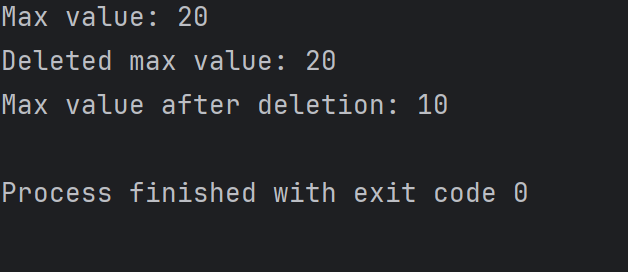
1.Implement a Python class MaxHeap that supports the following operations: insert, delete, and get\_max. Ensure the operations maintain the properties of a max-heap.

Solution:

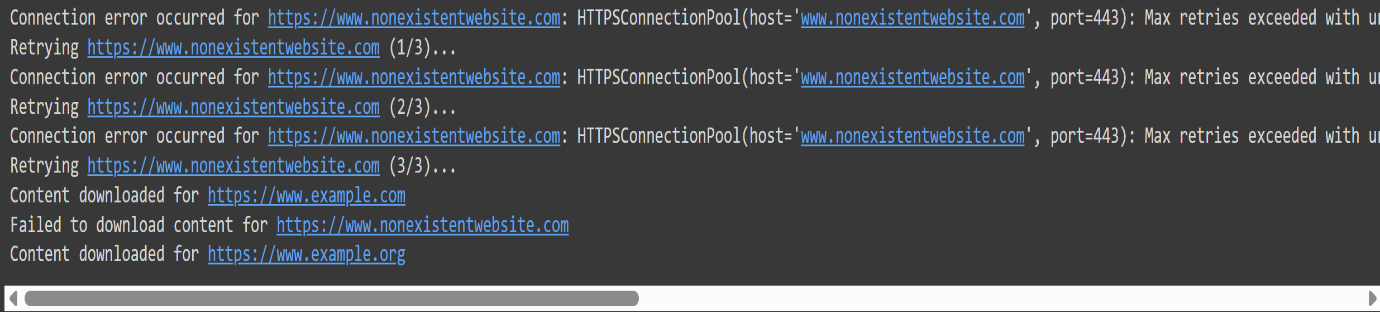
class MaxHeap:  
 def \_\_init\_\_(self):  
 self.heap = []  
  
 def insert(self, val):  
 self.heap.append(val)  
 self.\_heapify\_up(len(self.heap) - 1)  
  
 def delete(self):  
 if len(self.heap) == 0:  
 raise IndexError("delete from an empty heap")  
 if len(self.heap) == 1:  
 return self.heap.pop()  
 max\_val = self.heap[0]  
 self.heap[0] = self.heap.pop()  
 self.\_heapify\_down(0)  
 return max\_val  
  
 def get\_max(self):  
 if len(self.heap) == 0:  
 raise IndexError("get from an empty heap")  
 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  
  
 if left\_child\_index < len(self.heap) and self.heap[left\_child\_index] > self.heap[largest]:  
 largest = left\_child\_index  
 if right\_child\_index < len(self.heap) and self.heap[right\_child\_index] > self.heap[largest]:  
 largest = right\_child\_index  
 if largest != index:  
 self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]  
 self.\_heapify\_down(largest)  
max\_heap = MaxHeap()  
max\_heap.insert(10)  
max\_heap.insert(20)  
max\_heap.insert  
print("Max value:", max\_heap.get\_max()) # Output: Max value: 20  
print("Deleted max value:", max\_heap.delete()) # Output: Deleted max value: 20  
print("Max value after deletion:", max\_heap.get\_max()) # Output: Max value after deletion: 10

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.

Solution:

import requests  
from time import sleep  
def download\_content(urls):  
 results = {}  
  
 for url in urls:  
 attempts = 0  
 success = False  
 while attempts < 3 and not success:  
 try:  
 response = requests.get(url)  
 response.raise\_for\_status() # Raise an HTTPError for bad responses  
 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}")  
 finally:  
 attempts += 1  
 if not success:  
 print(f"Retrying {url} ({attempts}/3)...")  
 sleep(2) # Wait for 2 seconds before retrying  
  
 if not success:  
 results[url] = None  
  
 return results  
urls = [  
 "https://www.example.com",  
 "https://www.nonexistentwebsite.com",  
 "https://www.example.org"  
]  
  
downloaded\_content = download\_content(urls)  
for url, content in downloaded\_content.items():  
 if content is not None:  
 print(f"Content downloaded for {url}")  
 else:  
 print(f"Failed to download content for {url}")

3. 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.

4. 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.

Solution:

import pandas as pd

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

def clean\_and\_preprocess(df):

    # Separate features into numerical and categorical columns

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

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

    # Preprocessing for numerical data

    numerical\_transformer = Pipeline(steps=[

        ('imputer', SimpleImputer(strategy='mean')),

        ('scaler', StandardScaler())

    ])

# Preprocessing for categorical data

    categorical\_transformer = Pipeline(steps=[

        ('imputer', SimpleImputer(strategy='most\_frequent')),

        ('onehot', OneHotEncoder(handle\_unknown='ignore'))

    ])

# Bundle preprocessing for numerical and categorical data

    preprocessor = ColumnTransformer(

        transformers=[

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

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

        ])

    # Apply transformations

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

   # Convert the result back to a DataFrame for ease of use

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

    return df\_cleaned

data = {

    'age': [25, 30, np.nan, 40, 50],

    'salary': [50000, 60000, np.nan, 80000, 90000],

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

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

}

df = pd.DataFrame(data)

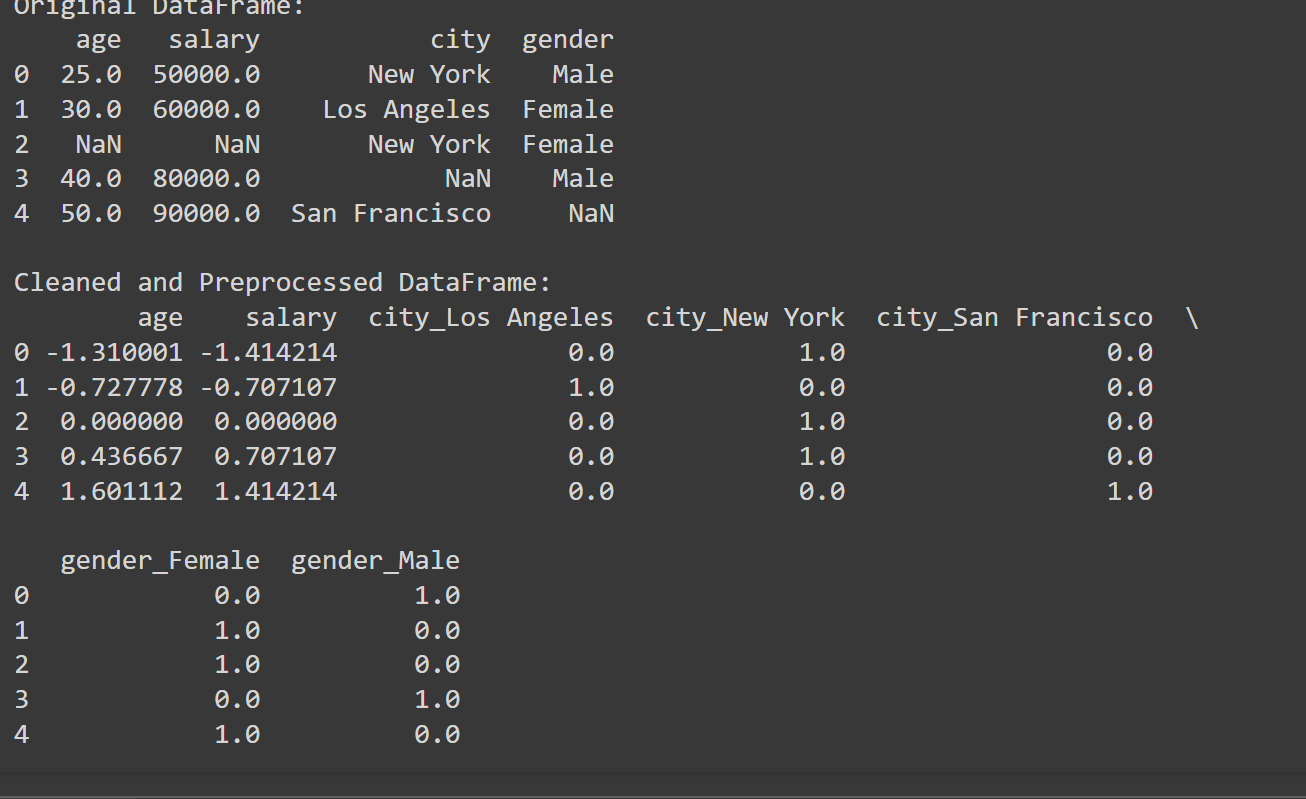
print("Original DataFrame:")

print(df)

df\_cleaned = clean\_and\_preprocess(df)

print("\nCleaned and Preprocessed DataFrame:")

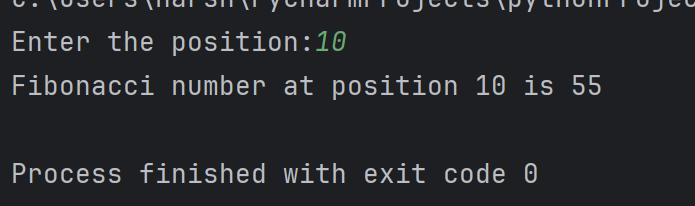
print(df\_cleaned)



5.Write a Python function to compute the nth Fibonacci number using recursion.

Solution:

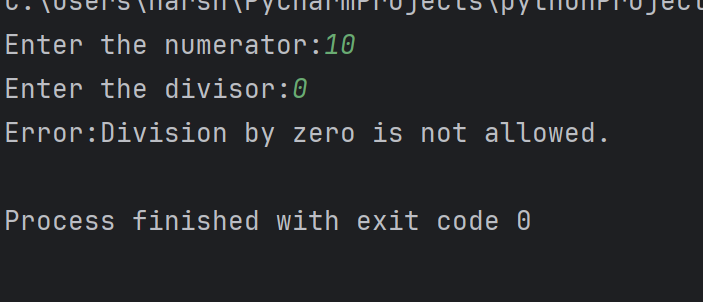
def fibonacci(n):  
 if n<0:  
 raise ValueError("Input should be non-negative integer")  
 elif n==0:  
 return 0  
 elif n==1:  
 return 1  
 else:  
 return fibonacci(n-1) + fibonacci(n-2)  
n=int(input("Enter the position:"))  
print(f"Fibonacci number at position {n} is {fibonacci(n)}")

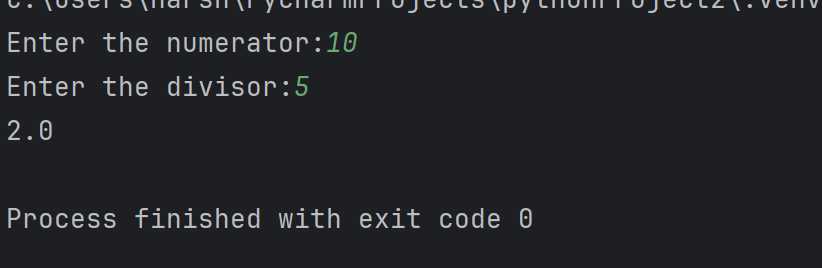


6.Write a Python function that divides two numbers and handles the case where the divisor is zero by returning a custom error message.

Solution:

def division(numerator,divisor):  
 try:  
 result=numerator/divisor  
 return result  
 except ZeroDivisionError:  
 return "Error:Division by zero is not allowed."  
numerator=float(input("Enter the numerator:"))  
divisor=float(input("Enter the divisor:"))  
result=division(numerator,divisor)  
print(result)

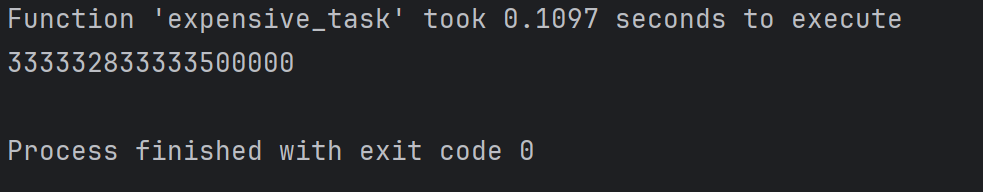




7.Write a Python decorator that measures the execution time of a function and logs it. Apply this decorator to a function that performs a computationally expensive task.

Solution:

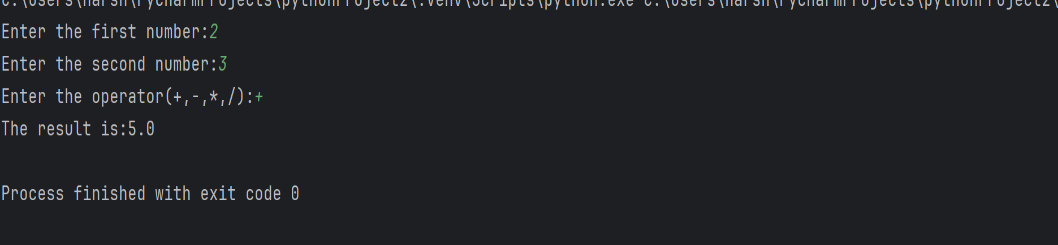
import time  
def log\_time(func):  
 def wrapper(\*args, \*\*kwargs):  
 try:  
 start\_time = time.time() # Record the start time  
 result = func(\*args, \*\*kwargs) # Execute the function and get result  
 end\_time = time.time() # Record the end  
 elapsed\_time = end\_time - start\_time # Calculate elapsed time  
 print(f"Function '{func.\_\_name\_\_}' took {elapsed\_time:.4f} seconds to execute")  
 return result  
 except Exception as e:  
 print(f"An error occurred: {e}")  
 raise e  
 return wrapper  
# Use the decorator on the function  
@log\_time  
def expensive\_task(n):  
 return sum(i\*\*2 for i in range(n))  
try:  
 # Testing the function with decorator  
 result = expensive\_task(1000000)  
 print(result)  
except Exception as e:  
 print(f"An error occurred: {e}")

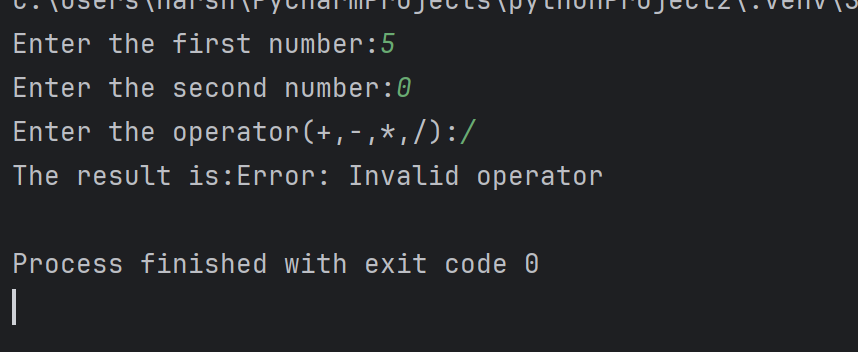


8.Write a Python function that takes two numbers and an operator (as a string) and performs the corresponding arithmetic operation (addition, subtraction, multiplication, or division).

Solution:

def calculate(n1,n2,operator):  
 if operator=='+':  
 return n1+n2  
 elif operator=='-':  
 return n1-n2  
 elif operator=='\*':  
 return n1\*n2  
 elif operator=='/':  
 if n2==0:  
 return "Error: Invalid operator"  
 return n1/n2  
 else:  
 return "Error: Invalid operator"  
n1=float(input("Enter the first number:"))  
n2=float(input("Enter the second number:"))  
operator=input("Enter the operator(+,-,\*,/):")  
result=calculate(n1,n2,operator)  
print(f"The result is:{result}")

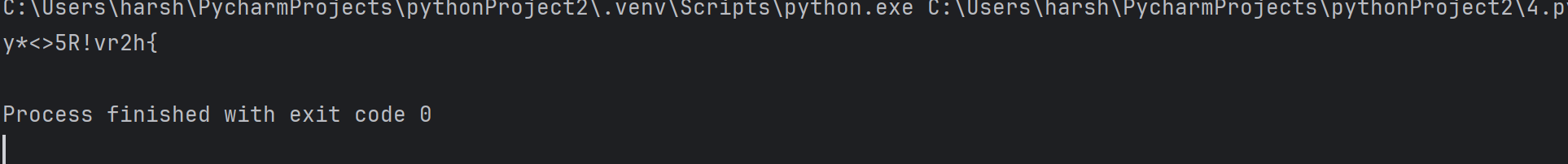




9.Write a Python function that generates a random password. The password should contain a mix of uppercase letters, lowercase letters, digits, and special characters.

Solution:

import random  
import string  
def random\_password(length=12):  
 if length<4:  
 raise ValueError("Password length should atleast 4")  
 #Characters to be used in password  
 characters=string.ascii\_letters+string.digits+string.punctuation  
 #password should have atleast one of each character type  
 password=[  
 random.choice(string.ascii\_uppercase),  
 random.choice(string.ascii\_lowercase),  
 random.choice(string.digits),  
 random.choice(string.punctuation)  
 ]  
 #Fill the password with random choices from all characters  
 password+=[random.choice(characters)for \_ in range(length-4)]  
 #Shuffle the password to mix all character types  
 random.shuffle(password)  
 #convert list to string  
 return''.join(password)  
print(random\_password(12))



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

Solution:

def transpose(matrix):  
 rows=len(matrix)  
 cols=len(matrix[0])  
 t=[[None]\*rows for \_ in range(cols)] #create a new matrix with rows&cols and initialize with None  
 for i in range(rows):  
 for j in range(cols):  
 t[j][i]=matrix[i][j]  
 return t  
def get\_input():  
 rows=int(input("Enter the number of rows:"))  
 cols=int(input("Enter the number of cols:"))  
 matrix=[] #initialize an empty matrix  
 for \_ in range(rows):  
 row=list(map(int,input().split()))  
 if len(row)!=cols:  
 print(f"Each row must have {cols} elements.")#Check the row has correct number of elements or not  
 return None  
 matrix.append(row)#add the row to the matrix  
 return matrix  
matrix=get\_input() #call the function  
if matrix: #if the matrix is valid  
 t=transpose(matrix) #Transpose the matrix  
 print("Transposed matrix is:")  
 for row in t:  
 print(row)

