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, value):

self.heap.append(value)

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

def delete(self):

if len(self.heap) == 0:

raise IndexError("delete from empty heap")

max\_value = self.heap[0]

self.heap[0] = self.heap[-1]

self.heap.pop()

if self.heap:

self.\_heapify\_down(0)

return max\_value

def get\_max(self):

if len(self.heap) == 0:

raise IndexError("get\_max from 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)

# Example usage

max\_heap = MaxHeap()

max\_heap.insert(10)

max\_heap.insert(20)

max\_heap.insert(15)

max\_heap.insert(30)

max\_heap.insert(40)

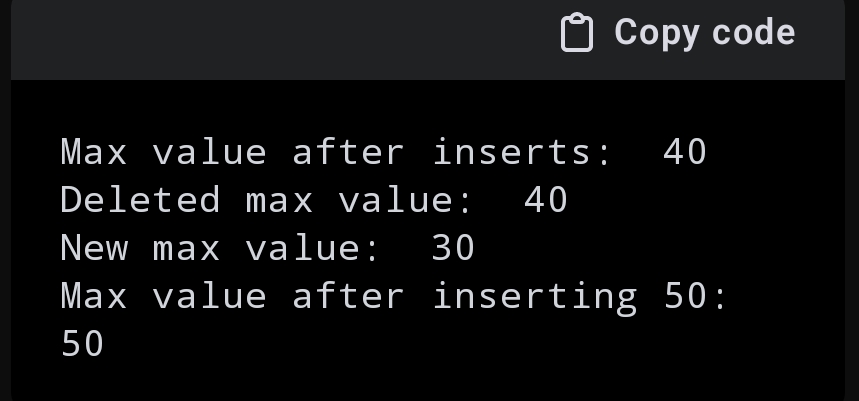
print("Max value after inserts: ", max\_heap.get\_max()) # Output: 40

deleted\_value = max\_heap.delete()

print("Deleted max value: ", deleted\_value) # Output: 40

print("New max value: ", max\_heap.get\_max()) # Output: 30

max\_heap.insert(50)

print("Max value after inserting 50: ", max\_heap.get\_max()) # Output: 50

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.

import requests

from requests.exceptions import HTTPError, Timeout, RequestException

def download\_content(urls):

results = {}

for url in urls:

attempt = 0

success = False

while attempt < 3 and not success:

try:

response = requests.get(url, timeout=5)

response.raise\_for\_status() # Check for HTTP errors

results[url] = response.text

success = True

except HTTPError as http\_err:

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

except Timeout as timeout\_err:

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

except RequestException as req\_err:

print(f"Request error occurred: {req\_err} - URL: {url}")

except Exception as err:

print(f"An error occurred: {err} - URL: {url}")

finally:

attempt += 1

if not success and attempt < 3:

print(f"Retrying... ({attempt}/3)")

if not success:

results[url] = None

print(f"Failed to download content from URL: {url} after 3 attempts")

return results

# Example usage

urls = [

"https://www.example.com",

"https://www.nonexistentwebsite.com",

"https://httpstat.us/500", # Simulates a server error

]

content = download\_content(urls)

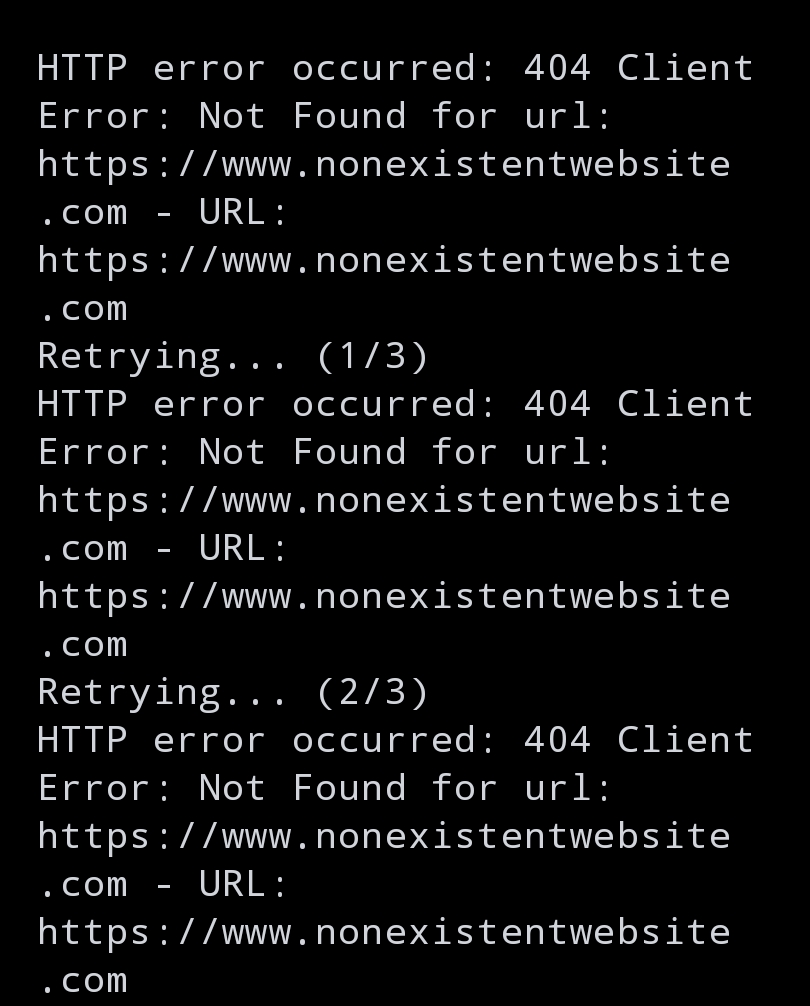
for url, result in content.items():

if result:

print(f"Content from {url}:\n{result[:100]}...\n") # Print the first 100 characters of the content

else:

print(f"Failed to retrieve content from {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.

import numpy as np

import pandas as pd

from sklearn.datasets import load\_boston

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 Boston housing dataset

boston = load\_boston()

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

data['PRICE'] = boston.target

# Define features and target variable

X = data.drop('PRICE', axis=1)

y = data['PRICE']

# Split the data 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 test 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)

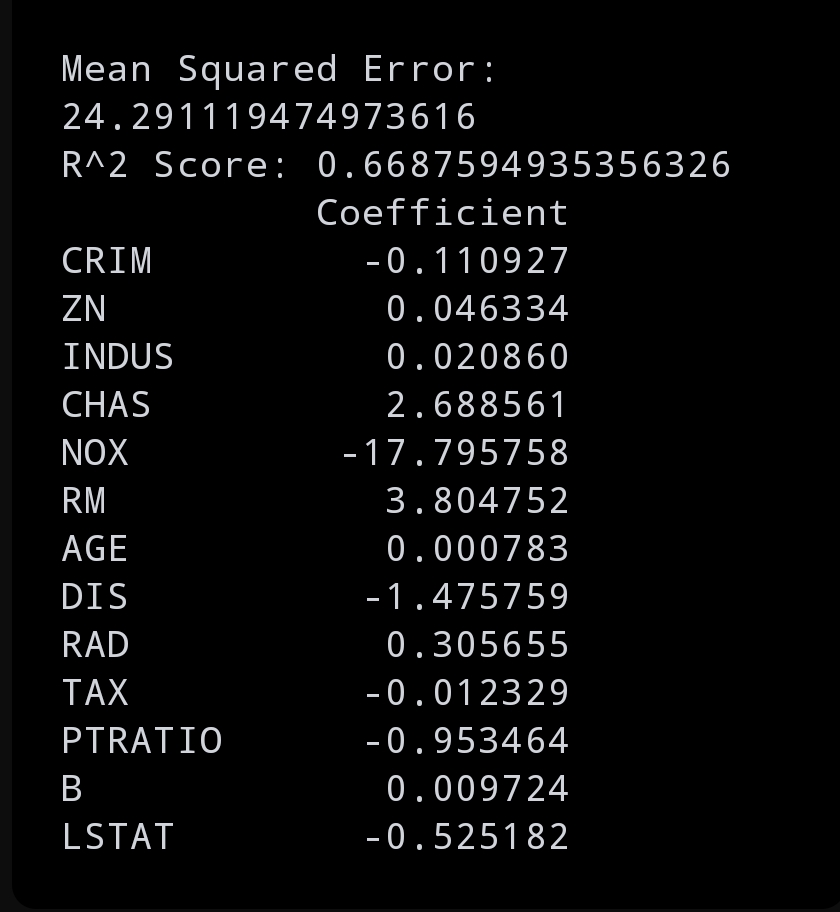
print(f"Mean Squared Error: {mse}")

print(f"R^2 Score: {r2}")

# Optionally, display the coefficients

coefficients = pd.DataFrame(model.coef\_, X.columns, columns=['Coefficient'])

print(coefficients)

L

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.

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

def preprocess\_dataframe(df):

# Handle missing values

for column in df.columns:

if df[column].dtype == 'object':

df[column].fillna(df[column].mode()[0], inplace=True)

else:

df[column].fillna(df[column].median(), inplace=True)

# Normalize numerical columns

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

scaler = MinMaxScaler()

df[numerical\_columns] = scaler.fit\_transform(df[numerical\_columns])

# Encode categorical columns

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

df = pd.get\_dummies(df, columns=categorical\_columns)

return df

# Example usage

data = {

'Age': [25, 30, np.nan, 35, 40],

'Salary': [50000, 60000, 65000, np.nan, 70000],

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

'City': ['New York', 'Los Angeles', np.nan, 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

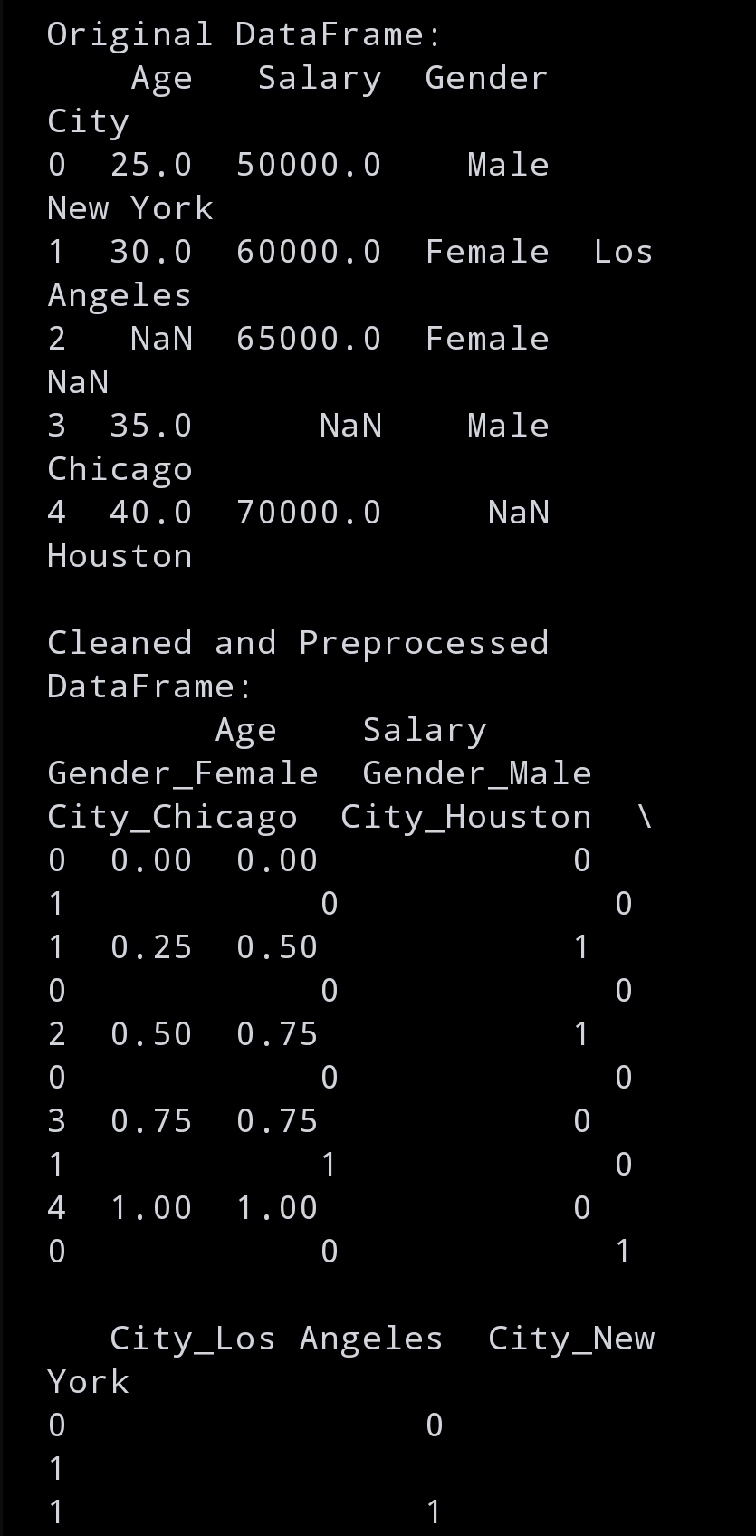
print("Original DataFrame:")

print(df)

cleaned\_df = preprocess\_dataframe(df)

print("\nCleaned and Preprocessed DataFrame:")

print(cleaned\_df)



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

def fibonacci(n):

if n <= 0:

raise ValueError("Input must be a positive integer.")

elif n == 1:

return 0

elif n == 2:

return 1

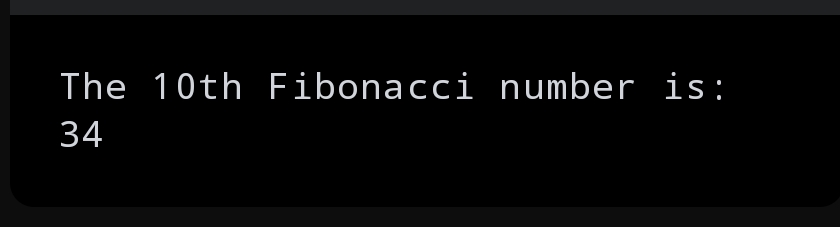
else:

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

# Example usage

n = 10

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



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(numerator, denominator):

try:

result = numerator / denominator

except ZeroDivisionError:

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

except TypeError:

return "Error: Both inputs must be numbers."

return result

# Example usage

numerator = 10

denominator = 0

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

numerator = 10

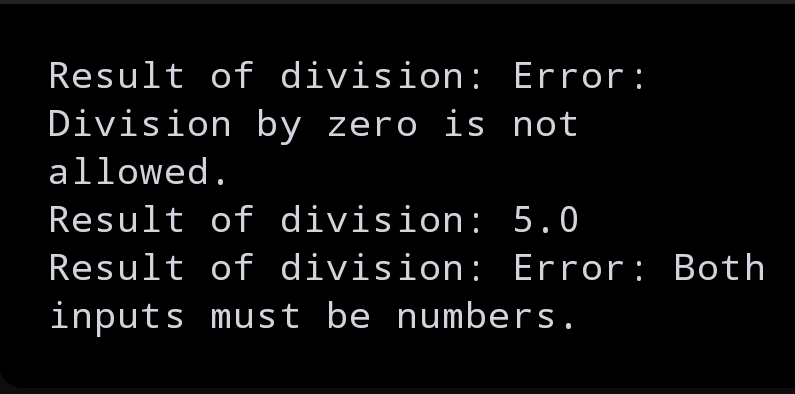
denominator = 2

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

numerator = 10

denominator = "a"

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



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 math

def execution\_time\_logger(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:.6f} seconds")

return result

return wrapper

@execution\_time\_logger

def compute\_factorial(n):

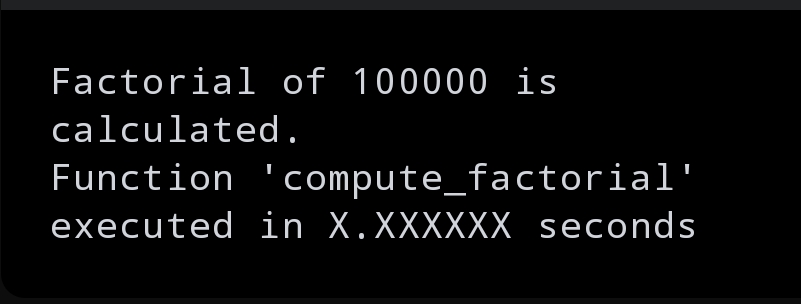
return math.factorial(n)

# Example usage

n = 100000

print(f"Factorial of {n} is calculated.")

compute\_factorial(n)



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 is not allowed."

return num1 / num2

else:

return "Error: Invalid operator. Use '+', '-', '\*', or '/'."

# Example usage

num1 = 10

num2 = 5

operators = ['+', '-', '\*', '/']

for op in operators:

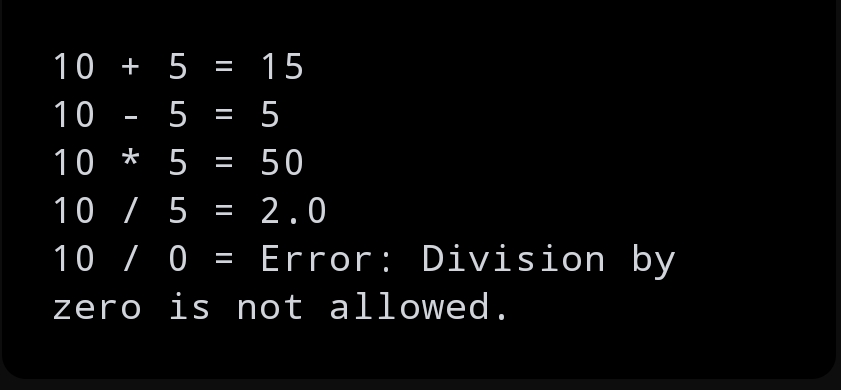
result = arithmetic\_operation(num1, num2, op)

print(f"{num1} {op} {num2} = {result}")

# Example with division by zero

num2 = 0

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



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\_password(length=12):

if length < 4:

raise ValueError("Password length should be at least 4 characters to include all character types.")

# Define character sets

lowercase = string.ascii\_lowercase

uppercase = string.ascii\_uppercase

digits = string.digits

special\_characters = string.punctuation

# Ensure the password contains at least one of each type

password = [

random.choice(lowercase),

random.choice(uppercase),

random.choice(digits),

random.choice(special\_characters)

]

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

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

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

# Shuffle the list to ensure a random order and convert to a string

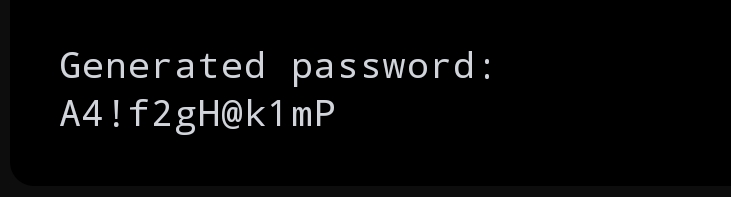
random.shuffle(password)

return ''.join(password)

# Example usage

password = generate\_password(12)

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



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

def transpose\_matrix(matrix):

if not matrix:

return []

num\_rows = len(matrix)

num\_cols = len(matrix[0])

# Initialize the transpose matrix with empty lists

transpose = [[] for \_ in range(num\_cols)]

# Fill the transpose matrix

for i in range(num\_rows):

for j in range(num\_cols):

transpose[j].append(matrix[i][j])

return transpose

# Example usage

matrix = [

[1, 2, 3],

[4, 5, 6],

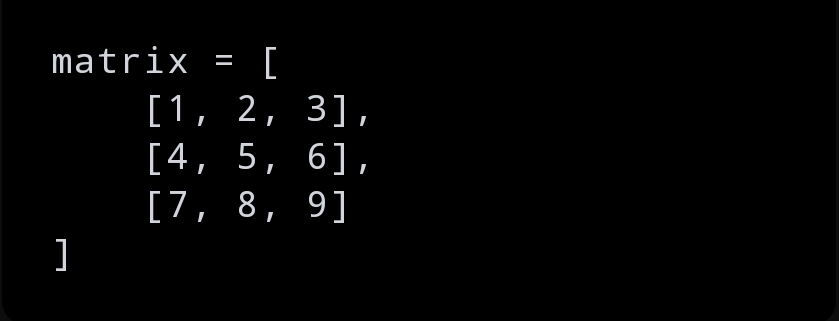
[7, 8, 9]

]

transposed\_matrix = transpose\_matrix(matrix)

for row in transposed\_matrix:

print(row)



The transpose of this is

