequence' represents a list, string or tuple. The 'function' is applied to every element of the 'sequence' and when the function returns True, the element is extracted otherwise it is ignored. Before using the filter() function, let's first write a function that tests whether a given number is even or odd.

Ex:

```
def is_even(x):
  if x%2==0:
    return True
  else:
    return False
```

Using Lambdas with map()

Function The map() function is similar to filter() function but it acts on each element of the sequence and perhaps changes the elements. The format of map() function is:

map(function, sequence)

The 'function' performs a specified operation on all the elements of the sequence and the modified elements are returned which can be stored in another sequence.

Ex:

```
def squares(x):
  return x*x
lst = [1, 2, 3, 4, 5]
lst1 = list(map(squares, lst))
print(lst1)
```

Using Lambdas with reduce() Function

The reduce() function reduces a sequence of elements to a single value by processing the elements according to a function supplied. The reduce() function is used in the format:

reduce(function, sequence)

For example, we write the reduce() function with a lambda expression, as:

```
lst = [1, 2, 3, 4, 5]
reduce(lambda x, y: x*y, lst)
```

The lambda function is taking two arguments and returning their product. Hence, starting from the 0th element of the list 'lst', the first two elements are multiplied and the product is obtained. Then this product is multiplied with the third element and the product is obtained.

9. Explain function decorators with an example program.

Python developers can extend and modify the behavior of a callable functions, methods or classes without permanently modifying the callable itself by using decorators. In short we can say they are callable objects which are used to modify functions or classes.

Function decorators are functions which accepts function references as arguments and adds a wrapper around them and returns the function with the wrapper as a new function.
Ex:

```
@decorator
def func(arg):
  return "value"
```

So from above, we can see a decorator is simply another function which takes a function as an argument and returns one.

Decorators basically “decoratre” or “wrap” another function and let you execute code before and after the wrapped function runs.

10. Explain different ways of creating lists with examples

We can use range() function to generate a sequence of integers which can be stored in a list. The format of the range() function is:

```
range(start, stop, stepsize)
```

If we do not mention the 'start', it is assumed to be 0 and the 'stepsize' is taken as 1.
Ex:

```
list1 = range(10)
for i in list1:
  print(i,',', end='')
print() #throw cursor to next line
#create list with integers from 5 to 9
list2 = range(5, 10)
for i in list2:
  print(i,',', end='')
print()
#create a list with odd numbers from 5 to 9
list3 = range(5, 10, 2) #step size is 2
for i in list3:
  print(i, ',' , end='')
```

11. Explain Indexing and Slicing operation for the list with example in python.

Indexing means referring to an element of an iterable by its position within the iterable. Each of a string’s characters corresponds to an index number and each character can be accessed using their index number.

We can access characters in a String in Two ways :

- Accessing Characters by Positive Index Number
- Accessing Characters by Negative Index Number

1. Accessing Characters by Positive Index Number: In this type of Indexing, we pass a Positive index(which we want to access) in square brackets. The index number start from index number 0 (which denotes the first character of a string).ex:

```
str = "Geeks for Geeks !"
 # accessing the character of str at 0th index
print(str[0])
```

2. Accessing Characters by Negative Index Number : In this type of Indexing, we pass the Negative index(which we want to access) in square brackets. Here the index number starts from index number -1 (which denotes the last character of a string). Ex:

```
# declaring the string
str = "Geeks for Geeks !"
# accessing the character of str at last index
print(str[-1])
```

Slicing in Python is a feature that enables accessing parts of sequence. In slicing string, we create a substring, which is essentially a string that exists within another string. We use slicing when we require a part of string and not the complete string.

Syntax :

string[start : end : step]

start : We provide the starting index.

end : We provide the end index(this is not included in substring).

step : It is an optional argument that determines the increment between each index for slicing.

Ex:

```
str ="Geeks for Geeks !"
# slicing using indexing sequence
print(str[: 3])
print(str[1 : 5 : 2])
```

12. Explain any five methods to process the list.

append(): Used for appending and adding elements to List.It is used to add elements to the last position of List.

Syntax: list.append (element)

Ex: List = ['Mathematics', 'chemistry', 1997, 2000]

List.append(20544)

print(List)

insert(): Inserts an elements at specified position.

Syntax: list.insert(<position, element)

Ex:

List = ['Mathematics', 'chemistry', 1997, 2000]

Insert at index 2 value 10087

List.insert(2,10087)

print(List)

extend(): Adds contents to List2 to the end of List1.

Syntax: List1.extend(List2)

Ex:

List1 = [1, 2, 3]

List2 = [2, 3, 4, 5]

Add List2 to List1

List1.extend(List2)

print(List1)

sum() : Calculates sum of all the elements of List.
Syntax: sum(List)
Ex: List = [1, 2, 3, 4, 5]
print(sum(List))

count(): Calculates total occurrence of given element of List.
Syntax: List.count(element)
Ex:
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.count(1))

13. Explain different ways of creating tuple and accessing tuple elements.

Creating Tuples:
We can create a tuple by writing elements separated by commas inside parentheses (). The elements can be of same datatype or different types. For example, to create an empty tuple, we can simply write empty parenthesis, as:

```
tup1 = () #empty tuple
```

If we want to create a tuple with only one element, we can mention that element in parentheses and after that a comma is needed, as:

```
tup2 = (10,)
```

Here is a tuple with different types of elements:

```
tup3 = (10, 20, -30.1, 40.5, 'Hyderabad', 'New Delhi')
```

We can create a tuple with only one of type of elements also, like the following:

```
tup4 = (10, 20, 30) #tuple with integers
```

If we do not mention any brackets and write the elements separating them by commas, then they are taken by default as a tuple. See the following example:

```
tup5 = 1, 2, 3, 4 #no braces
```

The point to remember is that if do not use any brackets, it will become a tuple and not a list or any other datatype.
It is also possible to create a tuple from a list. This is done by converting a list into a tuple using the tuple() function.

```
list = [1, 2, 3] #take a list
tpl = tuple(list) #convert list into tuple
print(tpl) #display tuple
```

Another way to create a tuple is by using range() function that returns a sequence.

```
tpl= tuple(range(4, 9, 2)) #numbers from 4 to 8 in steps of 2
print(tpl)
```

The preceding statements will give: (4, 6, 8)

Accessing the Tuple Elements:
Accessing the elements from a tuple can be done using indexing or slicing. This is same as that of a list.

```
tup = (50,60,70,80,90,100)
```

Indexing represents the position number of the element in the tuple. Now, tup[0] represents the 0th element, tup[1] represents the 1st element and so on.

```
print(tup[0])
```

The preceding will give: 50

```
print(tup[:])
print(tup[1:4])
```

```
print(tup[::2])
print(tup[-4:-1])
student = (10, 'Naresh kumar', 50,60,65,61,70)
rno, name = student[0:2]
print(rno, name)
marks = student[2:7]
for i in marks:
print(i)
```

14. Explain any five methods to process the tuple elements.

Function	Example	Description
len()	len(tpl)	Returns the number of elements in the tuple.
min()	min(tpl)	Returns the smallest element in the tuple.
max()	max(tpl)	Returns the biggest element in the tuple.
count()	tpl.count(x)	Returns how many times the element 'x' is found in tpl.
index()	tpl.index(x)	Returns the first occurrence of the element 'x' in tpl. Raises ValueError if 'x' is not found in the tuple.
sorted()	sorted(tpl)	Sorts the elements of the tuple into ascending order.sorted(tpl,reverse=True) will sort in reverse order.

15. Explain any five methods to process elements of the dictionary.

Method	Example	Description
clear()	d.clear()	Removes all key-value pairs from dictionary 'd'.
copy()	d1.d.copy()	Copies all elements from 'd' into a new dictionary 'd1'.
fromkeys()	d.fromkeys(s[,v])	Create a new dictionary with keys from sequence 's' and values all set to 'v'.
get()	d.get(k[,v])	Returns the value associated with key 'k'.If key is not found,it returns 'v'.
items()	d.items()	Returns an object that contains key-value pairs of 'd'.The pairs are stored as tuples in the object.
keys()	d.keys()	Returns a sequence of keys from the dictionary 'd'.
values()	d.values()	Returns a sequence of values from the dictionary 'd'.
update()	d.update(x)	Adds all elements from dictionary 'x' to 'd'.
pop()	d.pop(k[,v])	Removes the key 'k' and its value from 'd' and returns the value.If key is not found,then the value 'v' is returned.If key is not found and 'v' is not mentioned then 'KeyError' is raised.
setdefault()	d.setdefault(k[,v])	If key 'k' is found,its value is returned.If key is not found,then the k,v pair is stored into the dictionary 'd'.

16. Write a note on converting lists and strings into a dictionary.

Python Lists to Dictionary

When we have two lists, it is possible to convert them into a dictionary. For example, we have two lists containing names of countries and names of their capital cities.

countries = ["USA", "India", "Germany", "France"]
cities = ['Washington', 'New Delhi', 'Berlin', 'Paris']

We want to create a dictionary out of these two lists by taking the elements of 'countries' list as keys and of 'cities' list as values.
There are two steps involved to convert the lists into a dictionary. The first step is to create a 'zip' class object by passing the two lists to zip() function as:
z = zip(countries, cities)
The zip() function is useful to convert the sequences into a zip class object.
The second step is to convert the zip object into a dictionary by using dict() function.
d = dict(z)

ex:

```
countries = ["USA", "India", "Germany", "France"]
cities = ['Washington', 'New Delhi', 'Berlin', 'Paris']
z = zip(countries, cities)
d = dict(z)
print('{:15s} -- {:15s}'.format('COUNTRY', 'CAPITAL'))
for k in d:
  print('{:15s} -- {:15s}'.format(k, d[k]))
```

Python Strings to Dictionary
When a string is given with key and value pairs separated by some delimiter (or separator) like a comma (,) we can convert the string into a dictionary and use it as dictionary.
str = "Sujay=23, Rajesh=20, Lahari=19,Nithin=22"
To convert such a string into a dictionary, we have to follow 3 steps.
First, we should split the string into pieces where a comma is found using split() method and then brake the string at equals (=) symbol. This can be done using a for loop as:

```
for x in str.split(','):
  y= x.split('=')
```

Each piece of the string is available in 'y'. The second step is to store these pieces into a list 'lst' using append() method as: lst.append(y)
The third step is to convert the list into a dictionary 'd' using dict() function as: d = dict(lst)

UNIT III

Questions carrying 2 Marks

1. What is the purpose of the self variable?

we can use 'self' to refer to all the instance variables and instance methods. When an instance to the class is created, the instance name contains the memory location of the instance. This memory location is internally passed to 'self'.

2. Give the General format of defining constructor in Python.

Constructors are generally used for instantiating an object. The task of constructors is to initialize(assign values) to the data members of the class when an object of the class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

3. Differentiate between accessor methods and mutator methods.

- Accessor methods simply access or read data of the variables. They do not modify the data in the variables. Accessor methods are generally written in the form of getXXX() and hence they are also called getter methods.

- mutator methods are the methods which not only read the data but also modify them. They are written in the form of setXXX() and hence they are also called setter methods.

4. What is an accessor method? Give example.

Accessor methods simply access or read data of the variables. They do not modify the data in the variables. Accessor methods are generally written in the form of getXXX() and hence they are also called getter methods. For example,

```
def getName(self):
    return self.name
```

5. What is a mutator method? Give example.

mutator methods are the methods which not only read the data but also modify them. They are written in the form of setXXX() and hence they are also called setter methods. For example,

```
def setName(self, name):
  self.name = name
```

6. Differentiate between an instance variable and a class variable.

Instance variables are declared without static keyword. Class variables are common to all instances of a class. These variables are shared between the objects of a class. Instance variables are not shared between the objects of a class.

7. How to define a static method? Give example

Static methods are written with a decorator @staticmethod above them. Static methods are called in the form of classname.method().

Ex:

```
class Calculator:
  def add_numbers(num1, num2):
    return num1 + num2
# convert add_numbers() to static method
Calculator.add_numbers = staticmethod(Calculator.add_numbers)
sum = Calculator.add_numbers(5, 7)
print('Sum:', sum)
# Output: Sum: 12
```

8. Define method overloading and method overriding.

In the method overloading, methods or functions must have the same name and different signatures. Whereas in the method overriding, methods or functions must have the same name and same signatures.

9. What is polymorphism?

The literal meaning of polymorphism is the condition of occurrence in different forms. Polymorphism is a very important concept in programming. It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.

10. What is the purpose of the super() method?

The super() function is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the parent class.

11. What is an abstract class?

A class is called an Abstract class if it contains one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and its abstract methods must be implemented by its subclasses

12. What is an exception? List any two built-in exceptions in Python.

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

Exception	Description
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined

13. What is a regular expression? Give an example

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.
Ex: re.compile(r'm\w\w')

14. What do \w and $ represent in the regular expression?

\w Matches word characters.
$ Matches end of line.

15. Write a regular expression that matches a 5-character string starting with 'a' and ends with 's'

16. Write the output of the following Python code:

```
s= 'one two three four five nine ten eight seven 8 9'
result=re.findall(r '\b\w{4,}\b',s)
print(result)
```

output: ['three', 'four', 'five', 'nine', 'eight', 'seven']

17. List any two different ways to create a thread. [write any 2]

- Creating a thread without using a class
- Creating a thread by creating a sub class to Thread class
- Creating a thread without creating subclass to Thread class

18. What is a thread? Which method is used to return the name of the currently Running Thread?

Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

getName() − The getName() method returns the name of a thread.

19. What is the daemon thread?

The threads which are always going to run in the background that provides supports to main or non-daemon threads, those background executing threads are considered as Daemon Threads. The Daemon Thread does not block the main thread from exiting and continues to run in the background

20. What is the use of the join () method in thread class?

join() is used to join a thread with the main thread i.e. when join() is used for a particular thread the main thread will stop executing until the execution of joined thread is complete.

5 marks Questions

1. What is a class? How to create a class and object in Python? Give example

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together.

Syntax:

```
class classname:
'optional class documentation string'
class suite
```

The class has a documentation string, which can be accessed via class name. The class suit consists of a component statements defining class members, data attributes and functions.

Ex:

```
class Employee
    empcount=0
    def __init__(self,name,sal):
        self.name=name
        self.sal=sal
    Employee.empcount+=1
```

Creating instance objects: To create instance of a class, you call the class using class name and pass in whatever argument its __init__ method accepts.

Class Objects

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.

An object consists of :

- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:

```
class Dog:
```

```
    # A simple class
    # attribute
    attr1 = "mammal"
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)

# Driver code
# Object instantiation
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr1)
Rodger.fun()
```

In the above example, an object is created which is basically a dog named Rodger. This class only has two class attributes that tell us that Rodger is a dog and a mammal.

2. Write a note on the Constructor and self-variable.

Python Constructor: A constructor is a special method that is used to initialize the instance variables of a class. In the constructor, we create the instance variables and initialize them with some starting values. The first parameter of the constructor will be 'self' variable that contains the memory address of the instance. For example,

```
def __init__(self):
    self.name = 'Vishnu'
    self.marks = 900
```

Here, the constructor has only one parameter, i.e. 'self'. Using 'self.name' and 'self.marks', we can access the instance variables of the class.

Python Self Variable

'self' is a default variable that contains the memory address of the instance of the current class. So, we can use 'self' to refer to all the instance variables and instance methods. When an instance to the class is created, the instance name contains the memory location of the instance. This memory location is internally passed to 'self'.

1. The 'self' variable is used as first parameter in the constructor as:
   def __init__(self):

2. 'self' can be used as first parameter in the instance methods as:
   def talk(self):

Here, talk() is instance method as it acts on the instance variables. If this method wants to act on the instance variables, it should know the memory location of the instance variables. That memory location is by default available to the talk() method through 'self'.

3. Explain the different types of variables in Python with the example.

The variables are of 2 types:

- Instance variable
- Class variable or static variable

Instance variables are the variables whose separate copy is created in every instance. For example, if X is an instance variable and if we create 3 instances, there will be 3 copies of X in these 3 instances. When we modify the copy of X in any instance, it will not modify the other 2 copies.

```
#instance example
class Sample:
        def __init__(self):
                self.x=10
#create 2 instances
s1=Sample()
s2=Sample()
print('x in s1= ',s1.x)
print('x in s2= ',s2.x)

#modify x in s1
s1.modify()
print('x in s1= ',s1.x)
print('x in s2= ',s2.x)
```

Unlike instance variables, class variables are the variables whose single copy is available to all the instances of the class. If we modify the copy of class variable in an instance, it will modify all the copies in the other instances. For example, if X is a class variable and if we create 3 instances. The same copy of X is passed to these 3 instances. When we modify the copy of x in any instance using a class method, the modified copy is sent to the other 2 instances. Class variables are also called static variables.

```
class Sample:
        x=10

        @classmethod
        def modify(cls):
                cls.x+=1

#create 2 instances
s1=Sample()
s2=Sample()
print('x in s1= ',s1.x)
print('x in s2= ',s2.x)

#modify x in s1
s1.modify()
print('x in s1= ',s1.x)
print('x in s2= ',s2.x)
```

4. Explain the different types of methods in Python with an example.

The purpose of a method is to process the variables provided in the class or in the method. We already know that the variables declared in the class are called class variables (or static variables) and the variables declared in the constructor are called instance variables. We can classify the methods in the following 3 types:

1. Instance Methods
    - (a)Accessor methods
    - (b)Mutator methods
2. Class Methods
3. Static Methods

Instance Methods

Instance methods are the methods which act upon the instance variables of the class. Instance methods are bound to instances (or objects) and hence called as:
instancename.method().
Since instance variables are available in the instance, instance methods need to know the memory address of the instance. This is provided through 'self' variable by default as first parameter for the instance method. While calling the instance methods, we need not pass any value to the 'self' variable.

Instance methods are of two types:

(a)accessor methods and
(b)mutator methods.

Accessor methods simply access or read data of the variables. They do not modify the data in the variables. Accessor methods are generally written in the form of getXXX() and hence they are also called getter methods. For example,

```
def getName(self):
    return self.name
```

Here, getName() is an accessor method since it is reading and returning the value of 'name' instance variable. It is not modifying the value of the name variable.
On the other hand, mutator methods are the methods which not only read the data but also modify them. They are written in the form of setXXX() and hence they are also called setter methods. For example,

```
def setName(self, name):
    self.name = name
```

Here, setName() is a mutator method since it is modifying the value of 'name' variable by storing new name. In the method body, 'self.name' represents the instance variable 'name' and the right hand side 'name' indicates the parameter that receives the new value from outside.

Class Methods

These methods act on class level. Class methods are the methods which act on the class variables or static variables. These methods are written using @classmethod decorator above them. By default, the first parameter for class methods is 'cls' which refers to the class itself. For example, 'cls.var' is the format to refer to the class variable. These methods are generally called using the **classname.method()**.

Python Static Method

We need static methods when the processing is at the class level but we need not involve the class or instances. Static methods are used when some processing is related to the class but does not need the class or its instances to perform any work.

Setting environmental variables, counting the number of instances of the class or changing an attribute in another class, etc. are the tasks related to a class. Such tasks are handled by static methods. Also, static methods can be used to accept some values, process them and return the result. In this case the involvement of neither the class nor the objects is needed.

Static methods are written with a decorator @staticmethod above them. Static methods are called in the form of classname.method().

5. Write a note on inner classes with an example.

Writing a class within another class is called creating an inner class or nested class. For example, if we write class B inside class A, then B is called inner class or nested class. Inner classes are useful when we want to sub group the data of a class. For example, let's take a person's data like name, age, date of birth etc. Here, name contains a single value like 'Karthik', age contains a single value like '30' but the date of birth does not contain a single value. Rather, it contains three values like date, month and year. So, we need to take these three values as a sub group. Hence it is better to write date of birth as a separate class Dob inside the Person class. This Dob will contain instance variables dd, mm and yy which represent the date of birth details of the person.

```
class Person:
  def __init__(self):
    self.name = 'Karthik'
    self.db = self.Dob() #this is Dob object
```

EXAMPLE:

```
#inner class example
class Person:
  def __init__(self):
     self.name = 'Karthik'
     self.db = self.Dob()
  def display(self):
     print('Name=', self.name)
  class Dob:
   def __init__(self):
     self.dd = 10
     self.mm = 5
     self.yy = 1988
   def display(self):
     print('Dob= {}/{}/{}'.format(self.dd, self.mm, self.yy))
p = Person()
p.display()
x = p.db
x.display()
```

6. What is inheritance? How to implement inheritance in Python? Give an example

Inheritance is the capability of one class to derive or inherit the properties from another class. Inheritance allows us to define a class that inherits all the methods and properties from another class.

- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

```
class Person(object):

   # Constructor
   def __init__(self, name):
      self.name = name

   # To get name
   def getName(self):
      return self.name

   # To check if this person is an employee
   def isEmployee(self):
      return False


# Inherited or Subclass (Note Person in bracket)
class Employee(Person):

   # Here we return true
   def isEmployee(self):
      return True

# Driver code
emp = Person("Geek1")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

7. Explain with example overriding superclass constructor and method

When the programmer writes a constructor in the sub class, **the super class constructor is not available to the sub class**. In this case, only the sub class constructor is accessible from the sub class object. That means the sub class constructor is replacing the super class constructor. This is called **constructor overriding**. Similarly in the sub class, if we write a method with exactly same name as that of super class method, it will override the super class method. This is called **method overriding**.

A Python program to override super class constructor and method in sub class.

```
class Father:
  def __init__(self):
    self.property = 800000.00
  def display_property(self):
     print('Father\'s property=', self.property)
class Son(Father):
```

```
  def __init__(self):
    self.property = 200000.00
  def display_property(self):
    print('Child\'s property=', self.property)
s = Son()
s.display_property()
```

In this case, how to call the super class constructor so that we can access the father's property from the Son class? For this purpose, we should call the constructor of the super class from the constructor of the sub class using the super() method.

When there is a method in the super class, writing the same method in the sub class so that it replaces the super class method is called **'method overriding'**.
The programmer overrides the super class methods when he does not want to use them in sub class. Instead, he wants a new functionality to the same method in the sub class.

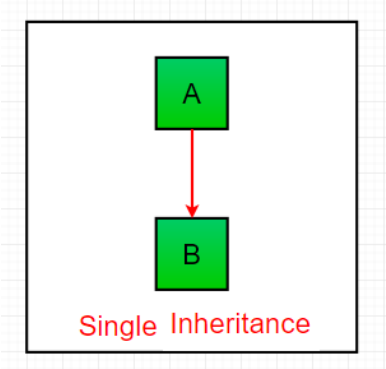
A Python program to override the super class method in sub class.

```
#method overriding
import math
class Square:
  def area(self, x):
    print('Square area= %.4f'% x*x)
class Circle(Square):
  def area(self, x):
    print('Circle area= %.4f'% (math.pi*x*x))
c = Circle()
c.area(15)
```

8. Explain any 2 types of inheritance in Python with an example for each

Types of Inheritance depends upon the number of child and parent classes involved. There are four types of inheritance in Python:

Single Inheritance: Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.



Example:

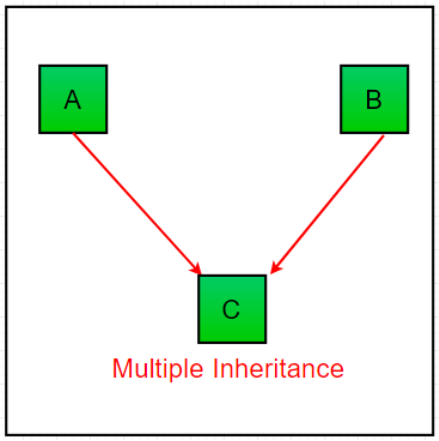
```
# Python program to demonstrate
# single inheritance

# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")

# Derived class
class Child(Parent):
    def func2(self):
        print("This function is in child class.")

# Driver's code
object = Child()
object.func1()
object.func2()
```

Multiple Inheritance: When a class can be derived from more than one base class this type of inheritance is called multiple inheritance. In multiple inheritance, all the features of the base classes are inherited into the derived class.



Example:

```
# Python program to demonstrate
# multiple inheritance


# Base class1
class Mother:
   mothername = ""
   def mother(self):
      print(self.mothername)

# Base class2
```

```
class Father:
    fathername = ""
    def father(self):
        print(self.fathername)

# Derived class
class Son(Mother, Father):
    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)

# Driver's code
s1 = Son()
s1.fathername = "RAM"
s1.mothername = "SITA"
s1.parents()
```

9. Explain multiple inheritance in Python with an example.[refer 8th ans]

10. Explain method overloading and overriding with example

Python Method Overloading

If a method is written such that it can perform more than one task, it is called method overloading.We see method overloading in the languages like Java. For example, we call a method as:

sum(10, 15)
sum(10, 15, 20)

In the first call, we are passing two arguments and in the second call, we are passing three arguments. It means, the sum() method is performing two distinct operations: finding sum of two numbers or sum of three numbers.This is called method overloading.

A Python program to show method overloading to find sum of two or three numbers.

```
#method overloading
class Myclass:
  def sum(self, a=None, b=None, c=None):
    if a!=None and b!=None and c!=None:
      print('Sum of three=', a+b+c)
    elif a!=None and b!=None:
      print('Sum of two=', a+b)
    else:
      print('Please enter two or three arguments')
#call sum() using object
m = Myclass()
m.sum(10, 15, 20)
m.sum(10.5, 25.55)
m.sum(100)
```

Here the sum() method is calculating sum of two or three numbers and hence it is performing more than one task. Hence it is an overloaded method. In this way, overloaded methods achieve polymorphism.

When there is a method in the super class, writing the same method in the sub class so that it replaces the super class method is called **'method overriding'**.
The programmer overrides the super class methods when he does not want to use them in sub class. Instead, he wants a new functionality to the same method in the sub class.

A Python program to override the super class method in sub class.

```
#method overriding
import math
class Square:
  def area(self, x):
    print('Square area= %.4f'% x*x)
class Circle(Square):
  def area(self, x):
    print('Circle area= %.4f'% (math.pi*x*x))
c = Circle()
c.area(15)
```

11. Explain abstract class and abstract method with an example.

An abstract method is a method whose action is redefined in the sub classes as per the requirement of the objects. Generally abstract methods are written without body since their body will be defined in the sub classes anyhow. But it is possible to write an abstract method with body also. To mark a method as abstract, we should use the decorator @abstractmethod. On the other hand, a concrete method is a method with body.

An abstract class is a class that generally contains some abstract methods. Since, abstract class contains abstract methods whose implementation (or body) is later defined in the sub classes, it is not possible to estimate the total memory required to create the object for the abstract class. So, PVM cannot create objects to an abstract class.

Once an abstract class is written, we should create sub classes and all the abstract methods should be implemented (body should be written) in the sub classes. Then, it is possible to create objects to the sub classes.

In next Program, we create Myclass as an abstract super class with an abstract method calculate(). This method does not have any body within it. The way to create an abstract class is to derive it from a meta class ABC that belongs to abc (abstract base class) module as:

class Abstractclass(ABC):

Since all abstract classes should be derived from the meta class ABC which belongs to **abc (abstract base class)** module, we should import this module into our program. A meta class is a class that defines the behavior of other classes.

The meta class ABC defines that the class which is derived from it becomes an abstract class. To import abc module's ABC class and abstractmethod decorator we can write as follows:

from abc import ABC, abstractmethod
or
from abc import *

Now, our abstract class 'Myclass' should be derived from the ABC class as:

```
classs Myclass(ABC):
@abstractmethod
def calculate(self, x):
pass #empty body, no code
```

This class has an abstract method calculate() that does not contain any code. We used @abstractmethod decorator to specify that this is an abstract method. We have to write sub classes where this abstract method is written with its body (or implementation).

12. What is an interface? Explain how to convert class as an interface with an example.

At a high level, an interface acts as a blueprint for designing classes. Like classes, interfaces define methods. Unlike classes, these methods are abstract. An abstract method is one that the interface simply defines. It doesn't implement the methods. This is done by classes, which then implement the interface and give concrete meaning to the interface's abstract methods.

Python's approach to interface design is somewhat different when compared to languages like Java, Go, and C++. These languages all have an interface keyword, while Python does not. Python further deviates from other languages in one other aspect. It doesn't require the class that's implementing the interface to define all of the interface's abstract methods.

13. Differentiate abstract classes and interfaces.

An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation

14. Explain exception handling in Python with an example.

The purpose of handling errors is to make the program robust. The word 'robust' means 'strong'. A robust program does not terminate in the middle. Also, when there is an error in the program, it will display an appropriate message to the user and continue execution. Designing such programs is needed in any software development. For this purpose, the programmer should handle the errors. When the errors can be handled, they are called exceptions. To handle exceptions, the programmer should perform the following three steps:
Step 1: The programmer should observe the statements in his program where there may be a possibility of exceptions. Such statements should be written inside a 'try' block. A try block looks like as follows:

```
try:
   statements
```

The greatness of try block is that even if some exception arises inside it, the program will not be terminated. When PVM understands that there is an exception, it jumps into an 'except' block.
Step 2: The programmer should write the 'except' block where he should display the exception details to the user. This helps the user to understand that there is some error in the program. The programmer should also display a message regarding what can be done to avoid this error. Except block looks like as follows:

```
except exceptionname:
     statements #these statements form handler
```

The statements written inside an except block are called 'handlers' since they handle the situation when the exception occurs.
Step 3: Lastly, the programmer should perform clean up actions like closing the files and terminating any other processes which are running. The programmer should write this code in the finally block. Finally block looks like as follows:

```
finally:
  statements
```

The specialty of finally block is that the statements inside the finally block are executed irrespective of whether there is an exception or not. This ensures that all the opened files are properly closed and all the running processes are properly terminated. So, the data in the files will not be corrupted and the user is at the safe-side.
Performing the above 3 tasks is called 'exception handling'.
A Python program to handle the ZeroDivisionError exception.

```
#an exception handling example
try:
  f = open("myfile", "w")
  a, b = [int(x) for x in input("Enter two numbers: ").split()]
  c = a/b
  f.write("writing %.2f into myfile" %c)
except ZeroDivisionError:
  print('Division by zero happened')
  print('Please do not enter 0 in denominator')
finally:
  f.close()
  print('File closed')
```

15. List and describe any five built-in exceptions in Python. [any 5]

Exception Class Name	**Description**
Exception	Represents any type of exception. All exceptions are sub classes of this class.
AssertionError	Raised when an assert statement gives error.
AttributeError	Raised when an attribute reference or assignment fails.
EOFError	Raised when input() function reaches end of file condition without reading any data.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when generator's close() method is called.
IOError	Raised when an input or output operation failed. It raises when the file opened is not found or when writing data disk is full.
ImportError	Raised when an import statement fails to find the module being imported.
IndexError	Raised when a sequence index or subscript is out of range.
KeyError	Raised when a mapping (dictionary) key is not found in the set of existing keys.

KeyboardInterrupt	Raised when the user hits the interrupt key(normally control-C or Delete).
NameError	Raised when an identifier is not found locally or globally.
RuntimeError	Raised when an error is detected that doesn't fall in any of the other categories.
StopIteration	Raised by an iterator's next() method to signal that there are no more elements.
SyntaxError	Raised when the compiler encounters a syntax error.Import or exec statements and input() and eval() functions may raise this exception.
IndentationError	Raised when indentation is not specified properly.
SystemExit	Raised by the sys.exit() function. When it is not handled, the Python interpreter exits.
TypeError	Raised when an operation or function is applied to an object of inappropriate datatype.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
ValueError	Raised when a built-in operation or function receives an argument that has right datatype but wrong value.
ZeroDivisionError	Raised when the denominator is zero in adivision or modulus operation.

16. Explain user-defined exceptions with suitable example

Like the built-in exceptions of Python, the programmer can also create his own exceptions which are called 'User- defined exceptions' or 'Custom exceptions'.We know Python offers many exceptions which will raise in different contexts. For example, when a number is divided by zero, the ZeroDivisionError is raised. Similarly, when the datatype is not correct, TypeError is raised.

Steps to be followed for User Defined Exceptions

1. Since all exceptions are classes, the programmer is supposed to create his own exception as a class. Also, he should make his class as a sub class to the in-built 'Exception' class.

2. **class MyException(Exception):**
3. **def __init__(self, arg):**
   self.msg = arg
   Here, 'MyException' class is the sub class for 'Exception' class.This class has a constructor where a variable 'msg' is defined.This 'msg' receives a message passed from outside through 'arg'.

4. The programmer can write his code; maybe it represents a group of statements or a function. When the programmer suspects the possibility of exception, he should raise his own exception using 'raise' statement as: **raise MyException('message')**

Here, raise statement is raising MyException class object that contains the given 'message'.

5. The programmer can insert the code inside a 'try' block and catch the exception using 'except' block as:
6. **try:**
7. **code**
8. **Except MyException as me:**
   print(me)

   Here, the object 'me' contains the message given in the raise statement.

A Python program to create our own exception and raise it when needed.

```
class MyException(Exception):
  def __init__(self,arg):
    self.msg = arg
def check(list):
    for i in list:
      print(i,end='\t')
      if(i<0 or i>=100):
        raise MyException('Exception Occured at %d'%i)
age=[25,78,65,94,112,77,49]
try:
 check(age)
except MyException as me:
 print(me)
```

17. Write a note on logging the exceptions.

Python provides a module 'logging' that is useful to create a log file that can store all error messages that may occur while executing a program.

There may be different levels of error messages. For example, an error that crashes the system should be given more importance than an error that merely displays a warning message. So, depending on the seriousness of the error, they are classified into 6 levels in 'logging' module, as shown in Table.

Table : Logging Error Level and their Predefined Numeric Values

Level	Numeric value	Description
CRITICAL	50	Represents a very serious error that needs high attention.
ERROR	40	Represents a serious error
WARNING	30	Represents a warning message,some caution is needed.
INFO	20	Represents a message with some important information.
DEBUG	10	Represents a message with debugging information.
NOTSET	0	Represents that the level is not set.

As we know, by default, the error messages that occur at the time of executing a program are displayed on the user's monitor.Only the messages which are equal to or above the level of a WARNING are displayed.That means WARNINGS, ERRORS and CRITICAL ERRORS are displayed.It is possible that we can set this default behavior as we need.

To understand different levels of logging messages, we are going to write a Python program. In this program, first we have to create a file for logging (storing) the messages. This is done using basicConfig() method of logging module as:

logging.basicConfig(filename='mylog.txt', level=logging.ERROR)

Here, the log file name is given as mylog.txt'.The level is set to ERROR. Hence the messages whose level will be at ERROR or above, (i.e. ERROR or CRITICAL) will only be stored into the log file. Once, this is done, we can add the messages to the 'mylog.txt' file as:

logging.methodname('message')

The methodnames can be critical(), error(), warning(), info() and debug(). For example, we want to add a critical message, we should use critical() method as:

logging.critical('System crash - Immediate attention required')

Now, this error message is stored into the log file, i.e. 'mylog.txt'.

18. Explain the sequence characters used in regular expressions. give example

Special sequences do not match for the actual character in the string instead it tells the specific location in the search string where the match must occur. It makes it easier to write commonly used patterns.

character	Description
\A	It returns a match if the specified characters are present at the beginning of the string.
\b	It returns a match if the specified characters are present at the beginning or the end of the string.
\B	It returns a match if the specified characters are present at the beginning of the string but not at the end.
\d	It returns a match if the string contains digits [0-9].
\D	It returns a match if the string doesn't contain the digits [0-9].
\s	It returns a match if the string contains any white space character.

\S	It returns a match if the string doesn't contain any white space character.
\w	It returns a match if the string contains any word characters.
\W	It returns a match if the string doesn't contain any word.
\Z	Returns a match if the specified characters are at the end of the string.

19. Explain the quantifiers used in regular expressions. Give an example.

In regular expressions, quantifiers match the preceding characters or character sets a number of times. The following table shows all the quantifiers and their meanings:

Quantifier	Name	Meaning
*	Asterisk	Match its preceding element zero or more times.
+	Plus	Match its preceding element one or more times.
?	Question Mark	Match its preceding element zero or one time.
{ *n* }	Curly Braces	Match its preceding element exactly n times.
{ *n* ,}	Curly Braces	Match its preceding element at least n times.
{ *n* , *m* }	Curly Braces	Match its preceding element from n to m times.

20. Explain the special characters used in a regular expression. Give an example.

[]	It represents the set of characters.
\	It represents the special sequence.
.	It signals that any character is present at some specific place.
^	It represents the pattern present at the beginning of the string.
$	It represents the pattern present at the end of the string.

21. Explain findall() and search() methods used to the regular expression.

re.findall()

Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found.

Example: Finding all occurrences of a pattern

```
# findall()
import re
string = """Hello my Number is 123456789 and
        my friend's number is 987654321"""

regex = '\d+'

match = re.findall(regex, string)
print(match)
```

re.search() : This method either returns None (if the pattern doesn't match), or a re.MatchObject contains information about the matching part of the string. This method stops after the first match, so this is best suited for testing a regular expression more than extracting data.

Example: Searching an occurrence of the pattern

```
# A Python program to demonstrate working of re.match().
import re

regex = r"([a-zA-Z]+) (\d+)"
match = re.search(regex, "I was born on June 24")

if match != None:
    print ("Match at index %s, %s" % (match.start(), match.end()))
```

```
    print ("Full match: %s" % (match.group(0)))
     print ("Month: %s" % (match.group(1)))
     print ("Day: %s" % (match.group(2)))
  else:
    print ("The regex pattern does not match.")
```

23. Explain creating a thread without using a class with an example.

We can create a thread by creating an object of Thread class and pass the function name as target for the thread as

T=Thread(target=functionname,[args=(arg1,arg2,.....)])

We are creating the Thread class object t that represents our thread. The target represents the function on which the thread will act. Args represents the tuple of arguments which are passed to the function.

Once the thread is created , it should be started by calling the start() method as:

t.start(). The thread t will jump to the target function

24. Explain creating a thread by creating a subclass to the Thread class. Give an example.

Thread class is already created by Python people in the threading module. We can make our class as a sub class to Thread so that we can inherit the functionality of Thread class.

class MyThread(Thread)

Here MyThread represents the subclass of Thread class . .Any methods of the Thread class are also available to the sub class. The Thread class has the run() method which is also available to the sub class So by overriding the run() method we can make the threads run our own run() method.

The step is to create the object of the MyThread class which contains a copy of the super class i.e the Thread class

t1=MyThread()

Here t1 is the object of MyThread class which represents our thread.
We can run this thread by calling the start() method as t1.start()

```
#creating our own thread
from threading import thread
class MyThread(Thread)
  def run(self):
    for i in range(1,6):
          print(i)
t1=MyThread()
t1.start()
t1.join()
```

25. Explain creating a thread without creating a subclass to the Thread class. Give an example.

We can create an independent class say MyThread that does not inherit from Thread class . Then we can create an object obj to MyThread class as

obj=MyThread('Hello')

The next step is to create a thread by creating an object to Thread class and specifying the method of the MyThread class as its Target as

t1=Thread (target=obj.display,arg(1,2))

Here t1 is our thread which is created as an object of thread class.Target represents the display() method of 'MyThread' object obj ,'args' represents a tuple of values passed to the method. When the thread t1 is started ,it will execute the display() method of MyThread class.

```
from threading import *
class MyThread :
  def _init_(self,str)
     self.str=str
def display(self,x,y)
   print(self.str)
     print('The args are :',x,y)
obj=MyThread('Hello')
t1=Thread(target=obj.display,args=(1,2))
t1.start()
```

26. List and describe any five methods associated with the Thread class [any 5]

Thread class methods

run() − The run() method is the entry point for a thread.

start() − The start() method starts a thread by calling the run method.

join([time]) − The join() waits for threads to terminate.

isAlive() − The isAlive() method checks whether a thread is still executing.

getName() − The getName() method returns the name of a thread.

setName() − The setName() method sets the name of a thread

name : This is the property that represents the threads name

setDaemon(flag):Make a thread a daemon if the flag is true.

27. What is thread synchronization? Explain different techniques used for thread synchronization?

When a thread is already acting on an object ,preventing any other thread from acting on the same object is called thread synchronization or thread safe. The object on which the threads are synchronized is called synchronized object or mutex(mutually exclusive lock)

Thread synchronization is done using the following techniques

- Using locks
- Using semaphores

Using locks

Locks :Locks can be used to lock the object on which the thread is acting.
We can create a lock by creating an object of Lock class as

l=lock()
To lock the current obect ,we should use acquire
l.Acquire()
To unlock or release the object we should use release
l.release()

Using semaphores

Semaphore: A semaphore is an object that provides synchronization based on a counter . A semaphore is created as an object of Semaphore class as
l=Semaphore(countervalue). If the counter value is not given ,the default is take as 1
When the acquire method is called the counter gets decremented by 1 and when the release method is called the counter gets incremented by 1
l.acquire()
l.release()

28. Explain notify() and wait() methods in thread communication.

notify() Method: When we want to send a notification to only one thread that is in waiting state then we always use notify() method. Here if one thread wants to be condition object up-gradation then notify() is used.

Syntax :condition_object.notify()

wait(time) Method: The wait() method can be used to make a thread wait till the notification got and also till the given time will end. In simple words we can say that thread is to wait until the execution of the notify() method is not done. We can use time in it if we set a certain time then the execution will stop until time overs after that it will execute still the instructions are remaining.

Syntax: condition_object.wait()

Parameter:Time

UNIT IV

Questions carrying 2 Marks

1. What is root window? How it is created in Python?

To display the graphical output, we need space on the screen. This space that is initially allocated to every GUI program is called 'top level window' or 'root window'.

```
from tkinter import *
root = Tk()
root.mainloop()
```

2. What is Canvas? How it is created in Python?

The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.

```
from tkinter import *
top = Tk()
top.geometry("200x200")
#creating a simple canvas.
```

```
c = Canvas(top,bg = "pink",height = "200",width = 200)
arc = c.create_arc((5,10,150,200),start = 0,extent = 150, fill= "white")
c.pack()
top.mainloop()
```

3. Write Python code to create rectangle and arc in canvas.

To create an arc we have the create_arc() method.
Id=c.create_arc(100,100,400,300,width=3,start=270,extent=180,outline='red',style='arc')

Similarly, to create a rectangle or square shaped box, we can use the create_rectangle() method as:

```
id = c.create_rectangle(500, 200, 700, 600, width=2, fill="gray",
    outline="black", activefill="yellow")
```

4. Write a Python code to display an image in the canvas.

```
from tkinter import *
root = Tk()
canvas = Canvas(root, width = 300, height = 300)
canvas.pack()
img = PhotoImage(file="ball.ppm")
canvas.create_image(20,20, anchor=NW, image=img)
mainloop()
```

5. Differentiate Canvas and frame.

A Canvas is primarily for drawing shapes and things on, whereas a frame is for organising a collection of tkinter widgets inside. While it's possible to put some tkinter widgets into a Canvas that's not its main use case, and a Frame isn't capable of drawing things like lines or circles or whatnot.

6. How to add a scrollbar to a Text widget?

it is useful to add scroll bars to the Text widget. A scroll bar is a bar that is useful to scroll the text either horizontally or vertically.For example, we can create a vertical scroll bar by creating an object to Scrollbar class as:
s = Scrollbar(root, orient=VERTICAL, command= t.yview)

7. Differentiate Label and Text Widget.

8. What is an entry widget? How it is created?

Entry widget is useful to create a rectangular box that can be used to enter or display one line of text.For example, we can display names, passwords or credit card numbers using Entry widgets.An Entry widget can be created as an object of Entry class as:
e1 = Entry(f, width=25, fg='blue', bg='yellow', font=('Arial', 14), show='*')

9. What is a spin box widget? How it is created?

A Spinbox widget allows the user to select values from a given set of values. The spin box appears as a long rectangle attached with arrowheads pointing towards up and down.The user can click on the arrowheads

to see the next value or previous value. To create a spin box with numbers ranging from 5 to 15, we can write the following statement:
s1 = Spinbox(f, from_= 5, to=15, textvariable=val1, width=15, fg='blue', bg='yellow', font=('Arial', 14, 'bold'))

10. List the values that can be assigned to selectmode property of listbox

The option 'selectmode' may take any of the following values:

- BROWSE
- SINGLE
- MULTIPLE
- EXTENDED

11. Write a Python code to read the source code of a web page.

```
import requests

url = input('Webpage to grab source from: ')
html_output_name = input('Name for html file: ')

req = requests.get(url, 'html.parser')

with open(html_output_name, 'w') as f:
  f.write(req.text)
  f.close()
```

12. Write a Python code to download an image from the internet.

```
import urllib.request

image_url = 'https://bit.ly/2XuVzB4' #the image on the web
save_name = 'my_image.jpg' #local name to be saved
urllib.request.urlretrieve(image_url, save_name)
```

13. Write a Python code to download a page

```
import urllib.request, urllib.error, urllib.parse
url = "The url of the page you want to download"
response = urllib.request.urlopen(url)
```

14. Write the Python statements used to connect the Gmail server and send a message.

```
server=smtplib.SMTP('smtp.gmail.com',587)

server.send_message(msg)
```

15. List any four types of databases used with Python.

- MySQL
- Oracle
- Microsoft SQL Server

- Microsoft Access
- Sybase
- Firebird

16. How to connect to MySQL database from Python.

To establish connection with MySQL database, we use the connect() method of *MySQLdb* module as:

```
conn = MySQLdb.connect(host='localhost', database='world',
user='root', password='nag123')
```

17. What is the use of cursor object?

Cursor class is an instance using which you can invoke methods that execute SQLite statements, fetch data from the result sets of the queries. You can create Cursor object using the cursor() method of the Connection object/class.

18. What are the methods used to get a single row and all the rows from the table?

cursor.fetchall() fetches all the rows of a query result. It returns all the rows as a list of tuples. An empty list is returned if there is no record to fetch.

cursor.fetchone() method returns a single record or None if no more rows are available.

19. How to connect to the oracle database from Python?

20. What are the modules used to work with MySQL and Oracle databases?

5 marks Questions

1. Explain the steps involved in creating a GUI application in Python with a suitable example.

Python offers tkinter module to create graphics programs. The tkinter represents 'toolkit interface' for GUI. This is an interface for Python programmers that enable them to use the classes of TK module of TCL/TK language. Let's see what this TCL/TK is. The TCL (Tool Command Language) is a powerful dynamic programming language, suitable for web and desktop applications, networking, administration, testing and many more. It is open source and hence can be used by any one freely. TCL language uses TK (Tool Kit) language to generate graphics. TK provides standard GUI not only for TCL but also for many other dynamic programming languages like Python. Hence, this TK is used by Python programmers in developing GUI applications through Python's tkinter module.
The following are the general steps involved in basic GUI programs:

1. First of all, we should create the root window. The root window is the top level window that provides rectangular space on the screen where we can display text, colors, images, components, etc.
2. In the root window, we have to allocate space for our use. This is done by creating a **canvas or frame**. So, canvas and frame are **child windows in the root window**.
3. Generally, we use **canvas** for displaying **drawings** like lines, arcs, circles, shapes, etc. We use **frame** for the purpose of displaying **components** like push buttons, check buttons, menus, etc. These components are also called 'widgets'.
4. When the user clicks on a widget like push button, we have to handle that event. It means we have to respond to the events by performing the desired tasks.

2. How to create a button widget and bind it to the event handler? Explain with example.

A push button is a component that performs some action when clicked. These buttons are created as objects of Button class as:

```
b = Button(f, text='My Button', width=15, height=2, bg='yellow', fg='blue',
activebackground='green', activeforeground='red')
```

We can also display an image on the button as: #first load the image into file1

```
file1 = PhotoImage(file="cat.gif")
#create a push button with image
b = Button(f, image=file1, width=150, height=100, bg='yellow', fg='blue', activebackground='green',
activeforeground='red')
```

A Python program to create a push button and bind it with an event handler function.

```
from tkinter import *
def buttonClick(self):
  print('You have clicked me')
root = Tk()
f = Frame(root, height=200, width=300)
f.propagate(0)
f.pack()
b = Button(f, text='My Button', width=15, height=2, bg='yellow', fg='blue',
activebackground='green', activeforeground='red')
b.pack()
b.bind("<Button-1>", buttonClick)
root.mainloop()
```

3. Write a Python GUI program to create three pushbuttons using Tkinter. BackgroundThe color of the frame should be different when different buttons are clicked.

```
from tkinter import *
class MyButton:
  def __init__(self, root):
    self.f = Frame(root, height=400, width=500)
    self.f.propagate(0)
    self.f.pack()
    self.b1 = Button(self.f, text='Red', width=15, height=2, command=lambda: self.buttonClick(1))
    self.b2 = Button(self.f, text='Green', width=15, height=2, command=lambda: self.buttonClick(2))
    self.b3 = Button(self.f, text='Blue', width=15, height=2, command=lambda: self.buttonClick(3))
    self.b1.pack()
    self.b2.pack()
    self.b3.pack()
  def buttonClick(self, num):
   if num==1:
    self.f["bg"]= 'red'
   if num==2:
    self.f["bg"]= 'green'
   if num==3:
    self.f["bg"]= 'blue'
```

```
root = Tk()
mb = MyButton(root)
root.mainloop()
```

3. Write a note on arranging Widgets in a frame using layout managers.

Once we create widgets or components, we can arrange them in the frame in a particular manner. Arranging the widgets in the frame is called 'layout management'. There are three types of layout managers.

- Pack layout manager
- Grid layout manager
- Place layout manager

Pack layout manager uses pack() method. This method is useful to associate a widget with its parent component. While using the pack() method, we can mention the position of the widget using 'fill' or 'side' options.

```
b.pack(fill=X)
b.pack(fill=Y)
```

The 'fill' option can take the values: X, Y, BOTH, NONE. The value X represents that the widget should occupy the frame horizontally and the value Y represents that the widget should occupy vertically. BOTH represents that the widget should occupy in both the directions. NONE represents that the widget should be displayed as it is. The default value is NONE. Along with 'fill' option, we can use 'padx' and 'pady' options that represent how much space should be left around the component horizontally and vertically.
For example, b1.pack(fill=Y, padx=10, pady=15)

Grid layout manager uses the grid() method to arrange the widgets in a two dimensional table that contains rows and columns. We know that the horizontal arrangement of data is called 'row' and vertical arrangement is called 'column'. The position of a widget is defined by a row and a column number. The size of the table is determined by the grid layout manager depending on the widgets size.
b1.grid(row=0, column=0, padx=10, pady=15)

Place layout manager uses the place() method to arrange the widgets. The place() method takes x and y coordinates of the widget along with width and height of the window where the widget has to be displayed. For example,
b1.place(x=20, y=30, width=100, height=50)

4. Write a note on i)Text Widget ii) Scrollbar Widget

Python Text Widget

Text widget is same as a label or message. But Text widget has several options and can display multiple lines of text in different colors and fonts.It is possible to insert text into a Text widget, modify it or delete it. We can also display images in the Text widget. One can create a Text widget by creating an object to Text class as:

t = Text(root, width=20, height=10, font=('Verdana', 14, 'bold'), fg='blue', bg='yellow', wrap=WORD)

Here, 't' represents the object of Text class.'root' represents an object of root window or frame.'width' represents the width of the Text widget in characters.'height' represents the height of the widget in lines.The option 'wrap' specifies where to cut the line.wrap=CHAR represents that any line that is too long will be broken at any character.wrap=WORD will break the line in the widget after the last word that fits in the line. wrap=NONE will not wrap the lines. In this case, it is better to provide a horizontal scroll bar to view the lines properly in the Text widget.

Python Scrollbar Widget

A scroll bar is a widget that is useful to scroll the text in another widget.For example, the text in the Text, Canvas, Frame or Listbox can be scrolled from top to bottom or left to right using scroll bars.There are two types of scroll bars.They are horizontal and vertical.The horizontal scroll bar is useful to view the text from left to right. The vertical scroll bar is useful to scroll the text from top to bottom.To create a scroll bar, we have to create Scrollbar class object as:

h = Scrollbar(root, orient=HORIZONTAL, bg='green', command=t.xview)

Here, 'h' represents the Scrollbar object which is created as a child to 'root' window.The option 'orient' indicates HORIZONTAL for horizontal scroll bars and VERTICAL indicates vertical scroll bars. 'bg' represents back ground color for the scroll bar.This option may not work in Windows since Window operating system may force some default back ground color for the scroll bar which will not be disturbed by the tkinter. The option 'command' represents the method that is to be executed. The method 'xview' is executed on the object 't'.Here, 't' may represent a widget like Text widget or Listbox.

5. Write a Python code to create 3 check buttons and display the selected option using a label.

```
from tkinter import *
class Mycheck:
  def __init__(self, root):
    self.f = Frame(root, height=350, width=500)
    self.f.propagate(0)
    self.f.pack()
    self.var1 = IntVar()
    self.var2 = IntVar()
    self.var3 = IntVar()
    self.c1 = Checkbutton(self.f, bg='yellow', fg= 'green', font=('Georgia', 20, 'underline'), text='Java',
variable= self.var1, command=self.display)
    self.c2 = Checkbutton(self.f, text='Python', variable= self.var2,command=self.display)
    self.c3 = Checkbutton(self.f, text='.NET', variable= self.var3, command=self.display)
#attach check boxes to the frame
    self.c1.place(x=50, y=100)
    self.c2.place(x=200, y=100)
    self.c3.place(x=350, y=100)
  def display(self):
    x = self.var1.get()
    y = self.var2.get()
    z = self.var3.get()
    str = ''
#catch user choice
    if x==1:
      str += 'Java '
    if y==1:
      str+= 'Python '
    if z==1:
      str+= '.NET '
    lbl = Label(text=str, fg='blue').place(x=50, y=150, width=200, height=20)
root = Tk()
#create an object to MyButtons class
mb = Mycheck(root)
#the root window handles the mouse click event
root.mainloop()
```

6. Write a note on i)Checkbox Widget ii) Radiobutton Widget

Python Checkbutton Widget

Check buttons, also known as check boxes are useful for the user to select one or more options from available group of options.Check buttons are displayed in the form of square shaped boxes. When a check button is selected, a tick mark is displayed on the button.We can create check buttons using the Checkbutton class as:

c1 = Checkbutton(f, bg='yellow', fg= 'green', font=('Georgia', 20, 'underline'), text='Java', variable= var1, command=display)

Here, 'c1' is the object of Checkbutton class that represents a check button.'f' indicates frame for which the check button becomes a child. 'bg' and 'fg' represent the back ground and fore ground colors used for check button.'font' represents the font name, size and style.'text' represents the text to be displayed after the check button. The option variable represents an object of IntVar() class.'command' represents the method to be called when the user clicks the check button.

Python Radiobutton Widget

A radio button is similar to a check button, but it is useful to select only one option from a group of available options.A radio button is displayed in the form of round shaped button. The user cannot select more than one option in case of radio buttons.When a radio button is selected, there appears a dot in the radio button.We can create a radio button as an object of the Radiobutton class as:

r1 = Radiobutton(f, bg='yellow', fg= 'green', font=('Georgia', 20, 'underline'), text='Male', variable= var, value=1, command=display)

The option 'text' represents the string to be displayed after the radio button.'variable' represents the object of IntVar class.'value' represents a value that is set to this object when the radio button is clicked.The object of IntVar class can be created as:

var = IntVar()

When the user clicks the radio button, the value of this 'var' is set to the value given in 'value' option, i.e. 1.It means 'var' will become 1 if the radio button 'r1' is clicked by the user.In this way, it is possible to know which button is clicked by the user.

7. Write a Python program to create Listbox with course names and display selected course in a textbox

```
from tkinter import *
class ListboxDemo:
  def __init__(self, root):
    self.f = Frame(root, width=700, height=400)
    self.f.propagate(0)
    self.f.pack()
    self.lbl = Label(self.f, text="Click one or more of the courses below:", font="Calibri 14")
    self.lbl.place(x=50, y=50)
    self.lb = Listbox(self.f, font="Arial 12 bold", fg='blue', bg='yellow', height=8,
selectmode=MULTIPLE)
    self.lb.place(x=50, y=100)
    for i in ["BCA", "BBA", "BCOM", "BSC", "BSW"]:
      self.lb.insert(END, i)
#bind the ListboxSelect event to on_select() method
      self.lb.bind('<<ListboxSelect>>', self.on_select)
```

```
#create text box to display selected items
      self.t = Text(self.f, width=40, height=6, wrap=WORD) self.t.place(x=300, y=100)
  def on_select(self, event):
    self.lst = []
    indexes = self.lb.curselection()

#retrieve the items names depending on indexes
#append the items names to the list box
for i in indexes:
  self.lst.append(self.lb.get(i))
#delete the previous content of the text box
  self.t.delete(0.0, END)
#insert the new contents into the text box
  self.t.insert(0.0, self.lst)
root = Tk()
root.title("List box demonstration.")
obj = ListboxDemo(root)
root.mainloop()
```

8. Explain the process of creating a Listbox widget with a suitable example. Also, explain different values associated with selectmode option.

A list box is useful to display a list of items in a box so that the user can select 1 or more items.To create a list box, we have to create an object of Listbox class, as:
lb = Listbox(f, font="Arial 12 bold", fg='blue', bg='yellow', height=8, width=24, activestyle='underline', selectmode=MULTIPLE)
Here, 'lb' is the list box object.The option 'height' represents the number of lines shown in the list box.'width' represents the width of the list box in terms of number of characters and the default is 20 characters.The option 'activestyle' indicates the appearance of the selected item.It may be 'underline', 'dotbox' or 'none'.The default value is 'underline'.The option 'selectmode' may take any of the following values:

- **BROWSE**: Normally, we can select one item (or line) out of a list box.If we click on an item and then drag to a different item, the selection will follow the mouse.This is the default value of 'selectmode' option.
- **SINGLE**: This represents that we can select only one item(or line) from all available list of items.
- **MULTIPLE**: We can select 1 or more number of items at once by clicking on the items. If an item is already selected, clicking second time on the item will un-select it.
- **EXTENDED**: We can select any adjacent group of items at once by clicking on the first item and dragging to the last item.

9. How to create a Menu in Python? Explain the process of creating menus in Python with Syntax and examples.

To create a menu, we should use the following steps:

1. First of all, we should create a menu bar as a child to root window. This is done using Menu class as: menubar = Menu(root)
2. This menu bar should be attached to the root window using config() method as: root.config(menu=menubar)
3. The next step is to create a menu with a group of menu items. For this purpose, first of all we should create Menu class object as:

filemenu = Menu(root, tearoff=0)

Here, 'filemenu' is the Menu class object. The option 'tearoff' can be 0 or 1.When this option is 1, the menu can be torn off.In this case, the first position (position 0) in the menu items is occupied by the tear-off element which is a dashed line. If this option value is 0, then this dashed line will not appear and the menu items are displayed starting from 0th position onwards.The next step is to add menu items to the filemenu object using the add_command() method as:

filemenu.add_command(label="New", command=donothing)

When we want to display a horizontal line that separates a group of menu items from another group of menu items, we can use the add_separator() method, as given in the following statement: **filemenu.add_separator()** After adding all menu items to filemenu object, we should give it a name and add it to menu bar using add_cascade() method as: **menubar.add_cascade(label="File", menu=filemenu)** The 'menu' option tells that the 'File' menu is composed of all menu items that are added already to filemenu object.

10. Explain the functions related to TCP/IP server and TCP/IP Client

11. Explain the functions related to UDP Server and UDP Client

12. Write a Python program to create a File server that receives file name and send the contents of a file.

13. Explain how to connect to the MySQL database and retrieve all the rows from a table with a suitable example.

14. Explain how records are inserted and deleted from the MySQL database with a suitable example

15. Explain how MySQL database table is created using Python code.

We can create the new table by using the CREATE TABLE statement of SQL. In our database PythonDB, the table Employee will have the four columns, i.e., name, id, salary, and department_id initially.

The following query is used to create the new table Employee.

```
import mysql.connector
myconn = mysql.connector.connect(host = "localhost", user = "root",password = "naveen",database = "PythonDB")
cur = myconn.cursor()

try:
   #Creating a table
   dbs = cur.execute("create table Employee(name varchar(20) not null, id int(20) not null primary key, salary float not null, Dept_id int not null)")
except:
   myconn.rollback()
myconn.close()
```

16. Write a note on using the Oracle database from Python.

17. Explain how records are inserted and deleted from the MySQL database with example