GlobalTrend\_Programming\_Assessment

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.

class MaxHeap:

    def \_\_init\_\_(self):

        self.heap = []

    def insert(self, data):

        self.heap.append(data)

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

    def delete(self, data):

        try:

            index = self.heap.index(data)

            self.\_swap(index, len(self.heap) - 1)

            removed\_value = self.heap.pop()

            if index < len(self.heap):

                self.\_heapify\_down(index)

                self.\_heapify\_up(index)

            return removed\_value

        except ValueError:

            return None

    def get\_max(self):

        if not self.heap:

            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.\_swap(index, parent\_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.\_swap(index, largest)

            self.\_heapify\_down(largest)

    def \_swap(self, i, j):

        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def \_\_str\_\_(self):

        return str(self.heap)

max\_heap = MaxHeap()

max\_heap.insert(92)

max\_heap.insert(116)

max\_heap.insert(5)

max\_heap.insert(63)

max\_heap.insert(3)

print("The Max-Heap after multiple insertions:", max\_heap)

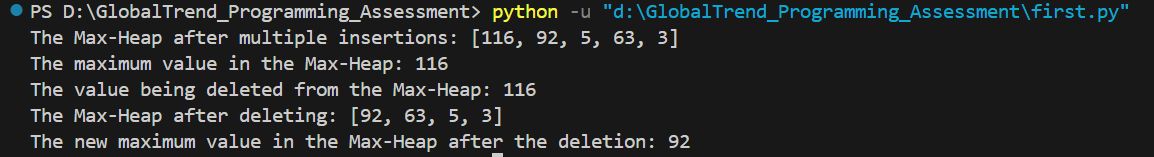
print("The maximum value in the Max-Heap:", max\_heap.get\_max())

print("The value being deleted from the Max-Heap:", max\_heap.delete(116))

print("The Max-Heap after deleting:", max\_heap)

print("The new maximum value in the Max-Heap after the deletion:", max\_heap.get\_max())

**Output:**

****

1. 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.

import requests

from time import sleep

def download\_urls(urls):

    results = {}

    max\_retries = 3

    backoff\_factor = 1

    for url in urls:

        attempt = 0

        success = False

        while attempt < max\_retries and not success:

            try:

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

                response.raise\_for\_status()

                results[url] = response.text

                success = True

            except requests.exceptions.Timeout:

                print(f"Timeout occurred while trying to access the URL: {url}. Retrying attempt {attempt + 1} of {max\_retries}.")

            except requests.exceptions.RequestException as e:

                print(f"Request failed for URL: {url}. Encountered error: {e}. Retrying attempt {attempt + 1} of {max\_retries}.")

            except Exception as e:

                print(f"An unexpected error occurred while accessing URL: {url}. Error details: {e}. Retrying attempt {attempt + 1} of {max\_retries}.")

            attempt += 1

            if not success and attempt < max\_retries:

                sleep(backoff\_factor \* attempt)

        if not success:

            results[url] = None

    return results

urls = [

    "https://www.globaltrend.co.in",

    "https://fakewebsite.in"

]

downloaded\_content = download\_urls(urls)

for url, content in downloaded\_content.items():

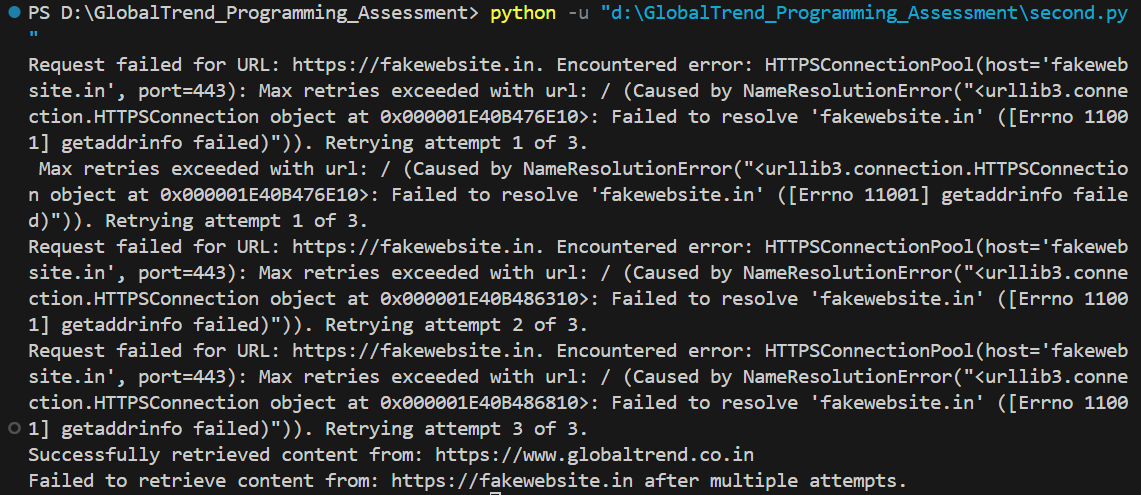
    if content:

        print(f"Successfully retrieved content from: {url}")

    else:

        print(f"Failed to retrieve content from: {url} after multiple attempts.")

**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.

import numpy as np

import pandas as pd

from sklearn.datasets import make\_regression

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

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

X, y = make\_regression(n\_samples=1000, n\_features=1, noise=10, random\_state=42)

X = pd.DataFrame(X, columns=['Feature'])

y = pd.Series(y, name='Target')

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

model = LinearRegression()

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

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

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

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

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

print(f"R-squared (R2) Score: {r2}")

plt.figure(figsize=(10, 6))

plt.scatter(X\_test, y\_test, alpha=0.7, edgecolors='b', s=100, label="Actual values")

plt.scatter(X\_test, y\_pred, alpha=0.7, edgecolors='r', s=100, label="Predicted values")

plt.plot(X\_test, y\_pred, color='red', lw=2, label="Regression Line")

plt.xlabel("Feature")

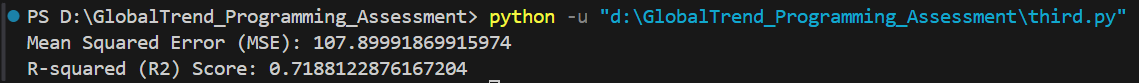
plt.ylabel("Target")

plt.title("Actual vs Predicted Values")

plt.legend()

plt.show()

**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):

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

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

    numerical\_transformer = Pipeline(steps=[

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

        ('scaler', StandardScaler())

    ])

    categorical\_transformer = Pipeline(steps=[

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

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

    ])

    preprocessor = ColumnTransformer(

        transformers=[

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

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

        ])

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

    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)

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

    return df\_cleaned

data = {

    'age': [25, 30, 35, np.nan, 40],

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

    'city': ['Mumbai', 'Delhi', 'Mumbai', 'Bangalore', 'Delhi']

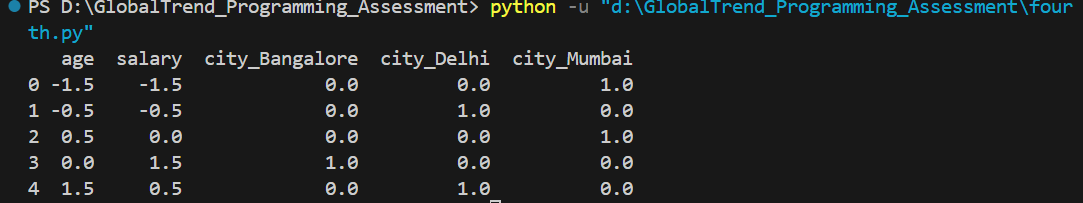
}

df = pd.DataFrame(data)

df\_cleaned = preprocess\_dataframe(df)

print(df\_cleaned)

**Output:**

****

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

def fibonacci(n):

    if n <= 0:

        return "Input should be a positive integer"

    elif n == 1:

        return 0

    elif n == 2:

        return 1

    else:

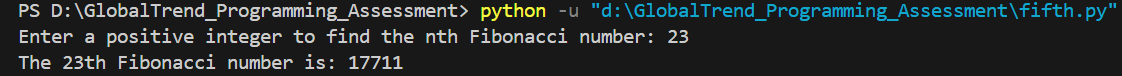
        return fibonacci(n - 1) + fibonacci(n - 2)

n = int(input("Enter a positive integer to find the nth Fibonacci number: "))

result = fibonacci(n)

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

**Output:**

****

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

def safe\_divide(numerator, denominator):

    try:

        result = numerator / denominator

        return result

    except ZeroDivisionError:

        return "Cannot divide by zero. Please provide a non-zero denominator."

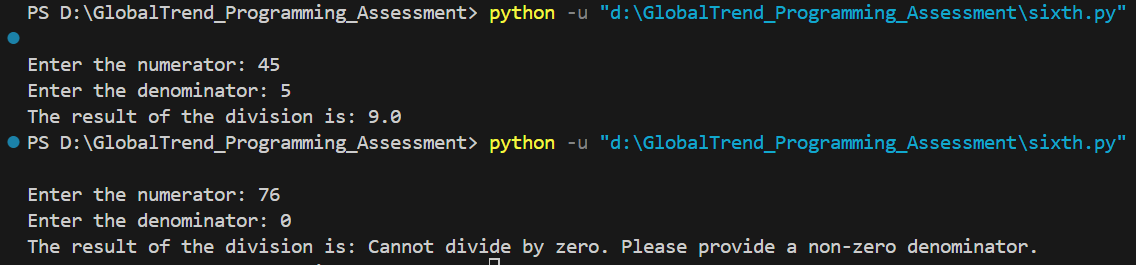
num = float(input("Enter the numerator: "))

den = float(input("Enter the denominator: "))

result = safe\_divide(num, den)

print(f"The result of the division is: {result}")

**Output:**

****

1. 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

def measure\_execution\_time(func):

    def wrapper(\*args, \*\*kwargs):

        start\_time = time.time()

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

        end\_time = time.time()

        execution\_time = end\_time - start\_time

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

        return result

    return wrapper

@measure\_execution\_time

def calculate\_square(num):

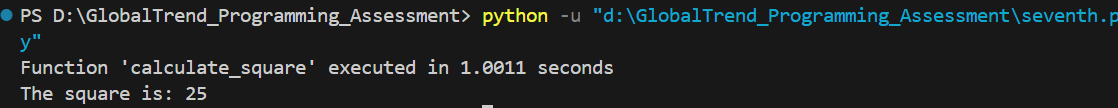
    time.sleep(1)

    return num \*\* 2

result = calculate\_square(5)

print(f"The square is: {result}")

**Output:**



1. 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 operations(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 is not allowed."

        else:

            return num1 / num2

    else:

        return "Error: Invalid operator"

num1 = float(input("Enter the first number: "))

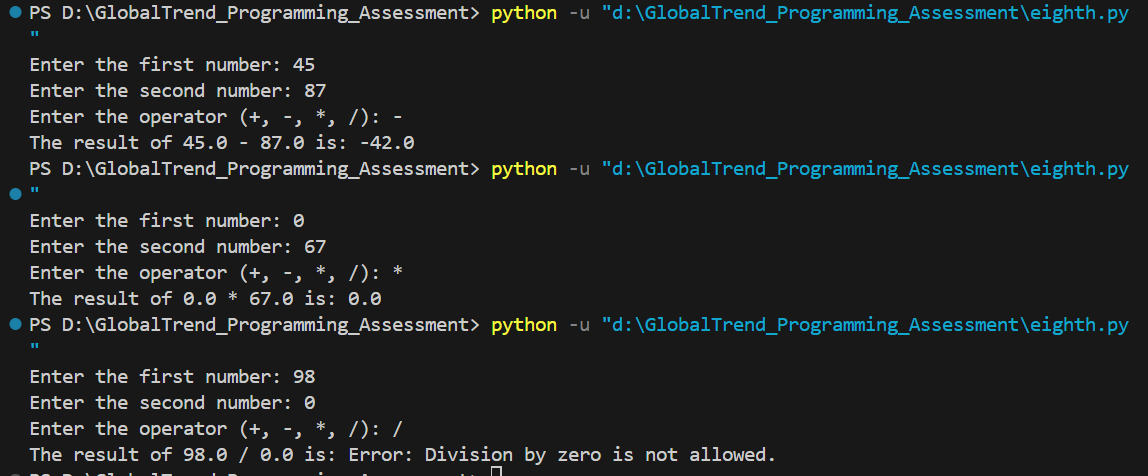
num2 = float(input("Enter the second number: "))

op = input("Enter the operator (+, -, \*, /): ")

result = operations(num1, num2, op)

print(f"The result of {num1} {op} {num2} is: {result}")

**Output:**

****

1. 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):

    characters = string.ascii\_letters + string.digits + string.punctuation

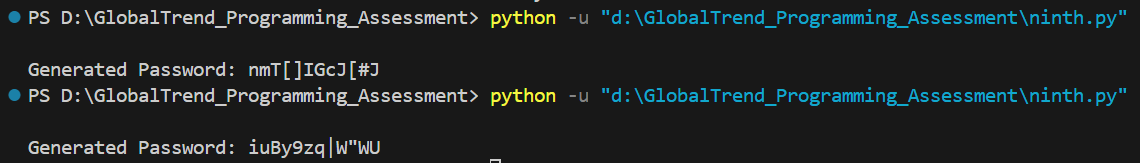
    password = ''.join(random.choice(characters) for \_ in range(length))

    return password

password = generate\_random\_password()

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

**Output:**



1. Write a Python function that takes a 2D list (matrix) and returns its transpose.
2. def transpose\_matrix(matrix):
3. rows = len(matrix)
4. cols = len(matrix[0])
5. transpose = [[0 for \_ in range(rows)] for \_ in range(cols)]
6. for i in range(rows):
7. for j in range(cols):
8. transpose[j][i] = matrix[i][j]
9. return transpose
10. matrix = [
11. [1, 2, 3],
12. [4, 5, 6],
13. [7, 8, 9]
14. ]
15. transposed\_matrix = transpose\_matrix(matrix)
16. for row in transposed\_matrix:
17. print(row)

**Output:**

****