In [13]:
```
f = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
print(f.read())
```

```
Hello! Welcome to Sanjivani.txt
This file is for testing purposes.
Good Luck!
```

In [16]:
```
f = open("D:\\Cyber.txt", "r")
print(f.read())
```

In [17]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
content = file.read()
print(content)
file.close()
```

```
Hello! Welcome to Sanjivani.txt
This file is for testing purposes.
Good Luck!
```

Writing to a File Writing to a file is done using file.write() which writes the specified string to the file. If the file exists, its content is erased. If it doesn't exist, a new file is created.

In [ ]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "w")
file.write("Hello, World!")
file.close()
```

Writing to a File in Append Mode (a) It is done using file.write() which adds the specified string to the end of the file without erasing its existing content.

In [ ]:
```
# Python code to illustrate append() mode
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", 'a')
file.write("This will add this line")
file.close()
```

Closing a File Closing a file is essential to ensure that all resources used by the file are properly released. file.close() method closes the file and ensures that any changes made to the file are saved.

In [ ]:
```
file = open("C://Users//Asus//OneDrive//Desktop\\Sanjivani.txt", "r")
# Perform file operations
file.close()
```

Advantages of File Handling in Python Versatility : File handling in Python allows us to perform a wide range of operations, such as creating, reading, writing, appending, renaming and deleting files. Flexibility : File handling in Python is highly flexible, as it allows us to work with different file types (e.g. text files, binary files, CSV files , etc.) and to perform different operations on files (e.g. read, write, append, etc.). User – friendly : Python provides a user-friendly interface for file handling, making it easy to create, read and manipulate files. Cross-platform : Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.Disadvantages of File Handling in Python Error-prone: File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.). Security risks : File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system. Complexity : File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely. Performance : File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.