**Anshul Bisht**

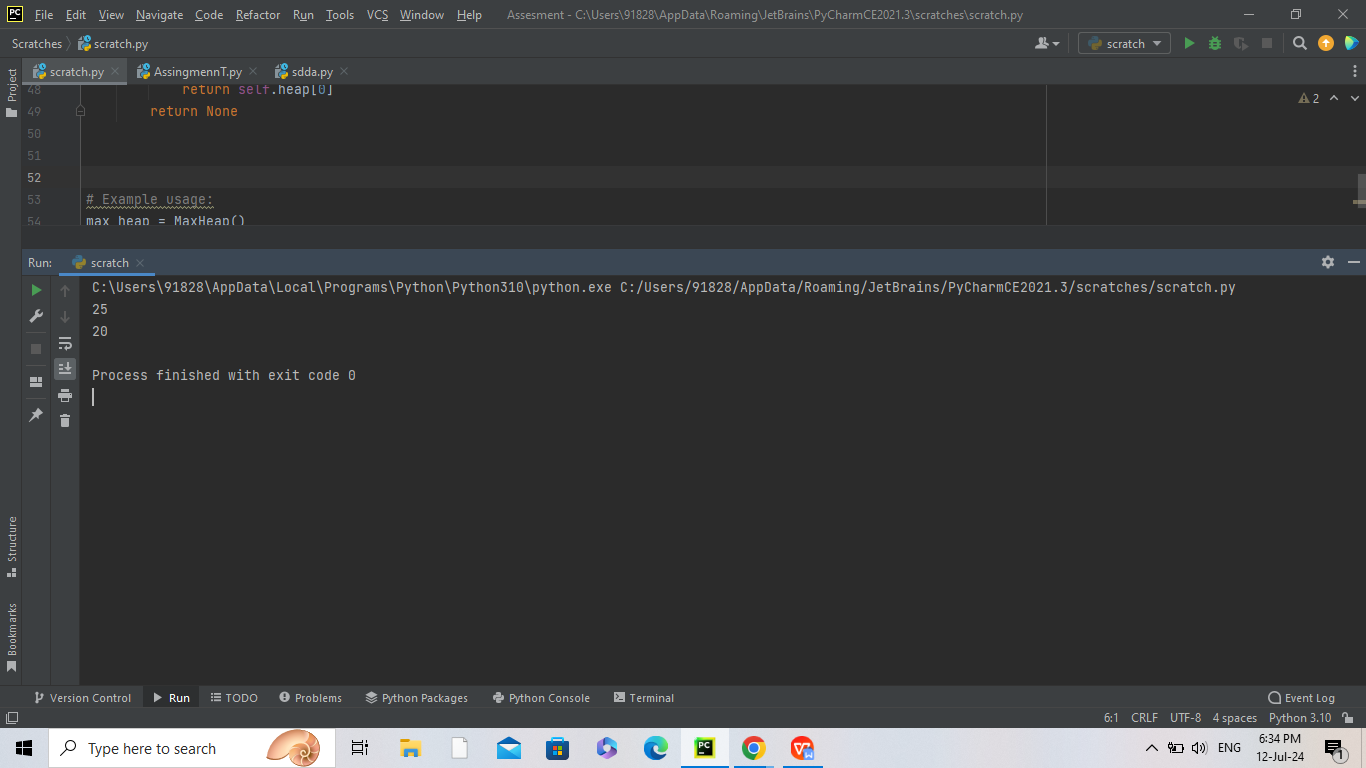
Here are the python programming Question and their answers

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.

Ans->

class MaxHeap:  
 def \_\_init\_\_(self):  
 self.heap = []  
  
 def \_heapify\_up(self, index):  
 parent\_index = (index - 1) // 2  
 # If the current node is greater than its parent, swap them  
 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]  
 # Recursively heapify up the parent node  
 self.\_heapify\_up(parent\_index)  
  
 def \_heapify\_down(self, index):  
 left\_child\_index = 2 \* index + 1  
 right\_child\_index = 2 \* index + 2  
 largest = index  
  
 # Check if the left child exists and is greater than the current node  
 if left\_child\_index < len(self.heap) and self.heap[left\_child\_index] > self.heap[largest]:  
 largest = left\_child\_index  
  
 # Check if the right child exists and is greater than the largest node so far  
 if right\_child\_index < len(self.heap) and self.heap[right\_child\_index] > self.heap[largest]:  
 largest = right\_child\_index  
  
 # If the largest node is not the current node, swap them and heapify down  
 if largest != index:  
 self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]  
 self.\_heapify\_down(largest)  
  
 def insert(self, val):  
 # Add the new value to the end of the heap  
 self.heap.append(val)  
 # Move the new value up to maintain the max-heap property  
 self.\_heapify\_up(len(self.heap) - 1)  
  
 def delete(self, val):  
 # Find the index of the value to be deleted  
 index = self.heap.index(val)  
 # Move the last element to the position to be deleted  
 self.heap[index] = self.heap[-1]  
 self.heap.pop()  
 # Heapify down from the index to maintain the max-heap property  
 self.\_heapify\_down(index)  
  
 def get\_max(self):  
 if self.heap:  
 return self.heap[0]  
 return None  
# Example usage:  
max\_heap = MaxHeap()  
max\_heap.insert(20)  
max\_heap.insert(25)  
max\_heap.insert(15)  
print(max\_heap.get\_max()) # Output: 25  
max\_heap.delete(25)  
print(max\_heap.get\_max()) # Output: 20

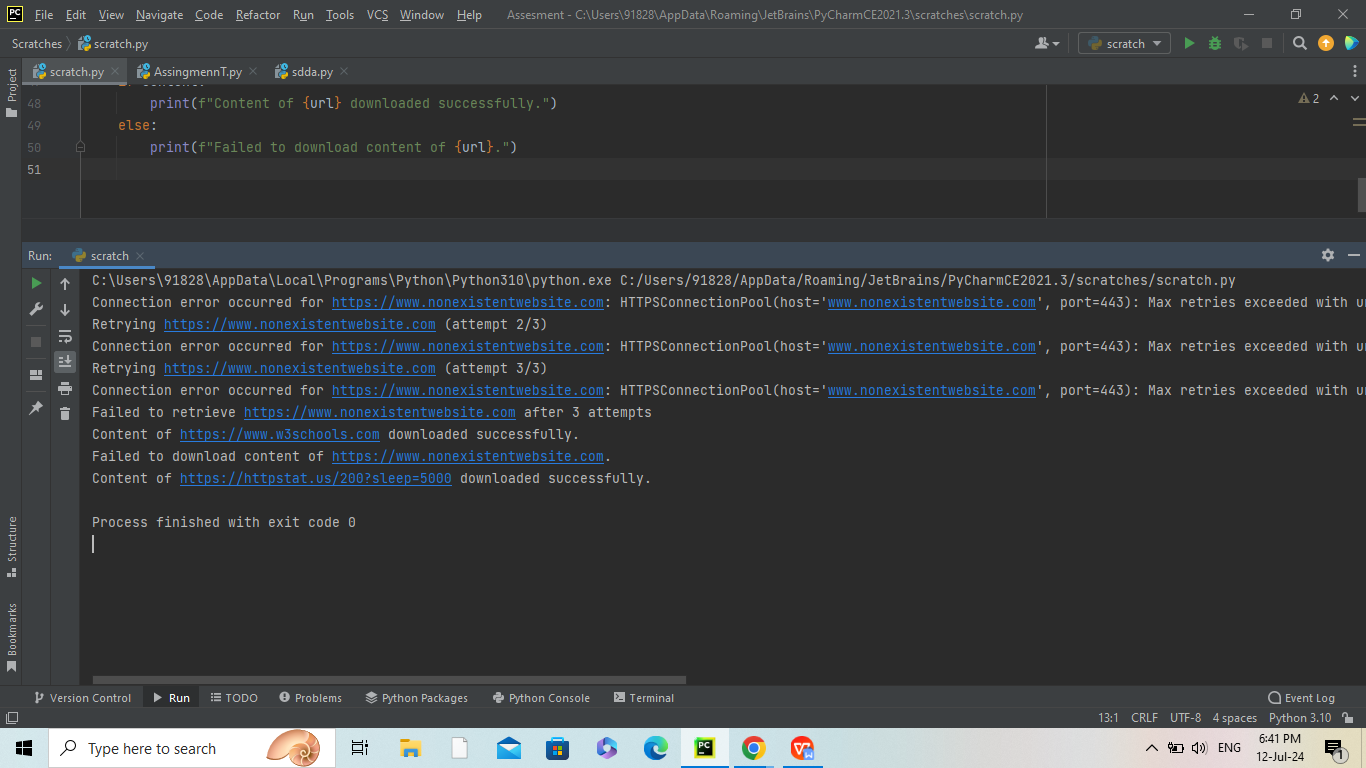
Output



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 = {}  
  
 for url in urls:  
 attempts = 0  
 success = False  
 while attempts < 3 and not success:  
 try:  
 response = requests.get(url, timeout=10)  
 response.raise\_for\_status()  
 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"Request error occurred for {url}: {req\_err}")  
 finally:  
 attempts += 1  
 if not success and attempts < 3:  
 print(f"Retrying {url} (attempt {attempts + 1}/3)")  
 sleep(2) # wait for 2 seconds before retrying  
  
 if not success:  
 results[url] = None  
 print(f"Failed to retrieve {url} after 3 attempts")  
  
 return results  
  
  
# Example usage:  
urls = [  
 "https://www.w3schools.com",  
 "https://www.nonexistentwebsite.com",  
 "https://httpstat.us/200?sleep=5000" # This URL will simulate a delayed response  
]  
contents = download\_urls(urls)  
  
for url, content in contents.items():  
 if content:  
 print(f"Content of {url} downloaded successfully.")  
 else:  
 print(f"Failed to download content of {url}.")

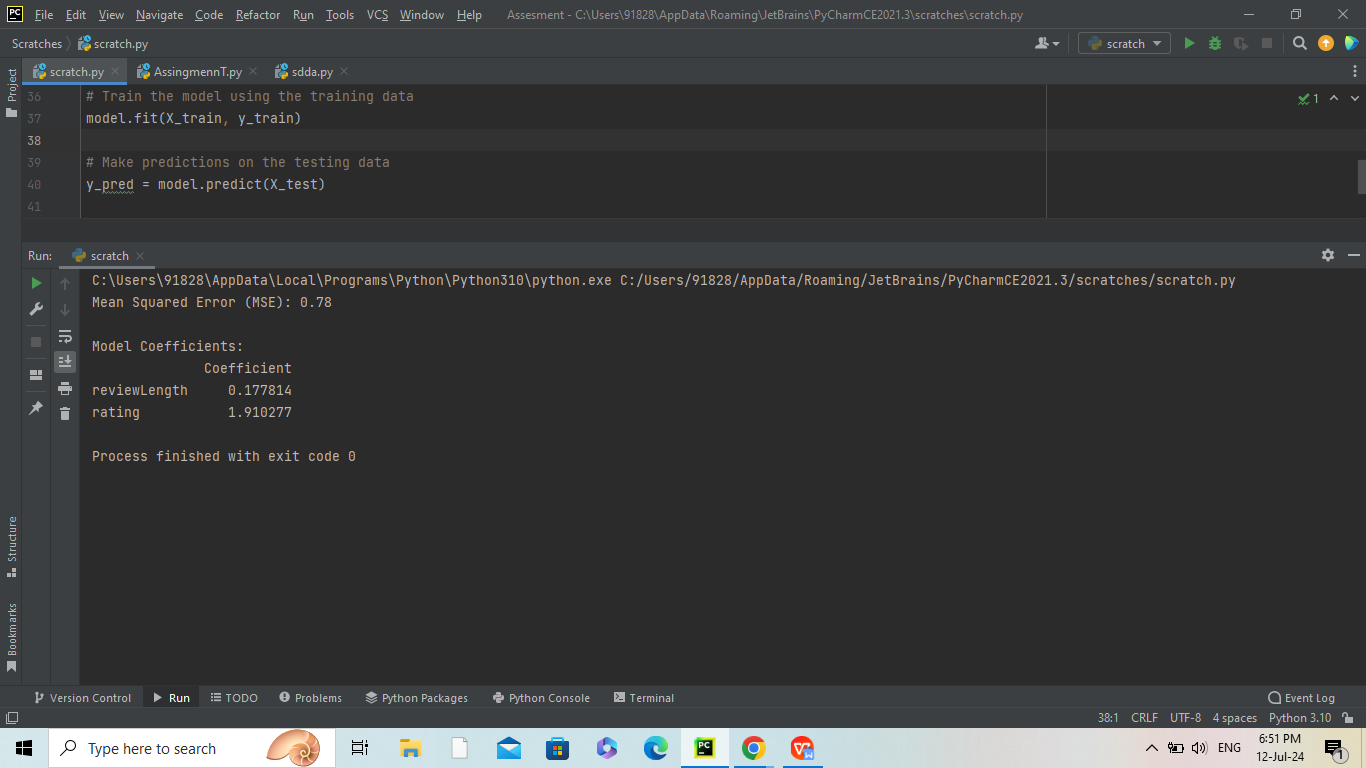
Output



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.

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  
  
# Assuming we have a dataset 'amazon\_reviews\_india.csv' with columns: 'reviewText', 'rating', 'helpfulnessScore'  
# Create a sample dataframe for illustration  
data = {  
 'reviewText': [  
 'This product is great!',  
 'Terrible quality, do not buy.',  
 'Decent product for the price.',  
 'Exceeded my expectations!',  
 'Not worth the money.'  
 ],  
 'rating': [5, 1, 3, 5, 2],  
 'helpfulnessScore': [10, 2, 7, 9, 3]  
}  
# Convert to DataFrame  
df = pd.DataFrame(data)  
  
# Feature Engineering: For simplicity, we will use review length and rating as features  
df['reviewLength'] = df['reviewText'].apply(len)  
  
# Define features and target variable  
X = df[['reviewLength', 'rating']]  
y = df['helpfulnessScore']  
  
# Split the dataset 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 using the training data  
model.fit(X\_train, y\_train)  
  
# Make predictions on the testing data  
y\_pred = model.predict(X\_test)  
  
# Evaluate the model's performance  
mse = mean\_squared\_error(y\_test, y\_pred)  
print(f"Mean Squared Error (MSE): {mse:.2f}")  
# Print model coefficients  
print("\nModel Coefficients:")  
coefficients = pd.DataFrame(model.coef\_, X.columns, columns=['Coefficient'])  
print(coefficients)

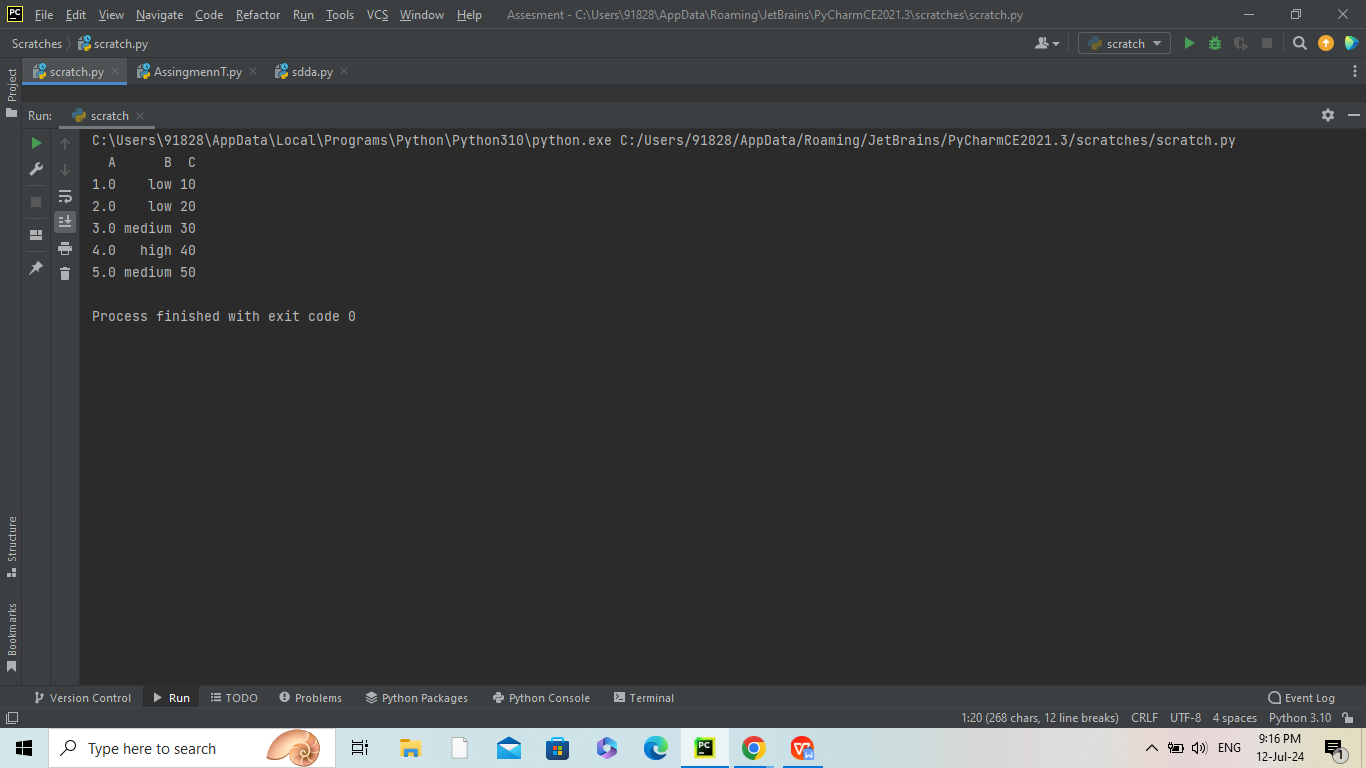
Output



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.

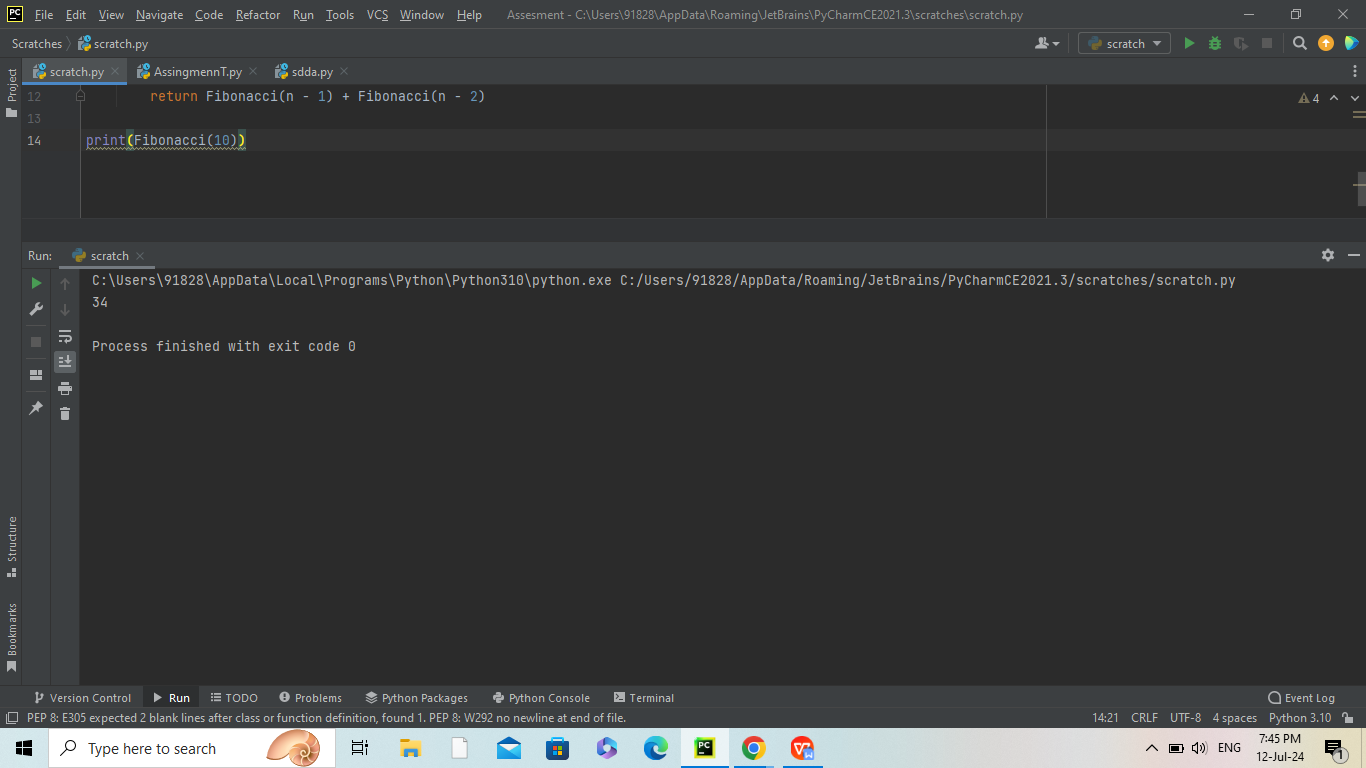
import pandas as pd  
from sklearn.preprocessing import StandardScaler, OneHotEncoder  
from sklearn.compose import ColumnTransformer  
from sklearn.pipeline import Pipeline  
from sklearn.impute import SimpleImputer  
  
  
def clean\_and\_preprocess(df):  
 # Identify categorical and numerical columns  
 categorical\_cols = df.select\_dtypes(include=['object', 'category']).columns  
 numerical\_cols = df.select\_dtypes(include=['int64', 'float64']).columns  
  
 # Separate DataFrame for transformation  
 df\_transformed = df.copy()  
  
 # Pipeline for numerical columns  
 numerical\_pipeline = Pipeline([  
 ('imputer', SimpleImputer(strategy='mean')), # Replace missing values with mean  
 ('scaler', StandardScaler()) # Normalize the numerical data  
 ])  
  
 # Pipeline for categorical columns  
 categorical\_pipeline = Pipeline([  
 ('imputer', SimpleImputer(strategy='most\_frequent')), # Replace missing values with most frequent value  
 ('encoder', OneHotEncoder(handle\_unknown='ignore')) # One-hot encode the categorical data  
 ])  
  
 # Apply transformations only to missing values, keeping existing values intact  
 for col in numerical\_cols:  
 if df\_transformed[col].isnull().any():  
 df\_transformed[col] = numerical\_pipeline.fit\_transform(df\_transformed[[col]])  
  
 for col in categorical\_cols:  
 if df\_transformed[col].isnull().any():  
 df\_transformed[col] = categorical\_pipeline.fit\_transform(df\_transformed[[col]])  
  
 return df\_transformed  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 data = {  
 'A': [1, 2, None, 4, 5],  
 'B': [None, 'low', 'medium', 'high', 'medium'],  
 'C': [10, 20, 30, 40, 50]  
 }  
 df = pd.DataFrame(data)  
 df\_cleaned = clean\_and\_preprocess(df)  
 print(df\_cleaned)

Output



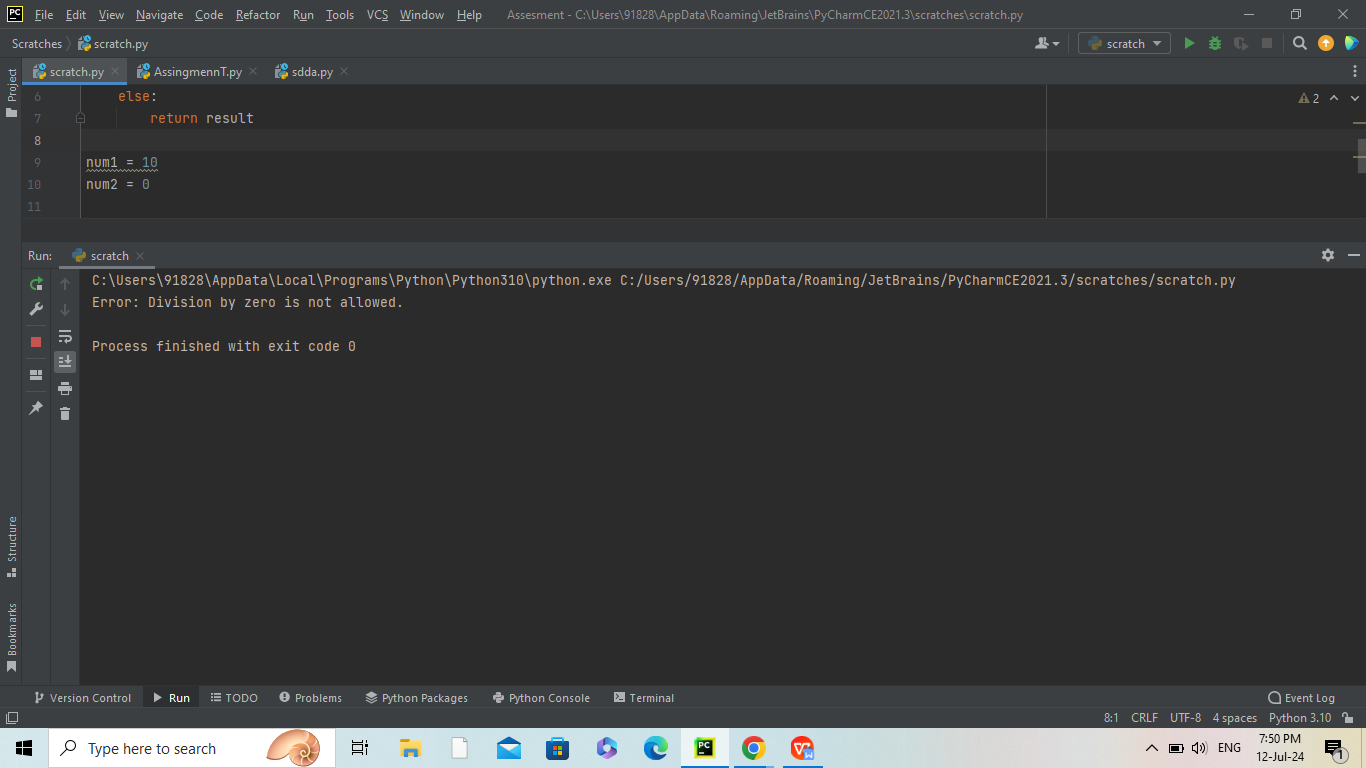
1. Write a Python function to compute the nth Fibonacci number using recursion.

#Fibonacci number using Recursion  
def Fibonacci(n):  
 if n <= 0:  
 print("Incorrect input")  
 # First Fibonacci number is 0  
 elif n == 1:  
 return 0  
 # Second Fibonacci number is 1  
 elif n == 2:  
 return 1  
 else:  
 return Fibonacci(n - 1) + Fibonacci(n - 2)  
  
print(Fibonacci(10))

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.

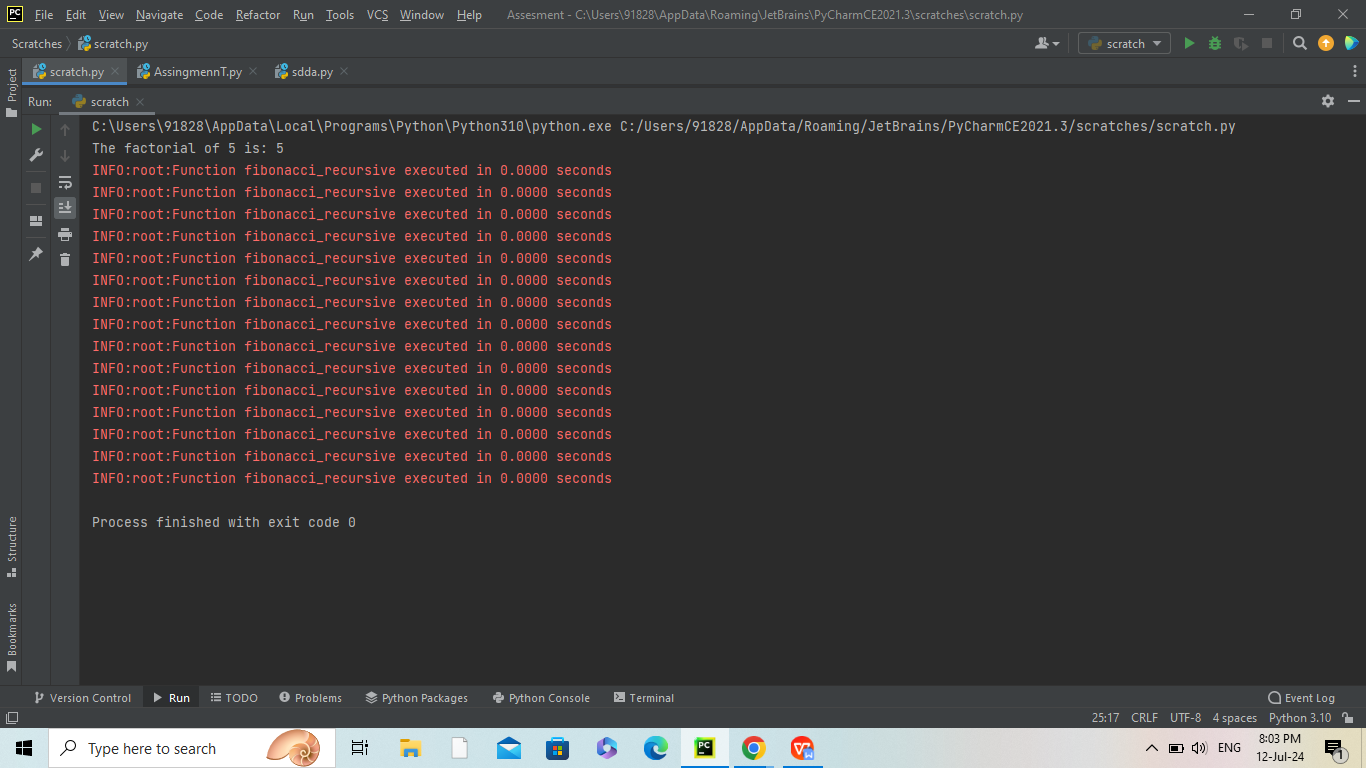
def divide\_numbers(dividend, divisor):  
 try:  
 result = dividend / divisor  
 except ZeroDivisionError:  
 return "Error: Division by zero is not allowed."  
 else:  
 return result  
  
num1 = 10  
num2 = 0  
  
result = divide\_numbers(num1, num2)  
print(result)



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 functools  
import logging  
  
# Setup logging  
logging.basicConfig(level=logging.INFO)  
  
# Define decorator function to measure execution time and log it  
def measure\_execution\_time(func):  
 @functools.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\_recursive(n):  
 if n <= 0:  
 return 0  
 elif n == 1:  
 return 1  
 else:  
 return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2)  
  
# Example usage:  
n = 5  
result = fibonacci\_recursive(n)  
print(f"The factorial of {n} is: {result}")

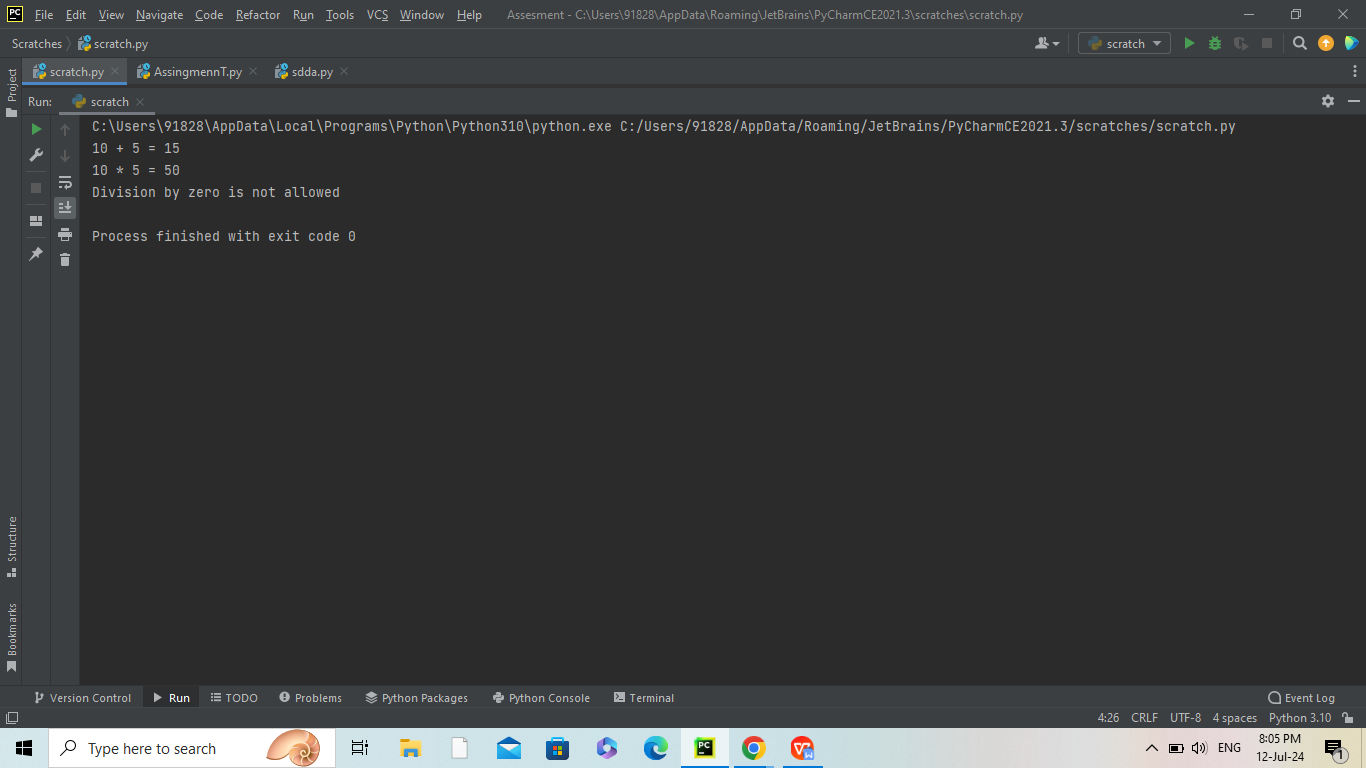
Output



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 arithmetic\_operation(num1, num2, operator):  
 if operator == '+':  
 result = num1 + num2  
 elif operator == '-':  
 result = num1 - num2  
 elif operator == '\*':  
 result = num1 \* num2  
 elif operator == '/':  
 if num2 == 0:  
 raise ValueError("Division by zero is not allowed")  
 result = num1 / num2  
 else:  
 raise ValueError(f"Unsupported operator: {operator}")  
  
 return result  
  
  
num1 = 10  
num2 = 5  
operator = '+'  
  
result = arithmetic\_operation(num1, num2, operator)  
print(f"{num1} {operator} {num2} = {result}")  
  
operator = '\*'  
result = arithmetic\_operation(num1, num2, operator)  
print(f"{num1} {operator} {num2} = {result}")  
  
num1 = 20  
num2 = 0  
operator = '/'  
  
try:  
 result = arithmetic\_operation(num1, num2, operator)  
 print(f"{num1} {operator} {num2} = {result}")  
except ValueError as e:  
 print(e)

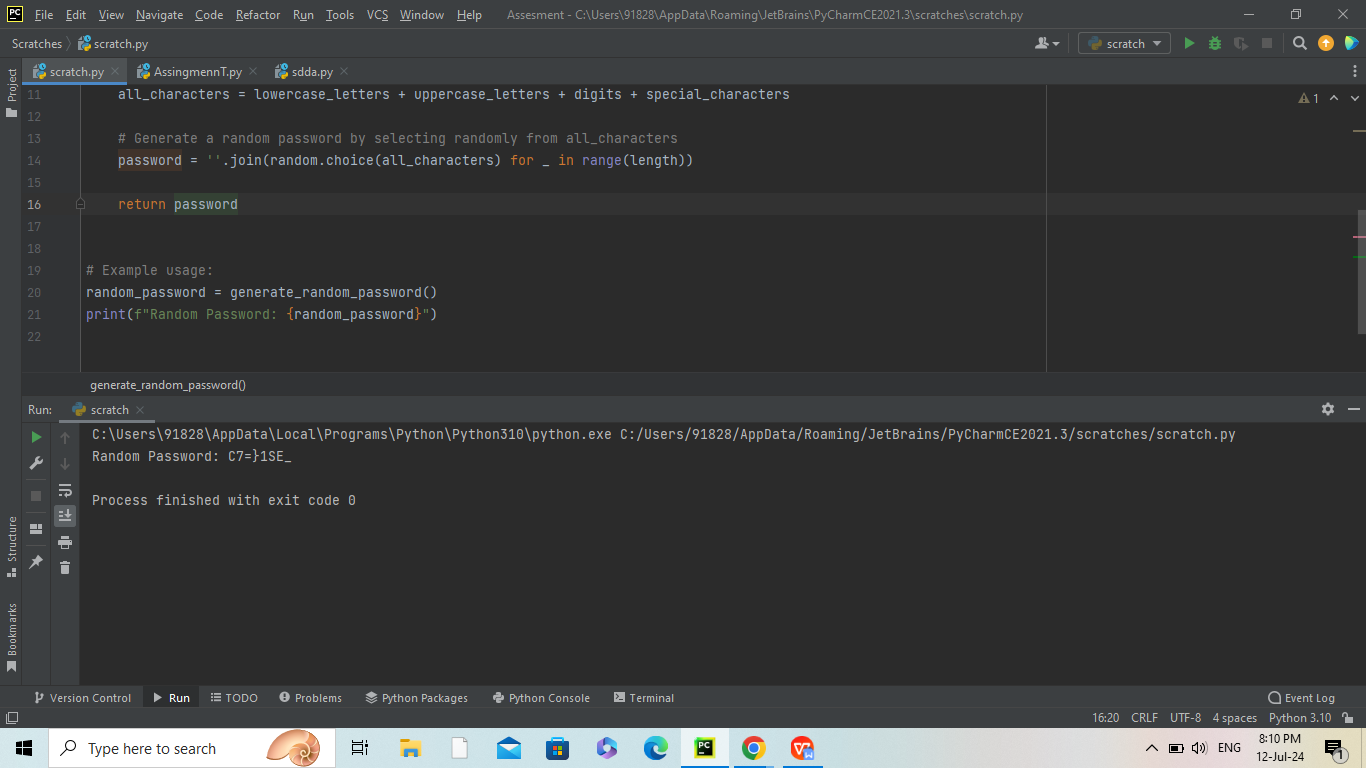
Output



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=8):  
 # Define the character sets  
 lowercase\_letters = string.ascii\_lowercase # lower case letters (a,b,c,d,....,z)  
 uppercase\_letters = string.ascii\_uppercase # upper case letters (A,B,C,D,...,Z)  
 digits = string.digits # Digits (1,2,3,4,5,6,7,8,9,0)  
 special\_characters = string.punctuation # Includes !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~  
  
 # Combine all character sets  
 all\_characters = lowercase\_letters + uppercase\_letters + digits + special\_characters  
  
 # Generate a random password by selecting randomly from all\_characters  
 password = ''.join(random.choice(all\_characters) for \_ in range(length))  
  
 return password  
  
  
# Example usage:  
random\_password = generate\_random\_password()  
print(f"Random Password: {random\_password}")

Output



1. 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 rows > 0 else 0  
  
 # Create a new matrix for the transpose  
 transpose = [[matrix[row][col] for row in range(rows)] for col in range(cols)]  
  
 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)

Output

