Section-2

T.A : Shereen el-gazzar

F# :

Originally developed at the University of Oxford,
it is deeply integrated with functional programming paradigms.

Part of .NET Family: F# is part of the .NET ecosystem,
enabling smooth interoperability with other .NET languages like C#
and VB.NET.

F#:

- **Fully Functional Language:** F# is designed primarily as a functional programming language, emphasizing the use of functions and supporting concepts like pure functions and lazy evaluation.

C#:

- **Multi-Paradigm Language:** C# is a multi-paradigm language that supports both object-oriented programming (OOP) and functional programming, but not as fully as F#.

1. F# Example:

fsharp

Copy code

```fsharp
let add x y = x + y
let result = add 3 5  // 8
```

2. C# Example:

csharp

Copy code

```csharp
using System;

class Program
{
    static void Main()
    {
        Func<int, int, int> add = (x, y) => x + y;
        int result = add(3, 5);  // 8
    }
}
```

Installation :

1. Install .NET SDK

   Download the **.NET SDK** from the official .NET website.

   Once installed, verify that everything is set up correctly by running the following command in the terminal:

   → dotnet -- version

2. Install Visual Studio Code

   download it from the official Visual Studio Code website.

3. Install the Ionide-F# Extension

4. Verify the Installation

After installing **Ionide-F#**, test if everything works by creating a new `.fsx` file (F# Script) and writing a simple F# code snippet:

```fsharp
let add x y = x + y
printfn "%d" (add 2 3)
```

Press `F5` to run the code in Visual Studio Code.

5. Create a New F# Project

You can use the **.NET CLI** to create a new F# project inside Visual Studio Code:

1. Open the **VS Code Terminal** (press `Ctrl +`).
2. Run the following commands to create a new F# console application:

```bash
dotnet new console -lang "F#" -o MyFSharpApp
cd MyFSharpApp
dotnet run
```

This will create and run a basic F# console application.

Variables in F#:

variables are defined using the **let** keyword unlike many other programming languages

variables in F# are **immutable (constant)** by default.
This means that once you assign a value to a variable, you cannot change it.

```
let x = 10
printfn "The value of x is: %d" x
```

Mutable Variables:

In some cases, you may need a variable whose value can change. In such situations, you can use the **mutable** keyword to define a mutable variable.

```
let mutable y = 5
y <- 10  // Reassigning a new value to the mutable variable y
```

Example:
Immutable Variable and New Calculated Value:

```
let baseValue = 20
let newValue = baseValue + 5
```

Example :
Using Mutable Variable in a Loop:

```
let mutable counter = 0
for i in 1..10 do
    counter <- counter + 1
```

Example : (functions)

Using Mutable Variable :

```
let calculate x =
    let mutable y = x + 10  // y is mutable
    y <- y * 2  // changing the value of y
    y  // returning the modified y

let result = calculate 5
printfn "The result is: %d" result
```

functions in F# are **immutable**, which means once a function is defined, it cannot be reassigned or modified.

- For example:

```
let multiply x = x * 2

// Trying to change the function definition will cause an error
// multiply <- fun x -> x + x  // This will result in an error

let result = multiply 5
printfn "The result is: %d" result
```

Note : in functions

If you specify a type, it follows the name of the parameter and is separated from the name by a colon. If you omit the type for the parameter, the parameter type is inferred by the compiler. For example, in the following function definition, the argument `x` is inferred to be of type `int` because 1 is of type `int`.

```
let f (x: int) = x + 1
```

Example of a Pure Function in F#

```
let add x y = x + y
```

F# functions automatically return the value of the last expression without needing an explicit return

Scope in F# :

```
let calculate x =
    let y = x + 10
    y * 2


let result = calculate 5
printfn "result is %d " result
```

Output:

```
x is 10
result is 30
```

```fsharp
let calculate x =
    let y = x + 10
    y * 2


let result = calculate 5
printfn "result is %d " result



printfn "The value of y is: %d" y
```

Output:

```
F# Compiler for F# 4.0 (Open Source Edition)
Freely distributed under the Apache 2.0 Open Source License


/HelloWorld.fs(18,33): error FS0039: The value or constructor 'y' is not defined
```

Data Types in F# :

- `int` : For integers.
- `float` : For floating-point numbers.
- `string` : For text or strings.
- `bool` : For boolean values (**true** or **false**).
- `list` : For lists of values.
- `tuple` : For grouping multiple values of different types.
- `unit` : For indicating the absence of a value.

Examples:

```
let x: int = 10
```

```
let y: float = 10.5
```

```
let greeting: string = "Hello, F#"
```

```
let isEven: bool = true
```

```
let numbers: int list = [1; 2; 3; 4; 5]
```

```
let person: string * int = ("Alice", 30)
```

```
let doNothing (): unit = ()
```

Control Flow in F# :

Control flow in F# includes:

- **Conditional statements** like `if-else`.
- **Loops** like `for` and `while`.
- **Pattern matching** using `match`.
- **Recursion** for self-referencing functions.

If-else:

```
if x > 0 then
    printfn "x is positive"
elif x < 0 then
    printfn "x is negative"
else
    printfn "x is zero"
```

Loops (for – while):

- **For loop:**

```
for i in 1..5 do
    printfn "i = %d" i
```

- **While loop:**

```
let mutable i = 0
while i < 5 do
    printfn "i = %d" i
    i <- i + 1
```

Pattern Matching:

Example 1 :

```
let describeNumber x =
    match x with
    | 0 -> "Zero"
    | 1 -> "One"
    | _ -> "Other"
```

Example 2 :

```
let describeDay day =
    match day with
    | "Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday" -> "Weekday"
    | "Saturday" | "Sunday" -> "Weekend"
    | _ -> "Not a valid day"
```

Recursion :

```
let rec factorial n =
    if n = 0 then 1
    else n * factorial (n - 1)
```

https://drive.google.com/drive/folders/1S7rAwjKosg_btafX2afBq9c1BhqGz-_p?usp=sharing

Thank You!