عنوان: **شناسه ها و استفاده از Let**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۱۷ ۳:۳۳
آدرس: www.dotnettips.info
برچسب‌ها: F#, Programming

F# هم مانند سایر زبان‌های برنامه نویسی از یک سری Data Type به همراه عملگر و Converter پشتیبانی می‌کند که در ابتدا لازم است یک نگاه کلی به این موارد بیندازیم. به دلیل آشنایی اکثر دوستان به این موارد و به دلیل اینکه تکرار مکررات نشود از توضیح در این موارد خودداری خواهم کرد.(در صورت مبهم بودن می‌توانید از قسمت پرسش و پاسخ استفاد نمایید)

Basic Literal

Type	Description	Sample Literals	.NET Name
bool	True/false values	true, false	System.Boolean
byte	8-bit unsigned integers	0uy, 19uy, 0xFFuy	System.Byte
sbyte	8-bit signed integers	0y, 19y, 0xFFy	System.SByte
int16	16-bit signed integers	0s, 19s, 0x0800s	System.Int16
uint16	16-bit unsigned integers	0us, 19us, 0x0800us	System.UInt16
int, int32	32-bit signed integers	0, 19, 0x0800, 0b0001	System.Int32
uint32	32-bit unsigned integers	0u, 19u, 0x0800u	System.UInt32
int64	64-bit signed integers	0L, 19L, 0x0800L	System.Int64
uint64	64-bit unsigned integers	0UL, 19UL, 0x0800UL	System.UInt64
nativeint	Machine-sized signed integers	0n, 19n, 0x0800n	System.IntPtr
unativeint	Machine-sized unsigned integers	0un, 19un, 0x0800un	System.UIntPtr
single, float32	32-bit IEEE floating-point	0.0f, 19.7f, 1.3e4f	System.Single
double, float	64-bit IEEE floating-point	0.0, 19.7, 1.3e4	System.Double
decimal	High-precision decimal values	0M, 19M, 19.03M	System.Decimal
bigint	Arbitrarily large integers	0I, 19I	Math.BigInt
bignum	Arbitrary-precision rationals	0N, 19N	Math.BigNum
unit	The type with only one value	()	Core.Unit

جدول بالا کاملا واضح است و برنامه نویسان دات نت نظیر C# با انواع داده ای بالا آشنایی دارند. فقط در مورد گزینه آخر unit در فصل‌های بعدی توضیح خواهم داد.

Arithmetic Operators (عملگرهای محاسباتی)

Operator	Description	Sample Use on int	Sample Use on float
+	Unchecked addition	1 + 2	1.0 + 2.0
-	Unchecked subtraction	12 - 5	12.3 - 5.4
*	Unchecked multiplication	2 * 3	2.4 * 3.9
/	Division	5 / 2	5.0 / 2.0
%	Modulus	5 % 2	5.4 % 2.0
-	Unary negation	-(5+2)	-(5.4+2.4)

Simple String (کار با نوع داده رشته ای)

Example	Kind	Type
"Humpty Dumpty"	String	string
"c:\\Program Files"	String	string
@"c:\Program Files"	Verbatim string	string
"xyZy3d2"B	Literal byte array	byte []
'c'	Character	char

بعد از بررسی موارد بالا حالا به معرفی شناسه‌ها می‌پردازم. شناسه‌ها در F# راهی هستند برای اینکه شما به مقادیر نام اختصاص دهید. برای اختصاص نام به مقادیر کافیست از کلمه کلیدی let به همراه یک نام و علامت = و یک عبارت استفاده کنید. چیزی شبیه به تعریف متغیر در سایر زبان‌ها نظیر C#. دلیل اینکه در F# به جای واژه متغیر از شناسه استفاده می‌شود این است که شما می‌توانید به یک شناسه تابعی را نیز اختصاص دهید و مقدار شناسه‌ها دیگر قابل تغییر نیست. در F# کلمه متغیر یک واژه نادرست است چون زمانی که شما یه یک متغیر مقدار اختصاص می‌دهید، مقدار اون متغیر دیگه قایل تغییر نیست. برای همین اکثر برنامه نویسان F# به جای استفاده از واژه متغیر از واژه مقدار یا شناسه استفاه می‌کنند. برای همین از واژه متغیر برای نام گذاری استفاده نمی‌شود. (البته در F# در بعضی مواقع ما شناسه‌ها رو دوباره تعریف می‌کنیم که چیزی شبیه به استفاده از متغیر هاست ولی با اندکی تفاوت. در این فصل تمرکز ما بر روی شناسه هایی است که مقدارشان تغییر نمی‌کند ولی در فصل برنامه نویسی دستوری به تفصیل در این باره توضیح داده شده است)

```
let x = 42
```

در بالا یک شناسه به نام x تعریف شد که مقدار 42 را دریافت کرد. در #F یک شناسه می‌تواند دارای یک مقدار معین باشد یا به یک تابع اشاره کند. این بدین معنی است #F معنی حقیقی برای تابع و پارامترهای آن ندارد و همه چیز رو به عنوان مقدار در نظر می‌گیرد.

```
let myAdd = fun x y -> x + y
```

کد بالا تعریف یک شناسه به نام myAdd است که به تابعی اشاره می‌کنه که دو پارامتر ورودی دارد و در بدنه آن مقدار پارامترها با هم جمع می‌شوند.(تعریف توابع به صورت مفصل بحث خواهد شد.) نکته جالب این است که تابع تعریف شده نام ندارد و #F **دقیقا با توابع همون رفتاری رو داره که با شناسه‌ها دارد.**

```
let raisePowerTwo x = x ** 2.0
```

در کد بالا شناسه ای تعریف شده است با نام raisePowerTwo که یک پارامتر ورودی داره به نام x و در بدنه آن (هرچیزی که بعد از = قرار گیرد) مقدار x رو به توان دو می‌کنه.

نام گذاری شناسه ها

برای نام گذاری شناسه‌ها نام انتخابی یا باید با Underscore شروع شود یا با حروف. بعد از آن می‌تونید از اعداد هم استفاده کنید.(نظیر سایر زبان‌های برنامه نویسی)

#F از unicode هم پشتیبانی می‌کنه یعنی می‌تونید متغیری به صورت زیر رو تعریف کنید.

```
let مسعود = ""
```

اگر احساس می‌کنید که قوانین نام گذاری در #F کمی محدود کننده است می‌تونید از علامت '' '' استفاده کنید و در بین این علامت هر کاراکتری که می‌خواهید رو قرار دهید و #F اونو به عنوان نام شناسه قبول خواهد کرد. برای نمونه

```
let ``more? `` = true
```

یا

```
let ``class`` = "style"
```

حتی امکان استفاده از کلمات کلیدی هم نظیر class به این روش وجود دارد.

محدوده تعریف شناسه ها

به دلیل اینکه در #F از {} به عنوان شروع و اتمام محدوده استفاده نمی‌شود دونستن و شناختن محدوده توابع بسیار مهم و ضروری است. چون اگر از شناسه ای که در یک محدوده در دسترس نباشد استفاده کنید با خطای کامپایلر متوقف خواهید شد.

همون بحث متغیرهای محلی و سراسری (در سایر زبان ها) در این جا نیز صادق است یعنی در #F شناسه‌های سراسری و محلی خواهیم داشت. تمام شناسه ها، چه اون هایی که در توابع استفاده می‌شوند و چه اونهایی که به مقادیر اشاره می‌کنند محدودشون از نقطه ای که تعریف می‌شوند تا جایی که اتمام استفاده از اونهاست تعریف شده است. برای مثال اگر یک شناسه رو در بالای فایل تعریف کنید که یک مقدار دارد تا پایان SourceFile قابل استفاده است.(به دلیل نبود مفهوم کلاس از واژه sourceFile استفاده کردم). هم چنین شناسه هایی که در توابع تعریف می‌شوند فقط در همون توابع قابل استفاده هستند.

حالا سوال این است که با نبودن {} چگونه محدوده خود توابع مشخص میشود؟

در #F با استفاده از فضای خای یا space محدوده شناسه‌ها و توابع رو مشخص می‌کنیم. برای روشن شدن مطلب به مثال زیر دقت کنید.

```
let test a b =
    let dif = b - a
```

```
    let mid = dif / 2
    mid + a

printfn "(test 5 11) = %i" (test 5 11)
printfn "(test 11 5) = %i" (test 11 5)
```

ابتدا اختلاف بین دو ورودی محاسبه می‌شود و در یک شناسه به نام dif قرار می‌گیرد. برای اینکه مشخص شود که این شناسه خود عضو یک تابع دیگر به نام test است از 4 فضای خالی استفاده شده است. در خط بعدی شناسه mid مقدار شناسه dif رو بر 2 تقسیم می‌کند. در انتها نیز مقدار mid با مقدار a جمع می‌شود و حاصل برگشت داده می‌شود.(انتهای بدنه تابع)
نکته مهم : به جای استفاده از فضای خالی(space) نمی‌تونید از TAB استفاده کنید.

LIGHTWEIGHT SYNTAX یا VERBOSE SYNTAX

در #F دو نوع سبک کد نویسی وجود دارد. یکی lightweight و دیگری Verbose. البته اکثر برنامه نویسان از سبک lightweight که به صورت پیش فرض در #F تعبیه شده است استفاده می‌کنند ولی آشنایی با سبک verbose نیز به عنوان برنامه نویس #F ضروری است. ما نیز به تبعیت از سایرین از سبک lightweight استفاده خواهیم کرد ولی یک فصل به عنوان مطالب تکمیلی اختصاص دادم که تفاوت این دو سبک را در طی چندین مثال بیان میکند.
همان طور که قبلا بیان شد #F بر اساس زبان OCaml پیاده سازی شده است. زبان OCaml مانند #F، یک زبان LIGHTWEIGHT SYNTAX نیست. LIGHTWEIGHT SYNTAX بدین معنی است محدوده شناسه‌ها بر اساس فضای خالی بین اون‌ها مشخص می‌شود نه با ;. (البته استفاده از ; به صورت اختیاری است)
بازنویسی مثال بالا

```
let halfWay a b =
let dif = b - a in
let mid = dif / 2 in
mid + a
```

برای اینکه کامپایلر #F متوجه شود که قصد کدنویسی به سبک lightweight رو نداریم، باید در ابتدای هر فایل از دستور زیر استفاده کنیم.

```
#light "off"
```