عنوان: **توابع(Function)**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۴:۴۴
آدرس: www.dotnettips.info
برچسب‌ها: F#, Programming

برنامه نویسی تابع گرا در یک جمله یعنی نوشتن توابع در پروژه و فراخوانی آن‌ها به همراه مقدار دهی به آرگومان‌های متناظر و دریافت خروجی در صورت نیاز. در F# پارامترهای یک تابع با پرانتز یا کاما از هم تمیز داده نمی‌شوند بلکه باید فقط از یک فضای خالی بین آن‌ها استفاده کنید.(البته می‌تونید برای خوانایی بهتر از پرانتز استفاده کنید)

```
let add x y = x + y
let result = add 4 5
printfn "(add 4 5) = %i" result
```

همان طور که می‌بینید تابعی به نام add داریم که دارای 2 پارامتر ورودی است به نام‌های x , y که فقط توسط یک فضای خالی از هم جدا شدند. حال به مثال دیگر توجه کنید.

```
let add x y = x + y
let result1 = add 4 5
let result2 = add 6 7
let finalResult = add result1 result2
```

در مثال بالا همان تابع add 2 بار فراخوانی شده است که یک بار مقدار خروجی تابع در یک شناسه به نام result1 و یک بار مقدار خروجی با مقادیر متفاوت در شناسه به نام result2 قرار گرفت. شناسه finalResult حاصل فراخوانی تابع add با مقادیر result1 , result2 است. می‌تونیم کد بالا رو به روش مناسب‌تری باز نویسی کنیم.

```
let add x y = x + y
let result =add (add 4 5) (add 6 7)
```

در اینجا برای خوانایی بهتر کد از پرانتز برای جداسازی مقدار پارامترها استفاده کردم.

خروجی توابع

کامپایلر F# آخرین مقداری که در تابع، تعریف و استفاده می‌شود را به عنوان مقدار بازگشتی و نوع آن را نوع بازگشتی می‌شناسد.

```
let cylinderVolume radius length : float =

   let pi = 3.14159
   length * pi * radius * radius
```

در مثال بالا خروجی تابع مقدار (length * pi * radius * radius) است و نو ع آن float می‌باشد.
یک مثال دیگر:

```
let sign num =
    if num > 0 then "positive"
    elif num < 0 then "negative"
    else "zero"
```

خروجی تابع بالا از نوع string است و مقدار آن با توجه به ورودی تابع positive یا negative یا zero خواهد بود.

تعریف پارامترهای تابع با ذکر نوع به صورت صریح

اگر هنگام تعریف توابع مایل باشید که نوع پارامترها را به صورت صریح تعیین کنید از روش زیر استفاده می‌کنیم.

```
let replace(str: string) =
```

```
    str.Replace("A", "a")
```

تعریف تابع به همراه دو پارامتر و ذکر نوع فقط برای یکی از پارامترها :

```
let addu1 (x : uint32) y =
    x + y
```

Pipe-Forward Operator

در #F روشی دیگری برای تعریف توابع وجود دارد که به pipe-Forward معروف است. فقط کافیست از اپراتور (<|) به صورت زیر استفاده کنید.

```
let (|>) x f = f x
```

کد بالا به این معنی است که تابعی یک پارامتر ورودی به نام x دارد و این پارامتر رو به تابع مورد نظر(هر تابعی که شما هنگام استفاده تعیین کنید) تحویل می‌دهد و خروجی را بر می‌گرداند. برای مثال

```
let result = 0.5 |> System.Math.Cos
```

یا

```
let add x y = x + y
let result = add 6 7 |> add 4 |> add 5
```

در مثال بالا ابتدا حاصل جمع 7 و 6 محاسبه می‌شود و نتیجه با 4 جمع می‌شود و دوباره نتیجه با 5 جمع می‌شود تا حاصل نهایی در result قرار گیرد. به نظر اکثر برنامه نویسان #F این روش نسبت به روش‌های قبلی خواناتر است. این روش همچنین مزایای دیگری نیز دارد که در مبحث Partial Functionها بحث خواهیم کرد.

Partial Fucntion Or Application

partial function به این معنی است که در هنگام فراخوانی یک تابع نیاز نیست که به تمام آرگومان‌های مورد نیاز مقدار اختصاص دهیم. برای نمونه در مثال بالا تابع add نیاز به 2 آرگومان ورودی داشت در حالی که فقط یک مقدار به آن پاس داده شد.

let result = add 6 7 |> add 4 دلیل برخورد #F با این مسئله این است که #F توابع رو به شکل مقدار در نظر می‌گیرد و اگر تمام مقادیر مورد نیاز یک تابع در هنگام فراخوانی تحویل داده نشود، از مقدار برگشت داده شده فراخوانی تابع قبلی استفاده خواهد کرد. البته این مورد همیشه خوشایند نیست. اما می‌تونیم با استفاده از پرانتز ر هنگام تعریف توابع مشخص کنیم که دقیقا نیاز به چند تا مقدار ورودی برای توابع داریم.

```
let sub (a, b) = a - b
let subFour = sub 4
```

کد بالا کامپایل نخواهد شد و خطای زیر رو مشاهده خواهید کرد.

```
prog.fs(15,19): error: FS0001: This expression has type
int
but is here used with type
'a * 'b
```

توابع بازگشتی

در مورد ماهیت توابع بازگشتی نیاز به توضیح نیست فقط در مورد نوع پیاده سازی اون در #F توضیح خواهم داد. برای تعریف

توابع به صورت بازگشتی کافیست از کلمه rec بعد از let استفاده کنیم(زمانی که قصد فراخوانی تابع رو در خود تابع داشته باشیم). مثال پایین به خوبی مسئله را روشن خواهد کرد.(پیاده سازی تابع فیبو ناچی)

```
let rec fib x =
match x with
| 1 -> 1
| 2 -> 1
| x -> fib (x - 1) + fib (x - 2)


printfn "(fib 2) = %i" (fib 2)
printfn "(fib 6) = %i" (fib 6)
printfn "(fib 11) = %i" (fib 11)
```

*درباره الگوی Matching در فصل بعد به صورت کامل توضیح خواهم داد.
خروجی برای مثال بالا به صورت خواهد شد.

```
(fib 2) = 1
(fib 6) = 8
(fib 11) = 89
```

توابع بازگشتی دو طرفه

گاهی اوقات توابع به صورت دوطرفه بازگشتی می‌شوند. یعنی فراخوانی توابع به صورت چرخشی انجام می‌شود. (فراخوانی یک تابع در تابع دیگر و بالعکس). به مثال زیر دقت کنید.

```
let rec Even x =
   if x = 0 then true
   else Odd (x - 1)
and Odd x =
   if x = 1 then true
   else Even (x - 1)
```

کاملا واضح است در تابع Even فراخوانی تابع Odd انجام می‌شود و در تابع Odd فراخوانی تابع Even. به این حالت mutual recursive می‌گویند.

ترکیب توابع

```
let firstFunction x = x + 1
let secondFunction x = x * 2
let newFunction = firstFunction >> secondFunction
let result = newFunction 100
```

در مثال بالا دو تابع به نام‌های firstFunction و secondFunction داریم. بااستفاده از (>>) دو تابع را با هم ترکیب می‌کنیم. خروجی بدین صورت محاسبه می‌شود که ابتدا تابع firstFucntion مقدار x را محاسبه می‌کند و حاصل به تابع secondFucntion پاس داده می‌شود. در نهایت یک تابع جدید به نام newFunction خواهیم داشت که مقدار نهایی محاسبه خواهد شد. خروجی مثال بالا 202 است.

توابع تودرتو

در F# امکان تعریف توابع تودرتو وجود دارد. بعنی می‌تونیم یک تابع را در یک تابع دیگر تعریف کنیم. فقط نکته مهم در امر استفاده از توابع به این شکل این است که توابع تودرتو فقط در همون تابعی که تعریف می‌شوند قایل استفاده هستند و محدوده این توابع در خود همون تابع است.

```
let sumOfDivisors n =
    let rec loop current max acc =

شروع تابع داخلی//

   if current > max then
            acc
```

```
        else
            if n % current = 0 then
                loop (current + 1) max (acc + current)
            else
                loop (current + 1) max acc

ادامه بدنه تابع اصلی//

 let start = 2
    let max = n / 2     (* largest factor, apart from n, cannot be > n / 2 *)
    let minSum = 1 + n  (* 1 and n are already factors of n *)
    loop start max minSum

printfn "%d" (sumOfDivisors 10)
```

در مثال بالا یک تابع تعریف کرده ایم به نام sumOfDivisors. در داخل این تابع یک تابع دیگر به نام loop داریم که از نوع بازگشتی است(به دلیل وجود rec بعد از let). بدنه تابع داخلی به صورت زیر است:

```
if current > max then
            acc
        else
            if n % current = 0 then
                loop (current + 1) max (acc + current)
            else
                loop (current + 1) max acc
```

خروجی مثال بالا برای ورودی 10 عدد 18 می‌باشد. مجموع مقصوم علیه‌های عدد 10 (1 + 2 + 5 + 10).

آیا می‌توان توابع را Overload کرد؟

در F# امکان overloading برای یک تابع وجود ندارد. ولی متدها را می‌توان overload کرد.(متدها در فصل شی گرایی توضیح داده می‌شود).

do keyword

زمانی که قصد اجرای یک کد را بدون تعریف یک تابع داشته باشیم باید از do استفاده کنیم. همچنین از do در انجام برخی عملیات پیش فرض در کلاس‌ها زیاد استفاده می‌کنیم.(در فصل شی گرایی با این مورد آشنا خواهید شد).

```
open System
open System.Windows.Forms

let form1 = new Form()
form1.Text <- "XYZ"

[<STAThread>]
do
   Application.Run(form1)
```