## 1 RegexCode

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import os

import re

def search_text_files(folder_path, regex_pattern):

    try:

        pattern = re.compile(regex_pattern)

    except re.error as e:

        print(f"Invalid regular expression: {e}")

        return

    print("\n--- Matching Lines ---\n")

    for filename in os.listdir(folder_path):

        if filename.endswith(".txt"):

            file_path = os.path.join(folder_path, filename)

            try:

                with open(file_path, 'r', encoding='utf-8') as file:

                    for line_number, line in enumerate(file, 1):

                        if pattern.search(line):

                            print(f"{filename} [Line {line_number}]: {line.st
rip()}")

            except Exception as e:

                print(f"Could not read file {filename}: {e}")
if __name__ == "__main__":

    folder = input("Enter the path to the folder: ").strip().strip('"')

    regex = input("Enter a regular expression to search for: ").strip()

    search_text_files(folder, regex)
```

O/P:MANOJ R, USN:1AY24AI068, SEC:O

Enter the path to the folder: C:\Users\91636\OneDrive\Desktop\New folder

Enter a regular expression to search for: panda

--- Matching Lines ---

New Text Document.txt [Line 1]: the ADJECTIVE panda walked to the NOUN and then

VERB. A nearby NOUN was unaffected by these events

Your Name.txt [Line 1]: the ADJECTIVE panda walked to the NOUN and then VERB. A nearb
y NOUN was unaffected by these events

## 2 MadLibs

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import re
def mad_libs(input_filename, output_filename):
    # Read the content of the file
    with open(input_filename, 'r') as file:
        content = file.read()
    placeholders = re.findall(r'\b(ADJECTIVE|NOUN|VERB|ADVERB)\b', content)
    for word in placeholders:
        article = "an" if word[0] in "AEIOU" else "a"
        user_input = input(f"Enter {article.lower()} {word.lower()}: ")
        # Replace only the first occurrence each time
        content = content.replace(word, user_input, 1)
    print("\nGenerated Mad Libs:\n")
    print(content)
    with open(output_filename, 'w') as file:
        file.write(content)
    print(f"\nSaved to '{output_filename}'")
input_file = input("Enter the path to your Mad Libs template file: ").strip()
output_file = "madlib_result.txt"
mad_libs(input_file, output_file)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Enter the path to your Mad Libs template file:  C:\Users\USER\Desktop\AIT\SEM
2\PYT
```

## 3 Selective Copy

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import os
import shutil
def copy_files_with_extension(source_folder, destination_folder, file_extensi
on):
    os.makedirs(destination_folder, exist_ok=True)
    file_extension = file_extension.lower()
    count = 0
    for foldername, subfolders, filenames in os.walk(source_folder):
```

```
        for filename in filenames:
            if filename.lower().endswith(file_extension):
                source_path = os.path.join(foldername, filename)
                destination_path = os.path.join(destination_folder, filename)
                if os.path.exists(destination_path):
                    base, ext = os.path.splitext(filename)
                    i = 1
                    while os.path.exists(destination_path):
                        destination_path = os.path.join(destination_folder, f
"{base}_{i}{ext}")
                        i += 1
                shutil.copy2(source_path, destination_path)
                print(f"Copied: {source_path} → {destination_path}")
                count += 1
    print(f"\nTotal files copied: {count}")
source = input("Enter the path to the source folder: ").strip()
destination = input("Enter the path to the destination folder: ").strip()
extension = input("Enter the file extension to search for (e.g., .pdf, .jpg):
").strip()
copy_files_with_extension(source, destination, extension)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Enter the path to the source folder:  C:\Users\USER\Desktop\AIT\SEM 2\PYT
Enter the path to the destination folder:  C:\Users\USER\Desktop\AIT\SEM 2\Sy
llabus
Enter the file extension to search for (e.g., .pdf, .jpg):  .pdf

Copied: C:\Users\USER\Desktop\AIT\SEM 2\PYT\Introduction to Python (YouTube @
ManojPN) Module 1.pdf → C:\Users\USER\Desktop\AIT\SEM 2\Syllabus\Introduction
to Python (YouTube @ManojPN) Module 1.pdf
Copied: C:\Users\USER\Desktop\AIT\SEM 2\PYT\Lab Manual.pdf → C:\Users\USER\De
sktop\AIT\SEM 2\Syllabus\Lab Manual.pdf

Total files copied: 2
```

## *4 Deleting Unneeded Files*

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import os
def find_large_files(folder_path, size_threshold_mb=100):
    size_threshold_bytes = size_threshold_mb * 1024 * 1024
    large_files_found = 0
    print(f"\nScanning for files larger than {size_threshold_mb}MB in:\n{os.p
ath.abspath(folder_path)}\n")
    for foldername, subfolders, filenames in os.walk(folder_path):
        for filename in filenames:
            try:
                file_path = os.path.join(foldername, filename)
                file_size = os.path.getsize(file_path)
                if file_size > size_threshold_bytes:
```

```
                    size_mb = file_size / (1024 * 1024)
                    print(f"{os.path.abspath(file_path)} - {size_mb:.2f} MB")
                    large_files_found += 1
            except (PermissionError, FileNotFoundError):
                continue
    if large_files_found == 0:
        print("No files larger than the specified size were found.")
    else:
        print(f"\nTotal large files found: {large_files_found}")
folder = input("Enter the path to the folder to scan: ").strip()
find_large_files(folder)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Enter the path to the folder to scan:  C:\Users\USER\Desktop\AIT\SEM 2\PYT


Scanning for files larger than 100MB in:
C:\Users\USER\Desktop\AIT\SEM 2\PYT

No files larger than the specified size were found.
```

## *5 Filling in the Gaps*

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import os
import re
def get_numbered_files(folder, prefix, extension):
    pattern = re.compile(rf'^{re.escape(prefix)}(\d+){re.escape(extension)}$'
)
    numbered_files = []

    for filename in os.listdir(folder):
        match = pattern.match(filename)
        if match:
            num = int(match.group(1))
            numbered_files.append((num, filename))
    return sorted(numbered_files)
def close_gaps(folder, prefix, extension):
    files = get_numbered_files(folder, prefix, extension)
    if not files:
        print("No matching files found.")
        return
    next_expected = 1
    for actual_num, filename in files:
        if actual_num != next_expected:
            new_name = f"{prefix}{str(next_expected).zfill(3)}{extension}"
            print(f"Renaming {filename} → {new_name}")
            os.rename(os.path.join(folder, filename), os.path.join(folder, ne
w_name))
        next_expected += 1
```

```
    print("Gaps closed.")
def insert_gap(folder, prefix, extension, insert_position):
    files = get_numbered_files(folder, prefix, extension)
    if not files:
        print("No matching files found.")
        return
    for num, filename in sorted(files, reverse=True):
        if num >= insert_position:
            new_num = num + 1
            new_name = f"{prefix}{str(new_num).zfill(3)}{extension}"
            print(f"Renaming {filename} → {new_name}")
            os.rename(os.path.join(folder, filename), os.path.join(folder, ne
w_name))
    print(f"Gap inserted at position {insert_position}.")
def main():
    print("Choose an operation:")
    print("1. Close gaps in numbered files")
    print("2. Insert a gap at a specific position")
    choice = input("Enter 1 or 2: ").strip()
    folder = input("Enter folder path: ").strip()
    prefix = input("Enter file prefix (e.g., spam): ").strip()
    extension = input("Enter file extension (e.g., .txt): ").strip()
    if choice == '1':
        close_gaps(folder, prefix, extension)
    elif choice == '2':
        pos = int(input("Enter position number to insert gap at (e.g., 2 for
spam002): "))
        insert_gap(folder, prefix, extension, pos)
    else:
        print("Invalid choice.")
if __name__ == "__main__":
    main()
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Choose an operation:
1. Close gaps in numbered files
2. Insert a gap at a specific position

Enter 1 or 2:  2
Enter folder path:  C:\Users\USER\Desktop\AIT\SEM 2\PYT
Enter file prefix (e.g., spam):  Lab
Enter file extension (e.g., .txt):  .pdf
Enter position number to insert gap at (e.g., 2 for spam002):  3

No matching files found.
```

## *6 Debugging Coin Toss*

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import random
guess = ''
```

```
while guess not in ('heads', 'tails'):
    print('Guess the coin toss! Enter heads or tails:')
    guess = input().lower()
toss = random.randint(0, 1)
toss_str = 'heads' if toss == 1 else 'tails'
if guess == toss_str:
    print('You got it!')
else:
    print('Nope! Guess again!')
    guess = ''
    while guess not in ('heads', 'tails'):
        guess = input().lower()
    if guess == toss_str:
        print('You got it!')
    else:
        print('Nope. You are really bad at this game.')
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Guess the coin toss! Enter heads or tails:

 Heads

Nope! Guess again!

 Tails

You got it!
```

## 7 Circle

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
class Rectangle:
    def __init__(self, corner, width, height):
        self.corner = corner
        self.width = width
        self.height = height
class Circle:
    def __init__(self, center, radius):
        self.center = center  # Point
        self.radius = radius
def distance(p1, p2):
    """Calculate Euclidean distance between two points"""
    return math.hypot(p1.x - p2.x, p1.y - p2.y)

def point_in_circle(circle, point):
    """Return True if point lies in or on the circle"""
```

```
    return distance(circle.center, point) <= circle.radius
def get_rect_corners(rect):
    """Return list of 4 corner points of the rectangle"""
    x, y = rect.corner.x, rect.corner.y
    return [
        Point(x, y),
        Point(x + rect.width, y),
        Point(x, y + rect.height),
        Point(x + rect.width, y + rect.height)
    ]
def rect_in_circle(circle, rect):
    """Return True if all corners of rect lie in or on circle"""
    return all(point_in_circle(circle, corner) for corner in get_rect_corners
(rect))
def rect_circle_overlap(circle, rect):
    """Return True if any corner of rect lies inside circle"""
    return any(point_in_circle(circle, corner) for corner in get_rect_corners
(rect))
circle = Circle(Point(150, 100), 75)
rect1 = Rectangle(Point(120, 80), 40, 30)
rect2 = Rectangle(Point(100, 50), 100, 100)
rect3 = Rectangle(Point(300, 300), 20, 20)
print("rect1 is inside circle:", rect_in_circle(circle, rect1))
print("rect2 is inside circle:", rect_in_circle(circle, rect2))
print("rect2 overlaps circle:", rect_circle_overlap(circle, rect2))
print("rect3 overlaps circle:", rect_circle_overlap(circle, rect3))
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
rect1 is inside circle: True
rect2 is inside circle: True
rect2 overlaps circle: True
rect3 overlaps circle: False
```

## 8 DateTime

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
from datetime import datetime, timedelta
def current_day_of_week():
    today = datetime.today()
    print("Today's date:", today.strftime("%Y-%m-%d"))
    print("Day of the week:", today.strftime("%A"))
def birthday_info():
    bday_str = input("Enter your birthday (YYYY-MM-DD): ")
    bday = datetime.strptime(bday_str, "%Y-%m-%d")
    now = datetime.now()
    age = now.year - bday.year
    if (now.month, now.day) < (bday.month, bday.day):
        age -= 1
    print(f"You are {age} years old.")
    next_birthday = datetime(now.year, bday.month, bday.day)
```

```
    if next_birthday < now:
        next_birthday = datetime(now.year + 1, bday.month, bday.day)
    time_left = next_birthday - now
    print(f"Time until next birthday: {time_left.days} days, "
          f"{time_left.seconds // 3600} hours, "
          f"{(time_left.seconds % 3600) // 60} minutes,"
          f"{time_left.seconds % 60} seconds.")
def double_day(birth1_str, birth2_str):
    birth1 = datetime.strptime(birth1_str, "%Y-%m-%d")
    birth2 = datetime.strptime(birth2_str, "%Y-%m-%d")
    if birth1 > birth2:
        older, younger = birth2, birth1
    else:
        older, younger = birth1, birth2
    diff = younger - older
    double_day = younger + diff
    print("Double Day (when older is twice as old):", double_day.date())
def n_times_day(birth1_str, birth2_str, n):
    birth1 = datetime.strptime(birth1_str, "%Y-%m-%d")
    birth2 = datetime.strptime(birth2_str, "%Y-%m-%d")
    if birth1 > birth2:
        older, younger = birth2, birth1
    else:
        older, younger = birth1, birth2
    diff = younger - older
    target_day = younger + diff / (n - 1)
    print(f"{n}-Times Day:", target_day.date())

# ----------- Run Example --------
if __name__ == "__main__":
    print("1. Current day of the week:")
    current_day_of_week()
    print("\n2. Birthday info:")
    birthday_info()
    print("\n3. Double Day Example:")
    double_day("1990-01-01", "2000-01-01")
    print("\n4. n-Times Day Example (n=3):")
    n_times_day("1990-01-01", "2000-01-01", 3)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
1. Current day of the week:
Today's date: 2025-06-15
Day of the week: Sunday

2. Birthday info:

Enter your birthday (YYYY-MM-DD):  2006-12-30

You are 18 years old.
Time until next birthday: 197 days, 13 hours, 21 minutes,45 seconds.
```

```
3. Double Day Example:
Double Day (when older is twice as old): 2009-12-31

4. n-Times Day Example (n=3):
3-Times Day: 2004-12-31
```

## 9 Average

```python
print("MANOJ R, USN:1AY24AI068, SEC:O")

class Time:
    def __init__(self, hours=0, minutes=0, seconds=0):
        self.hours = hours
        self.minutes = minutes
        self.seconds = seconds

    def total_seconds(self):
        """Returns total time in seconds"""
        return self.hours * 3600 + self.minutes * 60 + self.seconds

    @classmethod
    def from_seconds(cls, total_secs):
        """Creates a Time object from total seconds"""
        hours = total_secs // 3600
        remainder = total_secs % 3600
        minutes = remainder // 60
        seconds = remainder % 60
        return cls(int(hours), int(minutes), int(seconds))

    def print_time(self):
        """Nicely formats and prints time"""
        print(f"{self.hours:02d}:{self.minutes:02d}:{self.seconds:02d}")

    def __str__(self):
        return f"{self.hours:02d}:{self.minutes:02d}:{self.seconds:02d}"

def mul_time(time_obj, factor):
    """Multiplies a Time object by a number"""
    total_secs = time_obj.total_seconds() * factor
    return Time.from_seconds(total_secs)

def avg_pace(finishing_time, distance):
    """Returns average pace (time per unit distance)"""
    return mul_time(finishing_time, 1 / distance)

# ---------- Example Usage ----------
if __name__ == "__main__":
    # Suppose you ran a 10-mile race in 1 hour 30 minutes (01:30:00)
    finish_time = Time(1, 30, 0)
```

```
    distance = 10
    print("Finishing Time:", finish_time)
    pace = avg_pace(finish_time, distance)
    print("Average Pace (per mile):", pace)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O
Finishing Time: 01:30:00
Average Pace (per mile): 00:09:00

## 10 Atrribute

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
from __future__ import print_function, division
class Time:
    """Represents the time of day as seconds since midnight."""
    def __init__(self, hour=0, minute=0, second=0):
        self.seconds = hour * 3600 + minute * 60 + second
    def __str__(self):
        h, rem = divmod(self.seconds, 3600)
        m, s = divmod(rem, 60)
        return '%.2d:%.2d:%.2d' % (h, m, s)
    def print_time(self):
        print(str(self))
    def time_to_int(self):
        """Returns seconds since midnight."""
        return self.seconds
    def is_after(self, other):
        return self.seconds > other.seconds
    def __add__(self, other):
        if isinstance(other, Time):
            return self.add_time(other)
        else:
            return self.increment(other)
    def __radd__(self, other):
        return self.__add__(other)
    def add_time(self, other):
        assert self.is_valid() and other.is_valid()
        return int_to_time(self.seconds + other.seconds)
    def increment(self, seconds):
        return int_to_time(self.seconds + seconds)
    def is_valid(self):
        return 0 <= self.seconds < 24 * 3600
def int_to_time(seconds):
    """Creates a Time object from seconds since midnight."""
    return Time(0, 0, seconds)
def main():
    start = Time(9, 45, 0)
    start.print_time()
    end = start.increment(1337)
    end.print_time()
```

```
    print('Is end after start?')
    print(end.is_after(start))
    print('Using __str__')
    print(start, end)
    start = Time(9, 45)
    duration = Time(1, 35)
    print(start + duration)
    print(start + 1337)
    print(1337 + start)
    print('Example of polymorphism')
    t1 = Time(7, 43)
    t2 = Time(7, 41)
    t3 = Time(7, 37)
    total = sum([t1, t2, t3])
    print(total)
if __name__ == '__main__':
    main()
```

O/P:
```
MANOJ R, USN:1AY24AI068, SEC:O
09:45:00
10:07:17
Is end after start?
True
Using __str__
09:45:00 10:07:17
11:20:00
10:07:17
10:07:17
Example of polymorphism
23:01:00
```

## 11 Kangaroo

```
print("MANOJ R, USN:1AY24AI068, SEC:O")
class Kangaroo:
    def __init__(self):
        # Each kangaroo gets its own pouch list
        self.pouch_contents = []
    def put_in_pouch(self, item):
        self.pouch_contents.append(item)
    def __str__(self):
        contents = []
        for item in self.pouch_contents:
            if isinstance(item, Kangaroo):
                # Indent nested kangaroos
                contents.append("  " + str(item).replace("\n", "\n  "))
            else:
                contents.append(str(item))
        return "Kangaroo with pouch contents:\n" + "\n".join(contents)
# ----------- Test Code -----------
```

```
if __name__ == "__main__":
    kanga = Kangaroo()
    roo = Kangaroo()
    kanga.put_in_pouch("bottle")
    kanga.put_in_pouch("snack")
    kanga.put_in_pouch(roo)
    print("Kanga:")
    print(kanga)
    print("\nRoo:")
    print(roo)
```

O/P: MANOJ R, USN:1AY24AI068, SEC:O

```
Kanga:
Kangaroo with pouch contents:
bottle
snack
  Kangaroo with pouch contents:


Roo:
Kangaroo with pouch contents:
```