program 1

class MaxHeap:

def \_\_init\_\_(self):

self.heap = []

def insert(self, value):

self.heap.append(value)

self.\_heapify\_up(len(self.heap) - 1)

def delete(self):

if len(self.heap) == 0:

return None

if len(self.heap) == 1:

return self.heap.pop()

root = self.heap[0]

self.heap[0] = self.heap.pop()

self.\_heapify\_down(0)

return root

def get\_max(self):

if len(self.heap) == 0:

return None

return self.heap[0]

def \_heapify\_up(self, index):

parent\_index = (index - 1) // 2

if index > 0 and self.heap[index] > self.heap[parent\_index]:

self.heap[index], self.heap[parent\_index] = self.heap[parent\_index], self.heap[index]

self.\_heapify\_up(parent\_index)

def \_heapify\_down(self, index):

left\_child\_index = 2 \* index + 1

right\_child\_index = 2 \* index + 2

largest\_index = index

if left\_child\_index < len(self.heap) and self.heap[left\_child\_index] > self.heap[largest\_index]:

largest\_index = left\_child\_index

if right\_child\_index < len(self.heap) and self.heap[right\_child\_index] > self.heap[largest\_index]:

largest\_index = right\_child\_index

if largest\_index != index:

self.heap[index], self.heap[largest\_index] = self.heap[largest\_index], self.heap[index]

self.\_heapify\_down(largest\_index)

program 2:

import requests

from time import sleep

def download\_urls(urls):

results = {}

for url in urls:

success = False

attempts = 0

while not success and attempts < 3:

try:

response = requests.get(url, timeout=10)

response.raise\_for\_status()

results[url] = response.text

success = True

except requests.exceptions.HTTPError as http\_err:

print(f"HTTP error occurred for {url}: {http\_err}")

except requests.exceptions.ConnectionError as conn\_err:

print(f"Connection error occurred for {url}: {conn\_err}")

except requests.exceptions.Timeout as timeout\_err:

print(f"Timeout error occurred for {url}: {timeout\_err}")

except requests.exceptions.RequestException as req\_err:

print(f"An error occurred for {url}: {req\_err}")

attempts += 1

if not success:

sleep(1) # Optional: sleep for a second before retrying

if not success:

results[url] = None # Could not retrieve the URL content

return results

# Example usage:

urls = ["https://example.com", "https://nonexistent.url"]

contents = download\_urls(urls)

for url, content in contents.items():

if content:

print(f"Content for {url} downloaded successfully.")

else:

print(f"Failed to download content for {url}.")

program 3:

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

california = fetch\_california\_housing()

X = pd.DataFrame(california.data, columns=california.feature\_names)

y = pd.Series(california.target)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a linear regression model

model = LinearRegression()

# Train the model

model.fit(X\_train, y\_train)

# Make predictions on the testing set

y\_pred = model.predict(X\_test)

# Evaluate the model's performance

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2f}")

print(f"R² Score: {r2:.2f}")

program 4:

import pandas as pd

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

def clean\_and\_preprocess(df):

# Separate numerical and categorical columns

numerical\_cols = df.select\_dtypes(include=['int64', 'float64']).columns

categorical\_cols = df.select\_dtypes(include=['object']).columns

# Define preprocessing steps for numerical data

numerical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='mean')), # Handle missing values

('scaler', StandardScaler()) # Normalize numerical columns

])

# Define preprocessing steps for categorical data

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')), # Handle missing values

('onehot', OneHotEncoder(handle\_unknown='ignore')) # Encode categorical columns

])

# Combine preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_transformer, numerical\_cols),

('cat', categorical\_transformer, categorical\_cols)

])

# Apply preprocessing to the DataFrame

df\_clean = preprocessor.fit\_transform(df)

# Convert the processed array back to a DataFrame with appropriate column names

df\_clean = pd.DataFrame(df\_clean, columns=numerical\_cols.tolist() + preprocessor.named\_transformers\_['cat']['onehot'].get\_feature\_names\_out(categorical\_cols).tolist())

return df\_clean

# Example usage

data = {

'age': [25, 30, 35, None, 40],

'salary': [50000, 60000, 70000, 80000, None],

'city': ['New York', 'Los Angeles', 'San Francisco', None, 'Seattle']

}

df = pd.DataFrame(data)

cleaned\_df = clean\_and\_preprocess(df)

print(cleaned\_df)

program 5:

def fibonacci(n):

if n <= 0:

return "Input should be a positive integer."

elif n == 1:

return 0

elif n == 2:

return 1

else:

return fibonacci(n-1) + fibonacci(n-2)

# Example usage

n = 10

print(f"The {n}th Fibonacci number is: {fibonacci(n)}")

program 6:

def divide\_numbers(a, b):

try:

result = a / b

return result

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

# Example usage

numerator = 10

denominator = 0

print(f"Result of division: {divide\_numbers(numerator, denominator)}")

program 7:

import time

import logging

# Configure logging

logging.basicConfig(level=logging.INFO)

def timing\_decorator(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

execution\_time = end\_time - start\_time

logging.info(f"Execution time of {func.\_\_name\_\_}: {execution\_time:.4f} seconds")

return result

return wrapper

@timing\_decorator

def computationally\_expensive\_task(n):

return sum(i \* i for i in range(n))

# Example usage

result = computationally\_expensive\_task(10\*\*6)

print(f"Result: {result}")

program 8:

def arithmetic\_operation(a, b, operator):

if operator == '+':

return a + b

elif operator == '-':

return a - b

elif operator == '\*':

return a \* b

elif operator == '/':

try:

return a / b

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

else:

return "Error: Invalid operator. Please use one of the following: +, -, \*, /"

# Example usage

num1 = 10

num2 = 5

print(f"{num1} + {num2} = {arithmetic\_operation(num1, num2, '+')}")

print(f"{num1} - {num2} = {arithmetic\_operation(num1, num2, '-')}")

print(f"{num1} \* {num2} = {arithmetic\_operation(num1, num2, '\*')}")

print(f"{num1} / {num2} = {arithmetic\_operation(num1, num2, '/')}")

print(f"{num1} / 0 = {arithmetic\_operation(num1, 0, '/')}")

print(f"{num1} & {num2} = {arithmetic\_operation(num1, num2, '&')}")

program 9:

import random

import string

def generate\_random\_password(length=12):

if length < 4: # Ensure the password has at least one character from each category

return "Error: Password length should be at least 4."

# Define character sets

uppercase\_letters = string.ascii\_uppercase

lowercase\_letters = string.ascii\_lowercase

digits = string.digits

special\_characters = string.punctuation

# Ensure the password has at least one character from each category

password = [

random.choice(uppercase\_letters),

random.choice(lowercase\_letters),

random.choice(digits),

random.choice(special\_characters)

]

# Fill the rest of the password length with a mix of all character sets

all\_characters = uppercase\_letters + lowercase\_letters + digits + special\_characters

password += random.choices(all\_characters, k=length-4)

# Shuffle the list to ensure randomness

random.shuffle(password)

# Convert the list to a string

return ''.join(password)

# Example usage

password = generate\_random\_password(12)

print(f"Generated password: {password}")

program 10:

def transpose\_matrix(matrix):

# Check if the input is a valid 2D list

if not matrix or not isinstance(matrix, list) or not all(isinstance(row, list) for row in matrix):

return "Error: Invalid input. Please provide a 2D list (matrix)."

# Transpose the matrix

transposed\_matrix = [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

return transposed\_matrix

# Example usage

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

transposed = transpose\_matrix(matrix)

print("Original matrix:")

for row in matrix:

print(row)

print("\nTransposed matrix:")

for row in transposed:

print(row)