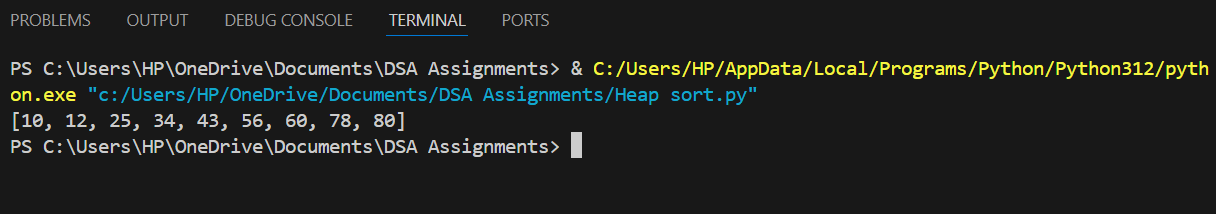
Programming Assignment:

1. Implement a Python class MaxHeap that supports the following operations: insert, delete, and get\_max. Ensure the operations maintain the properties of a max-heap.
2. class Heap:
3. def \_\_init\_\_(self):
4. self.heap=[]
6. def Createheap(self, list1):
7. for e in list1:
8. self.insert(e)
10. def insert(self, e):
11. index = len(self.heap)
12. parentIndex = (index-1)//2
13. while index > 0 and self.heap[parentIndex] < e:
14. if index == len(self.heap):
15. self.heap.append(self.heap[parentIndex])
16. else:
17. self.heap[index] = self.heap[parentIndex]
19. index = parentIndex
20. parentIndex = (index-1)//2
21. if index == len(self.heap):
22. self.heap.append(e)
23. else:
24. self.heap[index] = e
26. def top(self):
27. if len(self.heap) == 0:
28. raise EmptyHeapException()
29. return self.heap[0]
31. def delete(self):
32. if len(self.heap) == 0:
33. raise EmptyHeapException()
34. if len(self.heap) == 1:
35. return self.heap.pop()
36. max\_value = self.heap[0]
37. temp = self.heap.pop()
38. index = 0
39. leftChildIndex = 2\*index + 1
40. rightChildIndex = 2\*index + 2
41. while leftChildIndex < len(self.heap):
42. if rightChildIndex < len(self.heap):
43. if self.heap[leftChildIndex] > self.heap[rightChildIndex]:
44. if self.heap[leftChildIndex] > temp:
45. self.heap[index] = self.heap[leftChildIndex]
46. index = leftChildIndex
47. else:
48. break
49. else:
50. if self.heap[rightChildIndex] > temp:
51. self.heap[index] = self.heap[rightChildIndex]
52. index = rightChildIndex
53. else:
54. break
55. else: #No Right Child
56. if self.heap[leftChildIndex] > temp:
57. self.heap[index] = self.heap[leftChildIndex]
58. index = leftChildIndex
59. else:
60. break
61. leftChildIndex = 2\*index+1
62. rightChildIndex = 2\*index+2
63. self.heap[index] = temp
64. return max\_value
65. def heapsort(self, list1):
66. self.Createheap(list1)
67. list2 = []
68. try:
69. while True:
70. list2.insert(0, self.delete())
71. except EmptyHeapException:
72. pass
73. return list2
74. class EmptyHeapException(Exception):
75. def \_\_init\_\_(self, msg = "Empty Heap"):
76. self.msg = msg
77. def \_\_str\_\_(self):
78. return self.msg
79. list1 = [34, 56, 12, 78, 43, 25, 10, 80, 60]
80. h = Heap()
81. list1 = h.heapsort(list1)
82. print(list1)

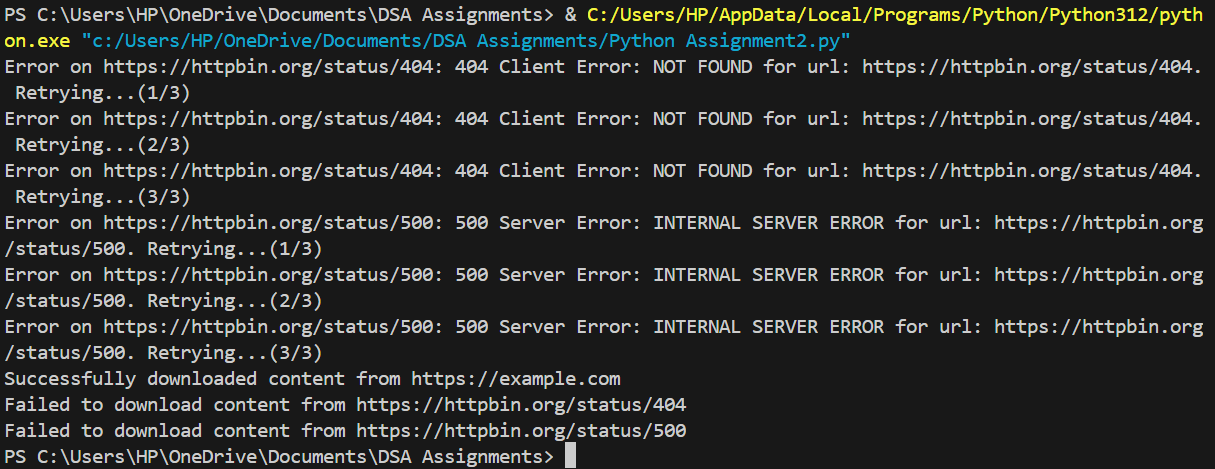
Output:



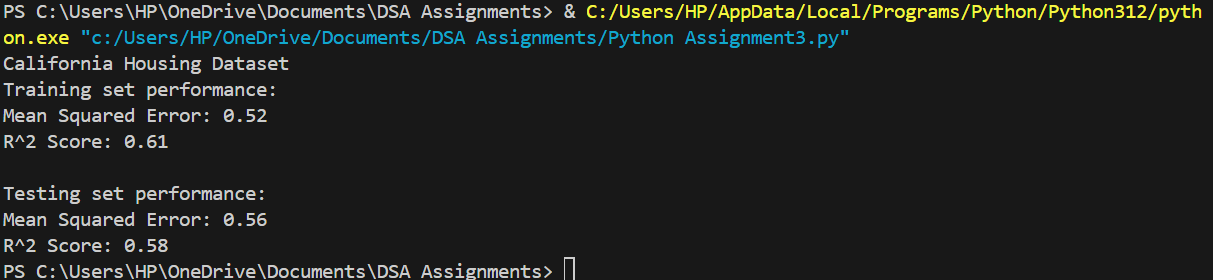
2.Write a Python function that takes a list of URLs, attempts to download their content, and retries up to 3 times if an error occurs. Use appropriate error handling to manage different types of exceptions.

1. import requests
2. from time import sleep
3. def download\_conten(urls, max\_retries = 3, timeout = 5):
4. result = {}
5. for url in urls:
6. attempt = 0
7. success = False
8. while attempt < max\_retries and not success:
9. try:
10. response = requests.get(url, timeout = timeout)
11. response.raise\_for\_status()
12. result[url] = response.content
13. success = True
14. except requests.exceptions.Timeout:
15. print(f"Timeout error on {url}. Retrying...({attempt+1}/{max\_retries})")
16. except requests.exceptions.TooManyRedirects:
17. print(f"Too many redirects on {url}. Skipping.")
18. break
19. except requests.exceptions.RequestException as e:
20. print(f"Error on {url}: {e}. Retrying...({attempt+1}/{max\_retries})")
21. attempt += 1
22. if not success and attempt < max\_retries:
23. sleep(1)
24. if not success:
25. result[url] = None
26. return result
27. urls = [
28. "https://example.com",
29. "https://httpbin.org/status/404",
30. "https://httpbin.org/status/500"
31. ]
32. content = download\_conten(urls)
33. for url, data in content.items():
34. if data:
35. print(f"Successfully downloaded content from {url}")
36. else:
37. print(f"Failed to download content from {url}")

Output:



1. Write a Python script that trains a simple linear regression model using scikit-learn. Use a dataset of your choice, split it into training and testing sets, and evaluate the model's performance.
2. import numpy as np
3. import pandas as pd
4. from sklearn.datasets import fetch\_california\_housing
5. from sklearn.model\_selection import train\_test\_split
6. from sklearn.linear\_model import LinearRegression
7. from sklearn.metrics import mean\_squared\_error, r2\_score
8. # Load the California Housing dataset
9. california = fetch\_california\_housing()
10. X = pd.DataFrame(california.data, columns=california.feature\_names)
11. y = pd.Series(california.target)
12. # Split the data into training and testing sets
13. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
14. # Create the linear regression model
15. model = LinearRegression()
16. # Train the model
17. model.fit(X\_train, y\_train)
18. # Make predictions
19. y\_train\_pred = model.predict(X\_train)
20. y\_test\_pred = model.predict(X\_test)
21. # Evaluate the model
22. train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)
23. test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)
24. train\_r2 = r2\_score(y\_train, y\_train\_pred)
25. test\_r2 = r2\_score(y\_test, y\_test\_pred)
26. print("California Housing Dataset")
27. print("Training set performance:")
28. print(f"Mean Squared Error: {train\_mse:.2f}")
29. print(f"R^2 Score: {train\_r2:.2f}")
30. print("\nTesting set performance:")
31. print(f"Mean Squared Error: {test\_mse:.2f}")
32. print(f"R^2 Score: {test\_r2:.2f}")

Output:

1. Using pandas, write a Python function to clean and preprocess a given DataFrame, which involves handling missing values, normalizing numerical columns, and encoding categorical columns.

import numpy as np

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 preprocess\_dataframe(df):

    # Separate numerical and categorical columns

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

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

    # Create a pipeline for numerical columns

    numerical\_pipeline = Pipeline(steps=[

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

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

    ])

    # Create a pipeline for categorical columns

    categorical\_pipeline = Pipeline(steps=[

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

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

    ])

    # Combine the numerical and categorical pipelines

    preprocessor = ColumnTransformer(transformers=[

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

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

    ])

    # Fit and transform the data

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

    # Get feature names for the transformed columns

    num\_feature\_names = numerical\_cols

    cat\_feature\_names = preprocessor.named\_transformers\_['cat']['onehot'].get\_feature\_names\_out(categorical\_cols)

    feature\_names = list(num\_feature\_names) + list(cat\_feature\_names)

    # Convert the transformed data back to a DataFrame

    df\_preprocessed = pd.DataFrame(df\_preprocessed, columns=feature\_names)

    return df\_preprocessed

# Example usage

data = {

    'age': [25, 30, 35, np.nan, 40],

    'salary': [50000, 60000, 80000, 70000, np.nan],

    'gender': ['Male', 'Female', np.nan, 'Female', 'Male'],

    'city': ['New York', 'San Francisco', 'Los Angeles', 'New York', np.nan]

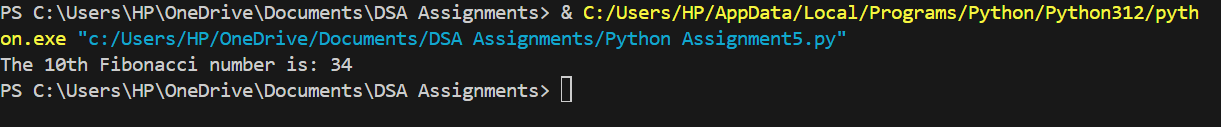
}

df = pd.DataFrame(data)

df\_cleaned = preprocess\_dataframe(df)

print(df\_cleaned)

1. Write a Python function to compute the nth Fibonacci number using recursion.
2. def fibonacci(n):
3. if n<= 0:
4. raise ValueError("Input should be a positive integer")
5. elif n==1:
6. return 0
7. elif n==2:
8. return 1
9. else:
10. return fibonacci(n-1)+fibonacci(n-2)
11. n=10
12. result = fibonacci(n)
13. print(f"The {n}th Fibonacci number is: {result}")

Output:

6. Write a Python function that divides two numbers and handles the case where the divisor is zero by returning a custom error message.

def divide\_numbers(dividend, divisor):

    try:

        result = dividend/divisor

        return result

    except ZeroDivisionError:

        return "Error: Division by zero is not allowed"

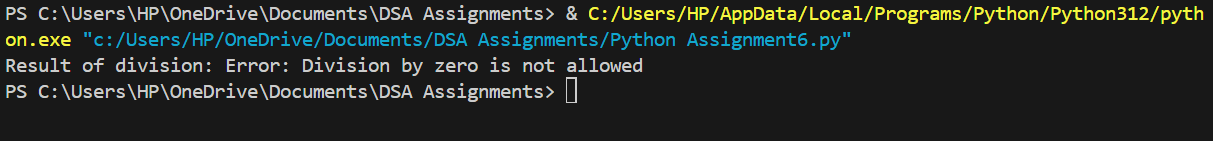
numerator = 10

denominator = 0

result = divide\_numbers(numerator, denominator)

print(f"Result of division: {result}")

Output:



7. Write a Python decorator that measures the execution time of a function and logs it. Apply this decorator to a function that performs a computationally expensive task.

import time

import logging

from functools import wraps

logging.basicConfig(level=logging.INFO)

def measure\_execution\_time(func):

    @wraps(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"Function '{func.\_\_name\_\_}' executed in {execution\_time:.4f} seconds")

        return result

    return wrapper

@measure\_execution\_time

def fibonacci(n):

    if n <= 0:

        raise ValueError("Input should be a positive integer")

    elif n == 1:

        return 0

    elif n == 2:

        return 1

    else:

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

n = 35

result = fibonacci(n)

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

8. Write a Python function that takes two numbers and an operator (as a string) and performs the corresponding arithmetic operation (addition, subtraction, multiplication, or division).

def arithmetic\_operation(num1, num2, operator):

    if operator == '+':

        return num1 + num2

    elif operator == '-':

        return num1 - num2

    elif operator == '\*':

        return num1 \* num2

    elif operator == '/':

        if num2 == 0:

            return "Error: Division by zero!"

        else:

            return num1 / num2

    else:

        return "Error: Invalid operator!"

result = arithmetic\_operation(10, 5, '+')

print("Result:", result)  # Output: Result: 15

result = arithmetic\_operation(10, 5, '-')

print("Result:", result)  # Output: Result: 5

result = arithmetic\_operation(10, 5, '\*')

print("Result:", result)  # Output: Result: 50

result = arithmetic\_operation(10, 5, '/')

print("Result:", result)  # Output: Result: 2.0

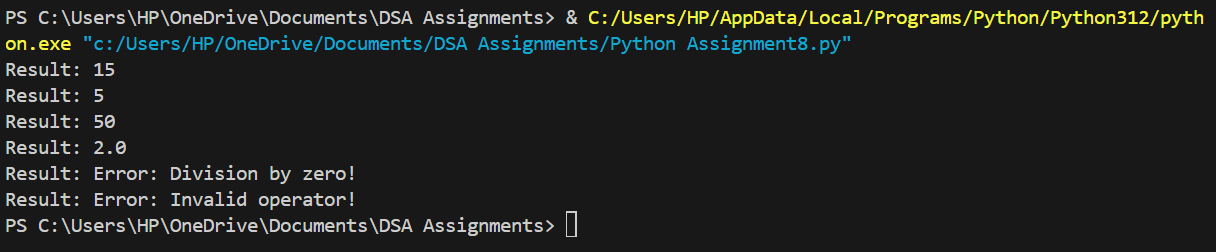
result = arithmetic\_operation(10, 0, '/')

print("Result:", result)  # Output: Result: Error: Division by zero!

result = arithmetic\_operation(10, 5, '^')

print("Result:", result)  # Output: Result: Error: Invalid operator!

Output:



9.Write a Python function that generates a random password. The password should contain a mix of uppercase letters, lowercase letters, digits, and special characters.

import random

import string

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

    # Define the character sets to choose from

    lowercase\_letters = string.ascii\_lowercase

    uppercase\_letters = string.ascii\_uppercase

    digits = string.digits

    special\_characters = string.punctuation  # includes punctuation and symbols

    # Combine all character sets

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

    # Generate a random password with the specified length

    password = ''.join(random.choice(all\_characters) for \_ in range(length))

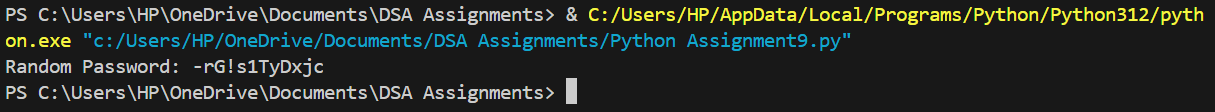
    return password

# Example usage:

password = generate\_random\_password()

print("Random Password:", password)

Output:



10. Write a Python function that takes a 2D list (matrix) and returns its transpose.

def transpose\_matrix(matrix):

    # Calculate dimensions of the matrix

    rows = len(matrix)

    cols = len(matrix[0]) if matrix else 0  # Number of columns in the first row

    # Initialize an empty transpose matrix

    transpose = []

    # Iterate through columns and rows to build the transpose

    for col in range(cols):

        transpose\_row = []

        for row in range(rows):

            transpose\_row.append(matrix[row][col])

        transpose.append(transpose\_row)

    return transpose

matrix = [

    [1, 2, 3],

    [4, 5, 6],

    [7, 8, 9]

]

transposed\_matrix = transpose\_matrix(matrix)

for row in transposed\_matrix:

    print(row)

Output:

