Teaching Functional Programming Paradigm with F#

Rong Li and Huabo Lu
School of Computing
Wichita State University
Wichita, Kansas
{Rong.Li, Huabo.Lu}@wichita.edu

Abstract

Functional Programming (FP) paradigm is a rising programming paradigm that varies from imperative and Object-Oriented Programming (OOP) paradigms. Most Computer Science undergraduate programs require OOP, while the functional programming paradigm is usually partially covered by a 3-credit-hour class that introduces multiple programming paradigms. F# (pronounced F Sharp) is a functional-first, general-purpose programming language. It runs on the .NET platform and is supported on Windows, Linux, and Mac OS operating systems. In this paper, we summarize our experience of using F# to teach the FP paradigm, list some sample codes, and make comparisons between the imperative paradigm and the FP paradigm.

Keywords

F#, Functional Programming, Programming Paradigm, Computer Science Curriculum

Introduction

Functional Programming (FP) paradigm is a programming paradigm that constructs computer programs by creating and using functions. Unlike imperative and Object-Oriented Programming (OOP) paradigms, which use programming statements and objects to control a program's running flow and states, FP uses functions to express computation logic. A key feature of FP is that functions are treated as first-class citizens [1], meaning that functions are treated as entities, and support a wide range of common operations typically including passing functions as arguments and returning function entities. FP is well-known for its strength in generating succinct, robust, and performant code.

Many general-purpose programming languages, such as Java and Python, support multiple programming paradigms, including OOP and FP. Their support for OOP is widely understood and is taught in introductory and intermediate programming classes. Meanwhile, FP receives limited coverage, usually in a 3-credit-hour class that introduces multiple programming paradigms after students have studied OOP.

Functional Programming had the attention of computer science programming classes back in the 1980s and 1990s. Crawford [2] presented their successful experience at Texas A&M in teaching FP in the freshman computer science courses, and Hughes [3] expanded the importance of FP beyond the classroom and to the real world. However, with the rise of new programming languages and the shift in the industry’s needs, Java and Python started to take the lead in the late 1990s and early 2000s [4], [5].

Recent years mark the next rise of Functional Programming. Ever-larger code repositories demand robustness and maintainability, whereas FP languages such as F#, Haskell, and Closure show their strength [6], [7]. Pedagogies of FP can be found at [8], [9].

Starting in 2022, we chose F# as the programming language to teach FP to undergraduate students in a programming paradigms class (3-credit-hour). F# is a functional-first, multiple-paradigm programming language that runs on the .NET platform [10]. Visual Studio 2022 (integrated development environment, IDE) and/or Visual Studio Code (code editor) are the tools of our choice. F# has been supported by the .Net platform since .Net 1.0 back in the early 2000s, so any recent .Net releases, such as .Net 6.0 and 7.0, can support F# with little to no problem.

The rest of the paper is organized as follows: we list selected topics of FP in Section Methods and make some comparisons of FP and OOP. Some student evaluation feedback is presented in Section Results, and we include future work in Section Summary.

Methods

The Functional Programming paradigm has a rich set of features. Due to the page limitation, we choose to list two features:

- Functions, the rule, and partial application.
- Function composition and pipelining.

Functions, The Rule, and Partial Application

Functions in F# (and many other FP languages) have one rule: they take one input and return one output. This seeming restriction benefits in many places, such as testing and concurrency. Students would naturally compare this rule with their experience of OOP, which supports functions with multiple parameters. For example:

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

defines a function named f. It takes one input, which is represented by x, of type int, and returns x + 1 as output.

A "counter-example":

```
let area (height: float) (base: float): float = height * base / 2.0
```

seems to be breaking the rule by allowing two inputs, height and base, to the function named area. F# can handle multiple parameters/arguments with no problem: it will generate the intended result, by breaking this function into two smaller one-parameter functions, such as:

```
let area_1 (height: float): float =
    let subArea (base: float): float =
        height * base / 2.0
    subArea
```

and to call this function:

```
let intermediateArea = area_1 height
let area_result = intermediateArea base
```

where the intermediateArea is a function as the result of calling area_1 with one argument height, then the intermediate function is called with one argument base to calculate the area of a triangle. The method of handling multiple parameters/arguments, while obeying the "one input, one output" rule, is referred to as *function currying* and is a FP character [11].

Function currying is in fact a case of *partial application*. If we call the original area function, which needs two arguments to return the calculated value, with only one argument at a time:

```
let area_missing_base = area height
let area_result = area_missing_base base
```

then the partial application is automatically triggered. The partial application allows fewer than the expected number of arguments to be passed to a function. As a result, a new/intermediate function, such as area_missing_base is created. This new function can be used with the rest of the expected arguments to complete the original function call.

As a comparison to OOP, Java did not support partial application until Java 8, which was released in 2014 [12]. It is referred to as the "functional interface" in Java's terms. Note that this feature is not covered in most introductory programming or OOP courses.

Our students had a learning curve when studying function currying and partial application. However, once understood, they appreciated that functions in FP can accept partial arguments and create intermediate functions that can be called when rest arguments are available. To give an analogy, partial application is like preheating the oven so it will be ready to bake any food (at that preheated temperature.)

Function Composition and Pipelining

Assume we need a function to calculate an expression 3 * x + 5, where the input x is an integer. In Java, this function can be written as (assume there is no integer overflow):

```
public static int calculate(int x) {
    return 3 * x + 5;
}
```

We can write something similar in F#, or use *Function Composition* to demonstrate that there are two calculations/functions:

```
let mul3 x = x * 3
let add5 x = x + 5
let mul3add5 = mul3 >> add5
mul3add5 2
```

There are three functions defined in the example. Function mul3 and add5 are obvious, whereas the third function, mul3add5, is defined by using function composition [13]. This feature allows compatible functions to be composed together, using the composition operator >>. Functions are compatible when the output from the first function is of the same type as the input of the second function. Our students liked this feature in their studies, as it enables a convenient way to create new functions from predefined functions.

Function Pipelining works differently: similar to function composition's "put functions together", function pipelining enables function calls to be chained together as successive operations. Note that function pipelining is designed to allow value/function to flow through the pipe, rather than composing a new function [14]:

```
let ten_mul3add5 = 10 |> mul3 |> add5
```

the value 10 will be passed through the function mul3, and the result (3 * 10 = 30) will be passed through the function add5, to form the final result 3 * 10 + 5 = 35. No new functions were created: the ten_mul3add5 is a value in this case.

Results

Our students showed mild to strong likes of F# functional programming language and a majority provided positive feedback, partially thanks to their familiarity with Visual Studio and .Net environment, which is the same tool/environment as their fundamental programming classes. Challenges exist, including the F# language's syntax, FP paradigm thinking, number of exercises, etc.

We found comparisons between different paradigms, OOP vs. FP in our case, to be effective in helping students learn new paradigms. Comparison can happen at all levels, such as at the statement level, module level, and project level.

Our class concluded the discussion of FP by completing a project using F# (learn by doing), and students presented an improved understanding of FP afterward.

Summary

We present our experience of teaching F# as a functional programming language to undergraduate students. The feedback from students provides us with the confidence to use this language further.

Comparisons between Functional Programming and other paradigms such as OOP add a good part to students' understanding. We plan to list more comparisons in the future, such as redoing some homework exercises from previous programming classes using F#. More applied learning projects, such as designing a web app, will find an opportunity to be added to our class. A high-quality reference, *F# for Fun and Profit* [15], will be formally introduced as a reference in the class syllabus.

Reference

[1] M. Scott, *Programming Language Pragmatics*, San Francisco, CA: Morgan Kaufmann Publishers, 2006.

[2] A. L. Crawford, "Functional programming for freshman computer science majors," ACM SIGCSE Bulletin, vol. 19, no. 1, pp. 165–169, Feb. 1987, doi: https://doi.org/10.1145/31726.31753.

[3] J. Hughes, "Why Functional Programming Matters," The Computer Journal, vol. 32, no. 2, pp. 98–107, Feb. 1989, doi: https://doi.org/10.1093/comjnl/32.2.98.

[4] C. M. Boroni, F. W. Goosey, M. T. Grinder, and R. J. Ross, "A paradigm shift! The Internet, the Web, browsers, Java and the future of computer science education," ACM SIGCSE Bulletin, vol. 30, no. 1, pp. 145–152, Mar. 1998, doi: https://doi.org/10.1145/274790.273181.

[5] Said Hadjerrouit, "A constructivist framework for integrating the Java paradigm into the undergraduate curriculum," Aug. 1998, doi: https://doi.org/10.1145/282991.283079.

[6] "Functional programming is finally going mainstream," GitHub. https://github.com/readme/featured/functional-programming (accessed Aug. 07, 2023).

[7] https://www.facebook.com/48576411181, "Why Functional Programming Should Be the Future of Software Development - IEEE Spectrum," spectrum.ieee.org. https://spectrum.ieee.org/functional-programming (accessed Aug. 07, 2023).

[8] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi, "The structure and interpretation of the computer science curriculum," Journal of Functional Programming, vol. 14, no. 4, pp. 365–378, Jul. 2004, doi: https://doi.org/10.1017/S0956796804005076.

[9] A. Khanfor and Y. Yang, "An Overview of Practical Impacts of Functional Programming," 2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), Dec. 2017, doi: https://doi.org/10.1109/apsecw.2017.27.

[10] "F# | Succinct, robust and performant language for .NET," Microsoft. https://dotnet.microsoft.com/en-us/languages/fsharp (accessed Aug. 07, 2023).

[11] "Using functions in F# - F#," learn.microsoft.com, Nov. 04, 2021. https://learn.microsoft.com/en-us/dotnet/fsharp/tutorials/using-functions?source=recommendations#curried-functions (accessed Aug. 07, 2023).

[12] "What's New in JDK 8," Oracle.com, 2018. https://www.oracle.com/java/technologies/javase/8-whats-new.html (accessed Aug. 07, 2023).

[13] "Functions - F#," learn.microsoft.com, Jun. 25, 2022. https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/functions/#function-composition (accessed Aug. 07, 2023).

[14] "Functions - F#," learn.microsoft.com, Jun. 25, 2022. https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/functions/#pipelines (accessed Aug. 07, 2023).

[15] "F# for fun and profit," fsharpforfunandprofit.com. https://fsharpforfunandprofit.com/ (accessed Aug. 07, 2023).

Rong Li (She/Her/Hers)

Rong Li obtained her Master of Science in Computer Networking degree from Wichita State University in 2011. She joined Wichita State University as an Assistant Engineering Educator in 2022. Before WSU, she was a Computer Science Instructor at Hutchinson Community College, Hutchinson, Kansas. She teaches programming classes such as Object-Oriented Programming, Data Structures, and Programming Paradigms. Her research includes Functional Programming and Computer Science Education.

Huabo Lu (He/Him/His)

Huabo Lu obtained his Doctor of Philosophy in Electrical Engineering and Computer Science degree from Wichita State University in 2018. He joined Wichita State University in 2019, as an Assistant Teaching Professor. Before joining WSU, he was an Assistant Professor at Southwestern College, Winfield, Kansas. His research includes Web Anonymity, Data Privacy, and Computer Science Education. He teaches design courses for computer science students.