1) What is Dictionary? Explain about Dictionary class properties.

Ans- Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is unordered, changeable and does not allow duplicates.

Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Properties:

- Dictionary Items: Dictionary items are unordered, changeable, and does not allow duplicates. Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

- Unordered: When we say that dictionaries are unordered, it means that the items does not have a defined order, you cannot refer to an item by using an index.

- Changeable: Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

- Duplicates Not Allowed: Dictionaries cannot have two items with the same key.

- Dictionary Length: To determine how many items a dictionary has, use the len() function,

2) Explain about a) Dictionary Operators b) Operators used with Dictionaries with examples.

Ans:

Dictionary Operators

The dictionary class supports some of the same operators that the list class supports. We already saw that the indexing operator ([]) can be used to access a value using the key as

the index:
>>> days['Fr']
'Friday'
The indexing operator can also be used to change the value corresponding to a key or to
add a new (key, value) pair to the dictionary:
>>> days
{'Fr': 'Friday', 'Mo': 'Monday', 'Tu': 'Tuesday',
'We': 'Wednesday', 'Th': 'Thursday'}
>>> days['Sa'] = 'Sat'
>>> days
{'Fr': 'Friday', 'Mo': 'Monday', 'Tu': 'Tuesday',
'We': 'Wednesday', 'Th': 'Thursday', 'Sa': 'Sat'}
The length of a dictionary (i.e., the number of (key, value) pairs in it) can be obtained using
the len function:
>>> len(days)
6
The in and not in operators are used to check whether an object is a key in the dictionary:
>>> 'Fr' in days
True
>>> 'Su' in days
False
>>> 'Su' not in days
True

Operation	**Explanation**
k in d	- True if k is a key in dictionary d, else False
k not in d	- False if k is a key in dictionary d, else True
d[k]	-Value corresponding to key k in dictionary d
len(d)	- Number of (key, value) pairs in dictionary d

Register number : 18GAEI6003

Name :Aishwarya B N

3.Explain any 5 Dictionary Methods

ANS- A Dictionary in Python is the unordered and changeable collection of data values that holds key-value pairs. Each key-value pair in the dictionary maps the key to its associated value making it more optimized. A Dictionary in python is declared by enclosing a comma-separated list of key-value pairs using curly braces({}). Python Dictionary is classified into two elements: Keys and Values.

Keys will be a single element

Values can be a list or list within a list, numbers, etc.

Syntax for Python Dictionary

Dict = { ' Tim': 18, xyz,.. }

Dictionary Methods :

Method	**Description**
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary.

Example

```
# Dictionary Methods
marks = {}.fromkeys(['Math', 'English', 'Science'], 0)
# Output: {'English': 0, 'Math': 0, 'Science': 0}
print(marks)
for item in marks.items():
    print(item)
```

 # Output:

```
['English', 'Math', 'Science']
print(list(sorted(marks.keys())))
```

4. Tuple objects can be Dictionary Key. Justify the statement with Example.

Answer:

According to the definition a dictionary key must be of a type that is immutable. For example, an integer, float, string, or Boolean as a dictionary key. However, neither a list nor another dictionary can serve as a dictionary key, because lists and dictionaries are mutable.

Instead a tuple which is immutable by definition can be used as a key in dictionary.

Thus “Tuple objects can be Dictionary Key” is justified.

Example:

```
# using tuple as a key

D1={}

D1[(1,”a”)] = “tuple”

Print(“Key”,’ : ‘,”Value”)

For x in d1.keys():

    Print (x,’ : ‘,d1[x])
```

Output :

```
 Key : Value

 (1,”a”):tuple
```

5 *.a] How to access the dictionary item?*
ANS-

<u>Accessing Values in Dictionary</u> :

To access dictionary elements, we can use the familiar square brackets along with the key to obtain its value.

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

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

```
# get vs [] for retrieving elements

my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack

print(my_dict['name'])

# Output: 26
```

```
print(my_dict.get('age'))
# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))
# KeyError
print(my_dict['address'])
```

Output

```
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

5b) Add Dictionary item

ANS-

Python dictionaries are an unordered collection of key value pairs. We add new key value pairs to an already defined dictionary. Below are the two approaches which we can use.

Assigning a new key as subscript

We add a new element to the dictionary by using a new key as a subscript and assigning it a value.

Example

```
CountryCodeDict = {"India": 91, "UK" : 44 , "USA" : 1}
print(CountryCodeDict)
CountryCodeDict["Spain"]= 34
print "After adding"
print(CountryCodeDict)
```

Output

Running the above code gives us the following result −

{'India': 91, 'USA': 1, 'UK': 44}

After adding

{'Spain': 34, 'India': 91, 'USA': 1, 'UK': 44}

Using the update() method

The update method directly takes a key-value pair and puts it into the existing dictionary. The key value pair is the argument to the update function. We can also supply multiple key values as shown below.

Example

```
CountryCodeDict = {"India": 91, "UK" : 44 , "USA" : 1, "Spain" : 34}

print(CountryCodeDict)


CountryCodeDict.update( {'Germany' : 49} )


print(CountryCodeDict)


# Adding multiple key value pairs

CountryCodeDict.update( [('Austria', 43),('Russia',7)] )


print(CountryCodeDict)
```

Output

Running the above code gives us the following result −

```
{'Spain': 34, 'India': 91, 'USA': 1, 'UK': 44}

{'Germany': 49, 'Spain': 34, 'India': 91, 'USA': 1, 'UK': 44}

{'USA': 1, 'India': 91, 'Austria': 43, 'Germany': 49, 'UK': 44, 'Russia': 7, 'Spain': 34}
```

By merging two dictionaries

We can also append elements to a dictionary by merging two dictionaries. Here again, we use the update() method but the argument to the method is a dictionary itself.

Example

```
CountryCodeDict1 = {"India": 91, "UK" : 44 , "USA" : 1, "Spain" : 34}

CountryCodeDict2 = {"Germany": 49, "Russia" : 7 , "Austria" : 43}

CountryCodeDict1.update(CountryCodeDict2)

print(CountryCodeDict1)
```

Output

Running the above code gives us the following result −

```
{'Austria': 43, 'Germany': 49, 'UK': 44, 'USA': 1, 'India': 91, 'Russia': 7, 'Spain': 34}
```

5 c) Remove Dictionary item

Ans :

Ways to delete python dictionary.

1) Python pop() function
2) Python del keyword
3) In-build python popitem() method
4) Dictionary comprehension along with python items()method

Python has in-built clear() method to delete a dictionary in python .The clear() method delete all the key-value pairs present in the dict and returns an empty dict.

Syntax:

Dict.clear()

1) Python pop() function
   The pop() method basically accepts a key to be deleted from the dictionary. It deletes the key as well as the value associated with the key from the dictand returns the updated dict.
   Syntax:
   dict.pop(key)

2) Python del keyword
   Python del is actually a keyword which is basically used for the deletion of objects.as we all are aware that python considers everything as an object , that is why we can easily use del to delete a dictionary in python by deleting elements individually.

   Python del keyword can also be used to delete a key-value pair from the input dictionary values.
   Syntax:
   del dict[key]

3) python popitem() method

python popitem() function can be used to delete a random or an arbitrary key-value pair from a python dict. the popitem function accepts no arguments and returns the deleted key-value pair from the dictionary.

Syntax:

Dict.popitem()

4) python dict comprehension along with items() method

python items() method along with dict comprehension can be used to delete a dictionary in python.
Python items() method basically takes no arguments and returns an object containing a list of all the key-value pairs in the particular dictionary.
Syntax:
Dict.items()

5. d) How to change dictionary item

ANS-

Change the value of a specific item in a dictionary by referring to its key name.

Example :

```
this_dict = { “brand” : “Ford” , “model” : “Mustang” , “year” : 1964 }

this_dict [ “year” ] = 2018                # To change the value of year,
```

6. Explain Looping through a dictionary

ANS-

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

There are multiple ways to iterate over a dictionary in Python.

- Iterate through all keys
- Iterate through all values

1.Iterate through all keys:

The order of states in the below code will change every time because the dictionary doesn't store keys in a particular order.

Example:

```
statesAndCapitals = {
                'Gujarat' : 'Gandhinagar',
                'Maharashtra' : 'Mumbai',
                'Rajasthan' : 'Jaipur',
                'Bihar' : 'Patna'
               }
 print('List Of given states:\n')
# Iterating over keys
for state in statesAndCapitals:
 print(state)
```

Output:

List Of given states:

Rajasthan

Bihar

Maharashtra

Gujarat

2.Iterate through all values:

Again, in this case, the order in which the capitals are printed in the below code will change every time because the dictionary doesn't store them in a particular order.

Example:

```
statesAndCapitals = {
                'Gujarat' : 'Gandhinagar',
                'Maharashtra' : 'Mumbai',
                'Rajasthan' : 'Jaipur',
                'Bihar' : 'Patna'
```

```
        }
print('List Of given capitals:\n')

# Iterating over values
for capital in statesAndCapitals.values():
    print(capital)
```

Output:

List Of given capitals:

Gandhinagar

Jaipur

Mumbai

Patna

7)Demonstrate: i) How a dictionary items can be represented as a list of tuples. ii) How tuples can be used as keys in dictionaries?

Ans:

i) Converting a dictionary into a list of tuples creates a list that contains each key-value pair in the dictionary as a tuple. For example, converting the dictionary {"a": 1, "b": 2, "c": 3} into a list of tuples results in [("a", 1), ("b", 2), ("c", 3)].

Call dict.items() with dict as a dictionary to return a list of key-value pairs as a view object.Call list(args) with args as the view object to convert args to a list.

Example:

```
a_dictionary = {"a": 1, "b": 2, "c": 3}
a_view = a_dictionary.items()
a_list = list(a_view)
print(a_list)
```

Output:

[('a', 1), ('b', 2), ('c', 3)]

ii) a tuple is a hashable value and can be used as a dictionary key.

Example:

```
coordinates = {(0, 0): 100, (1, 1): 200}
coordinates [(1, 0)] = 150
coordinates [(0, 1)] = 125

print(coordinates)

# {(0, 0): 100, (1, 1): 200, (1, 0): 150, (0, 1): 125}
```

8) Write a python program to check for presence of key in the Dictionary and find the sum of all it's values :

ANS-

```
# checking if the key exists in the dictionary
def checkKey(dict, key):

    if key in dict.keys():
        print("Present, ", end =" ")
        print("value =", dict[key])
    else:
        print("Not present")


# returning the sum of values in dictionary
def returnSum(myDict):

    sum = 0
    for i in myDict:
        sum = sum + myDict[i]

    return sum


# driver code
dict = {'a': 100, 'b':200, 'c':300}

key = 'b'
checkKey(dict, key)

print("Sum :", returnSum(dict))
```

OUTPUT :

```
Present, value = 200
Sum : 600
```

9. What is Dictionary? How it is different from list? Write a Python Program to Count the Occurrence of character in a String and print the Count.

Answer:

Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is unordered, changeable and does not allow duplicates.

A list is an ordered sequence of objects, whereas dictionaries are unordered sets. However, the main difference is that items in dictionaries are accessed via keys and not via their position.

```
# Python Program to Count Occurrence of a Character in a String
```

```
string = input("Please enter your own String : ")
char = input("Please enter your own Character : ")
i = 0
count = 0

while(i < len(string)):
    if(string[i] == char):
        count = count + 1
    i = i + 1

print("The total Number of Times ", char, " has Occurred = " , count)
```

10.What are python Sets and Set items ? Explain.

ANS-

- In Python, **Set** is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.
- The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.
- This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes like we do in lists.

```
# Python program to

# demonstrate sets


# Same as {"a", "b", "c"}

myset = set(["a", "b", "c"])

print(myset)


# Adding element to the set

myset.add("d")

print(myset)
```

Output:

{‘c’,’b’,’a’}

{‘d’,’c’,’b’,’a’}

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Sets are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created , you cannot change its items, but you can add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

Example:

Duplicate values will be ignored:

```
set = {"apple", "banana", "cherry", "apple"}

print(set)
```

Output:

{ ‘banana’,’cherry’,’apple’}

11] Explain how to remove duplicates from a list of set instructor?

Answer: To remove the duplicates from a list, you can make use of the built-in function set(). The specialty of set() method is that it returns distinct elements.
We have a list: [1,1,2,3,2,2,4,5,6,2,1]. The list has many duplicates which we need to remove and get back only the distinct elements. The list is given to the set() built-in function. Later the final list is displayed using the list() built-in function, as shown in the example below.
The output that we get is distinct elements where all the duplicates elements are eliminated.

```
my_list = [1,1,2,3,2,2,4,5,6,2,1]
```

```
my_final_list =set(my_list)

print(list(my_final_list))
```

Output:
[1, 2, 3, 4, 5, 6]

12. How to :

a) Access Set items

Answer:

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop or ask if a specified value is present in a set, by using the in keyword.

Example:

```
# Creating a set

set1 = set(["A", "B", "A" , "C"])
print("\nInitial set")  print(set1)

# Accessing element using
# for loop  print("\nElements of
set: ")  for i in set1:
    print(i, end=" ")
```

Output:

```
Initial set:

{'A', 'B', 'C'}


Elements of set:

A B C
```

12.How to :

b) Add Set items

Answer:

The set add() method adds a given element to a set if the element is not present in the set.

Syntax:

Set.add(elem)

Where add() method doesn't add an element to the set if it's already present in it otherwise it

Will get added to the set.

Example:

GEEK = {'g', 'e'}

GEEK.add('s')

Print('Letters are:', GEEK)

Output:

Letters are:{'e', 's', 'g'}

12. How to:
c) Remove Set items
ANS-

Method 1: Use of discard() method

The built-in method, discard() in Python, removes the element from the set only if the element is present in the set. If the element is not present in the set, then no error or exception is raised and the original set is printed.
If the element is present in the set:

```
# Python program to remove random elements of choice
# Function to remove elements using discard()
def Remove(sets):
    sets.discard(20)
    print (sets)

# Driver Code
sets = set([10, 20, 26, 41, 54, 20])
Remove(sets)
```

Output:
{41, 10, 26, 54}

Method 2: Use of remove() method
The built-in method, remove() in Python, removes the element from the set only if the element is present in the set, just as the discard() method does but If the element is not present in the set, then an error or exception is raised.
If the element is present in the set:

```
# Python program to remove random elements of choice
# Function to remove elements using remove()
def Remove(sets):
    sets.remove("aakash")
```

```
    print (sets)

# Driver Code
sets = set(["ram", "aakash", "kaushik", "anand", "prashant"])
Remove(sets)
```

Output:
{‘ram’, ‘anand’, ‘prashant’, ‘ kaushik’}

13.Write a note on a) Set Operators b) Set Methods

ANS-

A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).

a) Set Operations

- Set Union

  Union of A and B is a set of all elements from both sets.

  Union is performed using | operator. Same can be accomplished using the union() method.

  Example:

  ```
  # Set union method

  # initialize A and B

  A = {1, 2, 3, 4, 5}        B = {4, 5, 6, 7, 8}

  # use | operator

  print(A | B)
  ```

  Output - {1, 2, 3, 4, 5, 6, 7, 8}

- Set Intersection

  Intersection of A and B is a set of elements that are common in both the sets.

  Intersection is performed using & operator. Same can be accomplished using the intersection() method.

  Example:

  ```
  # Intersection of sets

  # initialize A and B

  A = {1, 2, 3, 4, 5}        B = {4, 5, 6, 7, 8}

  # use & operator
  ```

print(A & B)

Output - {4, 5}

- Set Difference

  Difference of the set B from set A(A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of elements in B but not in A.

  Difference is performed using - operator. Same can be accomplished using the difference() method.

Example:

```
# Difference of two sets

# initialize A and B

A = {1, 2, 3, 4, 5}        B = {4, 5, 6, 7, 8}

# use - operator on A

print(A - B)
```

Output - {1, 2, 3}

- Set Symmetric Difference

  Symmetric Difference of A and B is a set of elements in A and B but not in both (excluding the intersection).

  Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

Example:

```
# Symmetric difference of two sets

# initialize A and B

A = {1, 2, 3, 4, 5}        B = {4, 5, 6, 7, 8}

# use ^ operator

print(A ^ B)
```

Output - {1, 2, 3, 6, 7, 8}

b) Set methods

- add() - Adds an element to the set
- clear() - Removes all the elements from the set
- copy() - Returns a copy of the set
- difference() - Returns a set containing the difference between two or more sets
- difference_update() - Removes the items in this set that are also included in another, specified set
- discard() - Remove the specified item
- intersection() - Returns a set, that is the intersection of two other sets
- intersection_update() - Removes the items in this set that are not present in other, specified set(s)

- isdisjoint() - Returns whether two sets have a intersection or not
- issubset() - Returns whether another set contains this set or not
- issuperset() - Returns whether this set contains another set or not
- pop() - Removes an element from the set
- remove() - Removes the specified element
- symmetric_difference() - Returns a set with the symmetric differences of two sets
- symmetric_difference_update() - inserts the symmetric differences from this set and another
- union() - Return a set containing the union of sets
- update() - Update the set with the union of this set and others

14.Ilustrate the following set methods with example

a) Intersection() - Returns a set that is an intersection of two other sets.

b) union() - Returns a set containing the union of sets.

c) issubset() - Returns True or False depending on whether the set is a subset of another set.

d) difference() - Returns a set containing a difference of two or more sets.

e) update() - Update the set with the union of this set and others

f) discard() - Removes the specified item

Examples:

```
>>> a = {1,2,3,4}
>>> b = {2,4,6,8}
>>> a.intersection(b)
{2, 4}
>>> a.union(b)
{1, 2, 3, 4, 6, 8}
```

```
>>> a.issubset(b)
False
>>> c = {1,2}
>>> c.issubset(a)
True
>>> a.difference(b)
{1, 3}
>>> a.update(b)
>>> a
{1, 2, 3, 4, 6, 8}
>>> a.discard(8)
>>> a
{1, 2, 3, 4, 6}
```

15.Write a note on Random Module.

ANS-

Functions in the **random** module depend on a pseudo-random number generator function random(), which generates a random float number between 0.0 and 1.0.

- **random.random():** Generates a random float number between 0.0 to 1.0. The function doesn't need any arguments.

  >>>import random

  >>>random.random()

  0.645173684807533

- **random.randint():** Returns a random integer between the specified integers.

  >>>import random

  >>>random.randint(1,100)

  95

  >>> random.randint(1,100)

  49

- **random.randrange():** Returns a randomly selected element from the range created by the start, stop and step arguments. The value of start is 0 by default. Similarly, the value of step is 1 by default.

```
>>>random.randrange(1,10)

2

>>>random.randrange(1,10,2)

5

>>>random.randrange(0,101,10)

80
```

- **random.choice():** Returns a randomly selected element from a non-empty sequence. An empty sequence as argument raises an IndexError.

```
>>>import random

>>>random.choice('computer')

't'

>>>random.choice([12,23,45,67,65,43])

45

>>>random.choice((12,23,45,67,65,43))

67
```

- **random.shuffle():** This functions randomly reorders the elements in a list.

```
>>>numbers=[12,23,45,67,65,43]

>>>random.shuffle(numbers)

>>>numbers

[23, 12, 43, 65, 67, 45]

>>>random.shuffle(numbers)

>>>numbers

[23, 43, 65, 45, 12, 67]
```

16) Expand ASCII.Give the decimal ascii code and binary code for &,a,0,k,@,?,!.

ANS-

ASCII -American Standard Code For Information Interchange

Decimal ASCII code for

&-38

a-97

0-48

K-75

@-64

?-63

!-33

Binary ASCII code for

&-00100110

a-01100010

0-00000000

K-01001011

@-01000000

?-00111111

!-00100001

17) Explain ord() and chr() with examples.

ANS-

Pythons built-in function chr() is used for converting an Integer to a Character, while the function ord() is used to do the reverse, i.e, convert a Character to an Integer.

Example:

1) Res = ord('A')

print(Res)

Output: 65

2) Res = chr('A')

print(Res)

Output: A

18. Write a function encoding() that takes a string as input and prints the ASCII code—in

Decimal, hex, and binary notation—of every character in it.

ANS- encoding('dad')

Char	Decimal	Hex	Binary
D	100	64	1100100
A	97	61	1100001
D	100	64	1100100

Ans – def encoding(s):

```
For x in s:
    Print(x)
    Print(ord(x))
    Print(hex(ord(x)))
    Ch= format(ord(x),'b')
    Print(ch)

Print("Char Decimal Hex Binary")
Encoding('dad')
```

19.Write function char(low,high) that points the characters corresponding to ASCII decimal codes i for all values of i from low up to and including high.

>>>cha

r(62,67

) 62: >

63: ?

64: @

65: A

66: B

67: C

Solution: def char(low,high) :

```
        for i in
                range(low,
                high+1,1):
                print(i,":",c
                hr(i))

char(62,67)
```

20) What is a Class? How to define a class in Python? How to instantiate a class and how the class members are accessed?

→A **class** is a code template for creating objects. In **python** a **class** is created by the keyword **class**.

→ The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows −

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

< The class suite consists of all the component statements defining class members, data attributes and functions. >

→ Instantiating a class is creating a copy of the class which inherits all class variables and methods. Instantiating a class in Python is simple. To instantiate a class, we simply call the class as if it were a function, passing the arguments

that the __init__ method defines. The return value will be the newly created object.

Example

```
class Foo():

        def __init__(self,x,y):

            print x+y

f = Foo(3,4)
```

→

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows-

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

→example

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)
#This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
#This would create second object of Employee class"


emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

Output:

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

21)Differentiate class variables and instance variables.

Write a program that has a class Point with attributes as X and Y co-ordinates.

Create two objects of this class and find the mid-point of both the points. Add a

method reflex_x to class point ,which returns a new point ,which is the reflection

of the point about the x_axis.

Class Variables — Declared inside the class definition (but outside any of the instance methods). They are not tied to any particular object of the class, hence shared across all the objects of the class. Modifying a class variable affects all objects instance at the same time.

Instance Variable — Declared inside the constructor method of class (the __init__ method). They are tied to the particular object instance of the class, hence the contents of an instance variable are completely independent from one object instance to the other.

```
class Point:
    def reflex_x(P) :
        r = Point()
        r.x = p.x
        r.y = -p.y
       return(r)
  def midpoint(p1, p2) :
    m = point()
    m.x = (p1.x + p2.x)/2
    m.y = (p1.y + p2.y)/2
    return Point(mx, my)
p1 = point()
p2 = point()
p1.x =5
p1.y =6
p2.x = 10
p2.y =10
ref = p1.reflex_x()
print(ref.x, ",", ref.y)
mid = p1.midpoint(p2)
print(mid.x,mid.y)
```

22. write the python program to express instances as return values to define a class RECTANGLE with members width, height, corner_x, corner_y and member function : to find center, area, perimeter of a rectangle.

Ans:

```
class rectangle:
    def _init_(self, width, height, corner_x, corner_y):
        self.width = width
        self.height = height
        self.corner_x = corner_x
        self.corner_y = corner_y

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2*self.width + 2*self.height

    def center(self):
        return ((corner_x + self.width)/2), ((corner_y + self.height)/2)

    a=int(input("enter width:"))
    b=int(input("enter height:"))
    x=int(input(“enter corner_x”))
    y=int(input(“enter corner_y”))
    obj = rectangle(a,b,x,y)
    print("area=",obj.area)
    print("perimeter=",obj.perimeter)
    print("center=", obj.center)
```

output:

enter width : 2

enter height : 3

enter corner_x: 6

enter corner_y: 3

area =6

perimeter = 10

center(7.0 , 4.5)

23. What is the difference between method and function? Explain the working of the init method with suitable code.

A.)

M
e
t
h
o
d

1. Method is called by its name, but it is **associated to an object** (dependent).
2. A method is **implicitly passed the object** on which it is invoked.
3. It **may or may not return any data.**
4. A method **can operate on the data (instance variables) that is contained by the corresponding class**

Functions

1. Function is block of code that is also **called by its name**. (independent)
2. The function can have different parameters or may not have any at all. If **any data (parameters)** are passed, they are **passed explicitly**.
3. It **may or may not return any data.**
4. Function does not deal with Class and its instance concept.

Difference between method and function

1. Simply, function and method both look similar as they perform in almost similar way,
   but the key difference is the concept of '**Class and its Object**'.

2. Functions can be called **only by its name**, as it is defined independently. But methods c**an't be called by its name** only, we need to invoke the class by a reference of that class in which it is defined, i.e. method is defined within a class and hence they are dependent on that class.

The _____init___method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
class Person:
   def init (self, name):
      self.nam
   e = name
   def
   say_hi(self
   ):
      print('Hello, my name is', self.name)

p =
Person('J
ohn')
p.say_hi
()
```

In the above example, a person name John is created. While creating a person, "John" is passed as an argument, this argument will be passed to the _____init_method to initialize the object. The keyword self represents the instance of a class and binds the attributes with the given arguments. Similarly, many objects of Person class can be created by passing different names as arguments.

Question 24:

Develop a class BankAccount that supports the following methods:

- __init__( ): initializes the bank account balance to the value of the input argument, or to 0 if no input argument is give
- withdraw ( ):takes an amount as input and withdraws it from the balance
- Deposit( ) : takes an amount as input and adds it to the balance
- Balance( ): returns the balance on the account

```
>>> x= BankAccount(700)

>>>x.balance()

700

>>>x.withdraw(70)

>>>x.balance()
```

630

>>>x.deposit(7)

>>>x.balance()

637

Answer:

```
class BankAccount:
    def __init__(self, amount=0):
        self.amount = amount

    def withdraw(self, amount):
        if self.amount > amount:
            self.amount = self.amount - amount

    def deposit(self, amount):
        self.amount = self.amount + amount

    def balance(self):
        return self.amount
```

25) **Write a container class called PriorityQueue. The class should supports methods:**

- **insert(): Takes a number as input and adds it to the container**
- **min(): Returns the smallest number in the container**
- **removeMin(): Removes the smallest number in the container**
- **isEmpty(): Returns True if container is empty, False otherwise** The overloaded operator len() should also be supported. class PriorityQueue: def __init__(self):

```
            self.queue=[]
    def insert(self,item):
```

```
                        self.queue.append(item)
            def min(self):
   return min(self.queue)    def
removeMin(self):
            m=min(self.queue)
self.queue.remove(m)
  def size(self):      return len(self.queue)
       def isEmpty(self):            l=len(self.queue)
                        if l==0:
                    return True            else:
                                return False
```

```
>>> pq = PriorityQueue()
>>> pq.insert(3)
>>> pq.insert(1)
  >>> pq.insert(5)
  >>> pq.insert(2)
>>> pq.min()
  1
  >>> pq.removeMin()
>>> pq.min()
2
  >>> pq.size()
  3
  >>> pq.isEmpty()
  False
```

26. What are Container Classes ?explain with suitable example.

Solution: A container class is a class that is used to hold objects in memory or external storage. A container class acts as a generic holder. A container class has a predefined behaviour and a well-known interface. A container class is a supporting class whose purpose is to hide the topology used for maintaining the list of objects in memory.
Containers are any object that holds an arbitrary number of other objects. Generally, containers provide a way to access the contained objects and to iterate over them.

Examples of containers include tuple, list, set, dict; these are the built-in containers. More container types are available in the collection's module.

Example: CONTAINER CLASS QUEUE

A queue is a container type that abstracts a queue, such as a queue of shoppers in a supermarket waiting at the cashiers. In a checkout queue, shoppers are served in a first-in first-out (FIFO) fashion. A shopper will put himself at the end of the queue and the first person in the queue is the next one served by the cashier. More generally, all insertions must be at the rear of the queue, and all removals must be from the front.

Methods:

enqueue(item): Add item to the end of the queue

dequeue() :Remove and return the element at the front of the queue

isEmpty() :Returns True if the queue is empty, False otherwise

IMPLEMENTING A QUEUE CLASS:

The queue can be empty or contain an unbounded number of items. It also has to maintain the order of items, as that is essential for a (fair) queue. So, the built-in type which can be used to store, in order, an arbitrary number of items and allow insertions on one end and deletions from the other The list type certainly satisfies these constraints.

```
class Queue:
        #a classic queue
        class def init_(self):
                #instantiates an
                 empty list self.q =
                 []
        def isEmpty(self):
                 #returns True if queue is empty, False
                 otherwise return (len(self.q) == 0)
        def enqueue (self, item):
                #insert item at rear of
                 queue return
                 self.q.append(item)
        def dequeue(self):
                #remove and return item at front of
                 queue return self.q.pop(0)
```

27.Write a note on User defined Exceptions.

Python throws errors and exceptions, when there is a code gone wrong, which may cause program to stop abruptly. Python also provides exception handling method with the help of try-except. Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError and FileNotFoundError. A user can create his own error using exception class.

Creating User-defined Exception

Programmers may name their own exceptions by creating a new exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in "Error" similar to naming of the standard exceptions in python. For example:

```
# A python program to create
user-defined exception # class MyError
is derived from super class Exception
class MyError(Exception):
 #
  Construct
  or          or
  Initializer
  def___init
  (self,
  value):
      self.value = value
    #__str____is  to
    print()          the
    value    def__str
    (self):
       return(repr(self.value))
try:
    raise(MyError(3*2))
# Value of Exception is
stored in error except
MyError as error:
    print('A New Exception
occured: ',error.value) Output: A
New Exception occured: ', 6
```

28.

1.

Inheritance allows us to define a class that inherits all the methods and properties from another class

2. Difference

Multiple Inheritance vs Multilevel Inheritance	
Multiple Inheritance is an Inheritance type where a class inherits from more than one base class.	Multilevel Inheritance is an Inheritance type that inherits from a derived class, making that derived class a base class for a new class.
Usage	
Multiple Inheritance is not widely used because it makes the system more complex.	Multilevel Inheritance is widely used.
Class Levels	
Multiple Inheritance has two class levels namely, base class and derived class.	Multilevel Inheritance has three class levels namely, base class, intermediate class and derived class.

29) Class: A Class is like an object constructor, or a "blueprint" for creating objects.

Instantiation : Instantiating a class is creating a copy of the class which inherits all class variables and methods.

Object: An object is simply a collection of data (variables) and methods (functions) that act on those data.

Function Overloading: Overloading, in the context of programming, refers to the ability of a function or an operator to behave in different ways depending on the parameters that are passed to the function, or the operands that the operator acts on.

Data Member: A class variable or instance variable that holds data associated with a class and its objects

Reg No:18GAEI6038

Name : Raju K R

Operator Overloading in Python

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

Overloading binary + operator in Python :

When we use an operator on user defined data types then automatically a special function or magic function associated with that operator is invoked. Changing the behavior of operator is as simple as changing the behavior of method or function. You define methods in your class and operators work according to that behavior defined in methods. When we use + operator, the magic method __add__ is automatically invoked in which the operation for + operator is defined. There by changing this magic method's code, we can give extra meaning to the + operator.

Code:

```
class A:
    def __init__(self, a):
        self.a = a
    # adding two objects
    def __add__(self, o):
        return self.a + o.a
ob1 = A(1)
ob2 = A(2)
ob3 = A("Geeks")
ob4 = A("For")
print(ob1 + ob2)
print(ob3 + ob4)
```

Output :

```
3
GeeksFor
```

Overloading equality and less than operators :

```
class A:
    def __init__(self, a):
        self.a = a
    def __lt__(self, other):
```

```
            if(self.a<other.a):
                return "ob1 is lessthan ob2"
            else:
                return "ob2 is less than ob1"
        def __eq__(self, other):
            if(self.a == other.a):
                return "Both are equal"
            else:
                return "Not equal"
ob1 = A(2)
ob2 = A(3)
print(ob1 < ob2)
ob3 = A(4)
ob4 = A(4)
print(ob1 == ob2)
```

Output :

```
ob1 is lessthan ob2
Not equal
```

31.Define i)Class inheritance ii)Data hiding iii)Method overloading iv)child method v)class variable

i)Class inheritance:

The technique of creating a new class from an existing class is called class inheritance.

ii)Data hiding:

Data encapsulation, also called data hiding organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.

iii)Method Overloading:

Method overloading is a class having methods that are the same name with different arguments. Arguments different will be based on a number of arguments and types of arguments.

It is used in single class.

iv)Child Method:

A child method is a method present in child class derived from a parent class.

v)Class variable:

A variable that is shared by all the instances of a class.Class variables are defined within a class but outside any of the class's methods.

RAMYA S H

18GAEI6040 PYTHON PROGRAMMING ASSIGNMENT 2

32) Every Object has an associated Namespace. Justify the statement with Example.

Everything in Python is an object. All the data types such as numbers, strings, functions, classes are all objects.

Python has to keep track of all these objects and their names, and avoid conflicts in names it uses namespaces.

A namespace is a system tohave a unique name for each and every object in Python.

Python itself maintains a namespace in the form of a Python dictionary

Variables are names (identifiers) that map to objects. A *namespace* is a dictionary of variable names (keys) and their corresponding objects (values).

Every Object Has an Associated Namespace :-

Every Python object has its own separate namespace.

When we instantiate a new object of type Point and give it name point, as in

```
>>> point = Point()
```

a new namespace called point gets created as shown

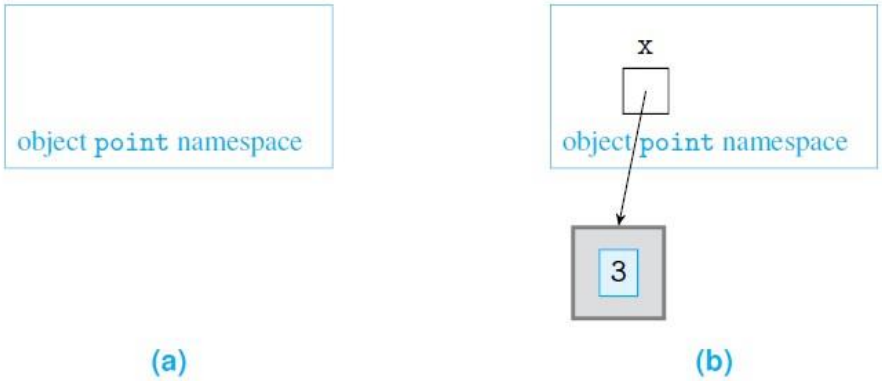


Figure 8.3 The namespace of an object. (a) Every `Point` object has a namespace. (b) The statement `point.x = 3` assigns 3 to variable `x` defined in namespace `point`.

suppose we create two point objects :-

```
>>> a = Point()
>>> a.setx(3)   // sets x coordinate of a point
>>> a.sety(4)  // sets y coordinate of a point


>>> b = Point()
>>> b.setx(5)
>>> b.sety(-2)
```

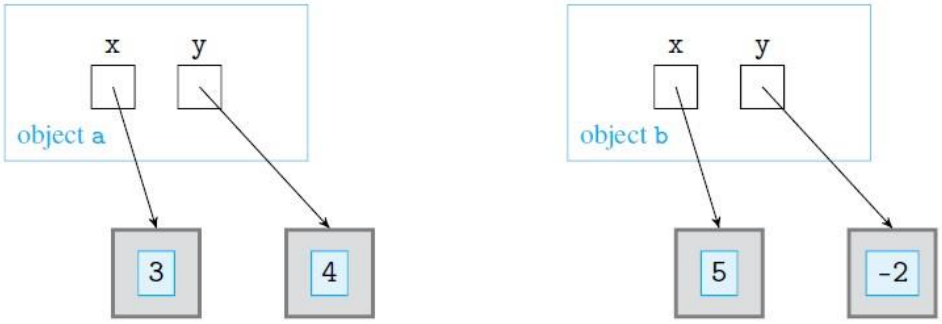


Figure 8.4 Instance variables. Each object of type `Point` has its own instance variables `x` and `y`, stored in the namespace associated with the object.

instance variables x and y can be accessed by specifying the appropriate instance(object) :

```
>>> a.x
3
>>> b.x
5
```

They can also be changed directly as well:

```
>>> a.x = 7
>>> a.x
7
```

PYTHON PROGRAMMING ASSIGNMENT

33.

Develop a new class called Animal that abstracts animals and supports three methods:

• setSpecies(species): Sets the species of the animal object to species.

• setLanguage(language): Sets the language of the animal object to language.

• speak(): Prints a message from the animal as shown

below. Here is how we want the class to behave:

>>> snoopy = Animal()

>>> snoopy.setpecies('dog')

>>> snoopy.setLanguage('bark')

>>> snoopy.speak() I am a dog and I bark.

Sol: We start the class definition with the first line:

class Animal:

Now, in an indented code block, we define the three class methods, starting with method setSpecies(). Even though the method setSpecies() is used with one argument (the animal species), it must be defined as a function that takes two arguments: the argument self that refers to the object invoking the method and the species argument:

```
def      setSpecies(self,
           species):
           self.species   =
           species
```

We have named the instance variable species the same as the local variable species. Because the instance variable is defined in the namespace self and the local variable is defined in the local namespace of the function call, there is no name conflict.

The implementation of method setLanguage() is similar to the implementation of setSpecies. The method speak() is used without input arguments; therefore, it must be defined with just input argument self.

CODE :

```
class Animal: #represents an animal
        def setSpecies (self, species): #sets the animal
```

```
        species self.spec = species
    def setLanguage (self, language): #sets the animal language
        self.lang = language
    def speak (self): #prints a sentence by the animal
        print ('I am a {} and I {}.'.format(self.spec, self.lang))
```

34) Define Constructor. Explain the types of constructor with examples?

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 class is created.

In Python the __init__ () method is called the constructor and is always called when an object is created.

Syntax of constructor declaration:

```
def __init__(self):
    # body of the constructor
```

Types of constructors:

default constructor: The default constructor is simple constructor which doesn't accept any arguments. Its definition has only one argument which is a reference to the instance being constructed.

parameterized constructor: constructor with parameters is known as parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Default constructor example:

```
class DemoClass:
```

```
        num = 10
      # non-parameterized constructor
        def __init__(self)
self.num = 999 # a
method          def
read_number(self):
print(self.num) #
creating object of the
class obj = DemoClass
()

# calling the instance method using the
object obj obj. read_number () Output:
999
```

Parameterized constructor

example:

```
class DemoClass:
num = 101

        # parameterized
constructor          def
__init__(self, data):
self.num = data

        # a method
def
```

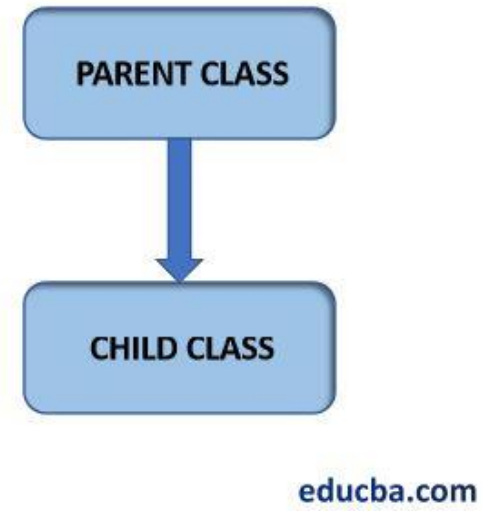
```
read_number(self)
:
        print(self.num)

# creating object of the class
# this will invoke parameterized
constructor obj = DemoClass (55)

# calling the instance method using the
object obj obj. read_number ()

# creating another object of the
class obj2 = DemoClass (66)

# calling the instance method using the
object obj obj2.read_number ()
```

Output:

```
55
66
```

36)a) Explain single inheritance

In python single inheritance, a derived class is derived only from a single parent class and allows class to derive behaviour and properties from a single base class. Thus enables code reusability of parent class and adding new features to a class makes code more readable, elegant and less redundant. And thus single inheritence is much more safer than multiple inheritance if it's done in right way and enables derived class to call parent class method and also to override parent classes existing methods.

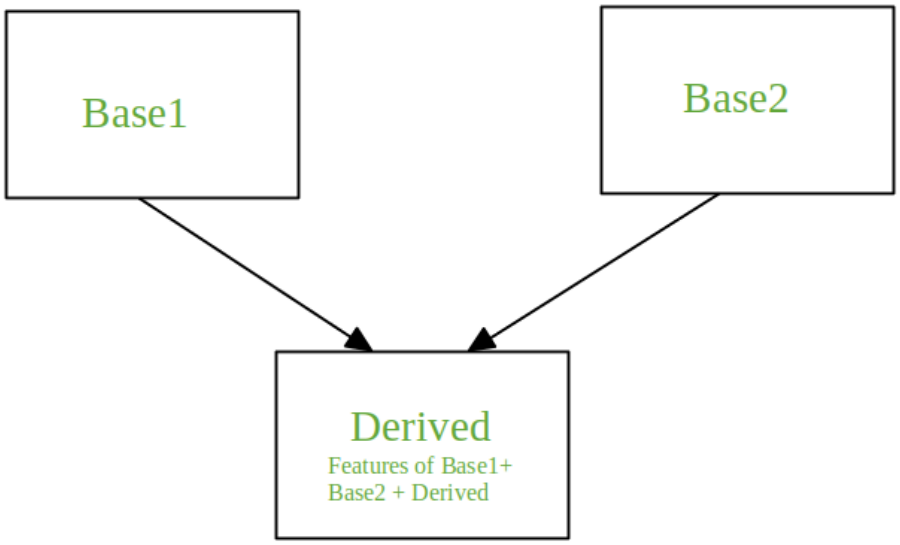


36. b.

Inheritance is the mechanism to achieve the re-usability of code as one class (child class) can derive the properties of another class(parent class).

Multiple Inheritance

When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case.



Syntax:

```
Class Base1:
      Body of the class
Class Base2:
        Body of the class
Class Derived (Base1, Base2):
    Body of the class
```

Python Multiple Inheritance Example

```
# Parent class 1
class TeamMember(object):
   def __init__(self, name, uid):
      self.name = name
      self.uid = uid

# Parent class 2
class Worker(object):
   def __init__(self, pay, jobtitle):
      self.pay = pay
      self.jobtitle = jobtitle

# Deriving a child class from the two parent classes
class TeamLeader(TeamMember, Worker):
   def __init__(self, name, uid, pay, jobtitle, exp):
      self.exp = exp
      TeamMember.__init__(self, name, uid)
      Worker.__init__(self, pay, jobtitle)
      print("Name: {}, Pay: {}, Exp: {}".format(self.name, self.pay, self.exp))

TL = TeamLeader('Jake', 10001, 250000, 'Scrum Master', 5)
```

The output is: Name: Jake, Pay: 250000, Exp: 5

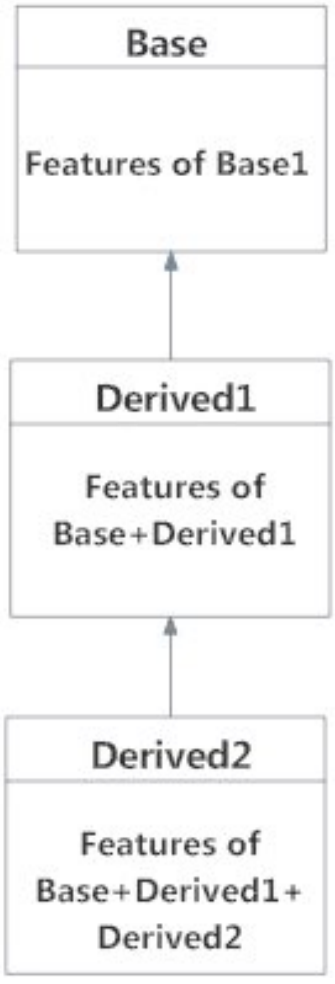
Spoorthi K J 18GAEI6047

A class can be derived from an already derived class.
In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

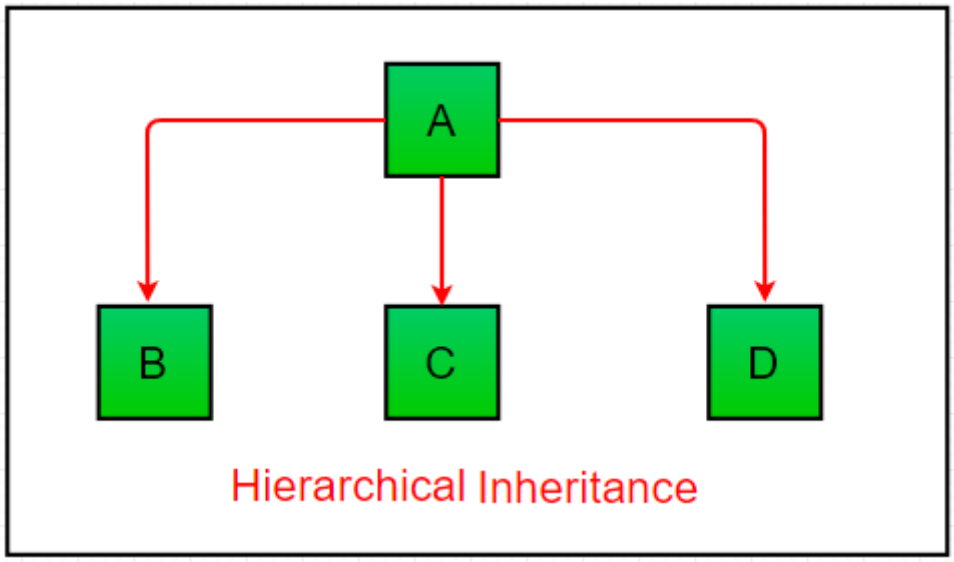
An example with corresponding visualisation is given below.

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

Here, the Derived1 class is derived from the Base class, and the Derived2 class is derived from the Derived1 class.



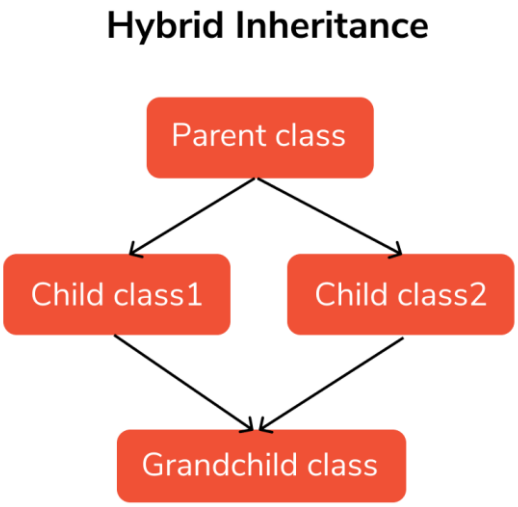
Hierarchical Inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.




```
# Python program to demonstrate
# Hierarchical inheritance


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

# Derived class1
class Child1(Parent):
      def func2(self):
          print("This function is in child 1.")

# Derivied class2
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")

# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

36 e) Explain Hybrid Inheritance with examples.

Hybrid Inheritance is a combination of more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.




```
# Python program to demonstrate hybrid inheritance

class School:
    def func1(self):
        print("This function is in school.")


class Student1(School):
    def func2(self):
        print("This function is in student 1. ")


class Student2(School):
    def func3(self):
        print("This function is in student 2.")


class Student3(Student1, Student2):
    def func4(self):
        print("This function is in student 3.")


# Driver's code

object = Student3()

object.func1()

object.func2()
```

Output:

This function is in school.

This function is in student 1.

In the above example, student1 and student2 inherit from the class School. Student3 inherits from both student1 and student2. This type of inheritance is called hybrid inheritance.

37. What is recursion? Explain the process of solving the problem recursively. Implement recursive method reverse() that takes a non-negative integer as input and prints the digits of n vertically, starting with the low-order digit.

```
>>>reverse(3124)
4
2
1
3
```

Ans: Recursion is a problem solving technique that expresses the solution to a problem in terms of solutions to subproblems of the original problem.
The process of solving the problem recursively:
Recursive Functions
Here is an example that illustrates what we mean by a function that calls itself:

```
def countdown(n):
       print(n)
       countdown(n-1)
```

In the implementation of function countdown(), the function countdown() is called. So,function countdown() calls itself. When a function calls itself, we say that it makes arecursive call.
Letâ€™s understand the behavior of this function by tracing the execution of function call countdown(3):
â€¢ When we execute countdown(3), the input 3 is printed and then countdown() is called on the input decremented by 1â€”that is, 3 1 = 2. We have 3 printed on the screen, and we continue tracing the execution of countdown(2).
â€¢ When we execute countdown(2), the input 2 is printed and then countdown() is called on the input decremented by 1â€”that is, 2 1 = 1. We now have 3 and 2 printed on the screen, and we continue tracing the execution of countdown(1).
â€¢ When we execute countdown(1), the input 1 is printed and then countdown() is called on the input decremented by 1â€”that is, 1 1 = 0. We now have 3, 2, and 1 printed on the screen, and we continue tracing the execution of countdown(0).
â€¢ When we execute countdown(0), the input 0 is printed and then countdown()is called on the input, 0, decremented by 1â€”that is, 0 1 = 1. We now have 3,2, 1, and 0 printed on the screen, and now we continue tracing the execution of countdown(-1).
â€¢ When we execute countdown(-1),...
It seems that the execution will never end. Letâ€™s check:

```
>>> countdown(3)
3
2
1
0
-  1
-  2
-  3
-  4
-  5
```

```
-  6
...
```

The behavior of the function is to count down, starting with the original input number. If we let the function call countdown(3) execute for a while, we get:

```
...
-  973
-  974
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
   countdown(3)
File "/Users/lperkovic/work/book/Ch10-RecursionandAlgs/ch10.py"...
countdown(n-1)
...
```

And after getting many lines of error messages, we end up with:

```
...
RuntimeError: maximum recursion depth exceeded.
```

OK, so the execution was going to go on forever, but the Python interpreter stopped it. We will explain why the Python VM does this soon. The main point to understand right now is that a recursive function will call itself forever unless we modify the function so there is a stopping condition.
To show this, suppose that the behavior we wanted to achieve with the countdown() function is really:

```
>>> countdown(3)
3
2
1
Blastoff!!!
or
>>> countdown(1)
1
Blastoff!!!
or
>>> countdown(0)
Blastoff!!!
```

Function countdown() is supposed to count down to 0, starting from a given input n; when 0 is reached, Blastoff!!! should be printed.To implement this version of countdown(), we consider two cases that depend onthe value of the input n. When the input n is 0 or negative, all we need to do is print'Blastoff!!!':

```
def countdown(n):
 'counts down to 0'
if n <= 0: # base case
   print('Blastoff!!!')
else:
... # remainder of function
```

We call this case the base case of the recursion; it is the condition that will ensure that the recursive function is not going to call itself forever.

The second case is when the input n is positive. In that case we do the same thing we did before:

```
print(n)
countdown(n-1)
```

How does this code implement the function countdown() for input value n > 0? The insight used in the code is this: Counting down from (positivenumber) n can be done by printing n first and then counting down from n 1. This fragment of code is called the recursive step.

With the two cases resolved, we obtain the recursive function:

```
def countdown(n):
   'counts down to 0'
 if n <= 0: # base case
     print('Blastoff!!!')
 else: # n > 0: recursive step
         print(n) # print n first and then
         countdown(n-1) # count down from n-1
```

```
def reverse(n):
 'prints digits of n vertically starting with low-order digit'
 if n<10:          #base case:one digit number
   print n
 else:             #n has atleast 2 digits
   print(n%10) #prints last digit of n
   reverse(n//10) #recursively print in reverse all but the last digit.
```

38.Write the Python Program

a) Implementing function pattern() that takes a nonnegative integer n and prints a number pattern:

>>> pattern(0) 0

>>> pattern(1) 010

>>> pattern(2)

0102010

>>> pattern(3)

010201030102010

>>> pattern(4) 010201030102010401020103010201O

```
def pattern(n):
    if n == 0:
        return "0" # base case
    return "{0}{1}{0}".format(reflected_palindrome(n-1), n)
```

```
>>>print(pattern(3))
010201030102010
```

b) Implement recursive method pattern2() that takes a nonnegative integer as input and prints the pattern shown next. The patterns for inputs 0 and 1 are nothing and one star, respectively:

>>> pattern2(0)

>>> pattern2(1) *

>>> pattern2(2) *

```
**
*
```

>>> pattern2(3) *

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

```
def pattern(number):
    if number == 0 or number ==  1:
        return "\n*\n"
    return "{0} {1} {0}".format(print_start(number-1), "*"*number).replace(" ", "")
```

SUPREETA (18GAEI6052)

39)Recursive is used to develop a virus scanner. Explain with examples.

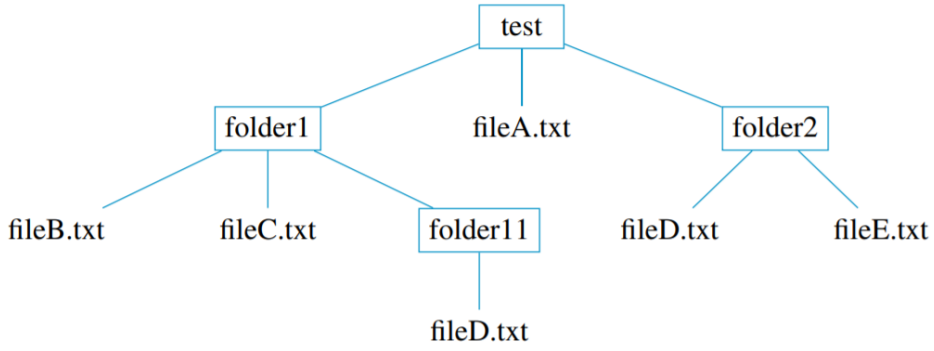
Ans: Recursion is used to develop a virus scanner, that is, a program that systematically looks at every file in the filesystem and prints the names of the files that contain a known computer virus signature. The signature is a specific string that is evidence of the presence of the virus in the file.

We use a dictionary to store the various virus signatures. It maps virus names to virus signatures:

```
>>> signatures = {'Creeper':'ye8009g2h1azzx33',
        'Code Red':'99dh1cz963bsscs3',
        'Blaster':'fdp1102k1ks6hgbc'}
```

The virus scanner function takes, as input, the dictionary of virus signatures and the pathname (a string) of the top folder or file. It then visits every file contained in the top folder, its subfolders, subfolders of its subfolders, and so on.

An example folder 'test' is shown in following figure together with all the files and folders that are contained in it, directly or indirectly.



The virus scanner would visit every file and could produce,

for example, this output:

```
File: test.zip
 >>> scan('test', signatures)
        test/fileA.txt,found virus Creeper
         test/folder1/fileB.txt, found virus Creeper
        test/folder1/fileC.txt, found virus Code Red
        test/folder1/folder11/fileD.txt, found virus Code
        Red test/folder2/fileD.txt, found virus Blaster
        test/folder2/fileE.txt, found virus Blaster
```

Because of the recursive structure of a filesystem (a folder contains files and other folders), we use recursion to develop the virus scanner function scan(). When the input pathname is the pathname of a file, the function should open, read, and search the file for virus signatures; this is the base case. When the input pathname is the pathname of a folder, scan() should recursively call itself on every file and subfolder of the input folder; this is the recursive step.

40. Explain run time analysis with example.

Sol. Run Time Analysis

The correctness of a program is the main concern. However, it is also important that the program is usable or even efficient. In this section, we continue the use of recursion to solve problems, but this time with an eye on efficiency. By this, we apply recursion to a problem that does not seem to need it and get a surprising gain in efficiency. Also, through this we take a problem that seems tailored for recursion and obtain an extremely inefficient recursive program.

Example:

Exponent function: The naïve method is been defined below to compute the exponent of the given value n times

```
def power(a, n):
        res = 1
        for i in range(n):
         res *= a
return res
```

If n is 10,000, then 10,000 multiplications are done in the naïve method. So we are going to develop a recursive function rpower() that takes inputs a and nonnegative integer n and returns a^n and check its efficiency.

```
def rpower(a,n):
        if n == 0:  # base case: n == 0
                return 1
         tmp = rpower(a, n//2)  # recursive step: n > 0
        If n % 2 == 0:
                return tmp*tmp
        else: # n % 2 == 1
                return a*tmp*tmp
```

We calculate the efficiency between the two functions by counting the number of operations i.e., by counting the number of multiplications and hence we infer that recursion led us to a way to do exponentiation that reduced the number of multiplications.

```
def timingAnalysis(func, start, stop, inc, runs):
        for n in range(start, stop, inc):
                acc = 0.0
        for i in range(runs):
                 acc += timing(func, n)
        formatStr = 'Run time of {}({}) is {:.7f} seconds.'
         print(formatStr.format(func.__name__, n, acc/runs))
```

Function timingAnalysis takes, as input, function func and numbers start, stop, inc, and runs. It first runs func on several inputs of size start and print the average run time. Then it repeats that for input sizes start+inc, start+2*inc, . . . up to input size stop.

Thus we analyze that run time of functions power() and rpower() as well as built-in operator ** by having different input sizes through the above function.

-Supriya P
18GAEI6053

Question No 41:

Write a Python Program for linear Search.

Soln:

```
A = [10,20,30,40,50]
search = 30
def linear(A,search):
    for i in range(len(A)):
        if(A[i] == search):
            return i
    return -1
print("the element is at position "+str(linear(A,search)))
```

output:

the element is at position 2

time complexity: O(n)

42.Write a Python Program for Binary Search.

Answer:

```
# Returns index of x in arr if present, else -1
Def binary_search(arr, low, high, x):
   If high >= low:
            Mid = (high + low) // 2
            If arr[mid] == x:
                    Return mid
            Elif arr[mid] > x:
                     Return binary_search(arr, low, mid – 1, x)
            Else:
                     Return binary_search(arr, mid + 1, high, x)
   Else:
         Return -1
Arr = [ 2, 3, 4, 10, 40 ]
X = 10
Result = binary_search(arr, 0, len(arr)-1, x)
If result != -1:
       Print(“Element is present at index”, str(result))
Else:
       Print(“Element is not present in array”)
```

Output:

```
Element is present at index 3
```

18GAEI6056

43 a. Python program for checking uniqueness of a list

```
# Python program to check if two
# to get unique values from list
# using traversal


# function to get unique values

def unique(list1):



    # intilize a null list

    unique_list = []



    # traverse for all elements

    for x in list1:

        # check if exists in unique_list or not
```

```
    if x not in unique_list:

        unique_list.append(x)

# print list

for x in unique_list:

b. print x
```

b.

Finding kth smallest item in unsorted list

```
Python3 program to find k'th smallest
# element



# Function to return k'th smallest
# element in a given array
 def kthSmallest(arr, n,
k):
```

```
    # Sort the given array

arr.sort()




    # Return k'th element in the

    # sorted array
     return arr[k-
1]



# Driver code
 if
__name__=='__main__':
     arr = [12, 3, 5, 7,
19]
     n =
len(arr)
     k =
2
     print("K'th smallest element
is",
           kthSmallest(arr, n,
k))
```

Output:

```
K'th smallest element is 5
```

C.

Computing more frequently occuring item in a list

Program to find most frequent

element in a list

```
def most_frequent(List):

    counter = 0

    num = List[0]



    for i in List:

        curr_frequency = List.count(i)

        if(curr_frequency> counter):

            counter = curr_frequency

            num = i
```

```
    return num



List = [2, 1, 2, 2, 1, 3]


print(most_frequent(List))  Output:
2
```

44. Write a python program for Tower of Hanoi Problem.

```
# Recursive Python function to solve tower of hanoi


def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
     if n == 1:
          print "Move disk 1 from rod",from_rod,"to rod",to_rod
          return
     TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
     print "Move disk",n,"from rod",from_rod,"to rod",to_rod
     TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)

# Taking user input.
n = int(input())
TowerOfHanoi(n, 'A', 'C', 'B')
# A, B, C are the name of rods
```

-Yashodhara Shastri
18GAEI6057

45.The recursive formula for computing the number of ways of choosing k items out of a set of n items, denoted C(n, k), is:

$$C(n,k) = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{if } n < k \\ C(n-1,k-1) + C(n-1,k) & \text{otherwise} \end{cases}$$

The first case says there is one way to choose no item; the second says that there is no way to choose more items than available in the set. The last case separates the counting of sets of k items containing the last set item and the counting of sets of k items not containing the last set item.

Write a recursive function combination() that computes C(n,k) using this recursive Formula.

>>> combinations(2, 1)

0

>>> combinations(1, 2)

2

>>> combinations(2, 5)

1

Ans:

```
Def combination(n , k):


   If k > n :

     Return 0

   If k==0 or k ==n :

      Return 1

    Return combination(n-1 , k-1)+combination(n-1 , k)
N = input(“enter the value of N:”)

K = input(“enter the value of K:”)

Print (combination(n,k) )
```

46) What is Tkinter Programming? What are the steps to create a GUI application using Tkinter? Explain with example.

=>Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

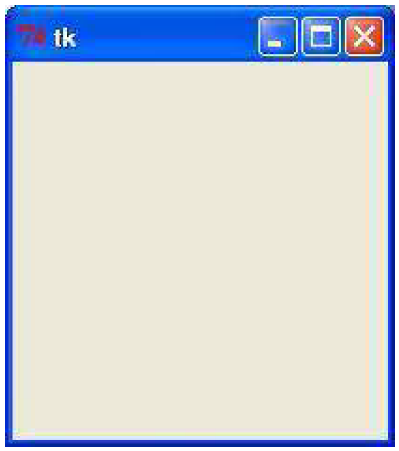
Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

- Import the Tkinter module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Example

```
#!/usr/bin/python3

import tkinter # note that module name has changed from Tkinter in Python 2 to tkinter in Python 3
top = tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

This would create a following window-



47. What are Tkinter Widgets? Explain any 5 with syntax.

Sol. Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter.

Operator	**Description**
Button	The Button widget is used to display the button in your application
Canvas	The Canvas widget is used to draw shapes, such as lines,ovals, polygons and rectangles, in your application
Checkbutton	The checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a containerwidget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.

48.

a) from tkinter import Tk, Label

```
root=Tk()
hello=Label(master=root,text="Hello GUI world")
hello.pack()
root.mainloop()
```

b) from tkinter import Tk , Label , PhotoImage

```
root=Tk()
photo=photoImage(file="peace.gif")
peace=Label(master=root, image=photo,width=300,height=180)
peace.pack()
root.mainloop()
```

Q> Illustrate multiple GUI widgets with example?

ANS>

1. TKinter Button : The Button widget is used to add buttons in a Python application

syntax to create this widget –

w = Button (master, option=value, ...)

example :

```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
def helloCallBack():
  tkMessageBox.showinfo( "Hello Python", "Hello World")
B = Tkinter.Button(top, text ="Hello", command = helloCallBack)
B.pack()
top.mainloop()
```

2. TKinter CheckButton: The Checkbutton widget is used to display a number of options to a user as toggle buttons.

syntax to create this widget –

w = Checkbutton (master, option, ...)

example:

```
from Tkinter import *
import tkMessageBox
import Tkinter

top = Tkinter.Tk()
CheckVar1 = IntVar()
CheckVar2 = IntVar()
C1 = Checkbutton(top, text = "Music", variable = CheckVar1, \
        onvalue = 1, offvalue = 0, height=5, \
        width = 20)
C2 = Checkbutton(top, text = "Video", variable = CheckVar2, \
```

```
            onvalue = 1, offvalue = 0, height=5, \
            width = 20)
C1.pack()
C2.pack()
top.mainloop()
```

3. TKinter Label: This widget implements a display box
where you can place text or images.

syntax to create this widget −

```
w = Label ( master, option, ... )
```

exaample:

```
from Tkinter import *
root = Tk()
var = StringVar()
label = Label( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

4. TKinter Message: This widget provides a multiline and noneditable object that
displays texts, automatically breaking lines and
justifying their contents.

syntax to create this widget −

```
w = Message ( master, option, ... )
```

example:

```
from Tkinter import *

root = Tk()
var = StringVar()
label = Message( root, textvariable=var, relief=RAISED )
```

```
var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

50. What is grid() method? How it is used in implementation of the phone dial GUI?

Ans: The `grid()` geometry manager organises widgets in a table-like structure in the parent widget. The master widget is split into rows and columns, and each part of the table can hold a widget. It uses

Option	Description
`column`	Specifies the column for the widget; default is column 0
`columnspan`	Specifies how many columns the widgets should occupy
`row`	Specifies the row for the widget; default is row 0
`rowspan`	Specifies how many rows the widgets should occupy

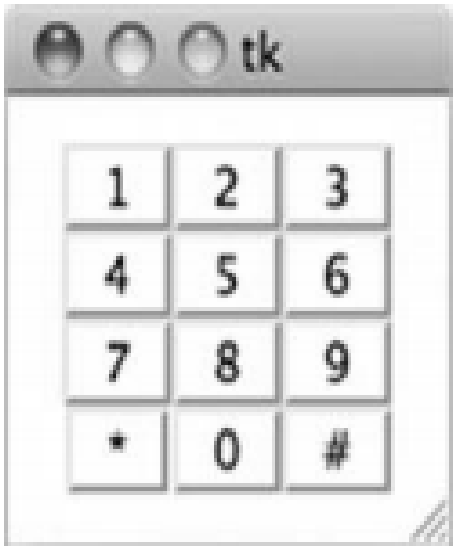
Syntax:

```
widget.grid(options)
```

When using method grid(), the master widget is split into rows and columns, and each cell of the resulting grid can store a widget. To place a widget in row r and column c, method grid() is invoked on the widget with the row r and column c as input arguments, as shown in this implementation of the phone dial GUI:

```
from tkinter import Tk, Label, RAISED
root = Tk()
labels = [['1', '2', '3'],      # phone dial label texts
          ['4', '5', '6'],      # organized in a grid
          ['7', '8', '9'],
          ['*', '0', '#']]

for r in range(4):       # for every row r = 0, 1, 2, 3
    for c in range(3):       # for every row c = 0, 1, 2
        # create label for row r and column c
        label = Label(root,
                      relief=RAISED,      # raised border
                      padx=10,            # make label wide
                      text=labels[r][c])  # label text
        # place label in row r and column c
        label.grid(row=r, column=c)

root.mainloop()
```

In lines 5 through 8, we define a two-dimensional list that stores in row r and column c the text that will be put on the label in row r and column c of the phone dial. Doing this facilitates the creation and proper placement of the labels in the nested for loop in lines 10 through 19. Note the use of the method grid() with row and column input arguments.



REG NO: 19GANSE060

51) Implement a GUI app that contains two buttons labelled “Local time” and “Greenwich time”. When the first button is pressed, the local time should be printed in the shell. When the second button is pressed, the Greenwich Mean time should be printed.

>>>

Local time

Day:08 Jul 2011

Time: 13:19:43 PM

Greenwich time

Day: 08 Jul 2011

Time: 18:19:46 PM

You can obtain the current Greenwich Mean Time using the function gmtime() from module time.

Ans :

```
from tkinter import Tk, Button
from time import gmtime,strftime
from time import strftime,localtime
import time
def Localtime():
obj=strftime('%a %d %b %Y\n Time: %H:%M:%S %p %Z\n', localtime())
print(obj)
```

```
def Greenwichtime():
obj= time.strftime("%a, %d %b %Y Time:%H:%M:%S %p %Z\n, time.gmtime())
print(obj)
root=Tk()
button1=Button(root,text='Local time',command=Localtime)
button2=Button(root, text='Greenwich time', command=Greenwichtime)
button1.pack()
button2.pack()
root.mainloop()
```

52 : Explain any 3 Entry and Text Widget methods

The Entry widget is used to accept single-line text strings from a user.

- If you want to display multiple lines of text that can be edited, then you should use the *Text* widget.
- If you want to display one or more lines of text that cannot be modified by the user, then you should use the *Label* widget.

Syntax

Here is the simple syntax to create this widget −

```
w = Entry( master, option, ... )
```

Parameters

- **master** − This represents the parent window.'
- **options** − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.
- Methods
- Following are commonly used methods for this widget −

Sr.No.	Method & Description
1	**delete (first, last=None)** Deletes characters from the widget, starting with the one at index first, up to but not including the character at position last. If the second argument is omitted, only the single character at position first is deleted.
2	**get()** Returns the entry's current text as a string.
3	**icursor (index)** Set the insertion cursor just before the character at the given index.
4	**index (index)** Shift the contents of the entry so that the character at the given index is the leftmost visible character. Has no effect if the text fits entirely within the entry.
5	**insert (index, s)** Inserts string s before the character at the given index.
6	**select_adjust (index)** This method is used to make sure that the selection includes the character at the specified index.
7	**select_clear()** Clears the selection. If there isn't currently a selection, has no effect.
8	**select_from (index)** Sets the ANCHOR index position to the character

	selected by index, and selects that character.
9	**select_present()** If there is a selection, returns true, else returns false.
10	**select_range (start, end)** Sets the selection under program control. Selects the text starting at the start index, up to but not including the character at the end index. The start position must be before the end position.
11	**select_to (index)** Selects all the text from the ANCHOR position up to but not including the character at the given index.
12	**xview (index)** This method is useful in linking the Entry widget to a horizontal scrollbar.
13	**xview_scroll (number, what)** Used to scroll the entry horizontally. The what argument must be either UNITS, to scroll by character widths, or PAGES, to scroll by chunks the size of the entry widget. The number is positive to scroll left to right, negative to scroll right to left.

Text widget

Text widgets provide advanced capabilities that allow you to edit a multiline text and format the way it has to be displayed, such as changing its color and font.

You can also use elegant structures like tabs and marks to locate specific sections of the text, and apply changes to those areas. Moreover, you can embed windows and images in the text because this widget was designed to handle both plain and formatted text.

Syntax

Here is the simple syntax to create this widget −

```
w = Text ( master, option, ... )
```

Parameters

- **master** − This represents the parent window.
- **options** − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Sr.No.	Option & Description
1	**bg** The default background color of the text widget.
2	**bd** The width of the border around the text widget. Default is 2 pixels.
3	**cursor** The cursor that will appear when the mouse is over the text widget.
4	**exportselection** Normally, text selected within a text widget is exported to be the selection in the window manager. Set exportselection=0 if you don't want that behavior.
5	**font**

	The default font for text inserted into the widget.
6	**fg** The color used for text (and bitmaps) within the widget. You can change the color for tagged regions; this option is just the default.
7	**height** The height of the widget in lines (not pixels!), measured according to the current font size.
8	**highlightbackground** The color of the focus highlight when the text widget does not have focus.
9	**highlightcolor** The color of the focus highlight when the text widget has the focus.
10	**highlightthickness** The thickness of the focus highlight. Default is 1. Set highlightthickness=0 to suppress display of the focus highlight.
11	**insertbackground** The color of the insertion cursor. Default is black.
12	**insertborderwidth** Size of the 3-D border around the insertion cursor. Default is 0.
13	**insertofftime** The number of milliseconds the insertion cursor is off during its blink cycle. Set this option to zero to suppress blinking. Default is 300.

14	**insertontime** The number of milliseconds the insertion cursor is on during its blink cycle. Default is 600.
15	**insertwidth** Width of the insertion cursor (its height is determined by the tallest item in its line). Default is 2 pixels.
16	**padx** The size of the internal padding added to the left and right of the text area. Default is one pixel.
17	**pady** The size of the internal padding added above and below the text area. Default is one pixel.
18	**relief** The 3-D appearance of the text widget. Default is relief=SUNKEN.
19	**selectbackground** The background color to use displaying selected text.
20	**selectborderwidth** The width of the border to use around selected text.
21	**spacing1** This option specifies how much extra vertical space is put above each line of text. If a line wraps, this space is added only before the first line it occupies on the display. Default is 0.

22	**spacing2** This option specifies how much extra vertical space to add between displayed lines of text when a logical line wraps. Default is 0.
23	**spacing3** This option specifies how much extra vertical space is added below each line of text. If a line wraps, this space is added only after the last line it occupies on the display. Default is 0.
24	**state** Normally, text widgets respond to keyboard and mouse events; set state=NORMAL to get this behavior. If you set state=DISABLED, the text widget will not respond, and you won't be able to modify its contents programmatically either.
25	**tabs** This option controls how tab characters position text.
26	**width** The width of the widget in characters (not pixels!), measured according to the current font size.
27	**wrap** This option controls the display of lines that are too wide. Set wrap=WORD and it will break the line after the last word that will fit. With the default behavior, wrap=CHAR, any line that gets too long will be broken at any character.
28	**xscrollcommand** To make the text widget horizontally scrollable, set this option to the set() method of the horizontal scrollbar.

29	**yscrollcommand** To make the text widget vertically scrollable, set this option to the set() method of the vertical scrollbar.

Methods

Text objects have these methods −

Sr.No.	Methods & Description
1	**delete(startindex [,endindex])** This method deletes a specific character or a range of text.
2	**get(startindex [,endindex])** This method returns a specific character or a range of text.
3	**index(index)** Returns the absolute value of an index based on the given index.
4	**insert(index [,string]...)** This method inserts strings at the specified index location.
5	**see(index)** This method returns true if the text located at the index position is visible.

Text widgets support three distinct helper structures: Marks, Tabs, and Indexes −

Marks are used to bookmark positions between two characters within a given text. We have the following methods available when handling marks −

Sr.No.	Methods & Description
1	**index(mark)** Returns the line and column location of a specific mark.
2	**mark_gravity(mark [,gravity])** Returns the gravity of the given mark. If the second argument is provided, the gravity is set for the given mark.
3	**mark_names()** Returns all marks from the Text widget.
4	**mark_set(mark, index)** Informs a new position to the given mark.
5	**mark_unset(mark)** Removes the given mark from the Text widget.

Tags are used to associate names to regions of text which makes easy the task of modifying the display settings of specific text areas. Tags are also used to bind event callbacks to specific ranges of text.

Following are the available methods for handling tabs −

Sr.No.	Methods & Description
1	**tag_add(tagname, startindex[,endindex] ...)** This method tags either the position defined by startindex, or a range delimited by the positions startindex and endindex.
2	**tag_config**

	You can use this method to configure the tag properties, which include, justify(center, left, or right), tabs(this property has the same functionality of the Text widget tabs's property), and underline(used to underline the tagged text).
3	**tag_delete(tagname)** This method is used to delete and remove a given tag.
4	**tag_remove(tagname [,startindex[.endindex]] ...)** After applying this method, the given tag is removed from the provided area without deleting the actual tag definition.

PYTHON ASSIGNMENT

53(A).Write a note on Entry Widget.

Answer: Entry widgets are the basic widgets of Tkinter used to get input, i.e. text strings, from the user of an application. This widget allows the user to enter a single line of text. If the user enters a string, which is longer than the available display space of the widget, the content will be scrolled.

This means that the string cannot be seen in its entirety. The arrow keys can be used to move to the invisible parts of the string. If you want to enter multiple lines of text, you have to use the text widget. An entry widget is also limited to single font.

The syntax of an entry widget is:

w = Entry(master, option, ...

OR

entry = tk.Entry(parent, options)

parent: The Parent window or frame in which the widget to display. Options: The various options provided by the entry widget.

"master" represents the parent window, where the entry widget should be placed. Like other widgets, it's possible to further influence the rendering of the widget by using options. The comma separated list of options can be empty.

54.

a) Write a Python script to concatenate following dictionaries to create a new one.
Sample Dictionary: dic1={1:10, 2:20} dic2={3:30, 4:40} dic3={5:50,6:60}
Expected Result: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

```
#!usr/bin/python3
dic1= {1:10, 2:20}
dic2= {3:30, 4:40}
dic3= {5:50, 6:60}
dic4 = {}
for d in (dic1, dic2, dic3): dic4.update(d) //The method update() add dictionaries
dic1,dic2,dic3 with key – values.
print(dic4)
```

Output:
{1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

b) Write a Python program to find the highest 3 values in a dictionary.

```
#!usr/bin/python3
from heapq import nlargest
dic4 = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
three_largest = nlargest(3, dic4, key=dic4.get)
 //3 --> specifies number of  times that nlargest will execute.
// dic4 --> dictionary
//key=dic4.get --> gets the key values.
print(three_largest)
```

Output:
[6, 5, 4]

55)
a) write a python program to check if a set is a subset of another set.
Code:-

```
print(“ check if a set is a subset of another set:\n”)
setx=set([“apple”,”mango])
sety=set([“mango”,”orange”])
setz=set([“mango”])
print(“x:”,setx)
print(“y:”,sety)
print(“z:”,setz,”\n”)
print(“ if x is subset of y”)
print(setx<=sety)
print(setx.isssubset(sety))
print(“ if y is subset of x”)
print(sety<=setx)
print(sety.isssubset(setx))
print(“\n if y is subset of z”)
print(sety<=setz)
print(sety.issubset(setz))
print(“if z is subset of y”)
print(setz<=sety)
print(setz.issubset(sety))
```

output:

```
  check if a set is a subset of another set
  x:{‘mango’,’apple’}
  y:{‘mango’,’orange’}
  z:{‘mango’}

 if x is subset of y
 false
 false
if y is subset of x
 false
 false

if  y is subset of z
false
false
if z is suset of y
 true
 true
```

b) write a python program to find maximum and minimum value in a set.

```
  #create a set
 set a =set([5,10,3,15,2,20])
 #find maximum value
 print (“maximum value is\n”)
 print(max(set a))
#find minimum value
print(“minimum value is\n”)
print(min(set a))
```

output:-

```
maximum value is
20
minimum value is
2
```

53. b) Write a note on Canvas Widget.

Tkinter Canvas widget is mainly used as a general-purpose widget which is used to draw anything on the application window in Tkinter.

- This widget is mainly used to draw graphics and plots, drawings, charts, and showing images.
- You can draw several complex layouts with the help of canvas, for example, polygon, rectangle, oval, text, arc bitmap, graphics, etc.
- Canvas widget is also used to create graphical editors.
- There are a number of options available to configure and control the Canvas Widget.

Tkinter Canvas Widget

The syntax of the canvas widget is given below:

W = Canvas(master, option=value)

In the above syntax, the master parameter denotes the parent window. You can use many options to change the layout of the canvas and these options are written as comma-separated key-values.

Tkinter Canvas Widget Options:

Following are the various options used with canvas widgets:

Option name	Description
Bg	This option is used to set the background color.
Bd	This option is mainly used to set the width of the border in pixels.
Cursor	Whether to use an arrow, dot, or circle on the canvas for the cursor, this option can be used.
Confine	This option is set to make the canvas non-scrollable outside the scroll region.
Height	This option is used for controlling the height of the canvas.
Width	This option is used to set the width of the widget.

Example:

```
from tkinter import *
#window named top
top =Tk( )
#set height and width of window
Top.geometry(“300*300”)
#creating a simple canvas with canvas widget
cv = Canvas(top,bg = “yellow”,height=”300”)
cv.pack( )
top.mainloop( )
```