Preparation Quiz for the PSPO I Certification

Francisco José Nardi Filho

April 1, 2021

Contents

1 Functional programming

Building software by composing pure functions, avoiding shared state, mutable data, and side effects.

1.1 Pure functions

Pure functions are deterministic (return of constant output for a given input). They have no side effects.

1.2 Side effects

Side effects are changes in the system via mutable state, database updates, web requests, IO, etc.

2 .fsx vs .fs files

There are different compiler warnings according to each of the two files.

- .fsx: This file extension is commonly used for scripts and interactive use of F#.
- .fs: This file extension is commonly used for a normal source of F# files.

3 Type annotations

Correct

```
// F# Interactive
// CodeLens extension
let myOne = 1                    // int
let myTwo = 2                    // int
```

Incorrect

```
// F# Interactive
// CodeLens extension
let myOne = 1.0                  // ???
let myTwo = 2                    // int
```

Incorrect

```
// F# Interactive
// CodeLens extension
let myOne: double = 1        // ???
let myTwo = 2                // int
```

Correct

```
// F# Interactive
// CodeLens extension
let myOne: double = 1.0      // double
let myTwo = 2.0              // float
```

4 Primitives

Example 1

```
let myOne = 1
let hello = "Hello"
let letterA = 'a'
```

The primitive types will be the same for all languages in the .NET Core environment.

4.1 Mutable/Assignment

Example 2

```
let isEnabled = true
isEnabled = false          // comparative clause
isEnabled <- false         // assignment clause
```

Values are immutable by default, so the previous clauses will not work.

Example 3

```
let mutable isEnabled = true
isEnabled = false          // comparative clause
isEnabled <- false         // assignment clause
```

To change the *isEnabled* value, we need to declare it as *mutable*.

5 Expressions vs Statements

- Functions are expressions.
- Expressions are values.
- Every new line is a new expression.

6 Functions

```
// int -> int -> int
let add x y = x + y
```

The return is automatically considered to be the last line of a function.

6.1 Lambda expressions

```
// int -> int -> int
let add' = fun x y -> x + y
```

6.2 Currying/Baking-In

```
// int -> int -> int
let add'' x = fun y -> x + y
```

Currying is a function that returns another, which in turn produces a new one until it returns a value.

```
// x is an early given parameter
let add'' x =
    // it produces a new function,
    // which takes y as another parameter
    fun y ->
        // the last line is the return,
        // a function that sums both parameters
        x + y
```

In F#, all functions are curried.

7 Partial Application

```
// int -> int
let add x y = x + y
let add5' = add 5
```

8 Function composition

```
let add3 number = number + 3.
let times2 number = number * 2.
let pow2 number = number ** 2.
```

9 Pipe operator

```
    // point free code
    let add3 = ( + ) 3.
    let times2 = ( * ) 2.
    let pow2 number = number ** 2.

    let operation' number =
        number
        |> add3
        |> times2
        |> pow2

    operation' 2.

(*
    val add3 : (float -> float)
    val times2 : (float -> float)
    val pow2 : number:float -> float
    val operation' : number:float -> float
    val it : float = 100.0
*)
```

10 Composition operator

```
let add3 = ( + ) 3.
let times2 = ( * ) 2.
let pow2 number = number ** 2.

let operation'' =
    add3
    >> times2
    >> pow2

operation'' 2.
```

11 Defining new operators

```
    let (>>) f g =
        fun x ->
            x
            |> f
            |> g
```

12 Hello World/Main function

Example 1

```
module Arithmetic =
    module public Addition =
        let add x y = x + y

open Arithmetic

let program =
    Addition.add 5 2
```

Example 2

```
module Arithmetic =
    module public Addition =
        let add x y = x + y

open Arithmetic

let program =
    Addition.add 5 2
```

13 Unit

```
open System
open System.Threading

[<EntryPoint>]
let main argv =

    printfn "How old are you?"
```

```
let year = Console.ReadLine()
printfn "You are %s years old" year

let currentTime() =
    DateTime.Now

currentTime ()
    |> printfn "Now is %O"

Thread.Sleep 2000

currentTime ()
    |> printfn "Now is %O"

0
```

14 Printing to Console

```
open System

// Record type
// Tuple
// Anonymous record

type Day = {DayOfTheMonth: int; Month: int}
type Person = {Name: string; Age: int}

let day = { DayOfTheMonth = 26; Month = 03}
let ben = { Name = "Ben"; Age = 26 }

printfn "%i-%i-2021" day.DayOfTheMonth day.Month
printfn "%s %i" ben.Name ben.Age
```

15 Pattern Matching

```
let yesOrNo bool =
    match bool with
    | true -> "Yes"
    | false -> "No"
```

```
// point free code
let yesOrNo' = function
    | true -> "Yes"
    | false -> "No"

// point free code
let isEven = function
    | n when n % 2 = 0 -> true
    | _ -> false
```

16 The "function" keyword

```
let isOne = function
    | 1 -> true
    | _ -> false

let isOne' number =
    number = 1

let isOne'' =
    (=) 1
```

17 Pattern matching with let and fun

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

18 Option type

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

19 Domain Errors vs Exceptions

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

20 Generics and SRTP

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

21 Inline keyword

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

22 Type members

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

23 Collections

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

24 Arrays

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

25 Lists

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

26 Collection libraries

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

27 List.head

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

28 Recursion / List.iter

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

29 List.map

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

30 List.fold

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

31 List.reduce

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

32 List.sum

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

33 Bind

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

34 Exception handling

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

35 Results / Error Modeling

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

36 Outtro

Lorem ipsum dolor sit amet, consectetur adipiscing elit.