عنوان: **لیست ها و آرایه ها در F#**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۷ ۱۶:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: F#, Programming

برای تعریف لیست در F# فقط کافیست از [] و برای جداسازی آیتم‌های موجود در لیست از عملگر :: (بخوانید cons) استفاده کنید. F# از لیست‌های خالی نیز پشتیبانی می‌کند. به مثال هایی از این دست توجه کنید

```
#1 let emptyList = []
#2 let oneItem = "one " :: []
#3 let twoItem = "one " :: "two " :: []
```

#1 تعریف یک لیست خالی
#2 تعریف یک لیست به همراه یک آیتم
#3 تعریف یک لیست به همراه دو آیتم
قبول دارم که دستورالعمل بالا برای مقدار دهی اولیه به لیست کمی طولانی و سخت است. برای همین می‌تونید از روش زیر هم استفاده کنید.

```
let shortHand = ["apples "; "pears"]
```

*کد بالا یک لیست با دو آیتم که از نوع رشته ای هستند تولید خواهد کرد.
می‌تونید از عملگر @ برای پیوستن دو لیست به هم نیز استفاده کنید.

```
let twoLists = ["one, "; "two, "] @ ["buckle "; "my "; "shoe "]
```

نکته : تمام آیتم‌های موجود در لیست باید از یک نوع باشند. یعنی امکان تعریف لیستی که دارای آیتم هایی با datatypeهای متفاوت باشد باعث تولید خطای کامپایلری می‌شود. اما اگر نیاز به لیستی دارید که باید چند datatype رو هم پوشش دهد می‌تونید از objectها استفاده کنید.

```
let objList = [box 1; box 2.0; box "three"]
```

در بالا یک لیست از objectها رو تعریف کرده ایم. فقط دقت کنید برای اینکه آیتم‌های موجود در لیست رو تبدیل به object کنیم از دستور box قبل از هر آیتم استفاده کردیم.

در هنگام استفاده از عملگرها @ و :: مقدار لیست تغییر نمی‌کند بلکه یک لیست جدید تولید خواهد شد.

```
#1 let one = ["one "]
#2 let two = "two " :: one
#3 let three = "three " :: two
#4 let rightWayRound = List.rev three

#5 let main() =
printfn "%A" one
printfn "%A" two
printfn "%A" three
printfn "%A" rightWayRound
```

#1 تعریف لیستی که دارای یک آیتم است.
#2 تعریف لیستی که دارای دو آیتم است(آیتم دوم لیست خود از نوع لیست است)
#3 تعریف لیستی که دارای سه آیتم است(ایتم دوم لیست خود از نوع لیستی است که دارای دو آیتم است)

از تابع List.rev برای معکوس کردن آیتم‌های لیست three استفاده کردیم و مقادیر در لیستی به نام rightWayRound قرار گرفت.

#5 تابع main برای چاپ اطلاعات لیست ها

بعد از اجرا خروجی زیر مشاهده می‌شود.

```
["one "]
["two "; "one "]
["three "; "two "; "one "]
["one "; "two "; "three "]
```

تفاوت بین لیست‌ها در #F و لیست و آرایه در دات نت(System.Collection.Generic)

	F#List	Net Array	Net List
#1 امکان تغییر در عناصر لیست	No	Yes	Yes
#2 امکان اضافه کردن عنصر جدید	No	No	Yes
#3 جستجو	On	O1	O1

#1 در #F بعد از ساختن یک لیست امکان تغییر در مقادیر عناصر آن وجود ندارد.

#2 در #F بعد از ساختن یک لیست دیگه نمی‌تونید یک عنصر جدید به لیست اضافه کنید.

#3 جستجوی در لیست‌های #F به نسبت لیست‌ها و آرایه‌های در دات نت کندتر عمل می‌کند.

استفاده از عبارات در لیست ها

برای تعریف محدوده در لیست می‌تونیم به راحتی از روش زیر استفاده کنیم

```
let rangeList = [1..99]
```

برای ساخت لیست‌ها به صورت داینامیک استفاده از حلقه‌های تکرار در لیست مجاز است.

```
let dynamicList = [for x in 1..99 -> x*x]
```

کد بالا معادل کد زیر در #C است.

```
for(int x=0;x<99 ; x++)
{
   myList.Add(x*x);
}
```

لیست‌ها و الگوی Matching

روش عادی برای کار با لیست‌ها در #F استفاده از الگوی Matching و توابع بازگشتی است.

```
let listOfList = [[2; 3; 5]; [7; 11; 13]; [17; 19; 23; 29]]

let rec concatList l =
match l with
| head :: tail -> head @ (concatList tail)
| [] -> []

let primes = concatList listOfList

printfn "%A" primes
```

در مثال بالا ابتدا یک لیست تعریف کردیم که دارای 3 آیتم است و هر آیتم آن خود یک لیست با سه آیتم است.(تمام آیتم‌ها از نوع داده عددی هستند). یک تابع بازگشتی برای پیمایش تمام آیتم‌های لیست نوشتم که در اون از الگوی Matching استفاده کردیم.
خروجی :

```
[2; 3; 5; 7; 11; 13; 17; 19; 23; 29]
```

ماژول لیست

در جدول زیر تعدادی از توابع ماژول لیست رو مشاهده می‌کنید.

نام تابع	توضیحات
List.length	تابعی که طول لیست را برمی گرداند
List.head	تابعی برای برگشت عنصر اول لیست
List.tail	تمام عناصر لیست را بر میگرداند به جز عنصر اول
List.init	یک لیست با توجه به تعداد آیتم ایجاد می‌کند و یم تابع را بر روی تک تک عناصر لیست ایجاد می‌کند.
List.append	یک لیست را به عنوان ورودی دریافت می‌کند و به لیست مورد نظر اضافه می‌کند و مجموع دو لیست را برگشت می‌دهد
List.filter	فقط عناصری را برگشت می‌دهد که شرط مورد نظر بر روی آن‌ها مقدار true را برگشت دهد
List.map	یک تابع مورد نظر را بر روی تک تک عناصر لیست اجرا می‌کند و لیست جدید را برگشت می‌دهد
List.iter	یک تابع مورد نظر را بر روی تک تک عناصر لیست اجرا می‌کند
List.zip	مقادیر دو لیست را با هم تجمیع می‌کند و لیست جدید را برگشت می‌دهد. اگر طول 2 لیست ورودی یکی نباشد خطا رخ خواهدداد
List.unzip	درست برعکس تابع بالا عمل می‌کند
List.toArray	لیست را تبدیل به آرایه می‌کند
List.ofArray	آرایه را تبدیل به لیست می‌کند

مثال هایی از توابع بالا

```
List.head [5; 4; 3]
List.tail [5; 4; 3]
List.map (fun x -> x*x) [1; 2; 3]
List.filter (fun x -> x % 3 = 0) [2; 3; 5; 7; 9]
```

Sequence Collection seq در #F یک توالی از عناصری است که هم نوع باشند. عموما از sequenceها زمانی استفاده میکنیم که یک مجموعه از داده‌ها با تعداد زیاد و مرتب شده داشته باشیم ولی نیاز به استفاده از تمام عناصر آن نیست. کارایی sequence در مجموعه‌های با تعداد زیاد از listها به مراتب بهتر است. sequenceها را با تابع seq می‌شناسند که معادل IEnumerable در دات

نت است. بنابر این هر مجمو عه ای که IEnumerable رو در دات نت پیاده سازی کرده باشد در #F با seq قابل استفاده است.

مثال هایی از نحوه استفاده seq

#1 seq بامحدوده 1 تا 100 و توالی 10

```
seq { 0 .. 10 .. 100 }
```

#2 استفاده از حلقه‌های تکرار برای تعریف محدوده و توالی در seq

```
seq { for i in 1 .. 10 do yield i * i }
```

#3 استفاده از <- به جای yield

```
seq { for i in 1 .. 10 -> i * i }
```

#4 استفاده از حلقه for به همراه شرط برای فیلتر کردن

```
let isprime n =
    let rec check i =
        i > n/2 || (n % i <> 0 && check (i + 1))
    check 2

let aSequence = seq { for n in 1..100 do if isprime n then yield n }
```

چگونگی استفاده از توابع seq

در این بخش به ارائه مثال هایی کاربردی‌تر از چگونگی استفاده از seq در #F می‌پردازیم. برای شروع نحوه ساخت یک seq خالی یا empty رو خواهم گفت.

```
let seqEmpty = Seq.empty
```

روش ساخت یک seq که فقط یک عنصر را برگشت می‌دهد.

```
let seqOne = Seq.singleton 10
```

برای ساختن یک seq همانند لیست‌ها می‌تونیم از seq.init استفاده کنیم. عدد 5 که بلافاصله بعد از تابع seq.init آمده است نشان دهنده تعداد آیتم‌ها موجود در seq خواهد بود. seq.iter هم یک تابع مورد نظر رو بر روی تک تک عناصر seq اجرا خواهد کرد.(همانند list.iter)

```
let seqFirst5MultiplesOf10 = Seq.init 5 (fun n -> n * 10)
Seq.iter (fun elem -> printf "%d " elem) seqFirst5MultiplesOf10
```

خروجی مثال بالا

```
0 10 20 30 40
```

با استفاده از توابع seq.ofList , seq.ofArray می‌تونیم seq مورد نظر خود را از لیست یا آرایه مورد نظر بسازیم.

```
let seqFromArray2 = [| 1 .. 10 |] |> Seq.ofArray
```

البته این نکته رو هم یادآور بشم که به کمک عملیات تبدیل نوع(type casting) هم می‌تونیم آرایه رو به seq تبدیل کنیم. به صورت زیر

```
let seqFromArray1 = [| 1 .. 10 |] :> seq<int>
```

برای مشخص کردن اینکه آیا یک آیتم در seq موجود است یا نه می‌تونیم از seq.exists به صورت زیر استفاده کنیم.

```
let containsNumber number seq1 = Seq.exists (fun elem -> elem = number) seq1
let seq0to3 = seq {0 .. 3}
printfn "For sequence %A, contains zero is %b" seq0to3 (containsNumber 0 seq0to3)
```

اگر seq پاس داده شده به تابع exists خالی باشد یا یک ArgumentNullException متوقف خواهید شد.

برای جستجو و پیدا کردن یک آیتم در seq می‌تونیم از seq.find استفاده کنیم.

```
let isDivisibleBy number elem = elem % number = 0
let result = Seq.find (isDivisibleBy 5) [ 1 .. 100 ]
printfn "%d " result
```

دقت کنید که اگر هیچ آیتمی در sequence با predicate مورد نظر پیدا نشود یک KeyNotFoundException رخ خواهد داد. در صورتی که مایل نباشید که استثنا رخ دهد می‌توانید از تابع seq.tryFind استفاده کنید. هم چنین خالی بودن sequence ورودی باعث ArgumentNullExceptionخواهد شد.

استفاده از lambda expression در توابع

lamdaExpressoion از توانایی‌ها مورد علاقه برنامه نویسان دات نت است و کمتر کسی است حاضر به استفاده از آن در کوئری‌های linq نباشد. در #F نیز می‌توانید از lambda Expression استفاده کنید. در ادامه به بررسی مثال هایی از این دست خواهیم پرداخت.

تابع skipWhile

همانند skipWhile در linq عمل می‌کند. یعنی یک predicate مورد نظر را بر روی تک تک عناصر یک لیست اجرا می‌کند و آیتم هایی که شرط برای آن‌ها true باشد نادیده گرفته میشوند و مابقی آیتم‌ها برگشت داده می‌شوند.

```
let mySeq = seq { for i in 1 .. 10 -> i*i }
let printSeq seq1 = Seq.iter (printf "%A ") seq1; printfn ""
let mySeqSkipWhileLessThan10 = Seq.skipWhile (
```

fun elem -> elem < 10

```
) mySeq
mySeqSkipWhileLessThan10 |> printSeq
```

می‌بینید که predicate مورد نظر برای تابع skipWhile به صورت lambda expression است که با رنگ متفاوت نمایش داده شده است.(استفاده از کلمه fun).

خروجی به صورت زیر است:

```
16 25 36 49 64 81 100
```

برای بازگرداندن یک تعداد مشخص از آیتم‌های seq می‌تونید از توابع seq.take یا seq.truncate استفاده کنید. ابتدا باید تعداد مورد نظر و بعد لیست مورد نظر را به عنوان پارامتر مقدار دهی کنید.

مثال:

```
let mySeq = seq { for i in 1 .. 10 -> i*i }
let truncatedSeq = Seq.truncate 5 mySeq
let takenSeq = Seq.take 5 mySeq

let printSeq seq1 = Seq.iter (printf "%A ") seq1; printfn ""


#1 truncatedSeq |> printSeq
#3 takenSeq |> printSeq
```

خروجی

```
1 4 9 16 25 //truncate
1 4 9 16 25 //take
```

Tuples

tuples در #F به گروهی از مقادیر بی نام ولی مرتب شده که می‌توانند انواع متفاوت هم داشته باشند گفته می‌شود. ساختار کلی آن به صورت (element , ... , element) است که هر element خود می‌تواند یک عبارت نیز باشد.(مشابه کلاس Tuple در #C که به صورت generic استفاده می‌کنیم)

```
// Tuple of two integers.
( 1, 2 )

// Triple of strings.
( "one", "two", "three" )

// Tuple of unknown types.
( a, b )

// Tuple that has mixed types.
( "one", 1, 2.0 )

// Tuple of integer expressions.
( a + 1, b + 1)
```

نکات استفاده از tuple

#1 می‌تونیم از الگوی Matching برای دسترسی به عناصر tuple استفاده کنیم.

```
let print tuple1 =
   match tuple1 with
    | (a, b) -> printfn "Pair %A %A" a b
```

#2 میتونیم از let برای تعربف الگوی tuple استفاده کنیم.

```
let (a, b) = (1, 2)
```

#3 توابع fst و snd مقادیر اول و دوم هر tuple رو بازگشت می‌دهند

```
let c = fst (1, 2) // return 1
let d = snd (1, 2)// return 2
```

#4 تابعی برای بازگشت عنصر سوم یک tuple وجود ندارد ولی این تابع رو با هم می‌نویسیم:

```
let third (_, _, c) = c
```

کاربرد tuple در کجاست

زمانی که یک تابع باید بیش از یک مقدار را بازگشت دهد از tupleها استفاده می‌کنیم. برای مثال

```
let divRem a b =
   let x = a / b
   let y = a % b
   (x, y)
```

خروجی تابع divRem از نوع tuple که دارای 2 مقدار است می‌باشد.