Question 1 : What is the difference between multithreading and multiprocessing?
Answer: 1. Multithreading:
- Multiple threads within a single process.
- Threads share the same memory space.
- Lightweight and faster to create.
- Limited by the Global Interpreter Lock (GIL) in Python, which can hinder performance for CPU-bound tasks.

2. Multiprocessing:
- Multiple processes, each with its own memory space.
- Processes do not share memory; data must be explicitly shared using inter-process communication (IPC) mechanisms.
- Heavier and slower to create compared to threads.
- Can fully utilize multiple CPU cores, making it suitable for CPU-bound tasks.

Question 2: What are the challenges associated with memory management in Python?
Answer: Challenges in Memory Management in Python:

1. Memory Leaks: Python's garbage collector may not always be able to detect circular references, leading to memory leaks.
2. Global Interpreter Lock (GIL): The GIL can introduce performance bottlenecks and limit the effectiveness of multithreading in CPU-bound tasks.
3. Memory Fragmentation: Frequent allocation and deallocation of memory can lead to fragmentation, reducing memory efficiency.
4. Reference Counting: Python's reference counting mechanism can lead to issues with cyclic garbage collection.
5. Large Data Structures: Handling large data structures can lead to memory issues if not managed properly.
6. External Resources: Managing external resources, such as file handles or network connections, requires careful handling to avoid memory leaks.

Question 3:Write a Python program that logs an error message to a log file when a division by zero exception occurs.
Answer: Logging Division by Zero Error:

```
import logging
 Configure logging
logging.basicConfig(filename='error.log', level=logging.ERROR)
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        logging.error("Division by zero error occurred.")
    else:
        return result
```

Test the function
divide(10, 0)
Log file (error.log):

```
ERROR:root:Division by zero error occurred.
```

Question 4:Write a Python program that reads from one file and writes its content to another file.
Answer: File Copy Program:

```
def copy_file(source_file, target_file):
    try:
        with open(source_file, 'r') as source:
            content = source.read()
        with open(target_file, 'w') as target:
            target.write(content)
        print(f"Content copied from {source_file} to {target_file} successfully.")
    except FileNotFoundError:
        print(f"File {source_file} not found.")

# Usage
source_file = 'source.txt'
target_file = 'target.txt'
copy_file(source_file, target_file)
```

Question 5: Write a program that handles both IndexError and KeyError using a try-except block.
Answer: Handling IndexError and KeyError:

```
def access_data(data, index=None, key=None):
    try:
        if isinstance(data, list) and index is not None:
            print(data[index])
        elif isinstance(data, dict) and key is not None:
            print(data[key])
        else:
            print("Invalid data type or missing index/key.")
    except IndexError:
        print("Index out of range.")
    except KeyError:
        print("Key not found.")

# Test the function
my_list = [1, 2, 3]
my_dict = {'a': 1, 'b': 2}access_data(my_list, 5)  # IndexError
```

```
access_data(my_dict, key='c')  # KeyError
access_data(my_list, 1)  # Valid index
access_data(my_dict, key='a')  # Valid key
```

Output:
Index out of range.
Key not found.
2
1

Question 6: What are the differences between NumPy arrays and Python lists?
Answer: NumPy Arrays vs Python Lists:

Differences:

1. Data Type: NumPy arrays are homogeneous (single data type), while Python lists are heterogeneous (multiple data types).
2. Memory: NumPy arrays are more memory-efficient due to their homogeneous nature.
3. Performance: NumPy arrays are faster for numerical computations due to vectorized operations.
4. Operations: NumPy arrays support element-wise operations and matrix operations.
5. Dimensions: NumPy arrays have built-in support for multi-dimensional arrays.

Question 7:Explain the difference between apply() and map() in Pandas.
Answer: Pandas Apply() vs Map():

Differences:

1. Purpose:
   - apply(): Applies a function to each row or column of a DataFrame.
   - map(): Applies a function element-wise to a Series.
2. Operation:
   - apply(): Can perform complex operations, including those that involve multiple columns or rows.
   - map(): Limited to element-wise operations.
3. Performance:
   - map(): Generally faster than apply() for simple operations.
   - apply(): Can be slower due to its flexibility and complexity.
4. Usage:
   - apply(): Often used for data transformation, aggregation, and feature engineering.
   - map(): Used for simple transformations, such as replacing values or performing element-wise operations.

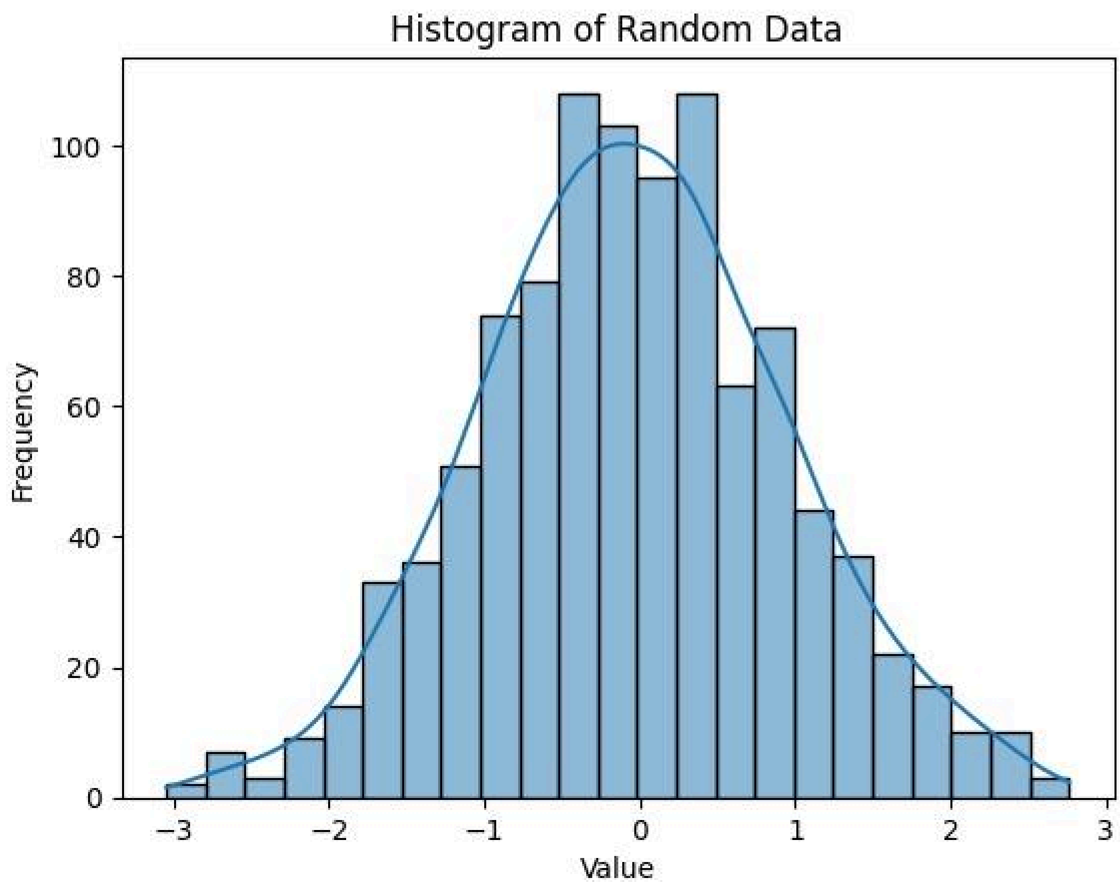
Question 8: Create a histogram using Seaborn to visualize a distribution.
Answer: import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Generate sample data
np.random.seed(0)
data = np.random.randn(1000)

# Create a histogram
sns.histplot(data, kde=True)

# Customize the plot
plt.title('Histogram of Random Data')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Show the plot
plt.show()



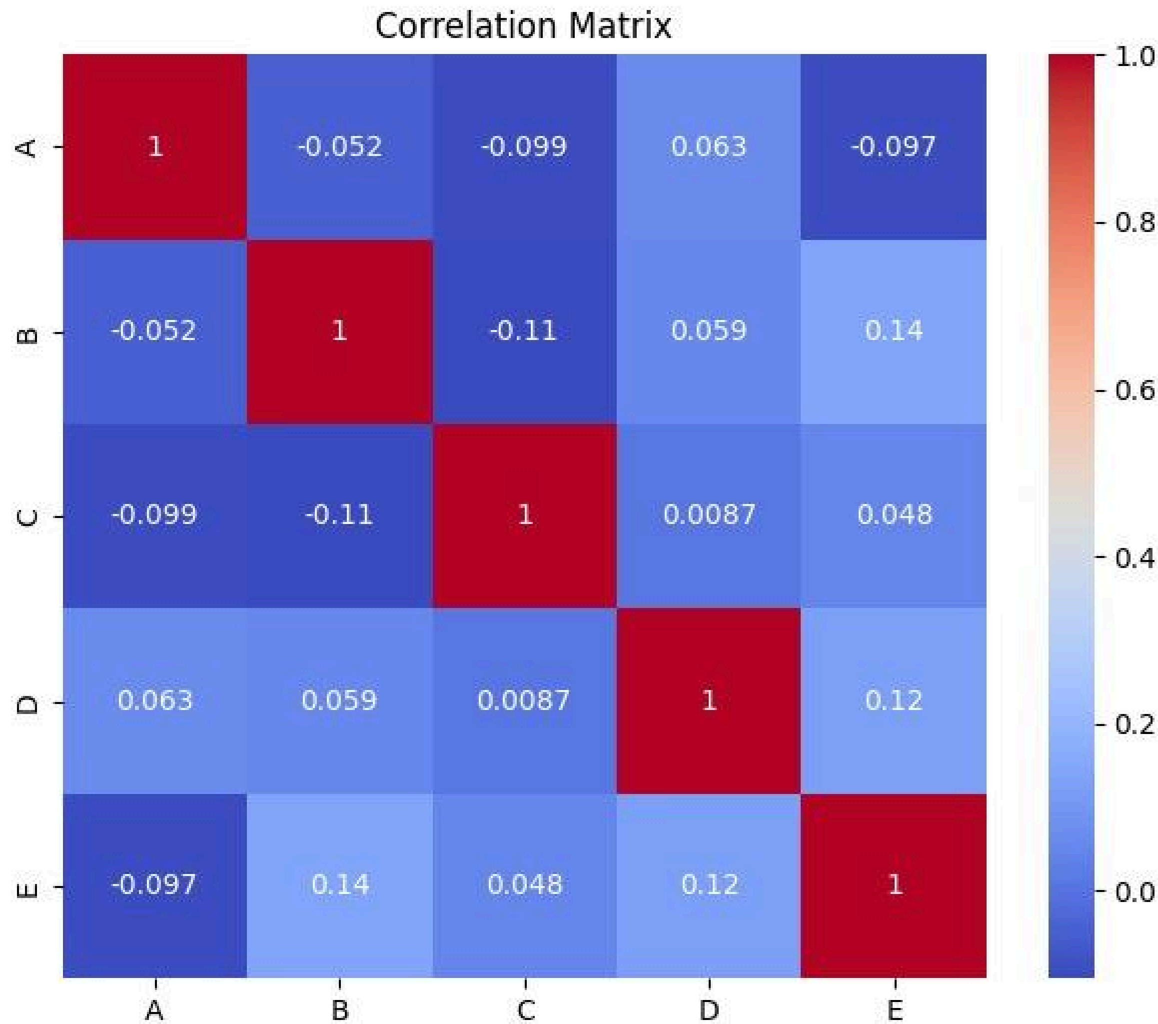
Question 9: Use Pandas to load a CSV file and display its first 5 rows.
Answer: import pandas as pd

```
# Load the CSV file
data = pd.read_csv("services.csv")

# Display the first 5 rows in tabular format
print("First 5 rows of the CSV file:\n")
print(data.head().to_string(index=False))
```

id	location_id	program_id	accepted_payments	alternate_name	application_process
1	1	NaN	NaN	NaN	Walk in or apply by phone.
2	2	NaN	NaN	NaN	Apply by phone for an appointment.
3	3	NaN	NaN	NaN	Phone for information (403-4300 Ext. 4322).
4	4	NaN	NaN	NaN	Apply by phone.
5	5	NaN	NaN	NaN	Phone for information.

Question 10: Calculate the correlation matrix using Seaborn and visualize it with a heatmap.
Answer: import seaborn as sns

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate sample data
np.random.seed(0)
data = np.random.randn(100, 5)
df = pd.DataFrame(data, columns=['A', 'B', 'C', 'D', 'E'])

# Calculate correlation matrix
corr_matrix = df.corr()

# Create heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', square=True)

# Set title
plt.title('Correlation Matrix')

# Show plot
plt.show()
```



Correlation Matrix
A
B
C
D
E
1
-0.052
-0.099
0.063
-0.097
-0.052
1
-0.11
0.059
0.14
-0.099
-0.11
1
0.0087
0.048
0.063
0.059
0.0087
1
0.12
-0.097
0.14
0.048
0.12
1
A
B
C
D
E
1.0
0.8
0.6
0.4
0.2
0.0