IMPLEMENTATION – EX 2 :

```
class LibraryItems:
    def __init__(self,dispitems):
        self.dispitems = dispitems

    def display(self):
        print()
        for i in self.dispitems:
            if isinstance(i, Author):
                i.display()
                continue
            print(i)

class Book(LibraryItems):
    def __init__(self,ISBN,DDS,subject,title,author):
        self.ISBN = ISBN
        self.DDS = DDS
        self.subject = subject
        self.title = title
        self.author = author
        self.dispitems = [ISBN,DDS,subject,title,author]
        super().__init__(self.dispitems)

    def display(self):
        print("The book details are :")
        super().display()
```

```
class Magazine(LibraryItems):
   def __init__(self,UPC,title,volume,issue_num):
      self.UPC = UPC
      self.title = title
      self.volume = volume
      self.issue_num = issue_num
      self.dispitems = [UPC,title,volume,issue_num]
      super().__init__(self.dispitems)

   def display(self):
      print("The magazine details are :")
      super().display()

class DVD(LibraryItems):
   def __init__(self,UPC):
            self.UPC = UPC
      self.dispitems = [UPC]
      super().__init__(self.dispitems)
   def display(self):
      print("The DVD details are :")
      super().display()

class CD(LibraryItems):
   def __init__(self,UPC,author):
          self.UPC = UPC
          self.author = author
          self.dispitems = [UPC,author]
          super().__init__(self.dispitems)
```

```
    def display(self):
        print("The CD details are :")
        super().display()

class Author:
    def __init__(self,fname,lname):
        self.fname = fname
        self.lname = lname
    def display(self):
            print(self.fname + ' ' + self.lname)

class Contributer:
    def __init__(self,fname,lname,books):
        self.fname = fname
        self.lname = lname
        self.books = books
        self.dispitems = [fname,lname,books]

    def display(self):
        print(f"Contributer name is : {self.fname} {self.lname}")
        print("Books donated along with quantities are:")
        for i in self.books:
            print(f"Book is : {i[0]} \nQuantity is : {i[1]}\n")

    def find_total(self):
        s = 0
        for i in self.books:
            s += i[1]
        return s
```

```
class Catalog:
    def __init__(self,items):
            self.items = items
    def find(self):
        option = int(input("1.Enter 1 if you want to search a Book.\n2.Enter 2 if you want to search a CD.\n3.Enter 3 if you want to search a Magazine.\n4.Enter 4 if you want to search a DVD.\n"))
        if option == 1:
            self.findbook()
        if option == 2:
            self.findCD()
        if option == 3:
            self.findMag()
        if option == 4:
            self.findDVD()

    def findbook(self):
        choice = int(input("Enter how you want to search :\n1. Enter 1 to search via ISBN.\n2. Enter 2 to search via certain subject.\n3. Enter 3 to search via title.\n4. Enter 4 to search via author last name.\n"))
        val = input("Enter value : ")
        found = False
        for i in self.items:
            if isinstance(i, Book):
                if choice == 1:
                    if i.ISBN == val:
                        found = True
                        i.display()
                        print()
                elif choice == 2:
                    if i.subject == val:
```

```
                    found = True
                    i.display()
                    print()
                elif choice == 3:
                    if i.title == val:
                        found = True
                        i.display()
                        print()
                elif choice == 4:
                    if i.author.lname == val:
                        found = True
                        i.display()
                        print()
        if not found:
            print("Book does not exist.\n")


    def findCD(self):
        choice = int(input("1. Enter 1 if you want to search via UPC.\n2. Enter 2 if you want to
search via author last name.\n"))
        val = input("Enter value : ")
        found = False
        for i in self.items:
            if isinstance(i,CD):
                if choice == 1:
                    if i.UPC == val:
                        found = True
                        i.display()
                        print()
                if choice == 2:
                    if i.author.lname == val:
```

```
                    found = True
                    i.display()
                    print()
        if not found:
            print("CD does not exist.\n")

    def findDVD(self):
        val = input("Enter UPC : ")
        found = False
        for i in self.items:
            if isinstance(i,DVD):
                if i.UPC == val:
                    found = True
                    i.display()
                    print()

        if not found:
            print("DVD does not exist.\n")

    def findMag(self):
        choice = int(input("1. Enter 1 to search via UPC.\n2. Enter 2 to search via Title.\n3. Enter 
3 to search via volume.\n4. Enter 4 to search via issue number.\n"))
        val = input("Enter value : ")
        found = False
        for i in self.items:
            if isinstance(i,Magazine):
                if choice == 1:
                    if i.UPC == val:
                        found = True
                        i.display()
                    print()
                elif choice == 2:
                    if i.title == val:
                        found = True
                        i.display()
                        print()
                elif choice == 3:
                    if i.volume == val:
                        found = True
                        i.display()
                        print()
                elif choice == 4:
                    if i.issue_num == val:
                        found = True
                        i.display()
                        print()
        if not found:
            print("Magazine does not exist.\n")

#driver code
if __name__ == '__main__':
    #The code provided here will not be executed when imported
    #writing down authors
    auth1 = Author('JK','Rowling')
    auth2 = Author('Arthur','Kingsley')

    #writing down books
    book1 = Book('a100','b2','fiction','Harry Potter',auth1)
    book2 = Book('a101','c2','History','Trojan Horse',auth2)
```

```
#writing down cd
cd1 = CD('ca100',auth1)
cd2 = CD('ca101',auth2)

#writing down magazines
mag1 = Magazine('ma100','The Moon','vol1','y155')
mag2 = Magazine('ma101','The Sun','vol2','z100')
dvd1 = DVD('da100')
dvd2 = DVD('da101')

#creating a catalog using given data
catalog = Catalog([book1,book2,cd1,cd2,mag1,mag2,dvd1,dvd2])

#finding book
catalog.findbook()
print()

#finding cd
catalog.findCD()
print()

#finding magazine
catalog.findMag()
print()

#finding dvd
catalog.findDVD()
print()
```

```
    #finding anything using common function
    catalog.find()
    print()
```

OUTPUT:

```
Enter how you want to search :
1. Enter 1 to search via ISBN.
2. Enter 2 to search via certain subject.
3. Enter 3 to search via title.
4. Enter 4 to search via author last name.
1
Enter value : a100
The book details are :
a100
b2
fiction
Harry Potter
JK Rowling
1. Enter 1 if you want to search via UPC.
2. Enter 2 if you want to search via author last name.
2
Enter value : ca101
CD does not exist.
```

IMPLEMENTATION – EX 3 :

Question 1:

```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def getPoint(self):
        return self.x, self.y

    def showPoint(self):
        print(f"Point: ({self.x}, {self.y})")

class Shape(Point):
    def __init__(self, x, y, vertices):
        super().__init__(x, y)
        self.vertices = vertices

    def identifyShape(self):
        num_vertices = len(self.vertices)
        if num_vertices == 3:
            return "Triangle"
        elif num_vertices == 4:
            side_lengths = []
            for i in range(4):
                x1, y1 = self.vertices[i]
                x2, y2 = self.vertices[(i + 1) % 4]
                side_length = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
            side_lengths.append(side_length)
          # Assuming a square has equal sides
          if all(side == side_lengths[0] for side in side_lengths):
            return "Square"
          # Assuming a rectangle has opposite sides of equal length
          elif side_lengths[0] == side_lengths[2] and side_lengths[1] == side_lengths[3]:
            return "Rectangle"
        return "Unknown Shape"


class Outlier(Shape):
    def checkIfPointInShape(self, x_point, y_point):
        # Assuming the shape is a square with vertices at (self.x, self.y), (self.x + side_length,
self.y),
        # (self.x, self.y + side_length), and (self.x + side_length, self.y + side_length)
        side_length = math.sqrt((self.vertices[1][0] - self.vertices[0][0]) ** 2 + (self.vertices[1][1]
- self.vertices[0][1]) ** 2)


        if self.x <= x_point <= self.x + side_length and self.y <= y_point <= self.y + side_length:
            print("Point falls within the area.")
        else:
            print("Point is an outlier.")


# Main method to demonstrate the classes
if __name__ == "__main__":
    # Create a Point
    point = Point(2, 3)
    point.showPoint()


    # Create a Shape (Assuming a square with vertices)
    vertices = [(2, 3), (4, 3), (2, 5), (4, 5)]
    shape = Shape(7, 9, vertices)

    print("Identified Shape:", shape.identifyShape())

    # Create an Outlier and check if a point falls within the shape

    outlier_point = Outlier(3, 4, vertices)

    x_point, y_point = 3.5,4.5

    print(f"Checking point ({x_point}, {y_point})")

    outlier_point.checkIfPointInShape(x_point, y_point)
```

OUTPUT:

```
Point: (2, 3)

Identified Shape: Rectangle

Checking point (3.5, 4.5)

Point falls within the area.
```

Question 2:

```
class Node:

    '''creating a class node '''


    def __init__(self, item = None, prev = None, next = None , parent =  None) :

        self.item = item

        self.left = prev

        self.right = next

        self.parent = parent


class Tree() :

    '''creating a tree data structure to perform operation like insert , search , delete and
traverse operations'''
```

```
    def __init__(self) :

        self.root = None

        self.size = 0


    def left (self,pos) :

        return pos.left


    def right(self ,pos):

        return pos.right


    def addroot(self,item) :                        # creates the root of the tree

        if self.root is not None :

            raise ValueError("root exits")

        root = Node(item)

        self.size = 1

        return root


    def addleft(self,item,pos) :                 # add the left node to the node by creating a new
node with item


        if pos is None:

            raise TypeError('Not a valid position.')

        if self.left(pos) is not None :

            raise ValueError("item is there")

        else:

            pos.left = Node(item,parent = pos)

            self.size += 1

            return pos.left
```

```
  def addright(self,item,pos) :                  # add the right node to the node by creating a new
node with item

    if pos is None:
      raise TypeError('Not a valid position.')
    if self.right(pos) is not None :
      raise ValueError("item is there")
    else:
      pos.right = Node(item,parent = pos)
      self.size += 1
      return pos.right

  def parent(self,pos) :
    return pos.parent

  def insert(self,element,pos) :          # insert the element in the tree
    if pos == None:
      self.root = self.addroot(element)

    while pos is not None:
      if pos.item > element :
        if pos.left is None :
          self.addleft(element,pos)
          break
        else :
          return (self.insert(element, pos.left))
      else :
        if pos.right is None :
          self.addright(element,pos)
          break
```

```
            else :
                return (self.insert(element,pos.right))


    def search(self,element,pos) :        # search the elrement in the tree
        if pos.item == element:
            return True
        elif pos.item > element :
          return (self.search(element, pos.left))
        elif pos.item < element :
                return (self.search(element,pos.right))
        else :
            return False


    def address(self,element,pos) :            # search the elrement in the tree
        if pos.item == element:
            return pos
        elif pos.item > element :
          return (self.address(element, pos.left))
        elif pos.item < element :
                return (self.address(element,pos.right))
        else :
            return None


    def findmin (self,pos) :        # returns the mininum element of the tree
        if pos.left is None :
            return pos
        else :
            return self.findmin(pos.left)
```

```
  def delete (self,element) :
    pos = self.address(element,self.root)
    Parent = self.parent(pos)

    if pos.left == None and pos.right == None :     # delete the node which has no child
      if Parent.left == pos :
        Parent.left = None
        self.size -= 1
      elif Parent.right == pos :
        Parent.right = None
        self.size -= 1

    elif pos.left != None and pos.right == None :    # delete the node which has left child
alone
      if Parent.left == pos :
        Parent.left = pos.left
        self.size -= 1
      else:
        Parent.right = pos.left
        self.size -= 1

    elif pos.left == None and pos.right != None :    # delete the node which has right child
alone
      if Parent.left == pos :
        Parent.left = pos.right
        self.size -= 1
      else:
        Parent.right = pos.right
        self.size -= 1
        elif pos.left != None and pos.right != None
            r = self.findmin(pos.right)
            pos.item = r.item
            r.item = 20000000
            self.delete(r.item)

    def traverse(self,pos):
        if pos is None :
            pos = self.root
        if pos is not None :
            if pos.left is not None :
                self.traverse(pos.left)
            print(pos.item)
            if pos.right is not None :
                self.traverse(pos.right)

a = Tree()
a.insert(6,a.root)
a.insert(5,a.root)
a.insert(8,a.root)
a.traverse(a.root)
print(a.search(5,a.root))
a.delete(6)
a.traverse(a.root)

from inheritanceTREES import Tree
import os

class FilesSystemTree(Tree):
    '''This subclass simulate a file system hierarchy where each node represents a directory or
file .'''

    def search_file (self, element ,pos) :
        ''' search for a file if it is present ,then return "true "or else return "false"
        if the other than file is given as input ,then it return "it is not a file" '''

        if os.path.isfile(element):
            return super().search(element ,pos)
        else:
            print("It is not a file")

    def add(self,element ,pos) :                                    # add the files or
directories in the tree
        super().insert(element ,pos)

    def display(self,x):                                    # display the files or directories
        super().traverse(x)

    '''def delete(self,item):                                    # delete the files or directories
        super().delete(item)'''

if __name__ == '__main__' :
    # creating a instance for child class
    a = FilesSystemTree()
    # add the files and directories
    a.add("e:\it lab",a.root)
    a.add("E:\it lab\SEM 3\Programming and Design Patterns\Lab",a.root)
    a.add("E:\it lab\SEM 3\Programming and Design Patterns\Lab\library.py",a.root)
    # displaying the files and directories

    a.display(a.root)

    # search a file

    print(a.search_file("E:\it lab\SEM 3\Programming and Design
Patterns\Lab\library.py",a.root))


    a.search_file("E:\it lab\SEM 3\Programming and Design Patterns\Lab",a.root)
# output : It is not a file


    # delete a file or directory

    a.delete("E:\it lab\SEM 3\Programming and Design Patterns\Lab")


    # display the final list after a deletion

    a.display(a.root)


from inheritanceTREES import Tree


class OrganisationTree (Tree):

    '''This subclass simulate a organisational hierarchy where each node represents an
employee or department .'''


    def add(self,element ,pos) :                                    # add the employees or
departments in the tree

        super().insert(element ,pos)


    def search_employee (self, element ,pos,position) :

        ''' search for a employee if he/she is present ,then return "true "or else return "false"

        if the other than employee is given as input ,then it return "not an employee" '''


        if position == "employee" :

            return super().search(element ,pos)
        else:

            print("not an employee")


    def display(self,x):                                                         # display the employees or
department

        super().traverse(x)


    '''def delete(self,item):                                                    # delete the employees or
department

        super().delete(item)'''




if __name__ == '__main__' :

    # creating a instance for child class

    a = OrganisationTree()


    # add the employees and departments

    a.add("maths",a.root)

    a.add("Ram",a.root)

    a.add("IT",a.root)


    # displaying the employees and departments

    a.display(a.root)


    # search an employee

    print(a.search_employee("Ram ",a.root,"employee"))

    a.search_employee("maths",a.root,"department")


    # delete an employee or departments

    a.delete("IT")
```

```
# display the final list after a deletion
a.display(a.root)
```

<u>OUTPUT:</u>

```
IT
Ram
maths
True
not an employee
Ram
Maths
```

IMPLEMENTATION – EX 4 :

Question 1:

```
import datetime
#class to represent individual notes
class Note:
    def __init__(self, content, tags=None):
        self.content = content
        self.creation_date = datetime.datetime.now() #records creation date
        self.tags = tags if tags else [] #initializes tags as an empty list if not provided

    def add_tag(self, tag):
        if tag not in self.tags:
            self.tags.append(tag)
            self.tags.sort() #sorts tags alphabetically for easier querying

    def remove_tag(self, tag):
        if tag in self.tags:
            self.tags.remove(tag)
            self.tags.sort() #sorts tags after removal

    def modify_content(self, new_content):
        self.content = new_content



    def __str__(self):
        return f"Created on: {self.creation_date}\nContent: {self.content}\nTags: {', 
'.join(self.tags)}"
```

```
#class to manage a collection of notes
class Notebook:
    def __init__(self):
        self.notes = []

    def add_note(self, note):
        self.notes.append(note)

    def delete_note(self, note):
        if note in self.notes:
            self.notes.remove(note)

    def search_notes(self, query):
        matching_notes = []
        for note in self.notes:
            #checks if the query is present in the note content or tags
            if query in note.content or query in note.tags:
                matching_notes.append(note)
        return matching_notes

    def __str__(self):
        return f"Number of Notes: {len(self.notes)}"

#class to represent tags associated with notes
class Tag:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return self.name

# Creating some tags
tag1 = Tag("work")
tag2 = Tag("personal")

# Creating notes
note1 = Note("Job search", ["work"])
note2 = Note("Grocery shopping", ["personal"])

# Creating a notebook
notebook = Notebook()

# Adding notes to the notebook
notebook.add_note(note1)
notebook.add_note(note2)

# Modifying a note
note1.modify_content("Job search from 3 PM")

# Adding and removing tags
note1.add_tag("important")
note2.remove_tag("personal")

# Searching for notes
results = notebook.search_notes("Job")
for result in results:
    print(result)
#creating more tags
```

```
tag3 = Tag("meetings")

tag4 = Tag("recipes")


# Create new notes

note3 = Note("Weekly team meeting", ["work", "meetings"])

note4 = Note("Spaghetti recipe", ["personal", "recipes"])

note5 = Note("Project deadline", ["work", "meetings", "important"])


# Adding new notes to the notebook

notebook.add_note(note3)

notebook.add_note(note4)

notebook.add_note(note5)


# Modifying a note's content

note4.modify_content("Spaghetti Carbonara recipe")




# Adding and removing tags from a note

note3.add_tag("important")

note5.remove_tag("important")


# Searching for notes by content

results_by_content = notebook.search_notes("meeting")

print("Search results by content:")

for result in results_by_content:

    print(result)


# Searching for notes by tags

results_by_tags = notebook.search_notes("work")
```

```
print("\nSearch results by tags:")
for result in results_by_tags:
    print(result)

# Deleting a note
notebook.delete_note(note4)

# Display the updated notebook
print("\nUpdated Notebook:")
print(notebook)
```

OUTPUT:

```
Created on: 2023-11-17 17:55:36.159521
Content: Job search from 3 PM
Tags: important, work
Search results by content:
Created on: 2023-11-17 17:55:36.159521
Content: Weekly team meeting
Tags: important, meetings, work

Search results by tags:
Created on: 2023-11-17 17:55:36.159521
Content: Job search from 3 PM
Tags: important, work
Created on: 2023-11-17 17:55:36.159521
```

```
Content: Weekly team meeting
Tags: important, meetings, work
Created on: 2023-11-17 17:55:36.159521
Content: Project deadline
Tags: meetings, work

Updated Notebook:
Number of Notes: 4
```

<u>Question 2:</u>

```
import datetime


# Date Module to create and display dates
def create_date(year, month, day):
    return datetime.date(year, month, day)


def display_date(date_obj):
    return date_obj.strftime("%d.%m.%Y")


# Current Module to get current time and date in various formats
def current_time():
    return datetime.datetime.now().strftime("%H:%M:%S")


def current_date(format="dd.mm.yyyy"):
    if format == "mm.dd.yyyy":
        return datetime.datetime.now().strftime("%m.%d.%Y")
    elif format == "string":
        return datetime.datetime.now().strftime("%A, %d %B %Y")
    else:
        return datetime.datetime.now().strftime("%d.%m.%Y")




# Convert Module converts hours to days, days to hours and man-hours to days
def convert_hrs_days(hours):
    return hours / 24

def convert_days_hrs(days):
    return days * 24

def convert_man_hrs_days(man_hours):
    return man_hours / 8

# Validity Module checks if a given time or date string is in valid format
def is_valid_time(time_str):
    try:
        datetime.datetime.strptime(time_str, "%H:%M:%S")
        return True
    except ValueError:
        return False

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



# Difference Module calculated differences between dates and times
def difference_with_current(date_obj):
    current_date = datetime.date.today()
    return (current_date - date_obj).days


def difference(date1, date2):
    return abs((date2 - date1).days)


def days_after(days):
    return datetime.date.today() + datetime.timedelta(days=days)


def days_before(days):
    return datetime.date.today() - datetime.timedelta(days=days)


def month_after(months):
    today = datetime.date.today()
    new_month = today.month + months
    new_year = today.year + new_month // 12
    new_month %= 12
    if new_month == 0:
        new_month = 12
    return today.replace(year=new_year, month=new_month)


def month_before(months):
    today = datetime.date.today()
    new_month = today.month - months
    new_year = today.year - new_month // 12
    new_month %= 12
    if new_month == 0:
        new_month = 12
    return today.replace(year=new_year, month=new_month)

# Registration Application
'''the registration application prompts the user to enter the student's details
including their name and dob in dd.mm.yyyy format. It checks whether the date of
birth entered by user is in the correct format. It then calculates the age of the
student based on the date provided. It checks if the claculated age is less than
or equal to 17 and if so displays that the student is eligible for U17 and prints
a registration confirmation,registration date and validity date(6 months from current date)
If not it prints that the student is not eligible.'''

def register_student():
    print("Enter student details:")
    name = input("Name: ")
    dob = input("Date of Birth (dd.mm.yyyy): ")  # Use the "dd.mm.yyyy" format

    if not is_valid_date(dob):
        print("Invalid date format. Use dd.mm.yyyy format.")
        return

    day, month, year = map(int, dob.split('.'))
    birth_date = create_date(year, month, day)
    today = datetime.date.today()
    # Calculate age correctly
    if (today.month, today.day) < (birth_date.month, birth_date.day):
        age = today.year - birth_date.year - 1
    else:
        age = today.year - birth_date.year

    if age <= 17:
        print(f"Registration successful for {name}.")
        registration_date = current_date()
        print(f"Registration Date: {registration_date}")
        six_months_validity = days_after(180)
        print(f"Registration Valid Until: {display_date(six_months_validity)}")
    else:
        print(f"Sorry, {name} is not eligible for the U17 category.")

if __name__ == "__main__":
    # Date module
    dob = create_date(2005, 5, 15) #creates a date object for may 15,2005
    formatted_date = display_date(dob) #formats the date as "15.05.2005"
    print(f"Formatted Date: {formatted_date}")

    #Current module
    current_time_str = current_time() #retreives the current time in "HH:MM:SS" format
    current_date_ddmmyyyy = current_date() #retrieves current date in "dd.mm.yyyy" 
format
    current_date_mmddyyyy = current_date("mm.dd.yyyy") #retrieves current date in 
"mm.dd.yyyy" format
    current_date_string = current_date("string") #retrieves current date as a string like 
"Monday, 01 January 2023"
    print(f"Current Time: {current_time_str}")
    print(f"Current Date (dd.mm.yyyy): {current_date_ddmmyyyy}")

    print(f"Current Date (mm.dd.yyyy): {current_date_mmddyyyy}")

    print(f"Current Date (string): {current_date_string}")


    #Convert module

    hours = 48

    days_from_hours = convert_hrs_days(hours) #converts 48 hours to 2 days

    days_to_hours = convert_days_hrs(5) #converts 5 days to 120 hours

    days_from_man_hours = convert_man_hrs_days(64) #converts 64 man-hours to 8 days

    print(f"Days from Hours (48 hours): {days_from_hours}")

    print(f"Days to Hours (5 days): {days_to_hours}")

    print(f"Days from Man-Hours (64 man-hours): {days_from_man_hours}")


    #Validity module

    time_str = "08:30:00"

    is_valid = is_valid_time(time_str) #checks if input is a valid time format

    date_str = "25.12.2022"

    is_valid_date_str = is_valid_date(date_str) #checks if input is a valid date format


    #Difference module

    date_obj = create_date(2022, 12, 25)

    days_difference = difference_with_current(date_obj) #calculates days until or since
December 25,2022

    date1 = create_date(2022, 12, 25)

    date2 = create_date(2023, 1, 10)

    days_diff = difference(date1, date2) #calculates days between December 25,2022 and
January 10,2023

    days_after_date = days_after(7) #calculates a date 7 days after the current date

    days_before_date = days_before(3) #calculates a date 3 days before the current date

    months_after_date = month_after(2) #calculates a date 2 months after the current date
    print(f"Is Valid Time (08:30:00): {is_valid}")
    print(f"Is Valid Date (25.12.2022): {is_valid_date_str}")
    print(f"Days Difference with Current (25.12.2022): {days_difference}")
    print(f"Days Difference (25.12.2022 to 10.01.2023): {days_diff}")
    print(f"Days After (7 days from today): {display_date(days_after_date)}")
    print(f"Days Before (3 days before today): {display_date(days_before_date)}")
    print(f"Months After (2 months from today): {display_date(months_after_date)}")

    #registration application
    register_student()  #above 17yrs
    register_student()  #below 17yrs
```

OUTPUT:

```
Formatted Date: 15.05.2005
Current Time: 18:02:28
Current Date (dd.mm.yyyy): 17.11.2023
Current Date (mm.dd.yyyy): 11.17.2023
Current Date (string): Friday, 17 November 2023
Days from Hours (48 hours): 2.0
Days to Hours (5 days): 120
Days from Man-Hours (64 man-hours): 8.0
Is Valid Time (08:30:00): True
Is Valid Date (25.12.2022): True
Days Difference with Current (25.12.2022): 327
Days Difference (25.12.2022 to 10.01.2023): 16
Days After (7 days from today): 24.11.2023
Days Before (3 days before today): 14.11.2023
Months After (2 months from today): 17.01.2024
Enter student details:
```

Name: Ram

Date of Birth (dd.mm.yyyy): 23.11.2004

Sorry, Ram is not eligible for the U17 category.

IMPLEMENTATION – EX 5 :

Question 1:

```
class Course:
    def __init__(self,course_code,course_name,credit_hours, *args,**kwargs):
        self.course_code = course_code
        self.course_name = course_name
        self.credit_hours = credit_hours
        self.additional_args = args
        self.additional_kwargs = kwargs

    def add_info(self):
        self.course_code = input("enter the course_code : ")
        self.course_name = input("enter the course_name: ")
        self.credit_hours = int(input("enter the credit_hours: "))
        self.additional_args = input("any additional info: ")
        self.additional_kwargs = input("any additional kwags: ")

    def display_info(self):
        print(f"Course_code:{self.course_code}")
        print(f"couse_name:{self.course_name}")
        print(f"credit_hours:{self.credit_hours}")
        if self.additional_args:
            print(f"additional_args:{self.additional_args}")

class Corecourse(Course):
    def __init__(self,*args,**kwargs):
        super().__init__(*args,**kwargs)

    def add_info(self):
        super().add_info()
        self.required_prerequistes = input("prerequistes: ")
    def display_info(self):
        print(f"required_prerequistes={self.required_prerequistes}")
        super().display_info()

class Elective(Course):
    def __init__(self,*args,**kwargs):
        super().__init__(*args,**kwargs)

    def add_info(self):
        super().add_info()
        self.available_terms_property = input("enter the lab properties: ")

    def display_info(self):
        super().display_info()
        print(f"available terms:{self.available_terms_property}")

class Labcourses(Course):
    def __init__(self,*args,**kwargs):
        super().__init__(*args,**kwargs)

    def add_info(self):
        super().add_info()
        self.lab_location = input("enter the lab location: ")

    def display_info(self):
        super().display_info()
        print(f"lab_location:{self.lab_location}")
```

```
# Create instances of CoreCourse, Elective, and Labcourses

# Create instances of CoreCourse, Elective, and Labcourses

core_course = Corecourse("CS100", "CS101", 3, required_prerequisites="CS100")

elective_course = Elective(["Fall", "Spring"], "MATH201", 4, available_terms_property="Fall
only")

lab_course = Labcourses("Lab Building 2", "CHEM301", 2, lab_location="Lab A")


# Add information to each course

core_course.add_info()

elective_course.add_info()

lab_course.add_info()


# Display course information

print("\nCore Course Information:")

core_course.display_info()

print("\nElective Course Information:")

elective_course.display_info()

print("\nLab Course Information:")

lab_course.display_info()
```

<u>OUTPUT:</u>

```
enter the course_code : C123

enter the course_name: MATHS

enter the credit_hours: 45

any additional info: ADVANCE MTH

any additional kwags:

prerequistes: NOTHING
```

Question 2:

```
class Movie:
    def __init__(self, title, director, year, genre, *args, **kwargs):
        self.title = title
        self.director = director
        self.year = year
        self.genre = genre
        self.additional_args = args
        self.additional_kwargs = kwargs

    def display_info(self):
        print(f"Title: {self.title}, Director: {self.director}, Year: {self.year}, Genre: {self.genre}")
        if self.additional_args:
            print(f"Additional Args: {self.additional_args}")
        if self.additional_kwargs:
            print(f"Additional Kwargs: {self.additional_kwargs}")

class Movielist(Movie):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.actor = ""
        self.actress = ""
        self.budget = ""
    def add_info(self):

        self.actor = input("Enter the actor of the movie: ")

        self.actress = input("Enter the actress of the movie: ")

        self.budget = input("Enter the budget of the movie: ")


    def display_info(self):

        super().display_info()

        print("Additional Information: ")

        print(f"Actor: {self.actor}, Actress: {self.actress}, Budget: {self.budget}")


# Create instances and use the classes

movie1 = Movielist("Leo", "Loki", 2023, "Action")

movie1.add_info()

movie1.display_info()


movie2 = Movielist("Jailer", "Nelson", 2022, "Adventure", "Worstfilm",
additional_info="Don't watch this movie this a shitt")

movie2.add_info()

movie2.display_info()
```

<u>OUTPUT:</u>

```
Enter the actor of the movie: VIJAY

Enter the actress of the movie: leo

Enter the budget of the movie: 500

Title: Leo, Director: Loki, Year: 2023, Genre: Action

Additional Information:

Actor: VIJAY, Actress: leo, Budget: 500
```

IMPLEMENTATION – EX 6 :

Question 1:

```
from abc import ABC

from abc import abstractmethod


class Loan(ABC):

   def __init__(self, loan_amount, account_type, location, borrower_details):

      self.borrower_details = borrower_details

      self.loan_amount = loan_amount

      self.account_type = account_type

      self.location = location


   @abstractmethod

   def calculate_interest(self):

      ...

   @abstractmethod

   def DisplayDetails(self):

      ...

   @abstractmethod

   def MonthlyPaymentInterest(self):

      ...

   @abstractmethod

   def MonthlyPaymentTotal(self):

      ...


class EducationLoan(Loan):

      def __init__(self, loan_amount, account_type, location, course_fee, borrower_details):

      super().__init__(loan_amount, account_type, location, borrower_details)

      self.course_fee = course_fee
```

```
    def calculate_interest(self):
        if self.location == "urban":
            location_factor = 1
        else:
            location_factor = 0.95
        if self.account_type == "Savings":
            account_type_factor = 1.05
        else:
            account_type_factor = 1
                base_interest_rate = 0.08
        return self.loan_amount * base_interest_rate * location_factor * account_type_factor

    def DisplayDetails(self):
        print(f"Borrower name is {self.borrower_details[0]}")
        print(f"Borrower age is {self.borrower_details[1]}")
        print(f"Borrower martial status is {self.borrower_details[2]}")

    def MonthlyPaymentInterest(self,years):
        interest = self.calculate_interest()
        return interest / (years * 12)

    def MonthlyPaymentTotal(self,years):
        interest_per_month = self.MonthlyPaymentInterest(years)
        loan_amt_per_month = self.loan_amount / (years * 12)
        return loan_amt_per_month + interest_per_month

class HomeLoan(Loan):
    def __init__(self, loan_amount, account_type, location, borrower_details):
        super().__init__(loan_amount, account_type, location, borrower_details)



    def calculate_interest(self):
        base_interest_rate = 0.06
        location_factor = 1.02 if self.location == "urban" else 1.0
        account_type_factor = 1.05 if self.account_type == "Savings" else 1.0
        return self.loan_amount * base_interest_rate * location_factor * account_type_factor

    def DisplayDetails(self):
        print(f"Borrower name is {self.borrower_details[0]}")
        print(f"Borrower age is {self.borrower_details[1]}")
        print(f"Borrower martial status is {self.borrower_details[2]}")

    def MonthlyPaymentInterest(self,years):
        interest = self.calculate_interest()
        return interest / (years * 12)

    def MonthlyPaymentTotal(self,years):
        interest_per_month = self.MonthlyPaymentInterest(years)
        loan_amt_per_month = self.loan_amount / (years * 12)
        return loan_amt_per_month + interest_per_month

class PersonalLoan(Loan):
    def __init__(self, loan_amount, account_type, location, borrower_details):
        super().__init__(loan_amount, account_type, location, borrower_details)

    def calculate_interest(self):
        base_interest_rate = 0.1
        location_factor = 1.05 if self.location == "urban" else 0.98
        account_type_factor = 1.08 if self.account_type == "Savings" else 1.0
        return self.loan_amount * base_interest_rate * location_factor * account_type_factor



    def DisplayDetails(self):
        print(f"Borrower name is {self.borrower_details[0]}")
        print(f"Borrower age is {self.borrower_details[1]}")
        print(f"Borrower martial status is {self.borrower_details[2]}")

    def MonthlyPaymentInterest(self,years):
        interest = self.calculate_interest()
        return interest / (years * 12)

    def MonthlyPaymentTotal(self,years):
        interest_per_month = self.MonthlyPaymentInterest(years)
        loan_amt_per_month = self.loan_amount / (years * 12)
        return loan_amt_per_month + interest_per_month

#driver code
if __name__ == '__main__':
    #The code provided here will not be executed when imported
    try:
        education_loan = EducationLoan(100000, "Savings", "urban", 8000,
["Ram",19,"Unmarried"])
        home_loan = HomeLoan(500000, "Current", "rural",["Vivek",50,"Married"])
        personal_loan = PersonalLoan(200000, "Savings", "urban",["Nikhil",28,"Married"])
        education_loan.DisplayDetails()
        print()
        home_loan.DisplayDetails()
        print()
        personal_loan.DisplayDetails()
        print()
        print(f"Education Loan Interest:{education_loan.calculate_interest()}")
        print()
        print(f"Home Loan Interest:{home_loan.calculate_interest()}")
        print()
        print(f"Personal Loan Interest:{personal_loan.calculate_interest()}")
        print()
        print(f"Education Loan Payment interest per month for 5 years :
{education_loan.MonthlyPaymentInterest(5)}")
        print()
        print(f"Home Loan Payment interest per month for 5 years :
{home_loan.MonthlyPaymentInterest(5)}")
        print()
        print(f"Personal Loan Interest per month for 5 years :
{personal_loan.MonthlyPaymentInterest(5)}")
        print()
        print(f"Education Loan total payment per month for 5 years :
{education_loan.MonthlyPaymentTotal(5)}")
        print()
        print(f"Home Loan total payment per month for 5 years :
{home_loan.MonthlyPaymentTotal(5)}")
        print()
        print(f"Personal Loan total payment per month for 5 years :
{personal_loan.MonthlyPaymentTotal(5)}")
        print()
    except Exception as e:
        print("Error:", str(e))
```

OUTPUT:

```
Borrower name is Ram
Borrower age is 19
Borrower martial status is Unmarried

Borrower name is Vivek
Borrower age is 50
Borrower martial status is Married

Borrower name is Nikhil
Borrower age is 28
Borrower martial status is Married

Education Loan Interest:8400.0
Home Loan Interest:30000.0
Personal Loan Interest:22680.0
Education Loan Payment interest per month for 5 years : 140.0
Home Loan Payment interest per month for 5 years : 500.0

Personal Loan Interest per month for 5 years : 378.0
Education Loan total payment per month for 5 years : 1806.6666666666667
Home Loan total payment per month for 5 years : 8833.333333333334
Personal Loan total payment per month for 5 years : 3711.3333333333335
```

Question 2:

```
import os
def SecureFileReader(fname,fpath):
    try:
        found = False
        for dir_path, dir_names, file_names in os.walk(fpath):
            if fname in file_names:
                found = True
                file=open(fname.txt, "r")
                a = file.readlines()
                for i in a:
                    print(i)
        if not found:
            raise FileNotFoundError("File does not exist.")
    except FileNotFoundError:
        print("File not found")
    except PermissionError:
        print("Required permissions not met")
    except Exception as e:
        print("An error occurred while reading the file.")
    else:
        print("File contents have been successfully printed.\n")

#driver code
SecureFileReader("Movie.py",r"Z:\Programming and Design Patterns")
print()
SecureFileReader("RandomFileDoesNotExist.txt",r"Z:\Programming and Design Patterns")
print()
SecureFileReader("textfile.txt",r"Z:\Programming and Design Patterns")
print()
```

OUTPUT:

```
Required permissions not met
File not found
HI
```

HOW

ARE YOU

Question 3:

```
class Calculator:
    def add(self,a,b):
        c=a+b
        return f"Addition of {a} and {b} is {c}"



    def subract(self,a,b):
        d=a-b
        return f"Subraction of {a} and {b} is {d}"

     def multiply(self,a,b):
        e=a*b
        return f"Multiplication of {a} and {b} is {e}"

     def divide(self,a,b):
        if b==0:
            raise ZeroDivisionError("Division by Zero is not Allowed")
        else:
            f=a/b
        return f"Division of {a} and {b} is {f}"

c=Calculator()
a=input("Enter Number1: ")
b=input("Enter Number2: ")
```

```
try:
   a=float(a)
   b=float(b)
   operation=input("Enter Operation: (Add, Sub, Mul, Div): ")
   if not ((isinstance(a ,int) or isinstance(b ,float)) and (isinstance(a ,float) or isinstance(b
,int))):
      raise TypeError("Both a and b must be Integer or Float")
   if operation=="Add":
      print(c.add(a,b))
   elif operation=="Sub":
      print(c.subract(a,b))
   elif operation=="Mul":
      print(c.multiply(a,b))
   elif operation=="Div":
      print(c.divide(a,b))
   else:
      raise ValueError("Invalid Operation, Please Enter Valid Operation like (Add, Sub, Mul,
Div)")
except (ZeroDivisionError,ValueError,TypeError) as error:
   print(error)
```

<u>OUTPUT:</u>

```
Enter Number1: 5
Enter Number2: 0
Enter Operation: (Add, Sub, Mul, Div): Div
Division by Zero is not Allowed


Enter Number1: 3
Enter Number2: g
could not convert string to float: 'g'
```

```
Enter Number1: 7

Enter Number2: 4

Enter Operation: (Add, Sub, Mul, Div): square root

Invalid Operation, Please Enter Valid Operation like (Add, Sub, Mul, Div)
```

IMPLEMENTATION – EX 7 :

Question 1:

```
class TextEditor:
    def __init__(self):
        # Initialize an empty text when an instance of TextEditor is created.
        self.text = ""

    def load_text(self, text):
        # Load the provided text into the text editor.
        self.text = text

    def get_statistics(self):
        # Calculate and return statistics about the text.
        char_count = len(self.text)  # Count characters
        word_count = len(self.text.split())  # Count words
        sentence_count = self.text.count('.') + self.text.count('!') + self.text.count('?')  # Count
sentences
        return char_count, word_count, sentence_count

    def count_word_frequencies(self, top_n):
        # Count and return the top N most frequent words in the text.
        words = self.text.split()
        word_freq = {}
        for word in words:
            word = word.strip('.,!?()[]{}":;')  # Remove punctuation
            word = word.lower()  # Convert to lowercase
            if word:
                word_freq[word] = word_freq.get(word, 0) + 1
        sorted_word_freq = sorted(word_freq.items(), key=lambda x: x[1], reverse=True)  # Sort
by frequency
```

```
        return sorted_word_freq[:top_n]



    def append_text(self, text_to_append):
        # Append the provided text to the end of the current text.
        self.text += text_to_append

    def insert_text(self, position, text_to_insert):
        # Insert the provided text at the specified position in the text.
        self.text = self.text[:position] + text_to_insert + self.text[position:]

    def search_and_replace(self, search_text, replace_text):
        # Search for a specific text and replace it with another text in the entire document.
        self.text = self.text.replace(search_text, replace_text)

    def delete_text(self, start, end):
        # Delete a portion of the text, specified by the start and end positions.
        self.text = self.text[:start] + self.text[end:]

    def categorize_text(self):
        # Categorize the text based on a specific logic, but this part is left as an exercise.
        # It's recommended to use external libraries for accurate categorization if needed.
        categorized_text = {}  # Store categorized text and their counts
        return categorized_text

# Example usage:
editor = TextEditor()
editor.load_text("Vicky waited for the train. The train was late. Mary and Samantha took the bus.")
char_count, word_count, sentence_count = editor.get_statistics()
```

```
print(f"Character Count: {char_count}")

print(f"Word Count: {word_count}")

print(f"Sentence Count: {sentence_count}")

top_words = editor.count_word_frequencies(3)

print(f"Top Words: {top_words}")

editor.append_text(" Appended Text.")

print("After Append:", editor.text)

editor.insert_text(10, "Inserted")

print("After Insert:", editor.text)

editor.search_and_replace("sample", "modified")

print("After Replace:", editor.text)

editor.delete_text(5, 14)

print("After Delete:", editor.text)
```

OUTPUT:

```
Character Count: 80

Word Count: 15

Sentence Count: 3

Top Words: [('the', 3), ('train', 2), ('vicky', 1)]

After Append:  Vicky waited for the train. The train was late. Mary and Samantha took the
bus. Appended Text.

After Insert:  Vicky waiInsertedted for the train. The train was late. Mary and Samantha
took the bus. Appended Text.

After Replace:  Vicky waiInsertedted for the train. The train was late. Mary and Samantha
took the bus. Appended Text.

After Delete:  Vickrtedted for the train. The train was late. Mary and Samantha took the
bus. Appended Text.
```

Question 2:

```
import re
from collections import Counter
```

```
class TextEditor:
    def __init__(self):
        self.text = ""

    def load_text(self, input_text):
        self.text = input_text

    def get_basic_stats(self):
        char_count = len(self.text)
        word_count = len(self.text.split())
        sentence_count = len(re.split(r'[.!?]', self.text))
        return char_count, word_count, sentence_count

    def count_word_frequencies(self, top_n):
        words = re.findall(r'\w+', self.text.lower())
        word_freq = Counter(words)
        return word_freq.most_common(top_n)

    def append_text(self, new_text):
        self.text += new_text

    def insert_text(self, position, new_text):
        self.text = self.text[:position] + new_text + self.text[position:]

    def search_and_replace(self, search_text, replace_text):
        self.text = self.text.replace(search_text, replace_text)

    def delete_text(self, start, end):
        self.text = self.text[:start] + self.text[end:]

    def categorize_text(self):
        categories = {
            "numbers": len(re.findall(r'\d+', self.text)),
            "alphabets": len(re.findall(r'[a-zA-Z]+', self.text)),
            "urls": len(re.findall(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\\(,]|(?:%[0-9a-fA-
F][0-9a-fA-F]))+', self.text)),
            "links": len(re.findall(r'www\.[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}', self.text)),
        }
        categories["others"] = len(self.text.split()) - sum(categories.values())
        return categories


if __name__ == "__main__":
    editor = TextEditor()
    input_text = """Unfortunately, the Department hasn't bothered to keep
    any of the old links,or provide cross-links into the new database-driven
    website in www.apple.com"""
    editor.load_text(input_text)

    char_count, word_count, sentence_count = editor.get_basic_stats()
    print(f"Character Count: {char_count}")
    print(f"Word Count: {word_count}")
    print(f"Sentence Count: {sentence_count}")

    top_words = editor.count_word_frequencies(5)
    print("Top 5 words and their frequencies:")
    for word, freq in top_words:
        print(f"{word}: {freq}")

    categories = editor.categorize_text()
    print("Categorized Text:")
    for category, count in categories.items():
        print(f"{category.capitalize()}: {count}")

    # Ensure total word count matches categorized count
    total_word_count = sum(categories.values())
    print(f"Total Word Count: {total_word_count}")
```

<u>OUTPUT:</u>

```
Character Count: 156
Word Count: 22
Sentence Count: 3
Top 5 words and their frequencies:
the: 3
links: 2
unfortunately: 1
department: 1
hasn: 1
Categorized Text:
Numbers: 0
Alphabets: 27
Urls: 0
Links: 1
Others: -6
Total Word Count: 22: 5
```

IMPLEMENTATION – EX 8 :

```
from getpass import getpass
import re
import pickle
import sys
from datetime import datetime
import json


class User:
    """Represents a user with a username and password."""


    def __init__(self, username: str, password: str):
        self.username = username
        self.password = password


def check_password_strength(password):
    """Check if the password meets certain strength criteria."""


    password_pattern = "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$"
    if re.match(password_pattern, password):
        return True
    return False


def serialize_user(user: User):
    """Serialize and store a user object in a file."""


    with open("users.pickle", "ab") as users_file:
        pickle.dump(user, users_file)
```

```
def check_credentials(username, password):
    """Check if the provided username and password match a registered user."""

    with open("users.pickle", "rb") as users_reader:
        while True:
            try:
                user = pickle.load(users_reader)
            except EOFError:
                return False
            if user.username == username and user.password == password:
                return True
            else:
                continue

def deserialize_users():
    """Deserialize and yield user objects from a file."""

    with open("users.pickle", "rb") as users_reader:
        while True:
            try:
                user = pickle.load(users_reader)
            except EOFError:
                return False
            yield user.username

def jsonify(message_str):
    """Serialize and store chat messages in a JSON file."""

    try:
        with open("messages.json", "r") as file:
            chat_data = json.load(file)
    except FileNotFoundError:
        chat_data = {}

    for username, messages in message_str.items():
        if username in chat_data:
            chat_data[username].extend(messages)
        else:
            chat_data[username] = messages

    with open("messages.json", "w") as message_writer:
        json.dump(chat_data, message_writer, indent=4)

def display_messages(chat_data):
    """Display chat messages for the current user."""

    username = ChatApp.current_user

    if username in chat_data:
        messages = chat_data[username]
        print(f"Messages for {username}")

        for message in messages:
            timestamp = message["timestamp"]
            message_text = message["message"]
            status = "Sent:\t\t" if message["type"] == "sent" else "Received:\t"
            if message.get("to"):
                display_name = f"To {message.get('to')}"
            else:
                display_name = f"From {message.get('from')}"
            print(f"{status}{timestamp} - {display_name}: {message_text}")
    else:
        print(f"No messages found for {username}")

def search_messages(chat_data, content):
    """Search for messages containing a specific phrase."""

    matching_messages = []
    messages = chat_data[ChatApp.current_user]
    for message in messages:
        if content in message["message"]:
            matching_messages.append(message)
    return matching_messages



class ChatApp:
    """A simple chat application."""
    usernames = []
    current_user = None

    def __init__(self):
        """Initialize the chat application."""

        user_db = open("users.pickle", "ab")
        user_db.close()
        ChatApp.usernames = list(deserialize_users())
```

```
def add_user(self):
    """Create a new user and store their credentials."""

    username = input("Please enter a username: ")
    print(
        "Note that your password contains minimum 8 characters and contain atleast\n"
        "1 uppercase character, 1 lowercase character,\n"
        "1 digit and 1 special character."
    )
    password = getpass("Please enter a password: ")
    re_password = getpass("Please re-enter your password: ")

    if password != re_password:
        print("Passwords are not the same!")
        return False
    elif username in ChatApp.usernames:
        print("Username already exists!")
        return False
    else:
        if not check_password_strength(password):
            print("Password is not strong enough!")
            return False
        else:
            user = User(username, password)
            ChatApp.usernames.append(user.username)
            serialize_user(user)
            return True

def login(self):
        """Log in with a username and password."""

        if ChatApp.current_user is not None:
            print("User already logged in! Please log out before logging in!")
            return False

        username = input("Please enter the username: ")
        password = getpass("Please enter the password: ")

        if username not in ChatApp.usernames:
            print("Invalid credentials! User not registered.")
            return False
        else:
            is_valid = check_credentials(username, password)
            if not is_valid:
                print("Invalid credentials!")
                return False
            else:
                ChatApp.current_user = username
                return True

    def send_message(self, user_choice, message_content):
        """Send a message to another user."""

        time_stamp = datetime.now().strftime("%d/%m/%Y, %H:%M:%S")
        sender = ChatApp.current_user
        receiver = ChatApp.usernames[user_choice]

        message = {}
        message[sender] = []
        message[receiver] = []

        contents_send = {
            "message": message_content,
            "timestamp": time_stamp,
            "to": receiver,
            "type": "sent",
        }
        contents_rec = {
            "message": message_content,
            "timestamp": time_stamp,
            "from": sender,
            "type": "received",
        }

        message[sender].append(contents_send)
        message[receiver].append(contents_rec)
        jsonify(message)

    def display_all_messages(self):
        """Display all chat messages for the current user."""

        try:
            with open("messages.json", "r") as file:
                chat_data = json.load(file)
        except FileNotFoundError:
            chat_data = {}
        display_messages(chat_data)
```

```
def search_message(self):
    """Search for messages containing a specific phrase and display the results."""
    if chatapp.current_user is None:
        print("You have not logged in! Log in first.")
        return
    search_phrase = input("Enter the message phrase to search: ")


    try:
        with open("messages.json", "r") as file:
            chat_data = json.load(file)
    except FileNotFoundError:
        chat_data = {}


    messages = search_messages(chat_data, content=search_phrase)


    if messages:
        for message in messages:
            timestamp = message["timestamp"]
            message_text = message["message"]
            status = "Sent:\t\t" if message["type"] == "sent" else "Received:\t"
            if message.get("to"):
                display_name = f"To {message.get('to')}"
            else:
                display_name = f"From {message.get('from')}"
            print(f"{status}{timestamp} - {display_name}: {message_text}")
    else:
        print("No messages found!")


def log_out(self):
        """Log out the current user."""
        ChatApp.current_user = None

    def exit(self):
        """Exit the chat application."""
        ChatApp.current_user = None
        sys.exit(0)


if __name__ == "__main__":
    chatapp = ChatApp()
    while True:
        print("=" * 26)
        print("1. Create a user")
        print("2. Login")
        print("3. Chat")
        print("4. Display messages")
        print("5. Log out")
        print("6. Search for a message")
        print("7. Exit")

        ch = input("Enter your choice:").strip()

        if ch == "1":
            chk = chatapp.add_user()
            if chk:
                print("User created successfully!")
            else:
                continue
```

```
elif ch == "2":
    chk = chatapp.login()

    if chk:
        print("User logged in successfully!")
        print(f"Current User: {chatapp.current_user}")
    else:
        continue

elif ch == "3":
    if chatapp.current_user is None:
        print("You have not logged in! Log in first.")
        continue

    print("Choose the user you want to chat with:")
    for idx, user in enumerate(ChatApp.usernames):
        print(idx + 1, "\t", user)

    chat_choice = int(input("Enter the choice: "))

    if not ((chat_choice >= 1) and (chat_choice <= len(ChatApp.usernames))):
        print("Please enter the correct choice!")
        continue

    message_content = input("Please provide the message to be sent: ")

    chatapp.send_message(chat_choice - 1, message_content)
elif ch == "4":
    if chatapp.current_user is None:
        print("You have not logged in! Log in first.")
        continue

    chatapp.display_all_messages()

elif ch == "5":
    chatapp.log_out()
    print("Logged out successfully.")

elif ch == "6":
    if chatapp.current_user is None:
        print("You have not logged in! Log in first.")
        continue

    chatapp.search_message()

elif ch == "7":
    print("Exiting...")
    chatapp.exit()
```

<u>OUTPUT:</u>

```
==========================
1. Create a user
2. Login
3. Chat
4. Display messages
5. Log out
```

```
6. Search for a message
7. Exit
Enter your choice:1
Please enter a username: ram
Note that your password contains minimum 8 characters and contain atleast
1 uppercase character, 1 lowercase character,
1 digit and 1 special character.
Please enter a password:
Please re-enter your password:
User created successfully!
==========================
1. Create a user
2. Login
3. Chat
4. Display messages
5. Log out
6. Search for a message
7. Exit
Enter your choice:1
Please enter a username: anbu
Note that your password contains minimum 8 characters and contain atleast
1 uppercase character, 1 lowercase character,
1 digit and 1 special character.
Please enter a password:
Please re-enter your password:
User created successfully!
==========================
1. Create a user
2. Login
```

```
3. Chat

4. Display messages

5. Log out

6. Search for a message

7. Exit

Enter your choice:2

Please enter the username: ram

Please enter the password:

User logged in successfully!

Current User: ram

=========================

1. Create a user

2. Login

3. Chat

4. Display messages

5. Log out

6. Search for a message

7. Exit

Enter your choice:3

Choose the user you want to chat with:

1       ram

2       anbu

Enter the choice: 1

Please provide the message to be sent: Hello Ram, How are you...

=========================

1. Create a user

2. Login

3. Chat

4. Display messages
```

```
5. Log out

6. Search for a message

7. Exit

Enter your choice:4

Messages for ram

Sent:       17/11/2023, 15:35:31 - To ram: Hello Ram, How are you...

Received:     17/11/2023, 15:35:31 - From ram: Hello Ram, How are you...

==========================

1. Create a user

2. Login

3. Chat

4. Display messages

5. Log out

6. Search for a message

7. Exit

Enter your choice:6

Enter the message phrase to search: Ram

Sent:       17/11/2023, 15:35:31 - To ram: Hello Ram, How are you...

Received:     17/11/2023, 15:35:31 - From ram: Hello Ram, How are you...

==========================

1. Create a user

2. Login

3. Chat

4. Display messages

5. Log out

6. Search for a message

7. Exit

Enter your choice:5

Logged out successfully.
```

```
==========================
1. Create a user
2. Login
3. Chat
4. Display messages
5. Log out
6. Search for a message
7. Exit
Enter your choice:7
Exiting...
```