Section 6

PL-3

T.A: Shereen El-gazzar

fold in Functional Programming:

What is fold ?

- A higher-order function that reduces a collection (list, array, etc.) to a single value.
- Combines elements using a provided function and an initial value.

Example: Sum of Numbers

```
let numbers = [1; 2; 3; 4]
let sum = List.fold (fun acc x -> acc + x) 0 numbers
printfn "%d" sum // Output: 10
```

Example: Product of Numbers

```
let numbers = [2; 3; 4]
let product = List.fold (fun acc x -> acc * x) 1 numbers
printfn "%d" product // Output: 24
```

Why Do We Need an Initial Value in fold ?!

The **initial value** is essential for:

1. **Starting the computation.**
2. **Handling empty lists** gracefully.
3. **Determining the result type.**

Behavior with Empty Lists:

- If the list is empty, fold returns the initial value directly.

```
let emptyList = []
let sum = List.fold (fun acc x -> acc + x) 0 emptyList
printfn "%d" sum // Output: 0
```

Choosing the Initial Value:

Operation	Initial Value
Sum or Subtraction	`0`
Multiplication	`1`
Creating a New List	Empty list `[]`

What is map
“ transforming Data ” ?

map → Applies a function to each element of a collection and returns a new collection with the transformed elements.

```
List.map (fun x -> transformation) collection
```

Example: Doubling Numbers

```
let numbers = [1; 2; 3; 4]
let doubledNumbers = List.map (fun x -> x * 2) numbers
printfn "%A" doubledNumbers // Output: [2; 4; 6; 8]
```

Visualizing map:

Original List	Function	Resulting List
`[1; 2; 3; 4]`	`x * 2`	`[2; 4; 6; 8]`

What is filter “Selecting Data ” ?

- Keep only the elements that satisfy a condition.

```
List.filter (fun x -> condition) collection
```

Example: Filtering Even Numbers

```
let numbers = [1; 2; 3; 4; 5]
let evenNumbers = List.filter (fun x -> x % 2 = 0) numbers
printfn "%A" evenNumbers // Output: [2; 4]
```

Visualizing filter:

Original List	Condition	Resulting List
`[1; 2; 3; 4; 5]`	`x % 2 = 0`	`[2; 4]`

What is the Pipeline Operator “|>” ?

A tool for **chaining functions** in a clean and readable way.

Passes the **result** of one operation as the **input** to the next function.

```
input |> function
```

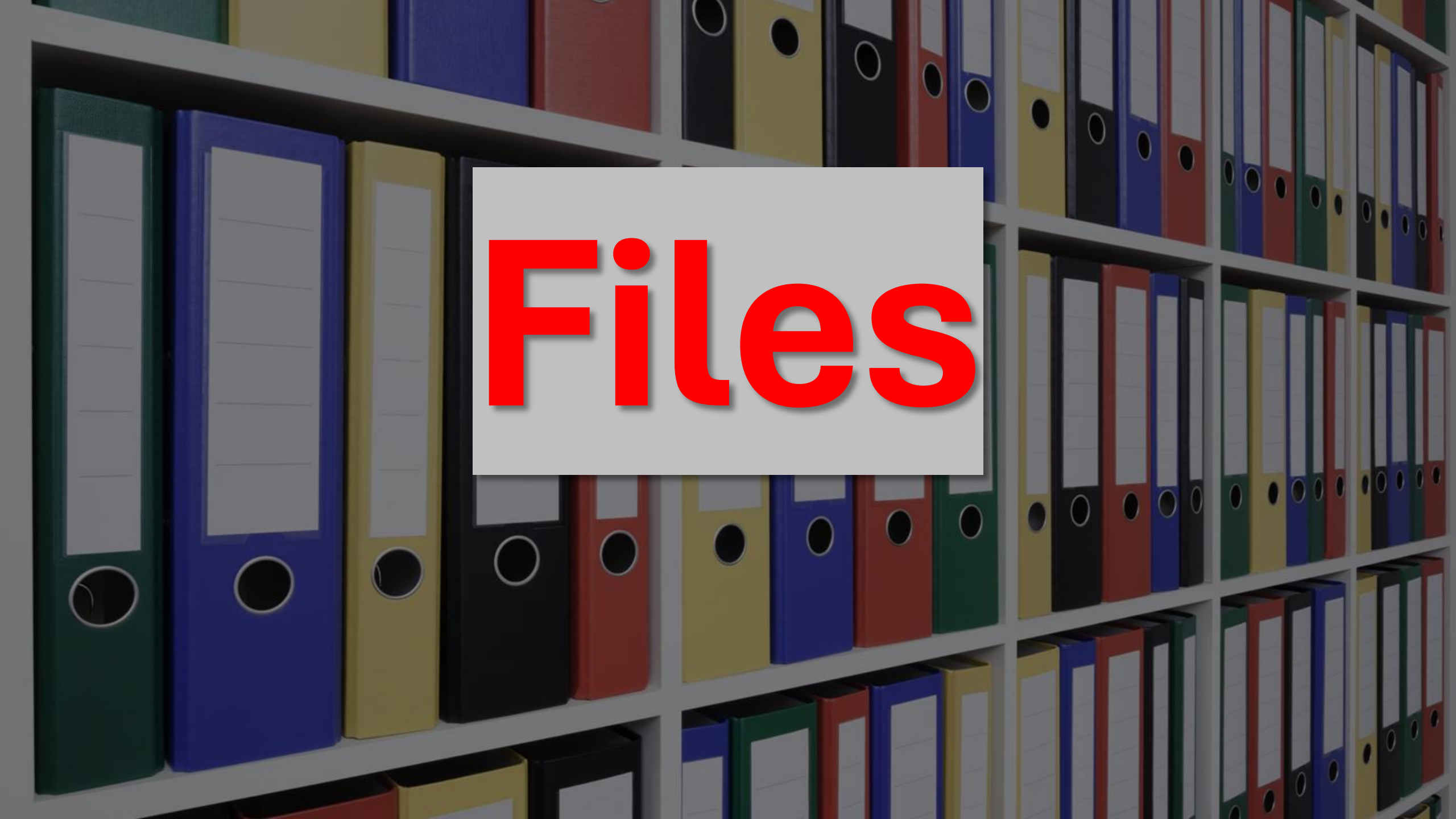
Example Without and With |> :

- Without pipeline:

```
let result = List.map (fun x -> x * 2) (List.filter (fun x -> x % 2 = 0) [1; 2; 3; 4; 5])
printfn "%A" result // Output: [4; 8]
```

- With pipeline:

```
let result =
    [1; 2; 3; 4; 5]
    |> List.filter (fun x -> x % 2 = 0)  // Step 1: Filter even numbers
    |> List.map (fun x -> x * 2)         // Step 2: Double the numbers
printfn "%A" result // Output: [4; 8]
```


Files

Reading Text from a File:

```
open System.IO

let readFile filePath =
    try
        let content = File.ReadAllText(filePath)
        printfn "File content: %s" content
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
readFile "sample.txt"
```

Reading File Line by Line

```
let readLines filePath =
    try
        let lines = File.ReadLines(filePath)
        for line in lines do
            printfn "%s" line
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
readLines "sample.txt"
```

Writing to a File

```
let writeFile filePath content =
    try
        File.WriteAllText(filePath, content)
        printfn "Data written to file successfully"
    with
    | ex -> printfn "Error: %s" ex.Message

// Example usage
writeFile "output.txt" "This is some text"
```

Appending Text to a File

```
let appendToFile filePath content =
    try
        File.AppendAllText(filePath, content)
        printfn "Data appended to file successfully"
    with
    | ex -> printfn "Error: %s" ex.Message

// Example usage
appendToFile "output.txt" "\nAppended text."
```

Reading File with StreamReader:

- **Regular reading**:
  Loads the entire content of the file into memory at once.

- **Stream-based reading**:
  Reads the file gradually, without loading it entirely into memory.

  This approach saves memory usage and improves efficiency when working with large files.

Reading File with StreamReader

```
open System.IO

let readFileWithStreamReader filePath =
    try
        use reader = new StreamReader(filePath)
        let content = reader.ReadToEnd()
        printfn "File content: %s" content
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
readFileWithStreamReader "sample.txt"
```

Using use vs Not Using use in F#:

- **Using use:**

  Automatically disposes of resources, ensuring safe and clean resource management.

  →No need to manually close the resource.

- **Without use:**

  Requires explicit disposal; and can lead to errors or memory leaks if not handled correctly.

Without use keyword:

```
open System.IO

let readFileWithoutStreamReader filePath =
    try
        // Manually manage the resource
        let reader = new StreamReader(filePath)

        while not reader.EndOfStream do
            let line = reader.ReadLine()
            printfn "%s" line

        // Manually closing the resource
        reader.Close()
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message
```

Without use keyword:

```
open System.IO

let readFileWithoutStreamReader filePath =
    try
        // Manually manage the resource
        let reader = new StreamReader(filePath)

        while not reader.EndOfStream do
            let line = reader.ReadLine()
            printfn "%s" line

        // Manually closing the resource
        reader.Close()
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message
```

Writing to a File with StreamWriter

```
let writeFileWithStreamWriter filePath content =
    try
        use writer = new StreamWriter(filePath)
        writer.Write(content)
        printfn "Data written to file using StreamWriter"
    with
    | ex -> printfn "Error: %s" ex.Message


// Example usage
writeFileWithStreamWriter "output_stream.txt" "This is written using StreamWriter"
```

Reading with Specific Encoding

```
let readFileWithEncoding filePath =
    try
        use reader = new StreamReader(filePath, System.Text.Encoding.UTF8)
        let content = reader.ReadToEnd()
        printfn "File content with UTF-8 encoding: %s" content
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
readFileWithEncoding "sample_utf8.txt"
```

Check if File or Directory Exists

```
let checkIfFileExists filePath =
    if File.Exists(filePath) then
        printfn "File exists"
    else
        printfn "File does not exist"

let checkIfDirectoryExists directoryPath =
    if Directory.Exists(directoryPath) then
        printfn "Directory exists"
    else
        printfn "Directory does not exist"

// Example usage
checkIfFileExists "sample.txt"
checkIfDirectoryExists "sample_directory"
```

Check if File or Directory Exists

```
let checkIfFileExists filePath =
    if File.Exists(filePath) then
        printfn "File exists"
    else
        printfn "File does not exist"

let checkIfDirectoryExists directoryPath =
    if Directory.Exists(directoryPath) then
        printfn "Directory exists"
    else
        printfn "Directory does not exist"

// Example usage
checkIfFileExists "sample.txt"
checkIfDirectoryExists "sample_directory"
```

List Files in a Directory:

```
let listFilesInDirectory directoryPath =
    try
        let files = Directory.GetFiles(directoryPath)
        for file in files do
            printfn "File: %s" file
    with
    | :? DirectoryNotFoundException -> printfn "Directory not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
listFilesInDirectory "some_directory"
```

List Files in a Directory:

```
let listFilesInDirectory directoryPath =
    try
        let files = Directory.GetFiles(directoryPath)
        for file in files do
            printfn "File: %s" file
    with
    | :? DirectoryNotFoundException -> printfn "Directory not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
listFilesInDirectory "some_directory"
```

Searching for a Term in a File Using F#:

```
open System.IO

let searchInFile filePath searchTerm =
    try
        use reader = new StreamReader(filePath)
        while not reader.EndOfStream do
            let line = reader.ReadLine()
            if line.Contains(searchTerm) then
                printfn "Found: %s" line
    with
    | :? FileNotFoundException -> printfn "File not found"
    | ex -> printfn "Error: %s" ex.Message

// Example usage
searchInFile "example.txt" "F#"
```


Any Questions?!

Thank YOU !