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

def delete(self):

if len(self.heap) == 0:

return None

if len(self.heap) == 1:

return self.heap.pop()

max\_value = self.heap[0]

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

self.\_bubble\_down(0)

return max\_value

def get\_max(self):

return self.heap[0] if self.heap else None

def \_bubble\_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.\_bubble\_up(parent\_index)

def \_bubble\_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.\_bubble\_down(largest)

def \_swap(self, i, j):

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

#example

heap = MaxHeap()

#inserting elements

heap.insert(30)

heap.insert(20)

heap.insert(10)

heap.insert(40)

heap.insert(50)

#printing maxium

print(heap.get\_max())

#deleting elements

print(heap.delete())

print(heap.delete())

#inserting elements

heap.insert(100)

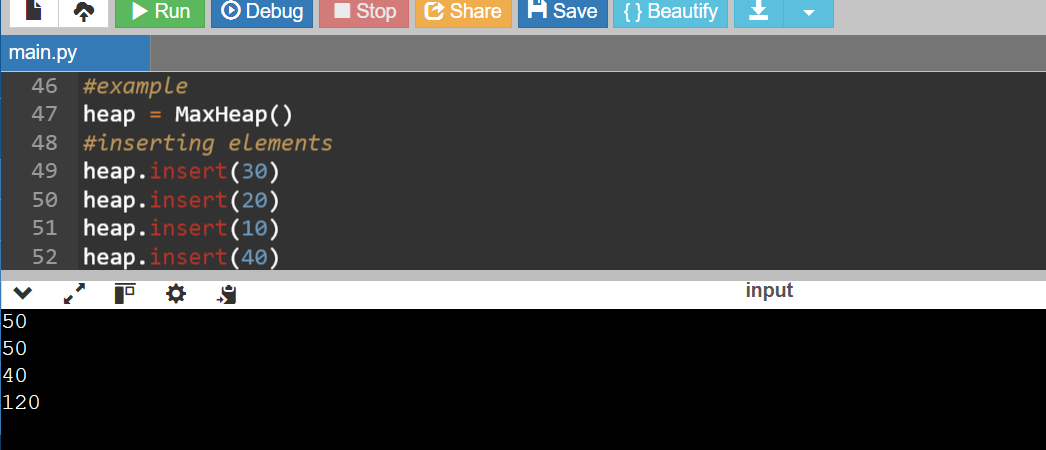
heap.insert(120)

heap.insert(80)

#printing maxium

print(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 fetch\_url\_content(url):

max\_attempts = 3

attempt = 0

while attempt < max\_attempts:

try:

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

response.raise\_for\_status()

return response.text

except requests.HTTPError as http\_err:

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

except requests.ConnectionError as conn\_err:

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

except requests.Timeout as timeout\_err:

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

except requests.RequestException as req\_err:

print(f"General error: {req\_err}")

attempt += 1

print(f"Attempt {attempt} of {max\_attempts} failed. Retrying in 2 seconds...")

sleep(2)

print(f"Failed to retrieve content from {url} after {max\_attempts} attempts.")

return None

def fetch\_multiple\_urls(urls):

results = {}

for url in urls:

print(f"Fetching: {url}")

content = fetch\_url\_content(url)

results[url] = content

return results

# Example usage

urls\_list = [

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

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

"https://www.nonexistentwebsite.com"

]

fetched\_contents = fetch\_multiple\_urls(urls\_list)

for url, content in fetched\_contents.items():

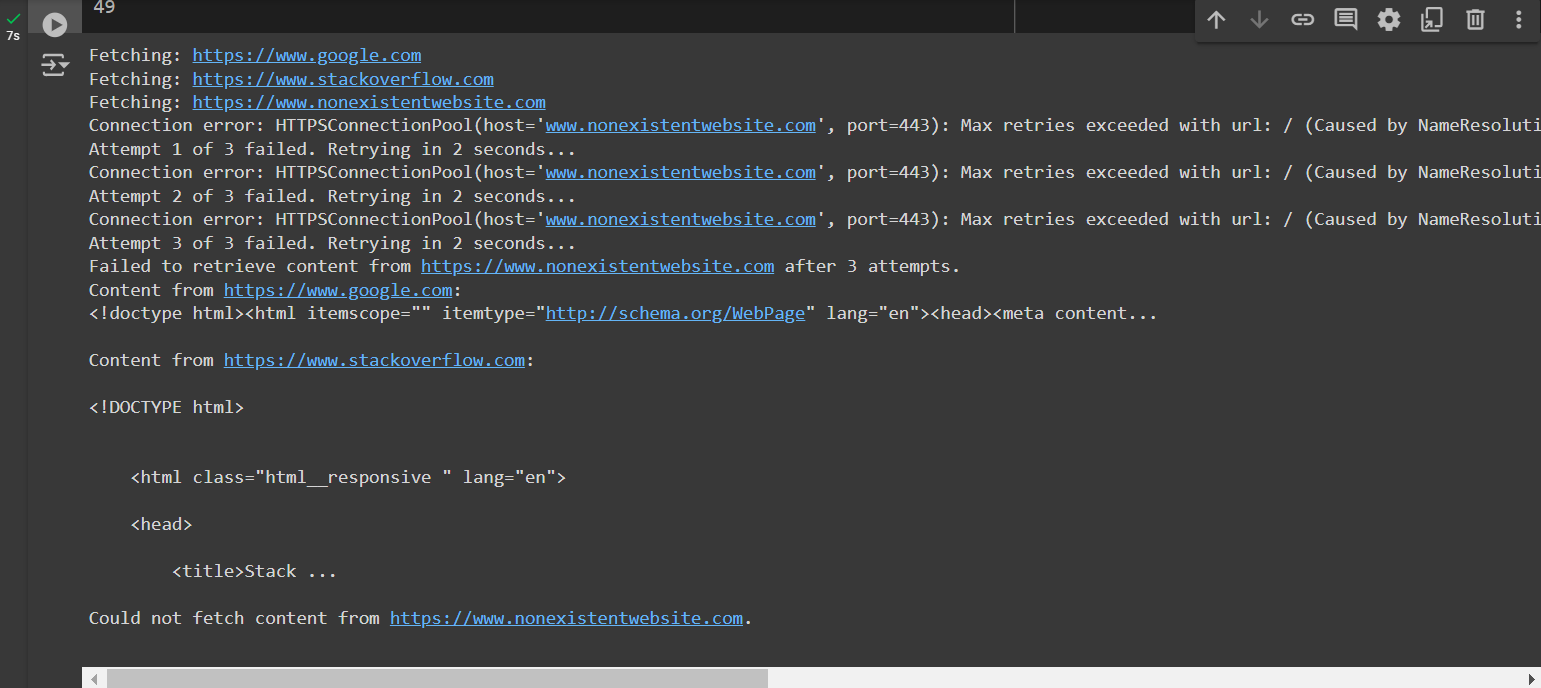
if content:

print(f"Content from {url}:\n{content[:100]}...\n")

else:

print(f"Could not fetch content from {url}.\n")

**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 pandas as pd

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

from sklearn import datasets

# Load the iris dataset

iris = datasets.load\_iris()

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

data['target'] = iris.target

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

y = data['target']

# Split the data

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

# Creating linear regression model

model = LinearRegression()

# Train model

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

#predictions on the test set

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

# Evaluate model

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

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

print("Mean Squared Error:", mse)

print("R-squared:", r2)

# Visualize the model's performance

plt.scatter(y\_test, y\_pred)

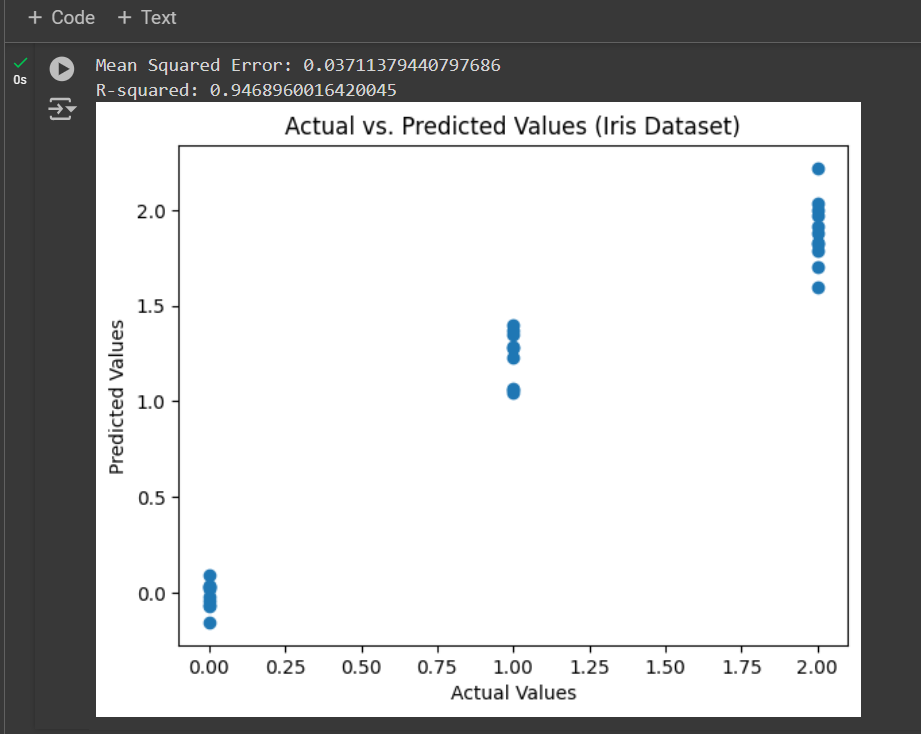
plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Actual vs. Predicted Values (Iris Dataset)")

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 pandas as pd

from sklearn.preprocessing import StandardScaler, OneHotEncoder

def preprocess\_dataframe(df):

# Handling missing values

for column in df.columns:

if df[column].isnull().any():

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

most\_common\_value = df[column].mode()[0]

df[column].fillna(most\_common\_value, inplace=True)

else:

mean\_value = df[column].mean()

df[column].fillna(mean\_value, inplace=True)

# Normalizing numerical columns

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

scaler = StandardScaler()

df[numeric\_features] = scaler.fit\_transform(df[numeric\_features])

# Encoding categorical columns

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

encoder = OneHotEncoder(sparse=False, drop='first')

encoded\_features = encoder.fit\_transform(df[categorical\_features])

encoded\_df = pd.DataFrame(encoded\_features, columns=encoder.get\_feature\_names\_out(categorical\_features))

# Dropping original categorical columns and combining with encoded columns

df.drop(columns=categorical\_features, inplace=True)

df = pd.concat([df, encoded\_df], axis=1)

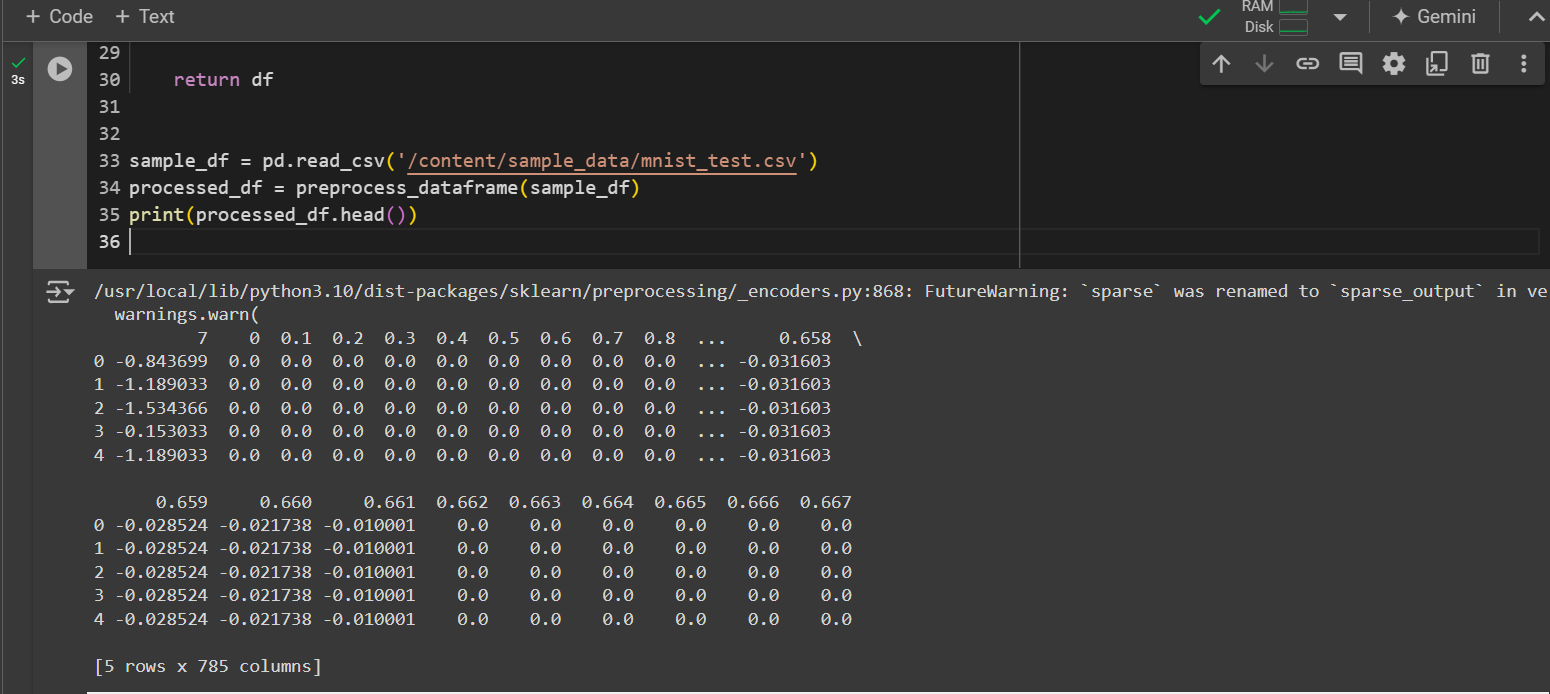
return df

sample\_df = pd.read\_csv('/content/sample\_data/mnist\_test.csv')

processed\_df = preprocess\_dataframe(sample\_df)

print(processed\_df.head())

**output :-**



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

def nth\_fibonacci(n):

if n <= 1:

return n

else:

return nth\_fibonacci(n-1) + nth\_fibonacci(n-2)

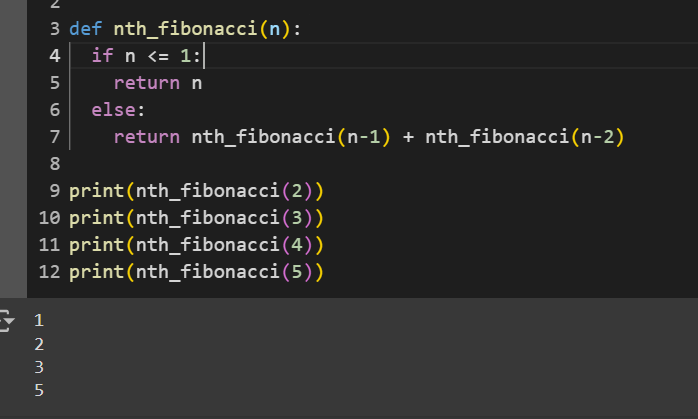
print(nth\_fibonacci(2))

print(nth\_fibonacci(3))

print(nth\_fibonacci(4))

print(nth\_fibonacci(5))

**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 division(dividend, divisor):

if divisor == 0:

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

else:

return dividend / divisor

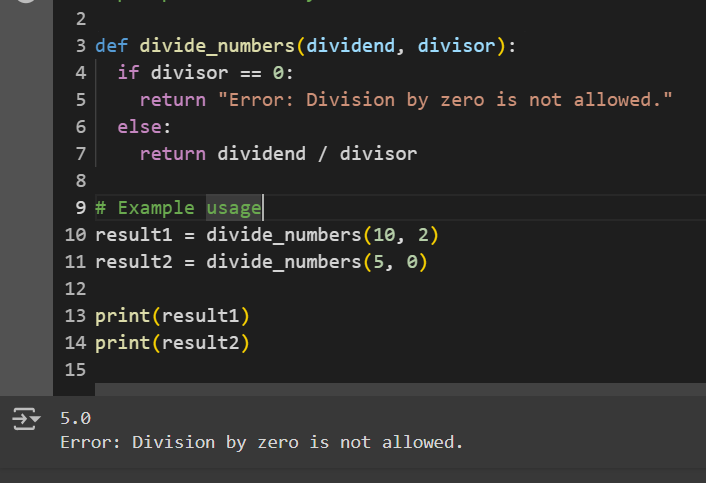
res1 = division(10, 2)

res2 = division (5, 0)

print(result1)

print(result2)

**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

import logging

# Configure logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')

def log\_execution\_time(func):

def wrapped\_func(\*args, \*\*kwargs):

start\_time = time.time()

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

end\_time = time.time()

duration = end\_time - start\_time

logging.info(f"Function '{func.\_\_name\_\_}' executed in {duration:.4f} seconds.")

return result, duration

return wrapped\_func

@log\_execution\_time

def sum\_of\_squares(n):

total = 0

for i in range(1, n + 1):

total += i \* i

return total

# Example

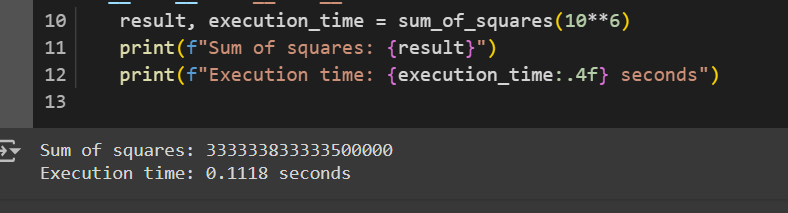
if \_\_name\_\_ == "\_\_main\_\_":

result, execution\_time = sum\_of\_squares(10\*\*6)

print(f"Sum of squares: {result}")

print(f"Execution time: {execution\_time:.4f} seconds")

**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 perform\_operation(a, b, op):

if op == '+':

return a + b

elif op == '-':

return a - b

elif op == '\*':

return a \* b

elif op == '/':

if b == 0:

raise ZeroDivisionError("Cannot divide by zero.")

return a / b

else:

raise ValueError(f"Invalid operator: {op}")

if \_\_name\_\_ == "\_\_main\_\_":

try:

print("10 + 5 =", perform\_operation(10, 5, '+'))

print("10 - 5 =", perform\_operation(10, 5, '-'))

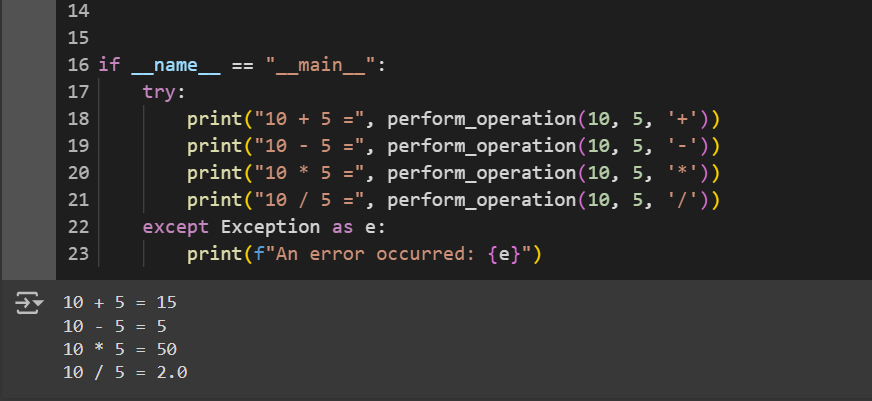
print("10 \* 5 =", perform\_operation(10, 5, '\*'))

print("10 / 5 =", perform\_operation(10, 5, '/'))

except Exception as e:

print(f"An error occurred: {e}")

**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):

if length < 4:

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

# Character categories

upper\_case\_letters = string.ascii\_uppercase

lower\_case\_letters = string.ascii\_lowercase

digits = string.digits

special\_characters = string.punctuation

# Start with one character from each category

password\_chars = [

random.choice(upper\_case\_letters),

random.choice(lower\_case\_letters),

random.choice(digits),

random.choice(special\_characters)

]

all\_characters = upper\_case\_letters + lower\_case\_letters + digits + special\_characters

password\_chars.extend(random.choice(all\_characters) for \_ in range(length - 4))

random.shuffle(password\_chars)

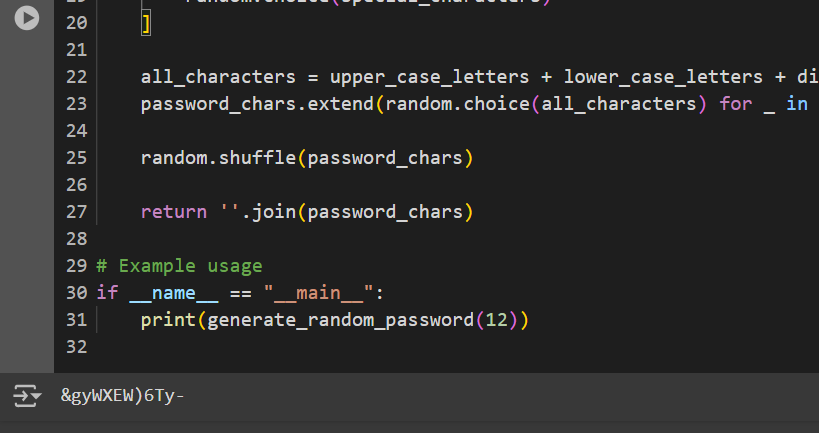
return ''.join(password\_chars)

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

print(generate\_random\_password(12))

**Output :-**



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

def transpose\_matrix(matrix):

"""

Returns the transpose of a given 2D list (matrix).

Args:

matrix (list of list): The matrix to transpose.

Returns:

list of list: The transposed matrix.

"""

# Check for an empty matrix

if not matrix:

return []

# Calculate number of rows and columns

num\_rows = len(matrix)

num\_cols = len(matrix[0])

# Create a new matrix with dimensions swapped

transposed = []

for col in range(num\_cols):

new\_row = []

for row in range(num\_rows):

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

transposed.append(new\_row)

return transposed

# Example usage

example\_matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

transposed\_example = transpose\_matrix(example\_matrix)

for row in transposed\_example:

print(row)

output :-

