```python
#ex 1
#vithula , 2210417 , IT-C
#implementing a circle class
class Circle():
    __slots__=['radius']

    def __init__(self,radius):
        self.radius=radius

    def circumference(self):
        return 2*3.14*self.radius

    def area(self):
        return 3.14*self.radius*self.radius

    def display(self):
        return 'radius,circuference,area is:',self.radius,self.circumference() ,self.area()

if __name__=="__main__":
    c=Circle(5)
    print(c.display())
```

```python
#ex 1
#vithula , 2210417 , IT-C
#implementing a student class
class Student():
    __slots__=['studentID','name','branch','age','phoneno','email']

    def __init__(self,studentID,name,branch,age,phoneno,email):
        self.studentID=studentID
        self.name=name
        self.branch=branch
        self.age=age
        self.phoneno=phoneno
        self.email=email

    def display(self):
        #returns a tuple containing all the details
        return self.studentID, self.name, self.branch,self.age,self.phoneno,self.email

    def updateno(self,value):
        self.phoneno=value

    def updateemail(self,value):
        self.email=value

if __name__=="__main__":
    s=Student(2210417,'vithula','IT',18,9940297114,'vithulasrinivasan@gmail.com')
    print(s.display())
    s.updateemail('vithula2210417@ssn.edu.in')
    print(s.display())
    s.updateno(9380959913)
    print(s.display())
```

```python
#ex 2 qn1
#vithula , 2210417 , IT-C
#implementing a point class
class Point:
    def __init__(self, xcod = 0, ycod = 0):
        self.x = xcod
        self.y = ycod

    def getPoint(self):
        self.x = int(input("Enter x coordinate: "))
        self.y = int(input("Enter y coordinate: "))

    def showPoint(self):
        result = f"({self.x}, {self.y})"
        return result

if __name__ == "__main__":
    #Creating a Point Object
    P = Point()

    #Getting x and y coordinates
    P.getPoint()

    #Displaying x and y coordinates
    print(P.showPoint())
```

```python
'''NAME: VITHULA S
REGISTER NO: 3122225002158
CLASS: IT-C
EX NO: 4'''

from binarytree import bt
class NewsArticle:
    def __init__(self, date, category, content):
        self.date = date
        self.category = category
        self.content = content

# Create BinaryTree instances for Political and Sports categories
political_tree = bt()
sports_tree = bt()

# Method to add a news article to the appropriate category tree
def add_news_article(date, category, content):
    article = NewsArticle(date, category, content)
    if category == "Political":
        political_tree.insert(article)
    elif category == "Sports":
        sports_tree.insert(article)

# Method to display news on a specific date and category
def display_news_on_date(category, date):
    tree = political_tree if category == "Political" else sports_tree
    articles = tree.inorder(tree.root)

    for article in articles:
        if article.date == date:
            print(f"Category: {article.category}, Date: {article.date}, Content: {article.content}")

# Method to retrieve news between specific dates for a given category
def retrieve_news_between_dates(category, start_date, end_date):
    tree = political_tree if category == "Political" else sports_tree
    articles = tree.inorder(tree.root)

    result = []
    for article in articles:
        if start_date <= article.date <= end_date:
            result.append(article)

    return result


# DRIVER CODE
if __name__=='__main__':
    # Add sample news articles
    add_news_article("2023-9-01", "Political", "Political news article 1")
    add_news_article("2023-10-02", "Political", "Political news article 2")
    add_news_article("2023-11-03", "Sports", "Sports news article 1")
    add_news_article("2023-10-04", "Sports", "Sports news article 2")

    # Display news on a specific date and category
    print("Displaying Political news on 2023-10-02:")
    display_news_on_date("Political", "2023-10-02")

    # Retrieve news between specific dates for a given category
    print("Retrieving Sports news between 2023-10-03 and 2023-10-05:")
    sports_articles = retrieve_news_between_dates("Sports", "2023-10-03", "2023-10-05")
    for article in sports_articles:
        print(f"Category: {article.category}, Date: {article.date}, Content: {article.content}")
```

```python
'''NAME: VITHULA S
REGISTER NO: 3122225002158
CLASS: IT-C
EX NO: 4'''

#to implement the binarytree module

from btnode import btnode

class bt():
    def __init__(self):
        self.root = None

    #Insertion
    def create_tree(self, values):
        if values:
            for value in values:
                self.insert(value)

    def _insert_recursive(self, node, value):
        if not node:
            return btnode(value)

        if not node.left:
            node.left = self._insert_recursive(node.left, value)
            return node

        if not node.right:
            node.right = self._insert_recursive(node.right, value)
            return node

        # If the current node has both children, we continue searching in the left subtree
        self._insert_recursive(node.left, value)
        return node

    def insert(self, value):
        if not self.root:
            self.root = btnode(value)
        else:
            self._insert_recursive(self.root, value)

    def inorder(self, current):
        if not current:
            return []
        return self.inorder(current.left) + [current.value] + self.inorder(current.right)


if __name__=="__main__":
    t1=bt()
    t1.create_tree([1,2,3,4,5,6,7])
    print(t1.inorder(t1.root))
```

```python
'''NAME: VITHULA S
REGISTER NO: 3122225002158
CLASS: IT-C
EX NO: 4'''

#to implement a binary search tree

class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    # Insertion
    def insert(self, value):
        if not self.root:
            self.root = Node(value)
        else:
            self._insert_recursive(self.root, value)

    def _insert_recursive(self, node, value):
        if value < node.value:
            if node.left is None:
                node.left = Node(value)
            else:
                self._insert_recursive(node.left, value)
        else: # value >= node.value
            if node.right is None:
                node.right = Node(value)
            else:
                self._insert_recursive(node.right, value)

    # In-order traversal
    def inorder_traversal(self):
        def _inorder(node):
            if not node:
                return []
            return _inorder(node.left) + [node.value] + _inorder(node.right)

        return _inorder(self.root)

    # Pre-order traversal
    def preorder_traversal(self):
        def _preorder(node):
            if not node:
                return []
            return [node.value] + _preorder(node.left) + _preorder(node.right)

        return _preorder(self.root)

    # Post-order traversal
    def postorder_traversal(self):
        def _postorder(node):
            if not node:
                return []
            return _postorder(node.left) + _postorder(node.right) + [node.value]

        return _postorder(self.root)

# DRIVER CODE
if __name__=='__main__':
    bst = BinarySearchTree()
    bst.insert(5)
    bst.insert(3)
    bst.insert(8)
    bst.insert(1)
    bst.insert(4)
    bst.insert(7)
    bst.insert(9)

    print("In-order:", bst.inorder_traversal())  # [1, 3, 4, 5, 7, 8, 9]
    print("Pre-order:", bst.preorder_traversal())  # [5, 3, 1, 4, 8, 7, 9]
    print("Post-order:", bst.postorder_traversal())  # [1, 4, 3, 7, 9, 8, 5]
```

```python
#ex 2 qn2
#vithula , 2210417 , IT-C
#Inherit Circle from point

class Circle:
    def __init__(self):
        self.radius = 0.0

    def getCircle(self):
        print("Enter the coordinates of Centre")
        x1 = int(input("Enter x coordinate of center: "))
        y1 = int(input("Enter y coordinate of center: "))

        print("Enter the coordinates of any point on circumference")
        x2 = int(input("Enter x coordinate of point: "))
        y2 = int(input("Enter y coordinate of point: "))

        self.calcRad(x1,y1,x2,y2)
        self.calcArea()

    def calcRad(self, x1, y1, x2, y2):
        self.radius = (((x2 - x1) ** 2) + ((y2 - y1) ** 2)) ** 0.5

    def calcArea(self):
        area = 3.14 * (self.radius ** 2)
        print(f"The radius of the circle is {self.radius}")
        print(f"Area of circle is {area}")

class Cone(Circle):
    def __init__(self):
        super().__init__()
        self.apex = None

    def getCone(self):
        self.getCircle()
        print("Enter the coordinates of apex")
        x = int(input("Enter the x coordinates of apex: "))
        y = int(input("Enter the y coordinate of apex: "))
        self.apex = (x, y)

    def calcVolume(self):
        height = ((self.radius ** 2) + (self.apex[1]**2)) ** 0.5
        volume = (1/3) * 3.14 * (self.radius**2) * (height)
        print(f"Apex Coordinates: {self.apex}")
        print(f"Height of the cone: {height}")
        print(f"Volume of the cone: {volume}")


if __name__ == "__main__":
    print("CIRCLE!!!")
    #Creating a circle object
    circle = Circle()

    #Printing details
    circle.getCircle()

    print("CONE!!!")
    #Creating a Cone object
    cone = Cone()

    #Printing details
    cone.getCone()
    cone.calcVolume()
```

```python
from Date import Date
from Convert import Convert
from Current import Current
from Difference import Difference
from Validity import Validity

class StudentCompetition:
    def __init__(self,name,dob,registration_date):
        self.name = name
        self.dob = dob
        self.registration_date = registration_date

    def is_registration_valid(self):
        if not Validity.is_valid_date(self.dob):
            return "Invalid date of birth"

        age_in_years = Difference.difference_with_current(self.dob)
        if age_in_years is None or age_in_years >17 *365:
            return "Invalid age"

        registration_age = Difference.difference_with_current(self.registration_date)
        if registration_age is None or registration_age < 0 or registration_age > 180:
            return "Registration expired"

        return "Registration is valid"

if __name__ == "__main__":

    dob = input("student's date of birth:")
    registration_date = input("enter registration date:")
    student = StudentCompetition(input("enter student's name:"), dob, registration_date)

    current = Current()
    print("Current Time:", current.current_time())

    print("Student Details:")
    print("Name:", student.name)
    print("Date of Birth:", student.dob)
    print("Registration Date:", student.registration_date)

    result = student.is_registration_valid()
    print("Registration Status:", result)
```

```python
from datetime import datetime

class Validity:
    def is_valid_time(time_str):
        try:
            datetime.strptime(time_str,'%H:%M:%S')
            return True
        except ValueError:
            return False

    def is_valid_date(date_str):
        try:
            datetime.strptime(date_str,'%d.%m.%Y')
            return True
        except ValueError:
            return False
```

```python
'''
VITHULA S
IT-C
EX-3
'''


class Convert:
    def Convert_hrs_days(hours):
        return hours/24

    def convert_days_hours(days):
        return days*24

    def convert_man_hrs_days(hours):
        return hours/8
```

```python
'''
VITHULA S
IT-C
EX-3

Date Module
â ¢ Create and display dates '''

class Date:
    def __init__(self,day,month,year):
        self.day = day
        self.month = month
        self.year = year


    def __str__(self):
        y = str(self.year)
        if int(self.month) == 2 and int(self.day)>28:
            return("February has only 28 days")

        if int(self.month) == 8:
            return(f"{self.day}:{self.month}:{self.year}")

        if int(self.month) %2 == 0 and int(self.day)> 30:
            return("Invalid date")

        if int(self.day)<=31 and int(self.month)<= 12 and len(str(self.year))== 4 :
            return(f"{self.day}:{self.month}:{self.year}")
        else:
            return("invalid date")


if __name__ == "__main__":
    date = Date(int(input("enter date:")),int(input("enter month:")),int(input("enter year:")))
    print(date)
```

```python
'''NAME: VITHULA S
REGISTER NO: 3122225002158
CLASS: IT-C
EX NO: 4'''



#to implement the btnode module



class btnode():

    #initialising the node with the given value
    def __init__(self,value):
        self.left=self.right=None
        self.value=value

    def display(self):
        return(self.value)

if __name__=='__main__':
    b1=btnode(6)
    print(b1.display())
```

```python
'''
VITHULA S
ex 5 pdp lab qn 1
Define a vector inheriting from standard list. Let the vector is restricted to having only integer
number in the list. Overload appropriate function and operator so that when any other type of element
is inserted the program raises an error.
'''


class Vector(list):
    def __init__(self, sequence=None):
        super().__init__()
        if sequence is not None:
            try:
                for item in sequence:
                    self.append(item)
            except TypeError:
                self.append(sequence)

    def append(self, item):
        if isinstance(item, int):
            super().append(item)
        else:
            raise TypeError('Only integers can be appended')

    def __add__(self, other):
        return sorted(Vector(set(self) | set(other)))

    def __sub__(self, other):
        union = set(self) | set(other)
        intersection = set(self) & set(other)
        return sorted(Vector(union - intersection))

    def get_ratios(self, other):
        if isinstance(other, Vector):
            if len(self) == len(other):
                result = Vector()
                for i in range(len(self)):
                    try:
                        result.append(self[i] // other[i])
                    except ZeroDivisionError:
                        result.append(0)
                return result
            else:
                raise IndexError("Vectors have different dimensions")
        else:
            raise TypeError(f"{other} is not a Vector object")


if __name__ == "__main__":
    v1 = Vector([1, 2, 3])
    v2 = Vector([3, 4, 5, 7])
    identity = Vector([1] * 4)

    v1.append(4)
    print("v1:", v1)
    print("v2:", v2)
    print("v1 + v2:", v1 + v2)
    print("v1 - v2:", v1 - v2)
    print("v1 / identity ratios:", v1.get_ratios(identity))

    try:
        v1.get_ratios(Vector([1, 0, 3, 2]))
    except ZeroDivisionError:
        print("Division by zero!")

    try:
        v1.get_ratios(Vector([1, 0, 3]))
    except IndexError as e:
        print(e)
```

```python
#ex 2 qn 3
#vithula , 2210417 , IT-C
#Implementing a Regular Polygon

from qn2 import Point

class Regular_Polygon(Point):
    def __init__(self):
        super().__init__()
        self.points_array = []
        self.num_sides = 0

    def getPolygon(self):
        self.num_sides = int(input("Enter the number of sides: "))
        print("Enter the coordinates of polygon's vertices")
        for i in range(self.num_sides):
            point = Point()
            point.getPoint()
            self.points_array.append(point)

    def showPolygonDetails(self):
        print(f"Number of sides: {self.num_sides}")
        print("Coordinates of the polygon's vertices:")
        for i, point in enumerate(self.points_array, start=1):
            print(f"Vertex {i}: ({point.x}, {point.y})")

class Square(Regular_Polygon):
    def __init__(self):
        super().__init__()  # Call the constructor of the base class (Regular_Polygon)
        self.side_length = 0

    def getSquareDetails(self):
        self.getPolygon()  # Reuse the getDetails method from the Regular_Polygon class
        self.side_length = float(input("Enter the side length of the square: "))

    def calcArea(self):
        area = self.side_length ** 2
        return area

    def calcPerimeter(self):
        perimeter = self.side_length * self.num_sides
        return perimeter


if __name__ == "__main__":
    print("Polygon!!!")
    # Create an instance of the Regular_Polygon class
    polygon = Regular_Polygon()

    # Get details of the regular polygon
    polygon.getPolygon()

    # Display the polygon details
    polygon.showPolygonDetails()

    print("_________________________________\n")
    print("SQUARE!!!")
    # Create an instance of the Square class
    square = Square()

    # Get details of the square
    square.getSquareDetails()

    # Display the square details
    square.showPolygonDetails()
    area = square.calcArea()
    perimeter = square.calcPerimeter()
    print(f"Area of the square: {area:.2f}")
    print(f"Perimeter of the square: {perimeter:.2f}")
```

```python
from datetime import datetime , timedelta

class Difference:
    def difference_with_current(date_str):
        try:
            current_date = datetime.now().date()
            input_date = datetime.strptime(date_str, '%d.%m.%Y').date()

            if (current_date.month,current_date.day) < (input_date.month , input_date.day):
                age = current_date.year - input_date.year - 1
            else:
                age = current_date.year - input_date.year

            return age
        except ValueError:
            return None

    def difference(date_str1,date_str2):
        try:
            date1 = datetime.strptime(date_str1 , '%d.%m.%Y').date()
            date2 = datetime.strptime(date_str2 , '%d.%m.%Y').date()
            return (date1 - date2).days
        except ValueError:
            return None

    def days_after(days):
        try:
            current_date = datetime.now().date()
            future_date = current_date + timedelta(days = days)
            return future_date.strftime('%d.%m.%Y')
        except ValueError:
            return None

    def days_before(days):
        try:
            current_date = datetime.now().date()
            past_date = current_date - timedelta(days = days)
            return past_date.strftime('%d.%m.%Y')
        except ValueError:
            return None

    def month_after(months):
        try:
            current_date = datetime.now().date()
            future_date = current_date + timedelta(days = months * 30)
            return future_date.strftime('%d.%m.%Y')
        except ValueError:
            return None

    def month_before(months):
        try:
            current_date = datetime.now().date()
            past_date = current_date - timedelta(days = months * 30)
            return past_date.strftime('%d.%m.%Y')
        except ValueError:
            return None
```

```python
'''
Author : VITHULA S
Reg no : 3122225002158
Sec    : IT-C
The ShowOff Function displays name and marks of the person,
only if the marks are above 60. Else print name and subject
 name and subject teacher name.'''

class ShowOff:
    def __init__(self,**kwargs):
        self.name = kwargs['name']
        self.marks = kwargs['mark']
        self.subject = kwargs['subject']
        self.teacher = kwargs['teacher']

    def __str__(self):
        if self.marks>60:
            return f"PASSED\nName:{self.name}\nMarks:{self.marks}"
        else:
            return f"FAILED\nName:{self.name}\nSubject:{self.subject}\nTeacher:{self.teacher}"

if __name__=='__main__':
    student1 =ShowOff(name="vithula",mark=99,subject="maths",teacher="Kalaivani Ma'am")
    print(student1)
    student2 =ShowOff(name="neeharika",mark=23,subject="dsa",teacher="Vasuki Ma'am")
    print(student2)
```

```
'''
VITHULA S
IT-C
EX-3

Current module which may helpful in getting
â ¢ Current time
â ¢ Current date default return format dd.mm.yyyy
â ¢ Current date in different formats mm.dd.yyyy
â ¢ Current date in string format '''

from datetime import datetime


class Current:
    def __init__(self):
        self.now = datetime.now()

    def current_time(self):
        return self.now.strftime('%H:%M:%S')

    def current_date(self):
        return self.now.strftime('%d.%m.%y')

    def current_date_m(self):
        return self.now.strftime('%m.%d.%y')

    def current_date_str(self):
        return self.now.strftime('%A,%B %d, %Y')

if __name__ == "__main__":
    current = Current()
    print(current.current_time())
    print(current.current_date())
    print(current.current_date())
    print(current.current_date_m())
    print(current.current_date_str())
```

```python
'''
VITHULA S
ex 5 pdp lab qn 2
Create a class â  Employeeâ  . The constructor must assign the first name and last name of an employee.
Define a function called from_string() which gets a single string from the user, splits and assigns to the
first and last name. For example, if the string is â  Seetha Ramanâ  , the function should assign first name
as Seetha and second name as Raman and the function returns the object. Can you design from_string()
as a class or instance function? Justify your response'''


class Employee:
    """Stores the details of an employee"""

    def __init__(self, fname=None, lname=None):
        """Constructor of the Employee class"""
        self.fname = fname
        self.lname = lname

    @classmethod
    def from_string(cls, name):
        """Create an Employee object from a full name string."""
        fname, lname = name.split()
        return cls(fname, lname)

    def __str__(self):
        """Return a string representation"""
        return "\nfirst name: {}\nlast name: {}\n".format(self.fname, self.lname)

if __name__ == "__main__":
    # Create Employee objects
    emp1 = Employee.from_string(input("\nEnter employee 1 full name: "))
    emp2 = Employee()
    emp2.from_string(input("\nEnter employee 2 full name: "))
    fname = input("\nEnter employee 3 first name: ")
    lname = input("Enter employee 3 last name: ")
    emp3 = Employee(fname, lname)

    # Display the employee details
    print("\nEmployee details:")
    print(emp1)
    print(emp2)
    print(emp3)
```

```python
'''
Author : VITHULA S
Reg : 3122225002158
sec  : IT-C
a. Average(*args)
 Write a function to find the average of 'n' numbers passed to the function '''


class Average:
    def __init__(self,*args):
        self.number = args

    def average(self):
        return sum(self.number)/len(self.number)
if __name__=='__main__':
    average=Average(1,2,3,4)
    print(average.average())
```

```python
'''
VITHULA S
ex 5 pdp lab qn 3
Implement the classes Movie() and MovieList() as described in the above figure. Override the
appropriate functions so that the MovieList is generated based on the genre assigned in the instance
variable when first object is created. For example, if the genre is defined as thriller, the list accepts only
thriller movies.
When two lists are given as input, the list with more number of movies are returned
'''

class Movie:
    def __init__(self, name, genre):
        self.name = name
        self.genre = genre

    def __str__(self):
        return "Movie name: {}\nGenre: {}".format(self.name, self.genre)

class MovieList(list):
    def __init__(self, genre):
        super().__init__()
        self.genre = genre

    def append(self, movie):
        if not isinstance(movie, Movie) or movie.genre != self.genre:
            raise TypeError("Only movies of the same genre can be added to the list")
        super().append(movie)

    def __str__(self):
        return "\nMovie list Genre: {}\nList: {}".format(self[0].genre, super().__str__())

    def __gt__(self, other):
        if not isinstance(other, MovieList):
            raise TypeError("Only MovieList objects can be compared")
        return self if len(self) >= len(other) else other

if __name__ == "__main__":

    crime_mov_1 = Movie("The Dark Knight", "Crime Thriller")
    crime_mov_2 = Movie("Heat", "Crime Thriller")
    adven_mov_1 = Movie("Indiana Jones and the Last Crusade", "Adventure")
    adven_mov_2 = Movie("Jurassic Park", "Adventure")

    crime_mov_lst = MovieList("Crime Thriller")
    crime_mov_lst.append(crime_mov_1)
    crime_mov_lst.append(crime_mov_2)

    adven_mov_lst = MovieList("Adventure")
    adven_mov_lst.append(adven_mov_1)
    adven_mov_lst.append(adven_mov_2)

    print("\nMovie List:")
    print(crime_mov_lst)
    print(adven_mov_lst)

    print("Greater > :")
    print(crime_mov_lst > adven_mov_lst)
```