Q.1 What is the difference between multithreading and multiprocessing?

Ans The main difference between **multithreading** and **multiprocessing** lies in how they handle tasks and system resources. **Multithreading** involves running multiple threads within a single process, where all threads share the same memory space. It allows tasks to run concurrently, making it suitable for I/O-bound operations like reading files or handling network requests. However, since threads share memory, they can interfere with each other if not managed properly. On the other hand, **multiprocessing** uses multiple processes, each with its own memory space and system resources. This approach is ideal for CPU-bound tasks, as processes can run truly in parallel on multiple CPU cores without affecting each other. While multiprocessing offers better performance for heavy computations, it consumes more memory compared to multithreading.

Q.2 What are the challenges associated with memory management in Python?

Ans. Memory management in Python faces several challenges due to its dynamic nature and automatic garbage collection. One major challenge is **memory leaks**, which occur when unused objects remain referenced, preventing the garbage collector from freeing them. Another issue is **fragmentation**, where frequent allocation and deallocation of memory create scattered free spaces, reducing efficiency. Python also uses **reference counting**, which can lead to problems with **circular references**—objects referring to each other—making them harder to clean up automatically. Additionally, **inefficient use of large data structures** like lists or dictionaries can consume excessive memory. Finally, Python's **Global Interpreter Lock (GIL)** can limit true parallelism in memory operations, especially in multithreaded programs. Managing these issues often requires careful coding, use of memory profiling tools, and optimizing data structures.

Q.3 Write a Python program that logs an error message to a log file when a division by zero exception occurs

Ans.
```
import logging


# Configure logging

logging.basicConfig(filename='error.log', level=logging.ERROR,

                    format='%(asctime)s - %(levelname)s - %(message)s')


try:

    # Example division

    numerator = 10

    denominator = 0

    result = numerator / denominator

except ZeroDivisionError as e:

    logging.error("Division by zero error occurred: %s", e)

    print("An error occurred. Check 'error.log' for details.")
```

Q.4 Write a Python program that reads from one file and writes its content to another file.

Ans. # Program to read from one file and write its content to another file

```
# Specify file names
source_file = "source.txt"
destination_file = "destination.txt"

try:
   # Open the source file in read mode
   with open(source_file, 'r') as src:
      content = src.read()

   # Open the destination file in write mode
   with open(destination_file, 'w') as dest:
      dest.write(content)

   print("File content copied successfully!")

except FileNotFoundError:
   print("Error: The source file was not found.")
except IOError:
   print("Error: There was an issue reading or writing the file."
```

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

Ans # Program to handle IndexError and KeyError

```
try:
   # Example list and dictionary
   numbers = [10, 20, 30]
   data = {"name": "Alice", "age": 25}
```

```
    # Intentionally causing errors
    print(numbers[5])       # This will raise IndexError
    print(data["address"])   # This will raise KeyError

except IndexError:
    print("Error: List index out of range. Please check your list index.")
except KeyError:
    print("Error: The specified key was not found in the dictionary.")
```

Q.6 What are the differences between NumPy arrays and Python lists?

Ans Here are the main differences between **NumPy arrays** and **Python lists** explained clearly:

- **1. Data type:**
  Python lists can store elements of different data types (e.g., integers, strings, floats), while NumPy arrays store elements of the **same data type**, making them more efficient for numerical computations.
- **2. Performance:**
  NumPy arrays are **faster and more memory-efficient** because they use fixed-type data and are implemented in **C**, whereas Python lists are slower and use more memory.
- **3. Operations:**
  Mathematical operations can be performed **directly on NumPy arrays** (like a + b or a * 2), while Python lists require explicit loops or list comprehensions.
- **4. Functionality:**
  NumPy provides a wide range of **built-in mathematical, statistical, and linear algebra functions** that are not available for lists.
- **5. Memory storage:**
  NumPy arrays use **contiguous blocks of memory**, which improves processing speed, while Python lists store references to objects, leading to more overhead.
- **6. Dimensionality:**
  NumPy supports **multi-dimensional arrays (matrices, tensors)**, while Python lists are primarily one-dimensional (though nested lists can mimic multi-dimensional arrays).

Q.7 Explain the difference between apply() and map() in Pandas.

Ans The main difference between **apply()** and **map()** in **Pandas** lies in their **scope** and **usage**:

- **1. Scope of operation:**
  - **map()** works **only on a Pandas Series** (a single column). It applies a function, dictionary, or mapping to each element of that Series.

    - **apply()** works on **both Series and DataFrames**. When used on a DataFrame, it can apply a function to **each row or column**.
- **2. Flexibility:**
    - **map()** is simpler and mainly used for **element-wise transformations**.
    - **apply()** is more **flexible and powerful** — it can apply complex functions that act on entire rows, columns, or even return new Series or DataFrames.

Q.8 : Create a histogram using Seaborn to visualize a distribution.

Ans import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = [12, 15, 20, 22, 25, 25, 28, 30, 32, 35, 36, 38, 40, 42, 45, 47, 50, 52, 55, 58]

# Create a histogram

sns.histplot(data, bins=8, kde=True, color='skyblue')

# Add labels and title

plt.title("Distribution of Data")

plt.xlabel("Value")

plt.ylabel("Frequency")

# Show the plot

plt.show()

Q.9 Use Pandas to load a CSV file and display its first 5 rows.

Ans import pandas as pd

# Load the CSV file

df = pd.read_csv("data.csv")   # Replace 'data.csv' with your file name or path

# Display the first 5 rows

```
print(df.head())
```

Q.10 : Calculate the correlation matrix using Seaborn and visualize it with a heatmap.

Ans