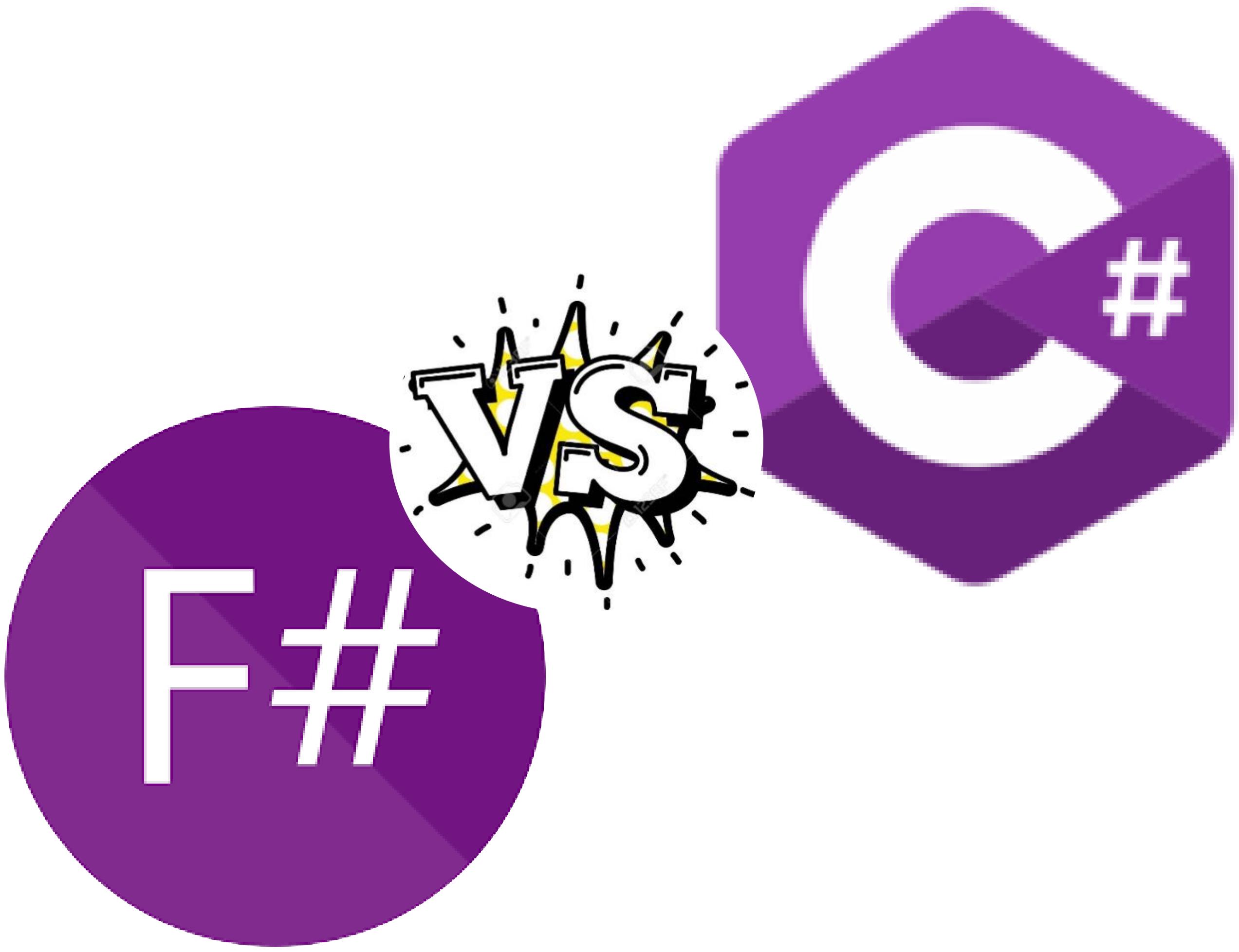
FUNCTIONAL PROGRAMMING SYNOPSIS



Miguel Ángel Herranz Marcos ……………,2018 EASV

INDEX

INTRODUCTION

In my opinion, if someone were interested in doing a comparison work between two programming languages, the first and fundamental step would be to define, or make clear, the concepts that characterize both languages.

F#

F# is a strongly typed, multi-paradigm programming language that encompasses functional, imperative, and object-oriented programming methods. F# is most often used as a cross-platform Common Language Infrastructure (CLI) language, but it can also generate JavaScript and graphics processing unit (GPU) code.

F# is a member of the ML language family and originated as a .NET Framework implementation of a core of the programming language OCaml, it has also been influenced by C#, Python, Haskell, Scala, and Erlang.

C#

C# is a general-purpose, multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed around 2000 by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

In this Synopsis you will find a series of simple examples of how to manipulate files in both languages and their corresponding comparison.

MANIPULATION OF FILE IN F#

F # has a small drawback, being a language of recent creation, the developer community has not advanced much, so the information regarding it is very scarce. The clearest and most basic examples of file manipulation are found at the time of writing and reading certain information from a simple text file (.txt).

F# has a WriteAllText(String, String) function that according to the Microsoft documentation, the function creates a new file, write the contents to the file, and then closes the file, but if the target file already exists, it is overwritten.

The first field (type string) that appears between parentheses is the absolute path, in which the name of the file is already included. The second field (type string) that appears is the content of the file.

Example:

```
open System
open System.IO

File.WriteAllText("C:\Users\Mi
PC\workspaceFP\FunctionalSynopsis\FunctionalSynopsis\SynopsisFile.tx
t","This Is a default message -> FUNCTIONAL IS LOVE")
```

Once created the file with the text, you should read the content of it, for this, I used the ReadAllText (String, String) function that is practically the same as the previous one, and later with "let" associated the contents of the file to a variable string to read the lines of the file in the PowerShell.

Example:

```
let str = File.ReadAllText(@"C:\Users\Mi
PC\workspaceFP\FunctionalSynopsis\FunctionalSynopsis\Prueba\Synopsis
File.txt")

printfn "%A" str

let vocals = ['a';'e';'i';'o';'u';'A';'E';'I';'O';'U']

let isVocals =
    fun count -> vocals |> List.contains count

let vCount =
    String.filter isVocals
    >> String.length

printfn "VOCALS -> %A" (vCount str)
```

The path at the time of writing and reading the file is exactly the same except for the character '@', try to use a variable to reduce work and time, however did not behave as I expected and that I generate more inconveniences than benefits , so I decided to leave it like that; however, in the other language, if I changed it.

In addition, I added a function this Christmas that counted the vowels introduced in the text of the file, using a list of characters in which both the uppercase and lowercase vowels were found. This is because unlike C # with (Lower () and Upper ()) I do not know a function that changes all the characters to the same format. And for this reason, I created a char List with the ALL vowels.

MANIPULATION OF FILE IN C#

When making a comparison between F # and C #, I have to be subject to what I do in both languages, for that reason when creating, writing and / or overwriting an external file in F #, I had to do the same in C #.

The function that I used in this case was a StreamWriter, having previously worked with the handling of data flows in different languages, I supposed that I will have more solid base.

In this case I could have used other functions like the File.WriteAllText, exactly the same as the case of F # (they share many similarities since both are developed by Microsoft), however in this way, only changing the second parameter of the StreamWriter (type bool) I can decide whether or not to overwrite the text of the file.

Example:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FunctionalSynopsisCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            //Declaration Variable
            int count = 0;
            string originalText, text2, pathFile = @"C:\Users\Mi
PC\workspaceFP\FunctionalSynopsisCSharp\FunctionalSynopsisCSharp\Pru
eba\SynopsisFile.txt";
            char[] vocals = { 'a', 'e',
'i','o','u','A','E','I','O','U' };
```

```
//Create, write or overwrite the File
            using (System.IO.StreamWriter file = new 
System.IO.StreamWriter(pathFile, false))
            {
                file.WriteLine("This is a default message -> C# IS 
LOVE");
            }
```

As there were several ways to do the required task and various functions with subtle differences, I wanted to teach each of them, so when reading the contents of the external file, this time I use the File.ReadAllText. How? By entering the file path, all the contents of the file are saved in a String variable, and this is displayed on the screen. Simple and basic but fulfills its function.

Example:

```
            //Read the File and Count the vocals
            originalText = System.IO.File.ReadAllText(pathFile);
            text2 = originalText.ToLower();

            count = text2.Count(x => vocals.Contains(x));

            Console.WriteLine("THE FILE WAS CREATED\n");

            Console.WriteLine(originalText);
            Console.WriteLine("VOCALS -> {0}\n", count);
```

Once the contents of the file are assigned to a variable string, the value is duplicated in another variable which will be transformed into a string of minuscule characters with .Lower () [this was done to omit the character duplication in Uppercase in the character list]. This transformed string is the one that will be manipulated through the use of LINQ to obtain the number of vowels (uppercase or lowercase) that appear in the text of the file.

One of the most notorious differences was that in the case of wanting to change the location of the file or if you wanted to create the file within a new directory, in the case done with F #, the insert of it in the path would be enough to generate it automatically; however, in the case of C #, it would show an error in not finding the full path. This is due more for the functions used in both cases, than for the languages themselves.

CONCLUSION

As it has been possible to verify, when carrying out diverse manipulations of file in the languages of development F # and C #, we do not find apparent differences (excepting obviously the SYNTAX of each language) if it is necessary to emphasize the notorious difference at the time of search and find, documentation, incidents, and solutions between both languages. Well, all this is why? The similarities are due to the fact that both languages are developed by the same company (Microsoft), and the differences are due to the importance and notoriety within the community of developers and the time in which each language carries within it. It is logical to find these difficulties when one of the languages we are dealing with developed in this century. And that was it.