→ write a python program to display contents of a file using with keyword.

```
file = input("enter filename")
try:
    with open(file,'r')
        print("File content",'s',f.read())
except FileNotFoundError as e:
    Print(" File not exist").
```

→ seek() & tell() functions in python (random access files)

→ seek():

This function is used to move file pointer to particular character

Syntax:- (offset)

file obj.seek(n) → n = no. of characters

file obj.seek(offset, whence)

whence

0 - Filebeg

1 - currentposition

2 - endposition.

→ Eg:- f.seek(5) → from 5th character beg

f.seek(6,1)

f.seek(-3,2).

→ tell():

this function is used to display current position of a file pointer.

Syntax:- file obj.tell()

UNIT-5

OOP, GUI & Database

→ Principles or Features of OOP:-

1. Object
2. class
3. Inheritance
4. Polymorphism
5. Data abstraction.
6. Data Encapsulation
7. Dynamic binding
8. Message passing.

Eg:- C++, Java, python, C# etc.

→ Polymorphism:-

Adhoc polymorphism or compile time: method Overloading
operator overloading

Runtime poly or classical : method overriding dynamic method dispatch, abstract

→ class:-

Syntax:-

```
class class_name:
    class_variable1 = value
    class_variable2 = value
    .....
    self, variable1 = value
    self, variable2 = value
    .....
    def method1(arg1, arg2):
        Statement1
        Statement2.
    def method2(self, arg1, arg2):
        Statement3
        Statement4
```

Differentiate Instance variables and class variables
Compare class methods with instance methods

1) object:-

Syntax:- obj = classname()

Program:- Student information.

```
Class A:
    self.no = None
    self.name = None
        address = None
    def getData(self):
        self.no = input(" enter rollno")
        self.name = input("enter name")
        self.address = input("enter address")
    def putData(self):
        print("number is", self.no)
        print("name is", self.name)
        print("address is", self.address)
```

```
s = A()
s.getData()
s.putData()
```

```
a = A()
v.getData()  s.getData()
v.putData()  s.putData()
```

o/p:- Enter rollno 120
Enter name: varshi
Enter address: Ggm
name is: 120
name is: varshi
address is: ongole.

⇒ Addition of two numbers:

```
class Add:
    def getData(self,x,y):
        self.a = x
        self.b = y
    def addition(self):
        self.c = self.a + self.b
    def display(self):
        print("Addition is:", self.c)
```

```
s = Add()
p = int(input("Enter value1:"))
q = int(input("Enter value2:"))
s.getData(p,q)
s.addition()
s.display()
```

o/p:- enter value1: 20
enter value2: 30
Addition is: 50

→ Write a python program to find out a factorial of a given number using classes and objects

Program:-

```
from math import *
class Fact:
    def getData(self):
        self.n=int(input("enter an integer"))
    def display(self):
        print("factorial is:",factorial(n))
r=Fact()
r.getData()
r.display()
```

o/p:- enter an integer 3
factorial is 6

→ Constructors:-

constructor is a type of method to initialize object members.

Properties of constructors:-

1) A constructor is created by using
   def __init__(self): //magic functions or dunder functions
2) no seperate calling for constructors, constructors ~~have~~ are automatically invoked at the time of object creation.

Types of constructors:-

1. Default or no argument constructors
2. parameterized constructors.

→ Default or no argument constructors:-

```
def __init__(self):
    statement1
    statement2
    .....
```

Ex program:-

```
class A:
    def __init__(self):
        print("hello...")
r=A()
```

o/p:- hello...

2) Parameterized Constructors:-

A constructor which contains arguments are called as Parameterized constructors.

Syntax: def __init__(self, arg1, arg2...):
    statement

Program:-

```
class A:
    def __init__(self,s):
        print("hello",s)
a=A("gec")
```

o/p:- hello gec.

⇒ Principles of Oop:-

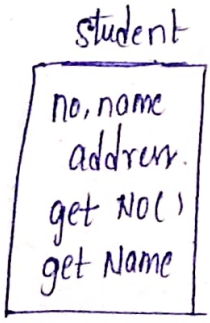
1: object:-

object is a runtime entity containing some properties. In Programming technology object is a collection of data members & functions.

→ Data members directly hold input data.
→ The operations performed on these data members are called as methods or member functions.

→ Eg's of object:-

Student, employee, mobile, television.



→ In python objects are created with the following syntax:

Obj = class_name()

Eg:- S = student()

2. class:-

class is a collection of objects with similar features

syntax:-

class class_name:

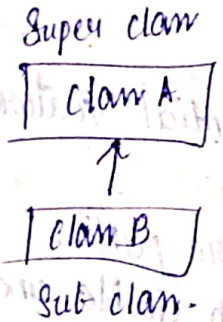
class variable { var1=value
. . .

def method1 (self, arg1, arg2..):
student1
student2
self.varv = value → Instance variable

class method { def method2 (arg1, arg2..):
student3
student4
. . .

3. Inheritance:- The process of deriving new class from already existing class.

→ New class is called as subclass or derived class or child class, Already existing class is called as Super class or base class or parent class.



→ In python inheritance is provided by using '( )' symbols

Eg:- class A(B):

→ Types of Inheritance:-

1) Single or simple Inheritance
2) multiple Inheritance
3) hierarchial Inheritance
4) multi-level Inheritance
5) multi path Inheritance
6) hybrid inheritance.

Note:- All above inheritances are supported in python

Polymorphism:- It is the concept of one name many forms.

Types of polymorphism:-

1. compile polymorphism
2. Run time polymorphism

→ By using the following mechanism python achive by using compile polymorphism.

1. method ~~overriding~~ overloading
2. operator overloading

Run time polymorphism:- By using the following mechanism python achieve by using runtime polymorphism

1. method overriding
2. Abstract class

→ Data Abstraction:-

The process of including essential features without affecting its background details.

→ By default all class members are public.

→ private members preceeds with double underscore(__)

ex:- class A:

__a = 40 #private

def __display(self): #private

#statment.

→ Data Encapsulation:- wrapping up of data and functions into single unit is called as data encapsulation

→ Dynamic Binding:- The call to the overridden method is resolved at runtime rather than compile time

ABC (Abstract Base class)

⇒ Message Passing:- objects are communicated with messages.

⇒ Default Constructor:-

Program:-
```
class Student():
    def __init__(self):
        self.no=input("enter roll no")
        self.name=input("enter name")
    def display(self):
        print("Roll number is:", self.no)
        print("Name is:", self.name)
s=Student()
s.display()
```

⇒ Parameterized Constructor:-

```
class Student():
    def __init__(self, no, name, address, mobile):
        self.no=no
        self.name=name
        self.address=address
        self.mobile=mobile
    def display(self):
        print("Roll number is:", self.no):
        print("Name is:", self.name)
        print("Address is:", self.address)
        print("mobile number is:", self.mobile)
no=input("enter roll no:")
name=input("enter name")
address=input("enter address")
mobile=input("enter mobile number")
s=Student(no, name, address, mobile)
s.display()
```

⇒ Polymorphism:-

1) Function overloading or method overloading:-

In python function overloading is achieved by default arguments of a function. Python does not allow more than one

function with same name like java.

Eg:- def add(a=5, b=7, c=8, d=9):
    print(a+b+c+d)

add() o/p:- 29
add(2) o/p:- 26
add(3,4) o/p:- 24
add(3,4,5) # 21

→ method Overloading:-

class Addition:
    def add(self, a=5, b=7, c=8, d=9):
        print(a+b+c+d)

a = Addition()
a.add() o/p:- 29
a.add(2) 26
a.add(3,4) 24
a.add(3,4,5) 21
a.add(1,2,3,4) 10.

→ Operator Overloading:-

+ magic function is __add__(self, val)
- " is __sub__(self, val)
* " is __mul__(self, val)
/ " " __floordiv__(self, val)
// " " __truediv__(self, val)
** " " __pow__(self, val)
o/p " " __lt__(self, val)
< " " __gt__(self, val)
> " " __le__(self, val)
<= " " __ge__(self, val)

→ Operator overloading is the concept of applying operators for all members of object.

→ object operators are directly applied to object.

Eg:-
```
class Op:
    def getData(self):
        self.a = int(input("Enter value1:"))
        self.b = int(input("Enter value2:"))
        self.c = int(input("Enter value3:"))
        self.d = int(input("Enter value4:"))
    def __mul__(self,r)
        r.a = self.a*r.a
        r.b = self.b*r.b
        r.c = self.c*r.c
        r.d = self.d*r.d
        return r
    def display(self):
        print(self.a, self.b, self.c, self.d)
s = Op()
s.getData(3,4,5,6)
v = Op()
v = getData()
x = s*v
x.display()
```

⇒ Inheritance:-

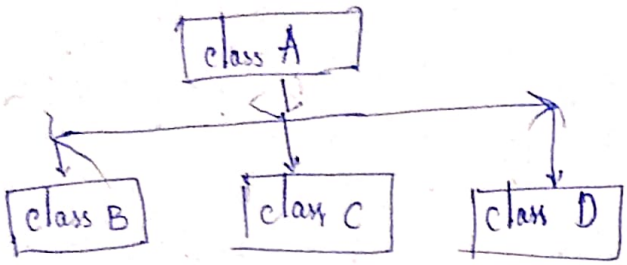
1) Single:- It contains one base class and one derived class.

base class

derived class.

Prog:-

Hierarchial Inheritance:-



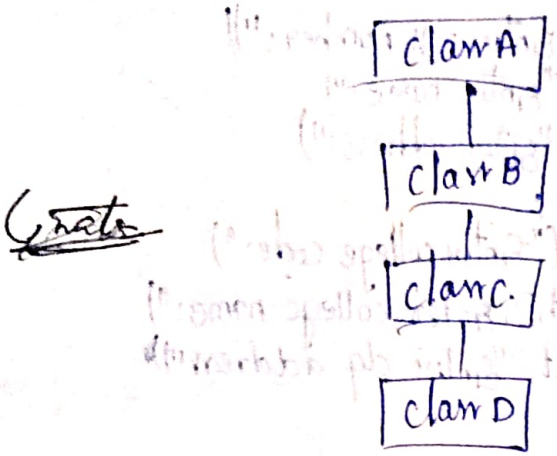
Def:- Hierarchial Inheritance contains only one base class and multiple / several derived classes.

Program:-

```
class A:
    def display1(self):
        print("A")

class B:
    def display2(self):
        print("B")

class C
    def display3(self):
        print("C")

class D
    def display4(self):
        print("D")

x = B()
S = C()
V = D()
x.display1()
x.display2()
s.display1()
s.display3()
v.display1()
v.display4()
```

```
class A:
    def getStudentDetails(self):
        self.no=int(input("Enter roll number:"))
        self.name = input("Enter name:")
        self.address = input("Enter address:")
    def getCollegeDetails(self):
        self.code = input("Enter college code:")
        self.cname = input("Enter college name:")
        self.address = input("Enter clg address:")
    def getMarks(self):
        self.m = [i for i in map(int, input("Enter 5 subjects
                                            marks:").split())]

class B(A):
    def display(self):
        print("Roll number is:", self.no)
        print("College code is:", self.code)
        print("Name is:", self.name)
        print("Address is:", self.address)
class C(A):
    def display(self):
        print("clg code is:", self.code)
        print("clg name is:", self.cname)
        print("clg address is:", self.caddress)
class D(A):
    def percentage(self):
        print("marks percentage is:", sum(self.m)/500*100)
r=B()
s=C()
v=D()
r.getStudentDetails()
s.getCollegeDetails()
v.getMarks()
r.display()
v.percentage()
s.display()
```

Multi-Level Inheritance:-



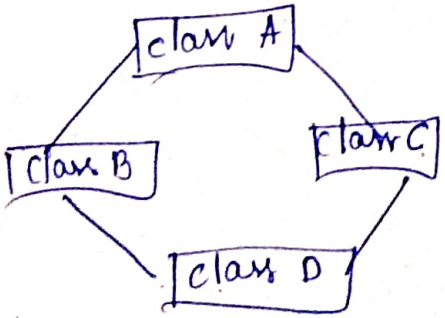
Note

Ex:- Program:-

```
class A:
    def display1(self):
        print("A")

class B(A):
    def display2(self):
        print("B")
class C(B):
    def display3(self):
        print("C")
class D(C):
    def display4(self):
        print("D")
x=D()
x.display1()
x.display2()
x.display3()
x.display4()
```

⇒ Combination of Hierarchial & Multi-level Inheritance

Multipath Inheritance:



Ex:-

```
class A:
    def display1(self):
        print("A")
class B(A):
    def display2(self):
        print("B")
class C(A):
    def display3(self):
        print("C")
class D(B,C):
    def display4(self):
        print("D")
x=D()
x.display1()
x.display2()
x.display3()
x.display4()
```

⇒ Overriding:- It is the concept of Run-time polymorphism. Base class method contains same method exist in derived class is called as Overriding method.

By using derived class object, user has to access only derived class method, ~~base~~ derived class method overrides the base class method.

Program:-

```
class Airtel:
    def getConnection(self):
        print("Airtel Connection")
class Idea(Airtel):
    def getConnection(self):
        print("Idea connection")
x=Idea()
x.getConnection()
```

→ By using super keyword invoke the overridden method into derived class.

eg:-
```
class Airtel:
    def getConnection(self):
        print("Airtel Connection")
class Idea(Airtel):
    def getConnection(self):
        super().getConnection()
        print("idea Connection")
r = Idea()
r.getConnection()
```

→ Data Hiding :-

Data hiding means Data Abstraction.

→ In python data hiding is achieved by using private variables or methods in a class.

→ for achieving private members outside the class, use mingling names and by using getter and setter blocks, access the private variables outside the class.

Eg: How to create private variables or methods.

```
def primeseries(n):
    c=0
    i=1
    k=1
    v=[]
    while k<=n:
        c=0
        for j in range(1,i+1):
            if i%j==0:
                c=c+1
        if c==2:
            k=k+1
            v.append(i)
        i=i+1
    return v
print(list(map(primeseries,[i for i in map(int,input().split(
))])))
```

```
k=int(input('enter a value'))
for i in range(1,k+1):
    print(i*(i+1))
```

2) notepad/shell

```
from SRGEC import *
v = getStudentInfo()
print("no is:", v[0])
print("name is:", v[1])
print("address is:", v[2])
print("caddress is:", v[3])
print("year of study:", v[4])
v=getStudentInfo("vijayawada")
print("no is:", v[0])
print("name is:", v[1])
print("address is:", v[2])
print("college address is:", v[3])
print("year of study:", v[4])
v = getStudentInfo("Guntur",3)
print("no is:", v[0])
print("name is:", v[1])
print("address is:", v[2])
print("college address is:", v[3])
print("year of study is:", v[4])
```

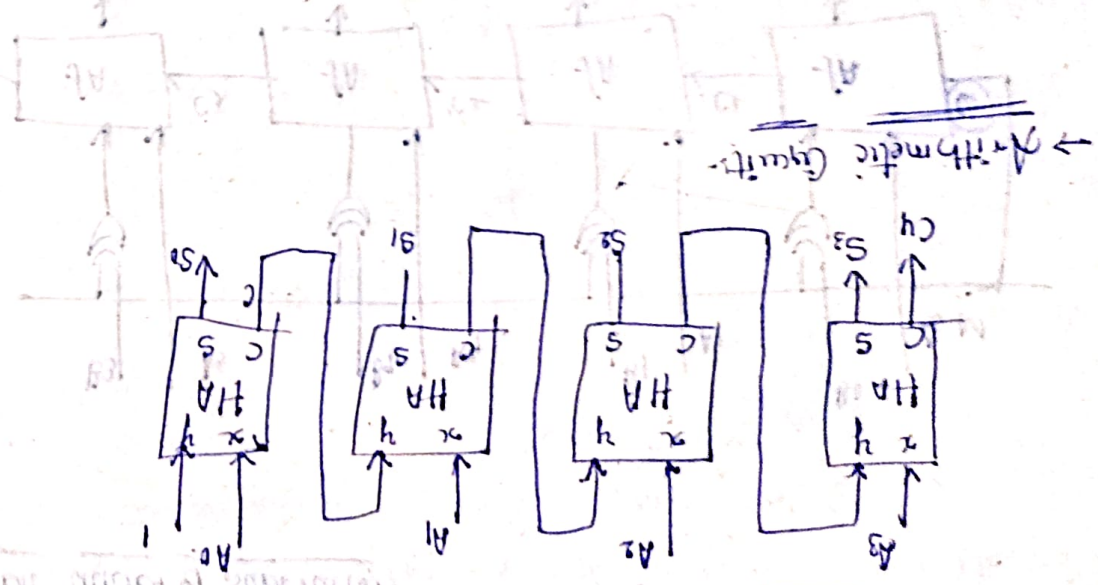
getStudent

```
def getStudentInfo(no, name, address, caddress = "GDLV",
                   ystudy = 2):
```

IIII

get stud

```
def getStudentInfo(caddress = "GDLV", ystudy = 2):
    no = int(input('enter no :"))
    name = input('enter name:")
    address = input('enter address:")
    return no, name, address, caddress, ystudy.
```



txt = "I am Varshitha from branch of Information Technology"

print(txt.find('a'))

O/p:- 30

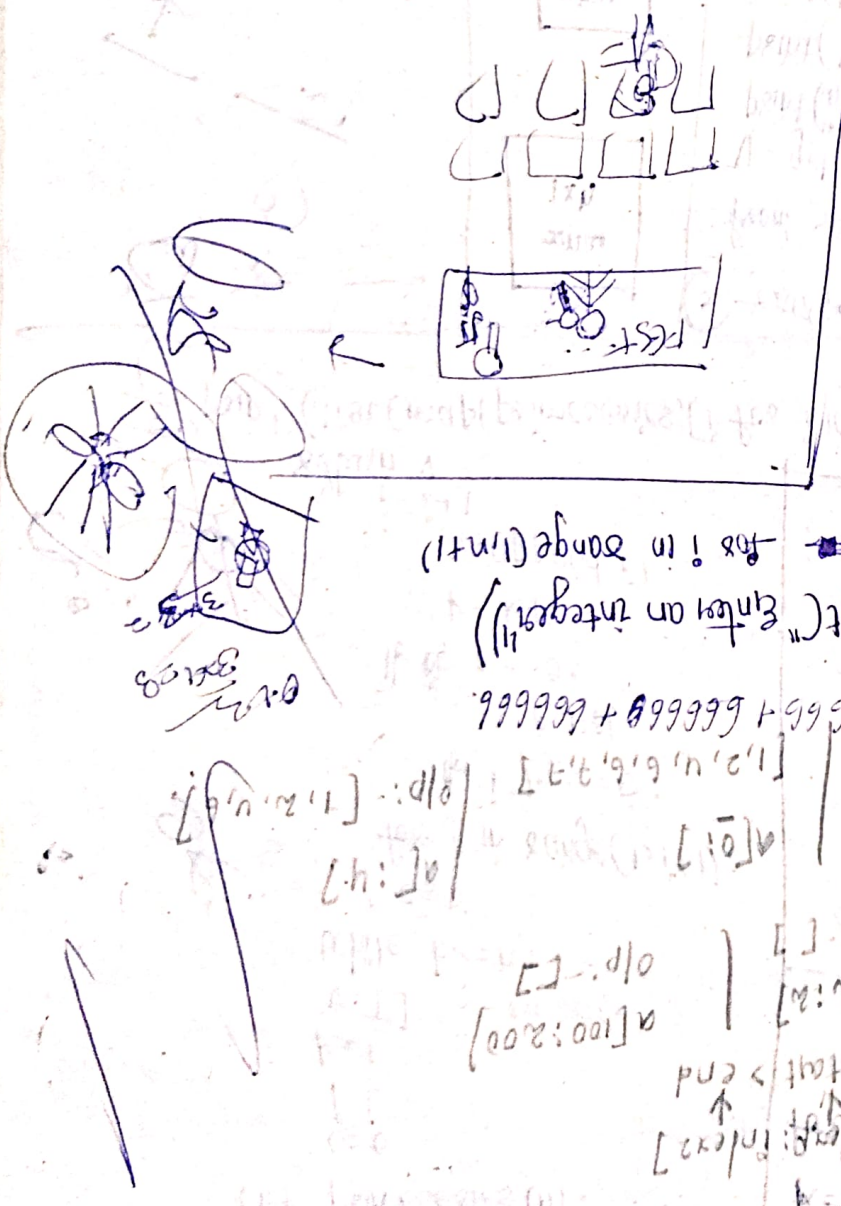
→ replace() | find() | rfind()

format() | splitlines()

txt = "Our class contains {:<8} students".

print(txt.format(200))

O/p:- Our class contains 200 students.



find
rfind - last index position

~~txt = "Ujwala is~~ membership operators in & not in

Tuple
↓
allows duplicate values

thistuple = ("apple")
tuple

thistuple = ("apple",)
↓
String

txt = "I am from Information Technology

thislist = list (("CSE", "IT", "ECE")).

thislist = ["ECE", "IT", "CSE"]
thislist.insert("ME", [1])
print(thislist).
o/p:- ["ECE", "ME", "IT", "CSE"]

"apple"
↓
tuple representation.

x = {"var", "ujw", "har"}
y = {"a", "c", "var"}
z = x.intersection_update(y)
print(z)
o/p:- "var"

IT, CSE, ME

Course = ("ECE", "IT", "CSE",

Tuple ✗ Expressions.
multiple ✗ Expressions are not allowed

dictionary → not allows duplicate values

x = {"varshitha", "ujwala", "Haripriya"}

y = {"pavvu", 3, "nandhu", 1}

z = x.symmetric_difference(y)
print(z).

o/p:-

8/04/23
12.

→ function → self defined block which returns a value
→ use : reusability.

→ function without name → anonymous function.

both arguments / variables refers to same address → aliasing

> help(list)

address → id(_)

without aliasing → use copy()

```
>a=[8,9]
>b=a.copy()
>b
 [8,9]
>a
 [8,9]
>b.append("u")
>b
 [8,9,'u']
>a
 [8,9]
```

with aliasing

```
>a=[8,9]
>b=a
>b
 [8,9]
>a
 [8,9]
>a.append("Hi")
>a
 [8,9,'Hi']
>b
 [8,9,'Hi'].
```

→ anonymous functions:-

↳ lambda Expressions :-

lambda arguments : expression / statements

↓

no need to write return type.

→ filter( ) → sequence of elements

filter(function or None, iterable).

def) check( ) → returns True or False.

" " → Blank / Zero → False

Otherwise → True.

map() filter() → used for all conditions

filter() → used for specific conditions

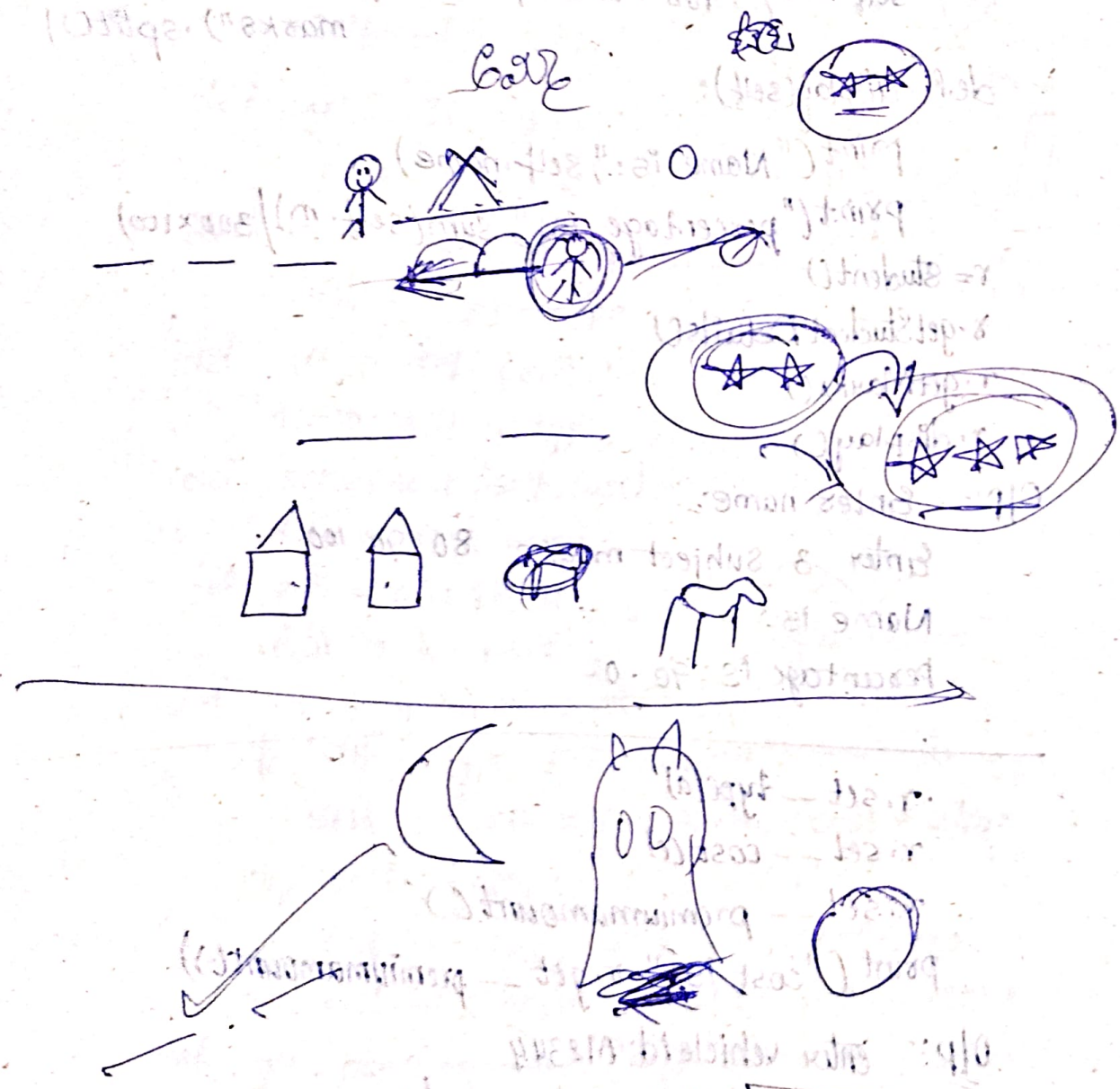
Pronic numbers → product of two consecutive integers

n = 3,

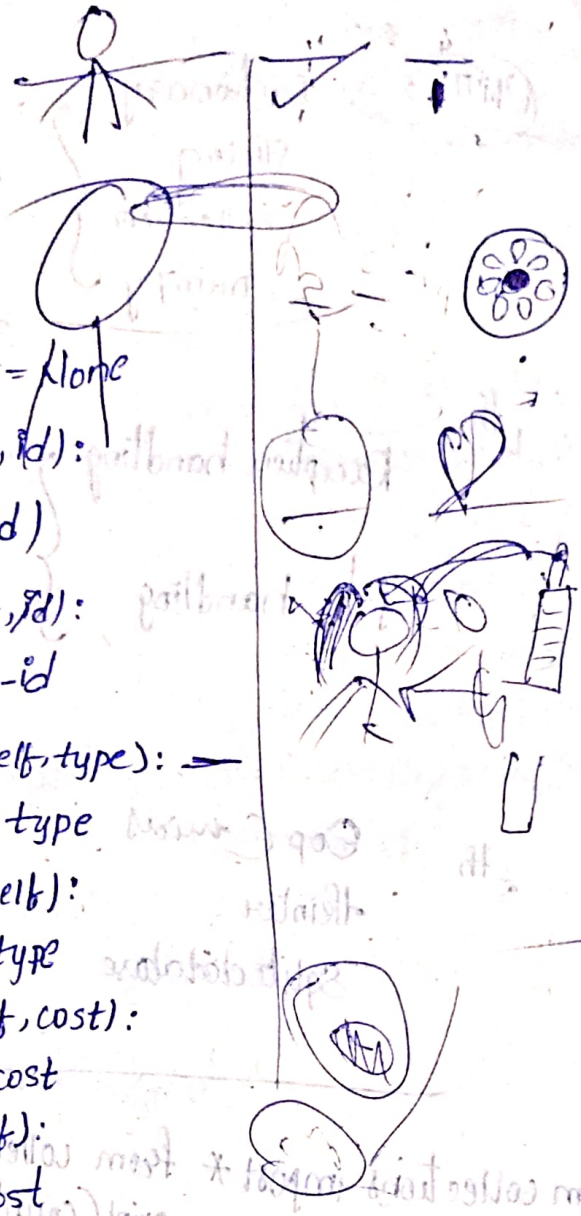
2 6 12

i * (i+1)

i



(20) # 20/6/23



```
class wecare:
    __id = None
    __type = None
    __cost = None
    __premiumamount = None
    def set__id(self, id):
        self.__id = (id)
    def get__id(self, id):
        return self.__id
    def set__type(self, type):
        self.__type = type
    def get__type(self):
        return self.__type
    def set__cost(self, cost):
        self.__cost = cost
    def get__cost(self):
        return self.__cost
    def set__premiumamount(self):
        if self.__type == "Two wheeler":
            self.__cost = self.__cost__cost * 2/100
        elif self.__type == "Four wheeler":
            self.__cost = self.__cost * 6/100
            self.__premiumamount = self.__cost
    def get_premiumamount(self):
        return self.__premiumamount
i = input("Enter vehicle id:")
a = input("Enter vehicle type:")
c = int(input("Enter cost:"))
x = wecare()
x.set__id(i)
```

(19)

Program:-

```
class Student:
    def getStudentDetails(self):
        self.name = input("Enter name:")
    def getMarks(self):
        self.m = [i for i in map(int, input("Enter 3 Subject
                                    marks").split()]
    def display(self):
        print("Name is:", self.name)
        print("percentage is:", sum(self.m)/300*100)
r = Student()
r.getStudentDetails()
r.getMarks()
r.display()
```

O/P:- Enter name:
Enter 3 Subject marks 80 90 100
Name is:
Percentage is: 90.0

```
r.set__type(a)
r.set__cost(c)
r.set__premiumamount()
print("cost is:", r.get__premiumamount())
```

O/P:- enter vehicleId: A1234
enter vehicle type: two wheely
enter cost: 200000
Cost is: 4000.0

```
import java.lang.*;
Class LinkedList() {
    Node head;
  //Node class
  class Node {
    int data;
    Node next;
    Node(int x) // param
    {
    data = x;
    next = null;
    }
  }
public Node insertBeginning(int data)
  {
   Node newNode = new Node(data);
   newNode.next = head;
     head = newNode;
     System.out.println("inserted" + data);
     return head;
  }
public void display()
  {
   System.out.println();
    Node node = head;
    while (node != null)
   {
    System.out.println(node.data + " ");
    node = node.next;
    }
    System.out.println();
   }
};
```

```
public class Main
{
public static void main(String
                    ...args)
{
LinkedList ll = new LinkedList();
   ll.insertBeginning(1);
   ll.insertBeginning(2);
   ll.insertBeginning(3);
   ll.insertBeginning(4);
   ll.insertBeginning(5);
   ll.insertBeginning(4);
   ll.display();
  }
}
```

UNIT-3 Dictionary
Slicing
collection
numpy
} 4 bits → obj
5m → Dep.
3 2

4th Exception handling
file handling
} + mcq's

5th Oop Concepts → oops-3
tkinter → tkinter-3
Sqlite database → database-3.
+ mcq's

```
from collections import *
m = [10,25,30]
d = defaultdict(int)
for i in m:
    d[i] += i
```

```
from collections import *
print(Counter('welcome'))
```

```
from collections import *
d1 = {}
d = OrderedDict()
d1[1] = 'csc'
d1[2] = 'it'
d[3] = 'eee'
d[4] = 'ece'
print(d)
print(d1)
```

Example programs :-

leap year or not.

Program:-

```
y=int(input("Enter year"))
if (y%4==0 and y%100!=1) or (y%400==0):
    print("leap year")
else:
    print("not leap year").
```

Swapping of given 2 values.

```
a=int(input("Enter first value"))
b=int(input("Enter second value"))
print("before swapping", a,b)
a,b=b,a
print("after swapping", a,b)
```

a=a+b b=a-b a=a-b / a=a^b / b=a^b / a=a^b

Even series.

```
n=int(input("enter an integer"))
for i in range(1,n):
    if i%2==0:
        print(i)
```

Prime or not

```
n=int(input("enter an integer"))
c=0
for i in range(1,n):
    if n%j==0:
        c=c+1
if c==2:
    print("prime")
else:
    print("not prime")
```

Prime series:-

```
n=int(input("enter an integer"))
c=0
for i in range(1,n+1):
    c=0
    for j in range(1,i+1):
        if i%j==0:
            c=c+1
    if c==2:
        print(i)
```

c!=2 → for non-prime

Pronic number

42 => 6x7

consecutive numbers product = number

20 => 4x5

6 => 3x2.

program:-

```
n=int(input("enter an integer"))
for i in range(1,n+1):
    if i*(i+1)==n:
        print("pronic number")
        break.
else:
    print("not pronic")
```

```
from collections import *
d= deque([34, 21, 1, 5, 6])
d.append(100)
d.appendleft(200)
d.pop()
print(d)
d.popleft()
print(d)
```

Vowels count

```
n=input("enter steing")
v='aeiouAEIOU'
c=0
for i in n:
    if i in v:
        c=c+1
print(c)
```

1:15 time

Reverse

```
n=int(input("enter an integer"))
s=0
while n>0:
    s=s*10+n%10
    n//=10
print(s)
if k==s:
    print("palindroome")
else
    print("not a palindroome")
```

#steong number.

```
n=int(input("enter int"))
k=n
s=0
from math import *
while n>0:
    s=s+factorial(n%10)
    n//=10
if s==k:
    print("steong")
else
    print("not steong")
```

#Armsteong number Imp

```
n=int(input("enter integer"))
k=n
m=len(str(n))
s=0
while n>0:
    s=s+((n%10)**m)
    n//=10
if s==k:
    print("Armsteong")
else:
    print("not an Armsteong")
```

Note: Ex: 153 count of digits = 3

$1^3+5^3+3^3 = 1+24+125 = 153$.

#perfect number

divisors sum = number.

```
n=int(input("enter integer"))
s=0
for i in range(1,n):
    if n%i==0:
        s=s+i
if n==s:
    print("perfect")
else:
    print("not perfect")
```

1
125
28
153

Pattern question :- V. Imp

Ex :- 256.

```
from sys import *
s=0
for i in range(1,len(argv)):
    k=int(argv[i])
    if k%2==0:
        s=s+k
```

even numbers Sum with command line Arguments.

---

write a python program that accepts an integer(n) & computes the value of n+nn+nnn. | Ex :- n=5 => 5+55+555

```
n=int(input("Enter number"))
s=0
for i in range(1,4):
    s=s+int(str(n)*i)
print(s)
```

555
55
+ 5
615

```
n=int(input('enter number'))
s=0
v=''
for i in range(1,4):
    s=s+int(str(n)*i)
    v=v+'+'+str(n)*i
print(v,'=',s)
    v[1:],
```

O/p :- 5+55+555 = 615

---

abc = 1,000,000.

abc

O/p :- (1,0,0) tuple format

a b = 10, 20

ab

O/p :- (10, 20)

a b c = 1000 2000 3000

Syntax Error :- invalid syntax

GST question :-

Types of user-defined functions

1. functions without parameters and without return type
2. functions without parameters & with return type
3. " with parameters & without return type
4. " with parameters & with return type

1)
```
def add()
    a=9
    b=8
    print(a+b)
add()
```

2)
```
def add():
    a=9
    b=9
    return a+b
print(add())
```

3)
```
def add(a,b):
    print(a+b)
add(9,8)
```

4)
```
def add(a,b):
    return a+b
print(add(9,8))
```

⇒ Types of Arguments. (5)

1. positional Arguments
2. default arguments
3. keyword arguments
4. arbitrary positional
5. arbitrary keyword

1) positional arguments:-
```
def add(a,b):
    print(a+b)
add(4,5)
```

2) default arguments:-
```
def add(a=1, b=2, c=3, d=6):
    print(a+b+c+d)
add(4,5,7,9)
add(8,9,10)
add(3,4)
add(6)
```

3) keyword arguments:-
```
def add(a,b,c,d):
    print(a+b+c+d)
add(b=3,c=1,a=10,d=9)
```

4) arbitrary positional arguments:-
pointers concept
```
def add(*p):
    print(p)
add(9,8,7,6,4,2)
```
default format is tuple.

5) arbitrary keyword argument:-
```
def add(**p):
    print(p)
add(a=9, b=8, c=11)
```
o/p :- dictionary format

Count no of instances of a class in python

Code:

```
Class A:
    c = 0
    def __init__(self):
        A.c += 1
A1 = A()
A2 = A()
A3 = A()
print(A.c)
```

Parameter passing techniques.

call by value: immutable → no affect of values

call by reference: mutable.

Polymorphism:- one name many forms.

1) compile time → method overloading
   operator overloading
2) runtime → method overriding
   Abstract Base class.

1) method Overloading:- same function → default functional arguments.

Ex:-
```
class Addition:
    def add(self,a=4,b=4,c=6):
        print("Addition is:",a+b+c)
x = Addition()
x.add()          # 14
x.add(10)        # 20
x.add(6,12)      # 24
```

2) Operator overloading:-

Ex:-
```
class Addition:
    def getdata(self):
        self.a=int(input("enter a value"))
        self.b=int(input("enter b value"))
        self.c=int(input("enter c value"))
        self.d=int(input("enter d value"))
    def __add__(self,x):
        x.a = self.a+x.a
        x.b = self.b+x.b
        x.c = self.c+x.c
        x.d = self.d+x.d
        return x
    def display(self):
        print(self.a,self.b,self.c,self.d)
r = Addition()
r.getdata()
v = Addition()
v.getdata()
x = Addition()
z = r+v
z.display()
```

value:- a=1, b=2, c=3, d=4.
a=2, b=3, c=4, d=5

O/P:- 3 5 7 9

method Overriding:-

Ex:-
```
class Airtel:
    def getConnection(self):
        print("Airtel Connect established")
class Jio(Airtel):
    def getConnection(self):
        print("Jio")
a=Jio()
a.getConnection()
```

o/p: Jio

with super() function.

```
super().getConnection()
```

o/p:- Airtel
Jio

~~class Ai~~

tkinter()

```
Tk()
pack()
mainloop()
```

1) Label:-
   obj = Label(master/container, options...)

2) Button:-
   obj = button(master, options)

3) Entry
   obj = Entry (master, options).

4) Combobox:- dropdownlist of options & displays them one at a time

Syntax:-
   obj = Combobox(master, options...)

5) Checkbutton:- select single/multiple list of items.
   obj = Checkbutton (master, options...)

6) Radiobutton:- one of many selection
   obj = Radiobutton(master, options...)

7) Spinbox:- alternative to Entry widget
   obj = Spinbox(master, options...)

8) text:- obj = text(master, options...)

9) menu:- obj = Menu(master, options...)

UNIT-I

Operators ✓
Control flow Statements ✓
Command line Arguments ✓
Data types
Literals

UNIT-2

~~Fun~~ Arguments ✓
parameters topic ✓
parameter passing techniques
Iterators
keywords

UNIT-3

numpy module ✓
list, tuple, dictionary, set } methods ✓

UNIT-4

Exceptions → user defined
predefined
lab programs
File handling → lab programs

UNIT-V

Inheritance, Polymorphism → examples
tkinter, textbox, button, widget, Combobox, Spinbox → lab program 22
Sqlite database - Student details