# Training Report Day-17

## 25 June 2024

### Introduction to File Handling

File handling is an essential aspect of programming that involves reading from and writing to files. Python provides built-in functions and modules to handle files, allowing you to create, read, write, and manipulate files in various ways.

It is a powerful and versatile tool that can be used to perform a wide range of operations. However, it is important to carefully consider the advantages and disadvantages of file handling when writing Python programs, to ensure that the code is secure, reliable, and performs well.

Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters, and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let’s start with the reading and writing files.

**Key Concepts**

**File Modes**: Modes specify the purpose of opening a file. Common modes include:

-'r': Read (default mode). Opens a file for reading.

-'w': Write. Opens a file for writing (creates a new file or truncates an existing file).

-'a': Append. Opens a file for appending (creates a new file if it doesn't exist).

-'b': Binary. Used with other modes to handle binary files.

-'+': Update. Opens a file for reading and writing.

**File Methods:**

open(): Opens a file and returns a file object.

read(): Reads the entire content of a file.

readline(): Reads a single line from a file.

readlines(): Reads all lines from a file and returns them as a list.

write(): Writes a string to a file.

writelines(): Writes a list of strings to a file.

close(): Closes the file.

**With Statement:** Ensures proper acquisition and release of resources, automatically closing the file when the block inside the with statement is exited.

## Python's "with open(…) as …" Pattern

Reading and writing data to files using Python is pretty straightforward. To do this, you must first open files in the appropriate mode. Here's an example of how to use Python's "with open(…) as …" pattern to open a text file and read its contents:

```
with open('data.txt', 'r') as f:
    data = f.read()
with open('data.txt', 'w') as f:
    data = 'some data to be written to the file'
    f.write(data)
```

```
def write_binary_file(file_path, data):
    try:
        with open(file_path, 'wb') as file:
            file.write(data)
            return "Write successful"
    except IOError as e:
        return f"IOError: {e}"

def read_binary_file(file_path):
    try:
```

```
        with open(file_path, 'rb') as file:
            data = file.read()
            return data
    except FileNotFoundError as e:
        return f"FileNotFoundError: {e}"
    except IOError as e:
        return f"IOError: {e}"

# Usage
binary_data = bytes([104, 101, 108, 108, 111])  # Binary data for "hello"
write_result = write_binary_file("example.bin", binary_data)
print(write_result)

read_result = read_binary_file("example.bin")
print(read_result)
```

**Advantages of File Handling in Python**

**Versatility:** File handling in Python allows you 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 you 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.

```
def read_file_by_line(file_path):
    try:
        with open(file_path, 'r') as file:
            for line in file:
                print(line.strip())
    except FileNotFoundError as e:
        print(f"FileNotFoundError: {e}")
    except IOError as e:
        print(f"IOError: {e}")

# Usage
read_file_by_line("/content/eid.txt")  # Output: (each line of example.txt)
```