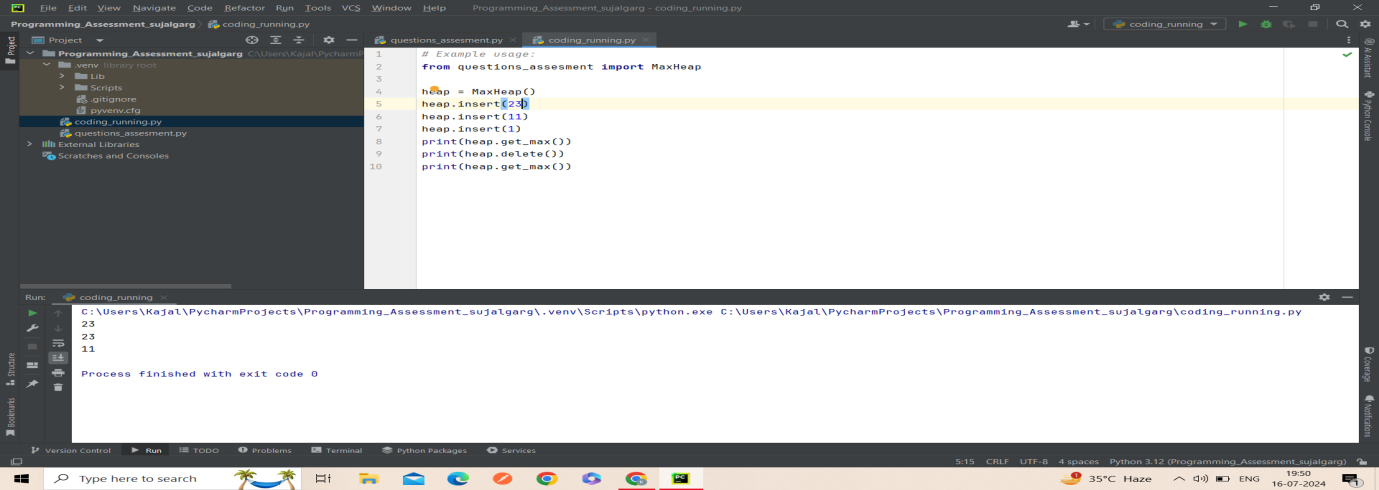
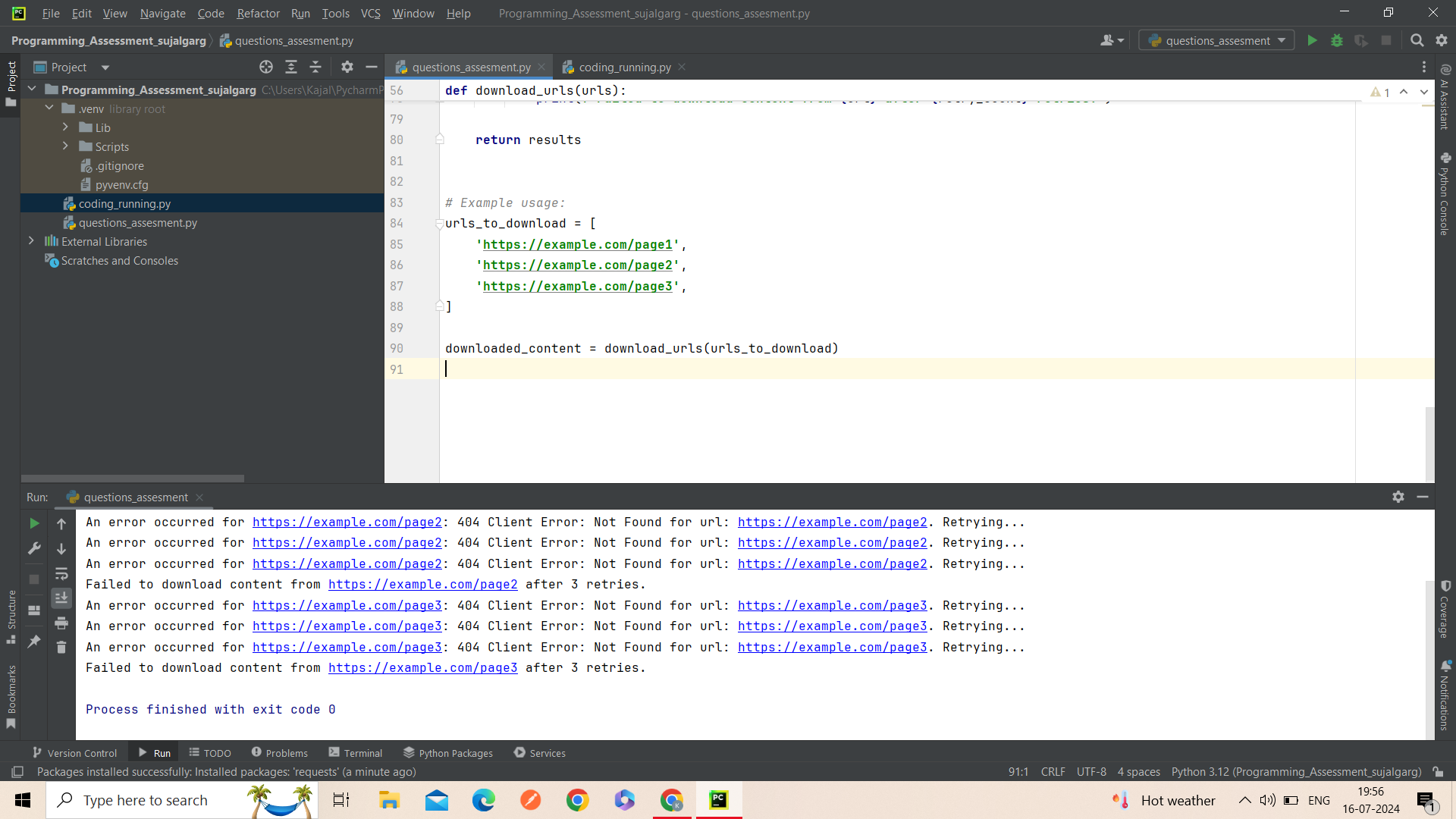
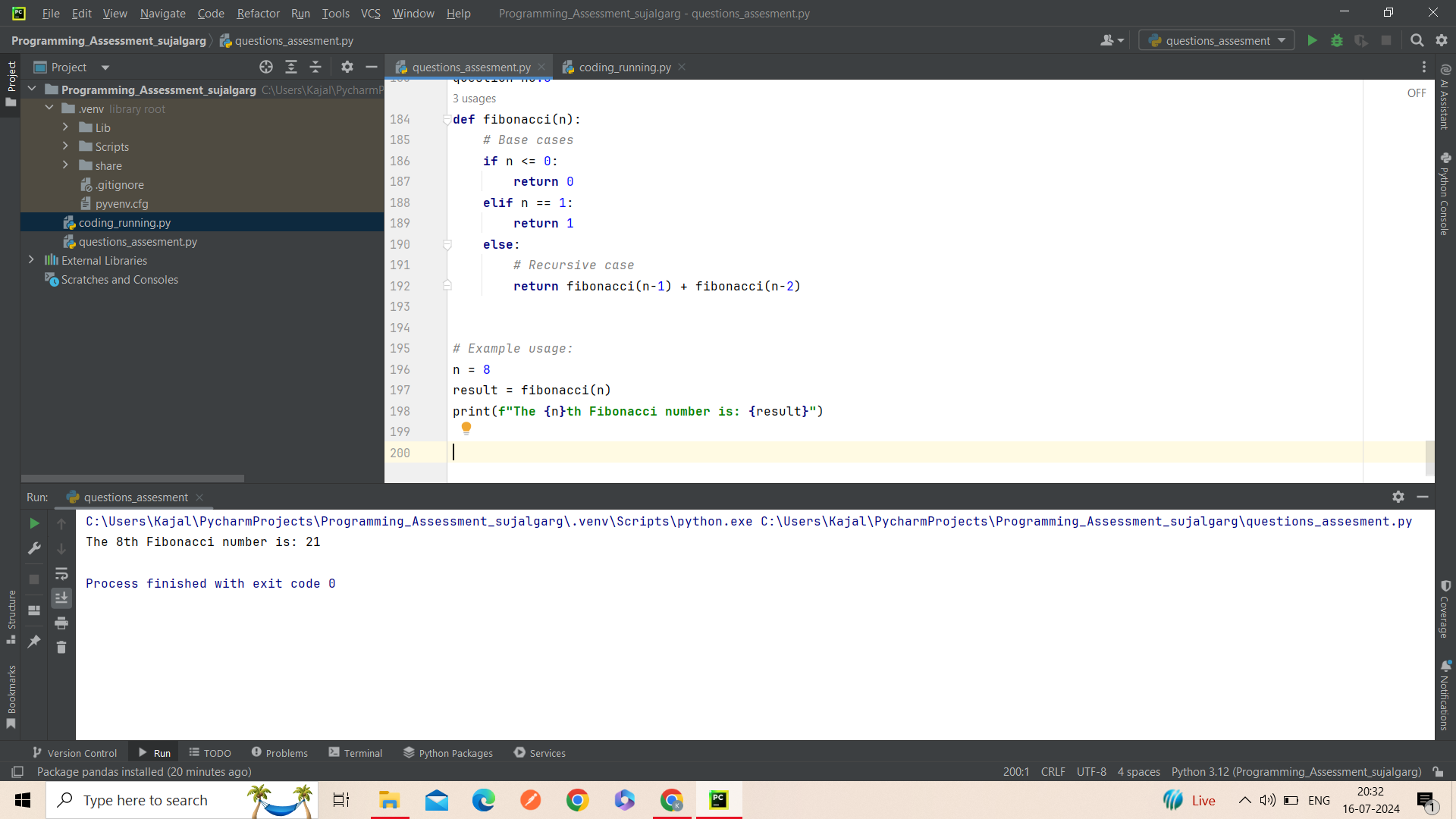
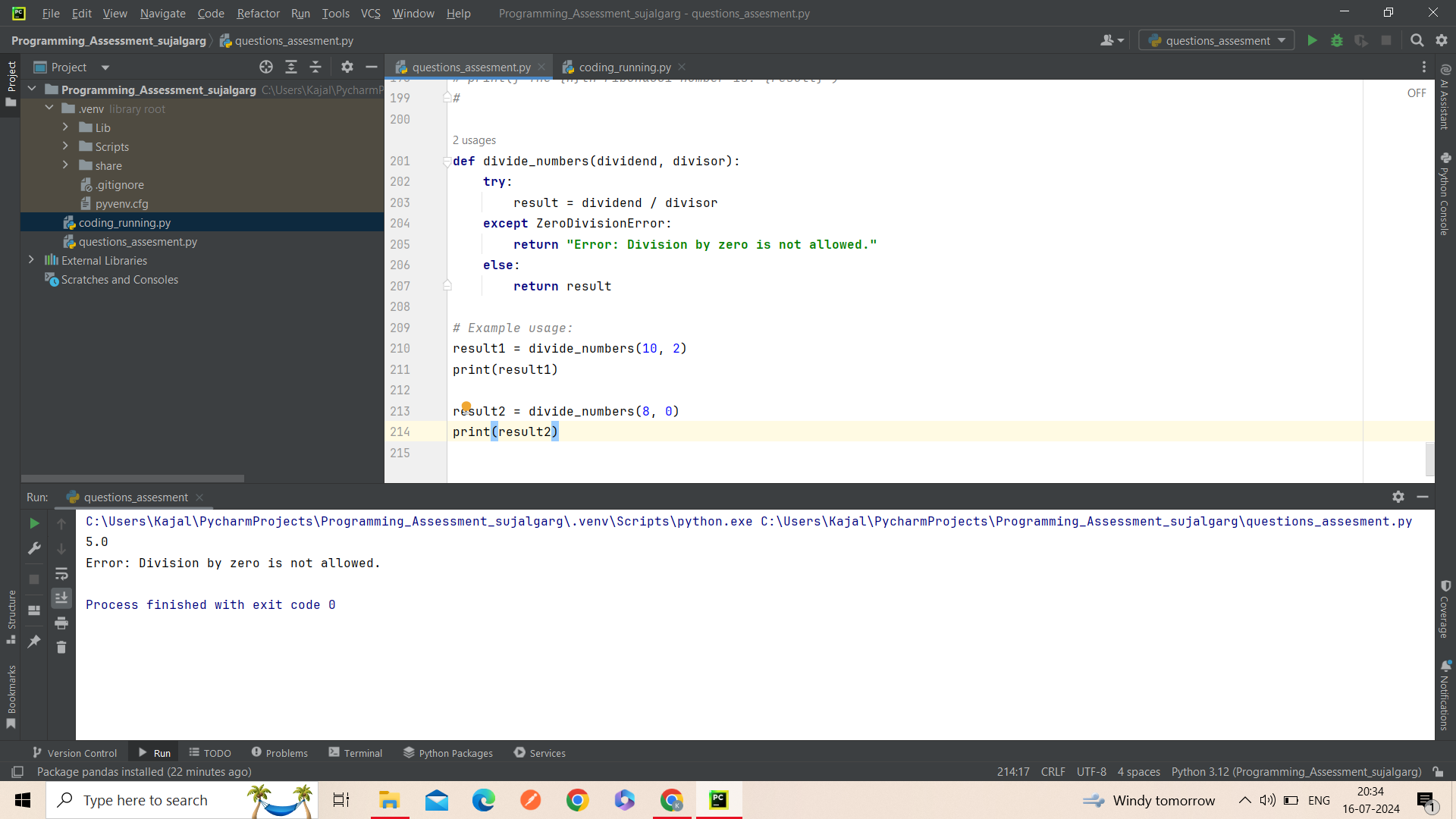
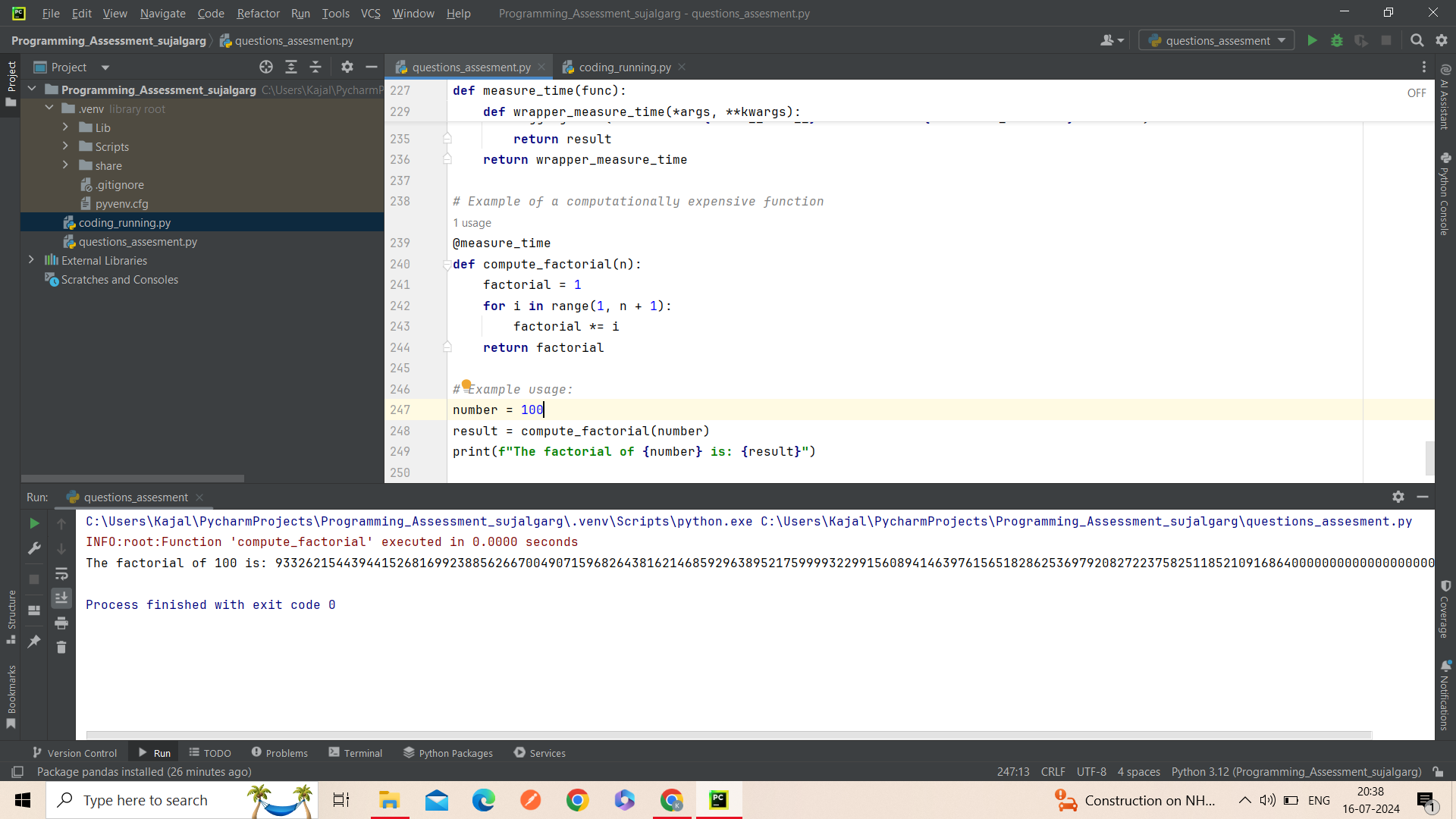
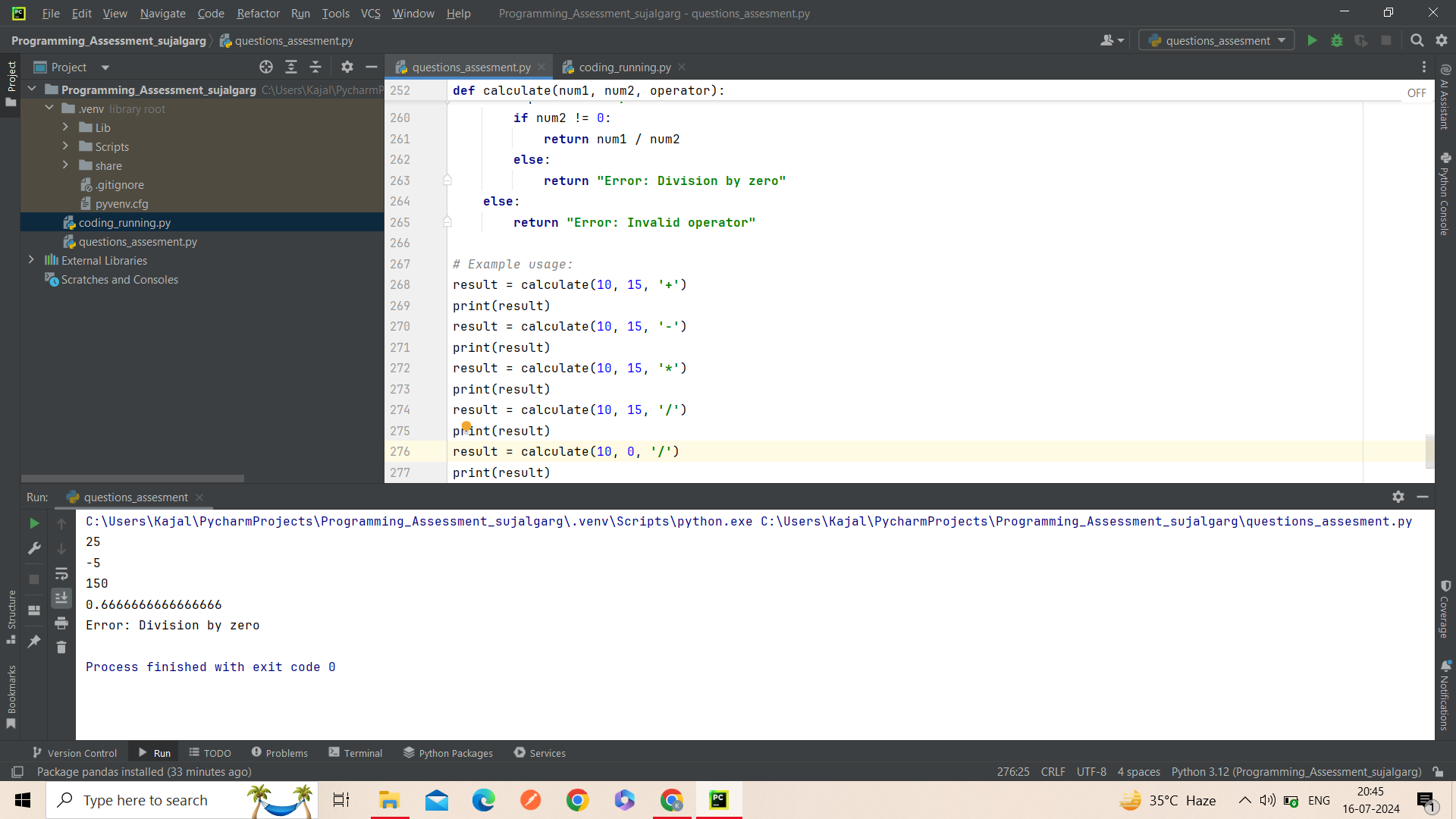
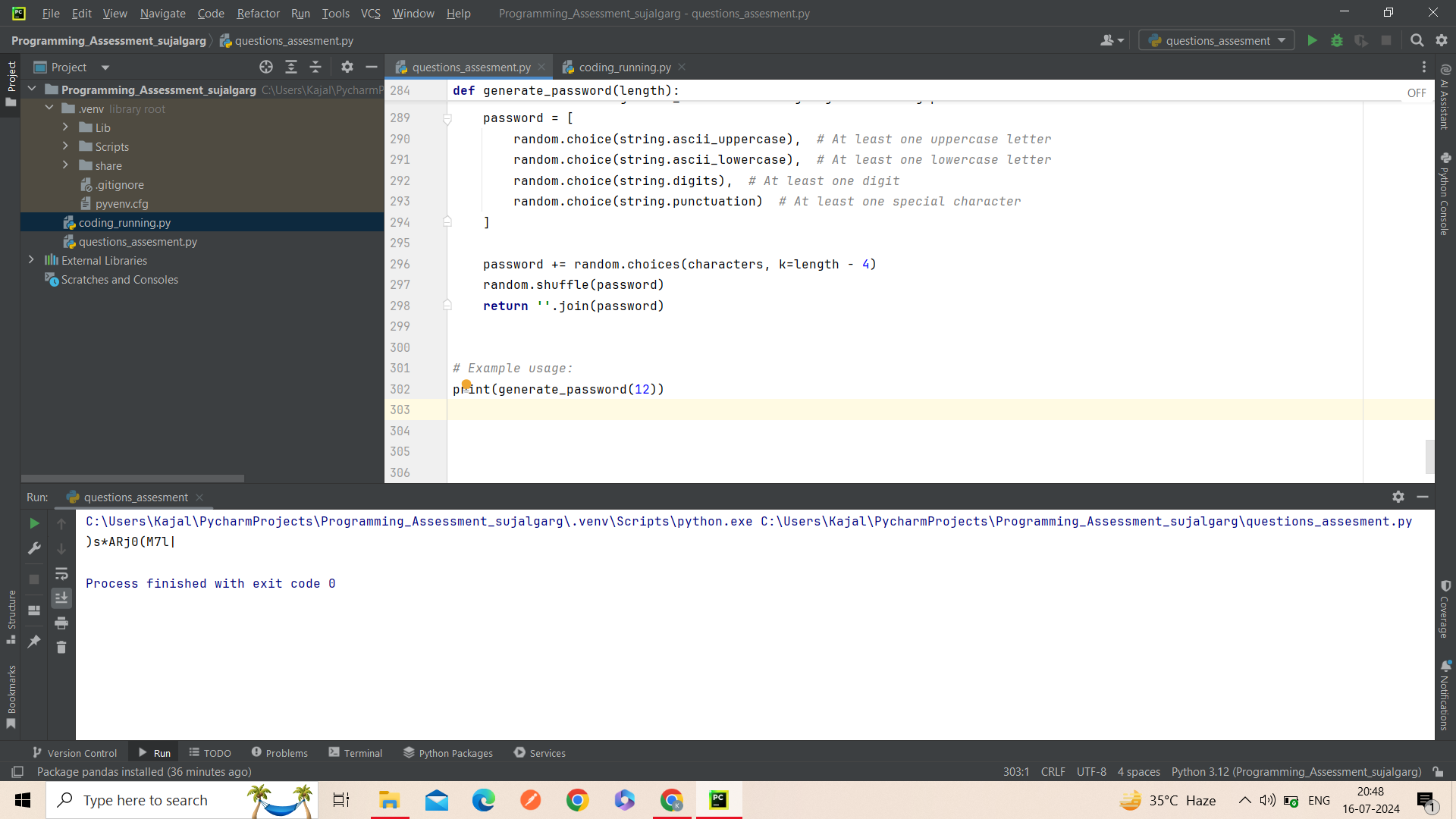
question no.1  
**class** MaxHeap:  
 **def** \_\_init\_\_(self):  
 self.heap = []  
  
 **def** insert(self, key):  
 self.heap.append(key)  
 self.\_heapify\_up(len(self.heap) - 1)  
  
 **def** delete(self):  
 **if** len(self.heap) == 0:  
 **raise** IndexError(**"Heap is empty"**)  
  
 max\_value = self.heap[0]  
 self.heap[0] = self.heap[-1]  
 **del** self.heap[-1]  
  
 **if** len(self.heap) > 0:  
 self.\_heapify\_down(0)  
  
 **return** max\_value  
  
 **def** get\_max(self):  
 **if** len(self.heap) == 0:  
 **raise** IndexError(**"Heap is empty"**)  
  
 **return** self.heap[0]  
  
 **def** \_heapify\_up(self, index):  
 parent\_index = (index - 1) // 2  
 **while** parent\_index >= 0 **and** self.heap[parent\_index] < self.heap[index]:  
 *# Swap parent and current node* self.heap[parent\_index], self.heap[index] = self.heap[index], self.heap[parent\_index]  
 index = parent\_index  
 parent\_index = (index - 1) // 2  
  
 **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:  
**from** questions\_assesment **import** MaxHeap  
  
heap = MaxHeap()  
heap.insert(23)  
heap.insert(11)  
heap.insert(1)  
print(heap.get\_max())  
print(heap.delete())  
print(heap.get\_max())

  
  
question no.2  
**import** requests  
**from** requests.exceptions **import** RequestException, Timeout  
  
  
**def** download\_urls(urls):  
 results = []  
  
 **for** url **in** urls:  
 retry\_count = 0  
 success = **False  
  
 while** retry\_count < 3 **and not** success:  
 **try**:  
 response = requests.get(url, timeout=5)  
 response.raise\_for\_status() *# Raise error for bad response status* results.append(response.content)  
 success = **True** print(**f"Downloaded content from {**url**}"**)  
 **except** Timeout:  
 print(**f"Timeout error occurred for {**url**}. Retrying..."**)  
 retry\_count += 1  
 **except** RequestException **as** e:  
 print(**f"An error occurred for {**url**}: {**e**}. Retrying..."**)  
 retry\_count += 1  
  
 **if not** success:  
 print(**f"Failed to download content from {**url**} after {**retry\_count**} retries."**)  
  
 **return** results  
  
  
Example usage:  
urls\_to\_download = [  
 **'https://example.com/page1'**,  
 **'https://example.com/page2'**,  
 **'https://example.com/page3'**,  
]  
  
downloaded\_content = download\_urls(urls\_to\_download)  
  
  
  
question.3  
*# Import necessary libraries***import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** creative.datasets **import** load\_boston  
**from** creative.model\_selection **import** train\_test\_split  
**from** creative.linear\_model **import** LinearRegression  
**from** creative.metrics **import** mean\_squared\_error, r2\_score  
  
*# Load the Boston Housing dataset*boston = load\_boston()  
  
*# Split the data into training and testing sets (80% train, 20% test)*X\_train, X\_test, y\_train, y\_test = train\_test\_split(boston.data, boston.target, test\_size=0.2, random\_state=42)  
  
*# Initialize the linear regression model*model = LinearRegression()  
  
*# Train the model using the training sets*model.fit(X\_train, y\_train)  
  
*# Make predictions using the testing set*y\_pred = model.predict(X\_test)  
  
*# Print model performance*print(**"Coefficients:"**, model.coef\_)  
print(**"Mean Squared Error:"**, mean\_squared\_error(y\_test, y\_pred))  
print(**"R-squared:"**, r2\_score(y\_test, y\_pred))  
  
*# Plot predicted vs actual values*plt.scatter(y\_test, y\_pred)  
plt.xlabel(**"Actual Prices"**)  
plt.ylabel(**"Predicted Prices"**)  
plt.title(**"Actual Prices vs Predicted Prices"**)  
plt.show()  
  
  
question no.4  
**import** pandas **as** pd  
**from** sklearn.preprocessing **import** StandardScaler  
  
  
**def** preprocess\_data(df):  
 *# Handling missing values* df = handle\_missing\_values(df)  
  
 *# Normalizing numerical columns* df = normalize\_numerical\_columns(df)  
  
 *# Encoding categorical columns* df = encode\_categorical\_columns(df)  
  
 **return** df  
  
  
**def** handle\_missing\_values(df):  
 *# Handling missing values in numerical columns by imputing with mean* numerical\_cols = df.select\_dtypes(include=[**'number'**]).columns  
 df[numerical\_cols] = df[numerical\_cols].fillna(df[numerical\_cols].mean())  
  
 *# Handling missing values in categorical columns by imputing with mode* categorical\_cols = df.select\_dtypes(include=[**'object'**]).columns  
 df[categorical\_cols] = df[categorical\_cols].fillna(df[categorical\_cols].mode().iloc[0])  
  
 **return** df  
  
  
**def** normalize\_numerical\_columns(df):  
 *# Standardize numerical columns (mean=0, std=1)* scaler = StandardScaler()  
 numerical\_cols = df.select\_dtypes(include=[**'number'**]).columns  
 df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])  
  
 **return** df  
  
  
**def** encode\_categorical\_columns(df):  
 *# One-hot encode categorical columns* df = pd.get\_dummies(df, columns=df.select\_dtypes(include=[**'object'**]).columns)  
  
 **return** df  
  
*# Example usage:  
# Assuming 'df' is your DataFrame to be cleaned and preprocessed*df = preprocess\_data(df)  
  
question no.5  
**def** fibonacci(n):  
 *# Base cases* **if** n <= 0:  
 **return** 0  
 **elif** n == 1:  
 **return** 1  
 **else**:  
 *# Recursive case* **return** fibonacci(n-1) + fibonacci(n-2)  
  
  
*# Example usage:*n = 8  
result = fibonacci(n)  
print(**f"The {**n**}th Fibonacci number is: {**result**}"**)  
  
  
  
question no.6  
**def** divide\_numbers(dividend, divisor):  
 **try**:  
 result = dividend / divisor  
 **except** ZeroDivisionError:  
 **return "Error: Division by zero is not allowed."  
 else**:  
 **return** result  
  
*# Example usage:*result1 = divide\_numbers(10, 2)  
print(result1)  
  
result2 = divide\_numbers(8, 0)  
print(result2)  
  
  
question no.7  
**import** time  
**import** functools  
**import** logging  
  
*# Configure logging (optional)*logging.basicConfig(level=logging.INFO)  
  
**def** measure\_time(func):  
 @functools.wraps(func)  
 **def** wrapper\_measure\_time(\*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\_time  
  
*# Example of a computationally expensive function*@measure\_time  
**def** compute\_factorial(n):  
 factorial = 1  
 **for** i **in** range(1, n + 1):  
 factorial \*= i  
 **return** factorial  
  
*# Example usage:*number = 100  
result = compute\_factorial(number)  
print(**f"The factorial of {**number**} is: {**result**}"**)



question no.8  
**def** calculate(num1, num2, operator):  
 **if** operator == **'+'**:  
 **return** num1 + num2  
 **elif** operator == **'-'**:  
 **return** num1 - num2  
 **elif** operator == **'\*'**:  
 **return** num1 \* num2  
 **elif** operator == **'/'**:  
 **if** num2 != 0:  
 **return** num1 / num2  
 **else**:  
 **return "Error: Division by zero"  
 else**:  
 **return "Error: Invalid operator"***# Example usage:*result = calculate(10, 15, **'+'**)  
print(result)  
result = calculate(10, 15, **'-'**)  
print(result)  
result = calculate(10, 15, **'\*'**)  
print(result)  
result = calculate(10, 15, **'/'**)  
print(result)  
result = calculate(10, 0, **'/'**)  
print(result)

  
  
question no.9  
**import** random  
**import** string  
  
  
**def** generate\_password(length):  
 **if** length < 4:  
 **return "Error: Password length should be at least 4"** characters = string.ascii\_letters + string.digits + string.punctuation  
 password = [  
 random.choice(string.ascii\_uppercase), *# At least one uppercase letter* random.choice(string.ascii\_lowercase), *# At least one lowercase letter* random.choice(string.digits), *# At least one digit* random.choice(string.punctuation) *# At least one special character* ]  
  
 password += random.choices(characters, k=length - 4)  
 random.shuffle(password)  
 **return ''**.join(password)  
  
  
*# Example usage:*print(generate\_password(12))

  
  
question no.10  
**def** transpose\_matrix(matrix):  
 *# Use zip and \* to unpack and transpose the matrix* transposed = [list(row) **for** row **in** zip(\*matrix)]  
 **return** transposed  
  
*# Example usage:*matrix = [  
 [1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]  
]  
  
transposed\_matrix = transpose\_matrix(matrix)  
print(transposed\_matrix)

