Joshua Catoe

CSCI 350

Reflection – Introduction to Functional Programming

Short Answer

1. Is F# a statically or dynamically typed language? What type of functional programming does F# use?

**F# is a statically typed language, and, as such, it uses typed functional programming.**

1. What are the five core concepts of functional programming?

**The five core concepts are functions, expressions, purity, referential transparency, and immutability.**

1. What does a function’s type signature tell you about the function?

**The type signature of a function gives the names and types of its parameters along with the type of data that the function produces.**

1. What is an expression and how do they differ from statements? Give an example of a mathematical expression.

**An expression is something that does an action and gives back a value. While statements only perform an action, expressions perform actions that result in a value.**

**An example mathematical expression is** x\*x**, which multiplies** x **with itself.**

1. What does the “unit” type indicate in a function signature?

**The** unit **type means that the function does not return a value.**

1. What does it mean for a function to be pure?

**A pure function has no side effects and its output only depends on its input.**

1. What is immutability?

**Immutability means that a value cannot be changed(mutated) in-place once they have been declared or evaluated.**

1. Does F# have support for value mutation?

**F# does support mutation, but it is not enabled by default.**

1. Give an example of imperative-style mutation and the equivalent in F#.

int x = 5;

x = x + 1;

let x = 5

let newx = x + 1

1. What might be a reason that immutability is so stressed in functional programming?

**Keeping data separate and clean probably leads to fewer errors from both the user and computer, thus making for much stronger and safer code.**

Essay

1. Explain the idea of type inference. What are some advantages and disadvantages? How does this pertain to automatic generalization?

**In F#, type inference is the process by which the compiler automatically determines the types of a function’s parameters and output. Although types are automatically determined, they can also be declared explicitly as needed. The biggest advantage of this is that coding is made more efficient since the programmer does not need to worry too much about the types of data they are using, but rather how they are using it. Disadvantages arise if/when the compiler does not pick the data type that was intended. Automatic generalization is when the compiler performs type inference and the type is determined to be generic. Data of the generic type can be used with any operator or function that is also generic, such as the greater than operator or** toString() **function.**

1. Write an F# function that adds 5 to a number if that number is even or adds 10 to that number if it is odd. Do the same in C#.

**int addFiveIfEven(int x)**

**{**

**if ( (x % 2) == 0)**

**{**

**x = x + 5**

**}**

**else**

**{**

**x = x + 10**

**}**

**return x;**

**}**

**let isEven x = x % 2 = 0**

**let addFiveIfEven number =**

**if isEven number then**

**number + 5**

**else**

**number + 10**

Reflection – First-class Functions

Short Answer

1. What are the four characteristics of a first-class function?

**The function can be bound to an identifier, it can be stored in a data structure, it can be passed as an argument to another function, and it can be returned from a function call.**

1. What does it mean for a function to be higher-order?

**A higher-order function accepts other functions as arguments, and/or return functions as their return value.**

1. What is a tuple? How does it differ from a list?

**A tuple is a group of unnamed, ordered values. The biggest difference between a tuple and list is that a tuple may contain mixed data types, while a list cannot.**

1. What two common operations in functional programming are made possible by passing functions as arguments?

**Map and filter operations are both common operations built upon functional arguments.**

1. Do functions in F# need to be explicitly curried? Why or why not?

**Functions in F# do not need to be explicitly curried, as any function with multiple parameters is curried implicitly.**

1. Given the function definition “let isEven = fun n -> (n % 2) = 0”, call the function using its definition.

**printfn “%b” ((fun n -> (n % 2) = 0) 37)**

1. Is this a valid declaration of a list of functions? “let funcs = [squareNum; isOdd]

**The declaration is not valid, as the functions have different type signatures (int and bool).**

1. What does the “fun” keyword indicate?

**The fun keyword indicates the declaration of a lambda expression.**

1. Are higher-order functions exclusive to functional programming? Are they supported in C#?

**Many different languages support higher-order functions, including C#.**

1. Give an example of calling a function and its argument from a tuple.

**let tuple = (cubeNum, num)**

**let result = ( (fst tuple) (snd tuple) )**

Essay

1. Explain currying in detail. What is the advantage of F#’s implicit currying vs explicit?

**Currying in F#, is the process by which a function with multiple parameters is broken down into a series of functions containing one parameter. In other words, each parameter of a function becomes the parameter of its own function, which is made using a lambda expression. This provides more flexibility of parameters, especially if they are going to be functions themselves. While explicit currying can be done at any time, F# provides implicit currying by default which, although it gives no benefit as far as speed, brings the advantage of being more concise and easier to read than explicit currying.**

1. Create a function that returns the value from a function call, then create another function that calls that function. Do the same in C#.

**let isEven =**

**let checkEven = fun x -> (x % 2) = 0**

**checkEven**

**let isThisNumEven = isEven 5**

**int checkEven(int x){**

**if ( (x % 2) = 0) {return 1;}**

**else {return 0;}**

**}**

**int isEven(int x) {**

**return checkEven(x);**

**}**

**bool isThisNumEven(int x){**

**if ( (isEven(x)) = 1) {return true;}**

**else {return false;}**

**}**