عنوان: **نکاتی درباره برنامه نویسی دستوری(امری)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۹ ۰:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: F#, Programming

در این فصل نکاتی را درباره برنامه نویسی دستوری در F# فرا خواهیم گرفت. برای شروع از mutale خواهیم گفت.

mutable Keyword

در فصل دوم(شناسه ها) گفته شد که برای یک شناسه امکان تغییر مقدار وجود ندارد. اما در F# راهی وجود دارد که در صورت نیاز بتوانیم مقدار یک شناسه را تغییر دهیم.در F# هرگاه بخواهیم شناسه ای تعریف کنیم که بتوان در هر زمان مقدار شناسه رو به دلخواه تغییر داد از کلمه کلیدی mutable کمک می‌گیریم و برای تغییر مقادیر شناسه‌ها کافیست از علامت (<-) استفاده کنیم. به یک مثال در این زمینه دقت کنید:

```
#1 let mutable phrase = "Can it change? "

#2 printfn "%s" phrase

#3 phrase <- "yes, it can."

#4 printfn "%s" phrase
```

در خط اول یک شناسه را به صورت mutable(تغییر پذیر) تعریف کردیم و در خط سوم با استفاده از (<-) مقدار شناسه رو update کردیم. خروجی مثال بالا به صورت زیر است:

```
Can it change?
yes, it can.
```

نکته اول : در این روش هنگام update کردن مقدار شناسه حتما باید مقدار جدید از نوع مقدار قبلی باشد در غیر این صورت با خطای کامپایلری متوقف خواهید شد.

```
#1 let mutable phrase = "Can it change?  "

#3 phrase <- 1
```

اجرای کد بالا خطای زیر را به همراه خواهد داشت.(خطا کاملا واضح است و نیاز به توضیح دیده نمی‌شود)

```
Prog.fs(9,10): error: FS0001: This expression has type
int
but is here used with type
string
```

نکته دوم :ابتدا به مثال زیر توجه کنید.

```
let redefineX() =
let x = "One"
printfn "Redefining:\r\nx = %s" x
if true then
   let x = "Two"
printfn "x = %s" x
printfn "x = %s" x
```

در مثال بالا در تابع redefineX یک شناسه به نام x تعریف کردم با مقدار "One". یک بار مقدار شناسه x رو چاپ می‌کنیم و بعد دوباره بعد از شرط true یک شناسه دیگر با همون نام یعنی x تعریف شده است و در انتها هم دو دستور چاپ. ابتدا خروجی مثال بالا رو با هم مشاهده می‌کنیم.

```
Redefining:
x = One
x = Two
x = One
```

همان طور که میبینید شناسه دوم x بعد از تعریف دارای مقدار جدید Two بود و بعد از اتمام محدوده(scope) مقدار x دوباره به One تغییر کرد.(بهتر است بگوییم منظور از دستور print x سوم اشاره به شناسه x اول برنامه است). این رفتار مورد انتظار ما در هنگام استفاده از روش تعریف مجدد شناسه هاست. حال به بررسی رفتار muatable در این حالت می‌پردازیم.

```
let mutableX() =
let mutable x = "One"
printfn "Mutating:\r\nx = %s" x
if true then
   x <- "Two"
printfn "x = %s" x
printfn "x = %s" x
```

تنها تفاوت در استفاده از mutable keyword و (<-) است. خروجی مثال بالا نیز به صورت زیر خواهد بود. کاملا واضح است که مقدار شناسه x بعد از تغییر و اتمام محدوده(scope) هم چنان Two خواهد بود.

```
Mutating:
x = One
x = Two
x = Two
```

Reference Cells

روشی برای استفاده از شناسه‌ها به صورت mutable است. با این روش می‌تونید شناسه هایی تعریف کنید که امکان تغییر مقدار برای اون‌ها وجود دارد. زمانی که از این روش برای مقدار دهی به شناسه‌ها استفاده کنیم یک کپی از مقدار مورد نظر به شناسه اختصاص داده می‌شود نه آدرس مقدار در حافظه.
به جدول زیر توجه کنید:

Definition	Description	Member Or Field
let (!) r = r.contents	مقدار مشخص شده را برگشت می‌دهد	(derefence operator)!
let (:=) r x = r.contents <- x	مقدار مشخص شده را تغییر می‌دهد	(Assignment operator)=:
let ref x = { contents = x }	یک مقدار را در یک reference cell جدید کپسوله می‌کند	ref operator
member x.Value = x.contents	برای عملیات get یا set مقدار مشخص شده	Value Property
let ref x = { contents = x }	برای عملیات get یا set مقدار مشخص شده	contents record field

یک مثال:

```
let refVar = ref 6

refVar := 50



printfn "%d" !refVar
```

خروجی مثال بالا 50 خواهد بود.

```
let xRef : int ref = ref 10

printfn "%d" (xRef.Value)
printfn "%d" (xRef.contents)

xRef.Value <- 11
printfn "%d" (xRef.Value)
xRef.contents <- 12
printfn "%d" (xRef.contents)
```

خروجی مثال بالا:

```
10
10
11
12
```

خصیصه اختیاری در #F

در #F زمانی از خصیصه اختیاری استفاده می‌کنیم که برای یک متغیر مقدار وجود نداشته باشد. option در #F نوعی است که می‌تواند هم مقدار داشته باشد و هم نداشته باشد.

```
let keepIfPositive (a : int) = if a > 0 then Some(a) else None
```

از None زمانی استفاده می‌کنیم که option مقدار نداشته باشد و از Some زمانی استفاده می‌کنیم که option مقدار داشته باشد.

```
let exists (x : int option) =
    match x with
    | Some(x) -> true
    | None -> false
```

در مثال بالا ورودی تابع exists از نوع int و به صورت اختیاری تعریف شده است.(معادل با ?int یا<int>Nullable در #C) در صورتی که x مقدار داشته باشد مقدار true در غیر این صورت مقدار false را برگشت می‌دهد.

چگونگی استفاده از option

مثال

```
let rec tryFindMatch pred list =
    match list with
    | head :: tail -> if pred(head)
                      then Some(head)
                      else tryFindMatch pred tail
    | [] -> None

let result1 = tryFindMatch (fun elem -> elem = 100) [ 200; 100; 50; 25 ] //برابر با 100 است

let result2 = tryFindMatch (fun elem -> elem = 26) [ 200; 100; 50; 25 ]// برابر با None است
```

یک تابع به نام tryFindMatch داریم با دو پارامتر ورودی. با استفاده از الگوی Matching از عنصر ابتدا تا انتها را در لیست (پارامتر

list) با مقدار پارامتر pred مقایسه می‌کنیم. اگر مقادیر برابر بودند مقدار head در غیر این صورت None(یعنی option مقدار ندارد) برگشت داده می‌شود.
یک مثال کاربردی تر

```
open System.IO
let openFile filename =
    try
        let file = File.Open (filename, FileMode.Create)
        Some(file)
    with
        | ex -> eprintf "An exception occurred with message %s" ex.Message
                None
```

در مثال بالا از optionها برای بررسی وجود یا عدم وجود فایل‌های فیزیکی استفاده کردم.

Enumeration

تقریبا همه با نوع داده شمارشی یا enums آشنایی دارند. در اینجا فقط به نحوه پیاده سازی آن در F# می‌پردازیم. ساختار کلی تعریف آن به صورت زیر است:

```
type enum-name =
   | value1 = integer-literal1
   | value2 = integer-literal2
  ...
```

یک مثال از تعریف:

```
type Color =
    | Red = 0
    | Green = 1
    | Blue = 2
```

نحوه استفاده

```
let col1 : Color = Color.Red
```

enums فقط از انواع داده ای sbyte, byte, int16, uint16, int32, uint32, int64, uint16, uint64, char پشتیبانی می‌کند که البته مقدار پیش فرض آن Int32 است. در صورتی که بخواهیم صریحا نوع داده ای را ذکر کنیم به صورت زیر عمل می‌شود.

```
type uColor =
    | Red = 0u
    | Green = 1u
    | Blue = 2u
let col3 = Microsoft.FSharp.Core.LanguagePrimitives.EnumOfValue<uint32, uColor>(2u)
```

توضیح درباره use

در دات نت خیلی از اشیا هستند که اینترفیس IDisposable رو پیاده سازی کرده اند. این بدین معنی است که حتما یک متد به نام dispose برای این اشیا وجود دارد که فراخوانی آن به طور قطع باعث بازگرداندن حافظه ای که در اختیار این کلاس‌ها بود می‌شود. برای راحتی کار در C# یک عبارت به نام using وجود دارد که در انتها بلاک متد dispose شی مربوطه را فراخوانی می‌کند.

```
using(var writer = new StreamWriter(filePath))
{

}
```

در F# نیز امکان استفاده از این عبارت با اندکی تفاوت وجود دارد.مثال:

```
let writeToFile fileName =
    use sw = new System.IO.StreamWriter(fileName : string)
    sw.Write("Hello ")
```

Units Of Measure

در #F اعداد دارای علامت و اعداد شناور دارای وابستگی با واحدهای اندازه گیری هستند که به نوعی معرف اندازه و حجم و مقدار و ... هستند. در #F شما مجاز به تعریف واحدهای اندازه گیری خاص خود هستید و در این تعاریف نوع عملیات اندازه گیری را مشخص می‌کنید. مزیت اصلی استفاده از این روش جلوگیری از رخ دادن خطاهای کامپایلر در پروژه است. ساختار کلی تعریف:

```
[<Measure>] type unit-name [ = measure ]
```

یک مثال از تعریف واحد cm:

```
[<Measure>] type cm
```

مثالی از تعریف میلی لیتر:

```
[<Measure>] type ml = cm^3
```

برای استفاده از این واحدها می‌تونید به روش زیر عمل کنید.

```
let value = 1.0<cm>
```

توابع تبدیل واحد ها

قدرت اصلی واحدهای اندازه گیری #F در توابع تبدیل است. تعریف توابع تبدیل به صورت زیر می‌باشد:

```
[<Measure>] type g                  تعریف واحد گرم
[<Measure>] type kg                 تعریف واحد کیلوگرم
let gramsPerKilogram : float<g kg^-1> = 1000.0<g/kg>    تعریف تابع تبدیل
```

یک مثال دیگر :

```
[<Measure>] type degC // دما بر حسب سلسیوس
[<Measure>] type degF // دما بر حسب فارنهایت

let convertCtoF ( temp : float<degC> ) = 9.0<degF> / 5.0<degC> * temp + 32.0<degF> // تابع تبدیل
سلسیوس به فارنهایت
let convertFtoC ( temp: float<degF> ) = 5.0<degC> / 9.0<degF> * ( temp - 32.0<degF>) // تابع تبدیل
فارنهایت به سلسیوس

let degreesFahrenheit temp = temp * 1.0<degF> // درجه به فارنهایت
let degreesCelsius temp = temp * 1.0<degC> // درجه به سلسیوس

printfn "Enter a temperature in degrees Fahrenheit."
let input = System.Console.ReadLine()
let mutable floatValue = 0.
if System.Double.TryParse(input, &floatValue)// اگر ورودی عدد بود
   then
      printfn "That temperature in Celsius is %8.2f degrees C." ((convertFtoC (degreesFahrenheit
floatValue))/(1.0<degC>))
   else
      printfn "Error parsing input."
```

خروجی مثال بالا :

```
Enter a temperature in degrees Fahrenheit.
90
That temperature in degrees Celsius is    32.22.
```