**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**

**Question 2:**

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.**