به نام خدا

آموزش برنامه نویسی به زبان HASKELL

فصل اول

صالح خزاعی

پاییز ۱۳۹۳

HASKELL **چیست ؟**

تعریف سایت رسمی Haskell از این زبان برنامه نویسی بدین صورت است :

“Haskell is a computer programming language. In particular, it is a *polymorphically statically typed, lazy, purely functional* language, quite different from most other programming languages.”

اما به زبان ساده تر زبان برنامه نویسی Haskell یک زبان Functional است. همانطور که در زبان های برنامه نویسی شی گرا مثل جاوا همه چیز شی هستند, در این زبان نیز همه چیز تابع هستند.

این زبان برنامه نویسی برای افرادی که با فرمول های ریاضی و الگوریتم ها سر و کار دارند مناسب است.

خُب , شروع کنیم ☺

اولین کاری که باید بکنیم اینه که پلتفرم Haskell رو دانلود و نصب کنیم

تنها کاری که لازمه انجام بدید اینه که برید به سایت http://www.haskell.org قسمت دانلود و Haskell رو دانلود کنید.

من توی سیستم عامل Windows با این زبان کار میکنم پس تمام توضیحاتم با این پیش فرض است که شما هم از سیستم عامل Windows استفاده می کنید.

ما برای کامپایل کردن برنامه هامون از کامپایلر GHC استفاده می کنیم (نگران نباشید وقتی از سایت رسمی هسکل نرم افزارش رو دانلود و نصب کنید این کامپایلر و سایر نرم افزار های مورد نیازتون هم نصب میشه)

این کامپایلر محیطی برای اجرای کد برای خودش داره (یک جور Shell) به اسم ghci . هروقت خواستید با این محیط کار کنید کافیه در محیط CMD تایپ کنید ghci (برای باز کردن محیط CMD کافیه توی منوی استارت تایپ کنید CMD ☺)

خب وقتی وارد محیط ghci شوید بعد از چند ثانیه خطی به صورت زیر میاد

```
Prelude>
```

این یعنی محیط نرم افزار اماده است برای اینکه شما دستوراتتون رو وارد کنید

ما فعلا می خواهیم در همین محیط با دستورات Haskell کار کنیم و چون نوشتن کد در فایل جداگانه و لود کردن و .... طول میکشه

پس از اعلان ghci استفاده می کنیم

برای تغییر اعلانتون کافیه خط زیر رو وارد کنید

```
:set prompt “ghci> “
```

خب برای شروع یک سری عملیات های ساده رو انجام میدیم برای مثال عملیات های ریاضی و خروجی های هرکدوم رو در زیر میتونید ببینید (خودتون هم تست کنید ☺)

```
ghci> 2 + 15
17
ghci> 49 * 100
4900
ghci> 1892 - 1472
420
ghci> 5 / 2
2.5
```

همچنین میتونیم از پرانتز هم استفاده کنیم دقت کنید که اعداد منفی حتما داخل پرانتز گذاشته بشن وگرنه کامپایلر سرتون داد میکشه !!!!
یعنی ۵ * -۳ غلط و ۵ * (-۳) درست است.
همچنین میتونیم عملیات های ساده منطقی رو هم انجام بدیم همانطور که احتمالا از زبان های دیگر برنامه نویسی با && و || آشنا هستید این دو عملگر به ترتیب برای AND و OR استفاده می شوند. همچنین از not برای منفی کردن استفاده میشه.

```
ghci> True && False
False
ghci> True && True
True
ghci> False || True
True
ghci> not False
True
ghci> not (True && True)
False
```

همچنین عملگر های تساوی به صورت زیر است

```
ghci> 5 == 5
True
ghci> 1 == 0
False
ghci> 5 /= 5
False
ghci> 5 /= 4
True
ghci> "hello" == "hello"
True
```

یعنی برای برابر بودن از عملگر == و برای نابرابری از عملگر /= استفاده می کنیم.

اما اگر دستوراتی مثل ۵ + "llama" یا ۵ == Trueرا وارد کنیم چه می شود؟ این از اون مواقعی است که کامپایلر سر شما فریاد خواهد کشید. اگر هنوز هم دوست دارید با این فریاد ها مواجه شوید کافیست یکی از مثال های بالا را تست کنید تا با این پیغام خطای بزرگ و ترسناک روبرو شوید.

```
No instance for (Num [Char])
arising from a use of `+' at <interactive>:1:0-9
Possible fix: add an instance declaration for (Num [Char])
In the expression: 5 + "llama"
In the definition of `it': it = 5 + "llama"
```

پیغام بالا به صورت خلاصه یعنی مقادیری که وارد کردید از یک نوع نیستند!

همانطور که گفتم زبان برنامه نویسی Haskell یک زبان Functional است و همه چیز اینجا Function هستند حتی همین عملیات های ریاضی که انجام دادید! مثلا همین عملیات ضرب (*) یک تابع است با دو ورودی که در اینجا این تابع به صورت infix صدا زده شده است. کسانی که با توابع در زبان های برنامه نویسی دیگر کار کرده باشند توابع در زبان هایی مثل C به صورت Prefix صدا زده می شوند همچنین همیشه بعد از نام تابع پرانتز و ورودی های تابع در داخل پرانتز ها با جداکننده ای مثل کاما وارد می شوند اما در زبان Haskell برای صدا زدن یک تابع ابتدا نام تابع را می نویسیم سپس یک کاراکتر فاصله می گذاریم و ورودی های تابع را وارد میکنیم کاراکتری که این ورودی ها را جدا می کند کاراکتر فاصله است.

برای مثال وقتی می خواهیم min یا max را حساب کنیم این توابع را به صورت زیر صدا می زنیم

```
ghci> min 9 10
9
ghci> min 3.4 3.2
3.2
ghci> max 100 101
101
```

اما اگر بخواهیم min بین دو عدد ۹ * ۹ و ۷ * ۲ را حساب کنیم ورودی ها را باید به صورت زیر وارد کنیم

```
ghci> min (9 * 9) (7 * 2)
```

برای یک مثال بهتر تابع succ را در نظر میگیریم

این تابع مقدار بعدی نسبت به ورودی را میدهد برای مثال succ 3 مقدار ۴ را میدهد و succ ‘a’ مقدار ‘b’ را در خروجی به ما میدهد.

فرض کنید می خواهیم succ (4 * 9) را محاسبه کنیم خروجی مورد انتظار ما از این کد مقدار ۳۷ است ولی چنانچه

اشتباها کد را به صورت succ 4 * 9 بنویسیم با مقدار ۴۵ روبرو خواهیم شد چون ابتدا مقدار succ 4 محاسبه می شود و سپس مقدار خروجی (۵) در عدد ۹ ضرب می شود

حال فرض کنید دو عدد را می خواهیم بر یکدیگر تقسیم کنیم. تابعی که این کار را برای ما انجام می دهد تابع Div است.

برای صدا زدن این تابع به صورت Prefix دستور Div 92 10 اما همه ما عادت کرده ایم عملگر تقسیم را بین دو عدد بگذاریم و همچنین تشخیص مقسوم و مقسوم علیه در این حالت آسان نیست. در این حالات که مقادیر ورودی تابع دو مقدار است می توانیم تابع را به صورت Infix یعنی ۹۲ `div` 10 صدا بزنیم.

دستور زیر را ببینید

```
Boo ( Boo 3 )
```

در نگاه اول به نظر می رسد تابع Boo با دو ورودی Boo و ۳ صدا زده شده است اما در واقع ابتدا تابع Boo با مقدار ۳ صدا زده شده است و مقدار خروجی آن به عنوان ورودی تابع Boo خروجی در نظر گرفته شده است.

تعریف توابع

حالا میخواهیم که یک تابع برای دو برابر کردن مقدار یک متغیر تعریف کنیم.

```
doubleMe x = 2 * x
```

راحت بود,نه؟ برای تعریف یک تابع کافیست ابتدا نام تابع را بنویسیم سپس با یک کاراکتر فاصله ورودی ها را بنویسیم سپس کاراکتر = را می گذاریم و کاری که تابع باید انجام بدهد را می نویسیم و خروجی ای که باید برگرداند را محاسبه می کنیم

حالا می خواهیم این تابع را در یک اسکریپت قرار دهیم و آنرا لود کنیم تا بتوانیم در ghci از آن استفاده کنیم

فایلی با نام ex1.hs بسازید و کد تابع خودتان را در آن قرار دهید و آنرا ذخیره کنید. (فایل های برنامه های ما به زبان Haskell باید با پسوند hs ذخیره شوند)

سپس در محیط ghci دستور زیر را وارد کنید

```
:l ex1
```

این دستور فایل ex1.hs را کامپایل و لود می کند. در صورت لود شدن صحیح فایل با پیغام زیر رو برو میشوید.

```
ghci> :l ex1
[1 of 1] Compiling Main             ( ex1.hs, interpreted )
Ok, modules loaded: Main.
```

حال می توانید از توابع تعریف شده خود استفاده کنید.

حالا میخواهیم تابعی بنویسیم که اعداد کوچک تر از ۱۰۰ را دو برابر کند و اعداد بزرگتر از ان را بدون تغییر بازگرداند

ایده شما چیست؟

```
doubleSmallNumber x = if x > 100
                      then x
                      else x*2
```

راحت بود نه؟ حالا یک چیز جالب تر ! شما می توانید از شرط ها به این صورت نیز استفاده کنید

```
doubleSmallNumber' x = (if x > 100 then x else x*2) + 1
```

اگر یکم دقت کنید متوجه کاراکتر ' در اسم تابع می شوید, در زبان Haskell برای نام گذاری توابع کاراکتر ' نیز مجاز است و شما می توانید از آن در نام گذاری توابع استفاده کنید

نکته: به جای نوشتن a=1 در اسکریپت و لود کردن آن میتوانیم در محیط ghci دستور let a = 1 را وارد کنیم.

لیست ها

در Haskell لیست ها همگن هستند ! یعنی شما می توانید لیستی از اعداد تعریف کنید یا لیستی از رشته ها ولی نمیتوانید لیستی از اعداد و رشته ها داشته باشید.

تعریف یک لیست به صورت زیر است

```
let a = [0,1,2,3,4,5,6]
```

همچنین رشته ها در Haskell لیستی از کاراکتر ها هستند برای مثال رشته ‘‘Hello’’ لیستی به صورت زیر است

```
[‘H’,’e’,’l’,’l’,’o’]
```

کار با لیست ها

ترکیب دو لیست

برای ترکیب دو لیست از کاراکتر ++ استفاده می کنیم. برای مثال

```
[1,2,3,4] ++ [5,6,7,8]
[1,2,3,4,5,6,7,8]
“Hello” ++ “ “ ++ “World!”
“Hello World!”
```

همچنین وقتی می خواهیم یک عنصر را به اول لیست اضافه کنیم از کاراکتر : استفاده می کنیم

```
1:[2,3,4,5,6]
[1,2,3,4,5,6]
‘A’:” Small Cat”
“A Small Cat”
```

همچنین [] به عنوان لیست خالی در نظر گرفته می شود وقت کنید که لیست های [[]] و [[],[],[]] خالی نیستند بلکه عناصری که دارند لیست های خالی هستند.

برای انتخاب یک عنصر بر اساس شماره محل قرار گیری آن از عملگر !! استفاده می کنیم شماره محل قرار گیری اولین عنصر ۰ است

```
[1,2,3,4,5,6]  !! 5
6
```

شما می توانید با عملگر های > >= < <= لیست ها را در صورت قابل مقایسه بودن مقایسه کنید

```
[3,2,1] > [1,2,3]
True
```

در این مقایسه ها ابتدا عنصر اول لیست اول با عنصر اول لیست دوم مقایسه میشود سپس عنصر دوم هر دو لیست و به همین ترتیب جلو می رود

توابع برای کار با لیست

head

این تابع عنصر ابتدای لیست را برمیگرداند

last

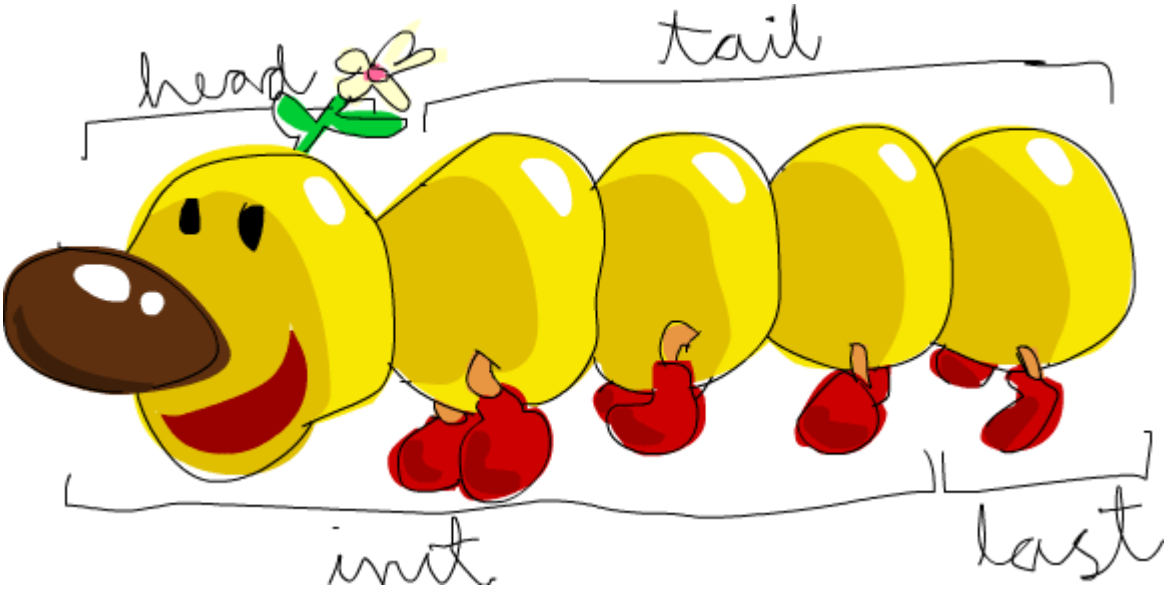
این تابع عنصر انتهای لیست را بر میگرداند

tail

این تابع کل لیست را بدون عنصر ابتدای لیست برمیگرداند

init

این تابع کل لیست را بدون عنصر انتهای لیست برمیگرداند



مراقب باشید که استفاده از تابع های init , last , tail , head روی لیست های خالی منجر به خطا خواهد شد.

length

طول لیست را برمیگرداند

null

این تابع چک می کند که لیست خالی است یا خیر.

reverse

این تابع لیست را برعکس می کند

take

این تابع یک عدد می گیرد و به همان تعداد از ابتدای لیست جدا می کند و برمیگرداند

```
take 3 [1,2,3,4,5,6]
[1,2,3]
take 1 [1,2,3]
[1]
```

drop

این تابع شبیه تابع take عمل می کند با این تفاوت که به تعدادی که به تابع داده شده از ابتدای تابع را در نظر نمی گیرد و بقیه تابع را برمیگرداند

maximum

این تابع بزرگترین مقدار داخل یک لیست را برمیگرداند

minimum

این تابع کوچکترین مقدار داخل یک لیست را برمیگرداند

sum

این تابع جمع مقادیر داخل لیست را برمیگرداند

product

این تابع ضرب مقادیر داخل لیست را برمیگرداند

elem

این تابع یک عنصر و یک لیست میگیرد و وجود داشتن عنصر در لیست را بررسی می کند

```
4 `elem` [1,2,3,4]
True
```

فرض کنید می خواهید لیستی از اعداد ۱ تا ۲۰ درست کنید؟ ایده شما چیست؟ یکی یکی اعداد را وارد کنید؟ اگر ۱ تا ۱۰۰ را میخواستید چی؟

برای اینکار در Haskell روش های خیلی راحتی وجود دارد

برای لیست اعداد ۱ تا ۲۰ کافیست دستور زیر را بزنید

```
[1..20]
```

حال اگر حروف کوچک از 'g' تا 'w' را میخواستید

['g'..'w']

اعداد زوج بین ۱ تا ۲۰

[2,4..20]

از ۱ تا ۲۰ , ۳ تا ۳ تا

[1,4..20]

دقت کنید که شما نمی توانید با این روش لیستی از توان های ۲ را بسازید زیرا فقط و فقط باید دو عدد را در ابتدا مشخص کنید که step مشخص شود و بعد از .. سقف را مشخص کنید.

همچنین برای لیستی از اعداد از ۲۰ تا ۱ باید به صورت زیر عمل کنید

[20,19..1]

در Haskell می توانید لیست های نا متناهی تعریف کنید برای مثال وقتی می خواهید ۲۴ عدد ابتدای مضارب ۱۳ را داشته باشید

take 24 [13,26..]

Haskell وقتی شما لیست های نامتناهی را تعریف می کنید انها را نمیسازد بلکه وقتی انها را می سازد که شما بخواهید با آن کاری انجام دهید

cycle

این تابع یک لیست می گیرد و آن را به صورت نامتناهی تکرار می کند

take 3 (cycle [1,2,3])
[1,2,3,1,2,3,1,2,3]

repeat

این تابع یک عنصر میگیرد و آن را در یک لیست به صورت نامتناهی تکرار می کند

take 6 (repeat 5)
[5,5,5,5,5,5]

replicate

این تابع یک عنصر و تعداد تکرار را میگیرد و در یک لیست به آن تعداد عنصر را تکرار می کند

replicate 3 5
[5,5,5]

list comprehension

حال می خواهیم عبارت های ریاضیاتی مثل $S = \{ 2 \cdot x \mid x \in \mathbb{N},\ x \leq 10 \}$ را در Haskell وارد کنیم

برای مثال همین عبارت بالا

```
[ 2*x | x <- [1,2..10] ]
[2,4,6,8,10,12,14,16,18,20]
```

حالا فرض کنید یک شرط هم میخواهیم بهش اضافه کنیم

```
ghci> [ 2*x | x <- [1,2..10] , x*2 < 15 ]
[2,4,6,8,10,12,14]
ghci> [ x | x <- [100,105..500] , x `mod` 7 == 3 ]
[115,150,185,220,255,290,325,360,395,430,465,500]
```

حالا میخواهیم عبارتی وارد کنیم که تمام اعداد فرد بزرگتر از ۱۰ را با "BANG!" و اعداد فرد کوچکتر از ۱۰ را با "BOOM!" جایگزین کنیم و اگر عددی فرد نبود آنرا از لیست بیرون بیاندازیم.

```
ghci> let boomBangs xs = [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd x]
ghci> boomBangs[7..13]
["BOOM!","BOOM!","BANG!","BANG!"]
```

حال میخواهیم ضرب اعداد دو مجموعه در یکدیگر را ببینیم

```
[ x*y | x <- [1,2,3] , y <- [4,5,6] ]
[4,5,6,8,10,12,12,15,18]
```

ترکیب اسم ها و صفت ها چطوره؟

```
ghci> let nouns = ["hobo","frog","pope"]
ghci> let adjectives = ["lazy","grouchy","scheming"]
ghci> [adjective ++ " " ++ noun | adjective <- adjectives, noun <- nouns]
["lazy hobo","lazy frog","lazy pope","grouchy hobo","grouchy frog",
"grouchy pope","scheming hobo","scheming frog","scheming pope"]
```

Tuples

تعریف یک چند تایی کاملا شبیه تعریف لیست است با این تفاوت که برای تعریف یک چند تایی باید به جای [] ان را داخل () قرار دهید

فرق اساسی یک چندتایی (Tuple) و لیست در این است که شما می توانید چند تایی ای از عدد و رشته داشته باشید.

و همچنین وقتی لیستی از چند تایی ها تعریف می کنید همه چند تایی هایتان باید شبیه یکدیگر باشند ! یعنی برای مثال باید همه انها دوتایی ای از عدد و عدد باشند. در غیر اینصورت کامپایلر به شما خطایی شبیه خطای زیر را نشان خواهد داد

```
Couldn't match expected type `(t, t1)'
against inferred type `(t2, t3, t4)'
In the expression: (8, 11, 5)
In the expression: [(1, 2), (8, 11, 5), (4, 5)]
In the definition of `it': it = [(1, 2), (8, 11, 5), (4, 5)]
```

توابع کار با چندتایی ها

fst

این تابع یک **دوتایی** میگیرد و عنصر اول آن را بازمیگرداند

snd

این تابع یک **دوتایی** میگیرد و عنصر دوم آن را بازمیگرداند

توجه کنید که تابع های fst و snd فقط برای کار با دوتایی ها استفاده میشود.

zip

این تابع دو لیست میگیرد و آن ها را به صورت لیستی از دوتایی ها ترکیب میکند. اگر طول یک لیست از طول لیست دیگر بزرگتر باشد عناصر اضافی ان را در نظر نمیگیرد

```
ghci> zip [1,2,3,4,5] [5,5,5,5,5]
[(1,5),(2,5),(3,5),(4,5),(5,5)]
ghci> zip [1 .. 5] ["one", "two", "three", "four", "five"]
[(1,"one"),(2,"two"),(3,"three"),(4,"four"),(5,"five")]
```

حال میخواهیم عبارتی بنویسیم که تمام مثلث های ممکن که اضلاع آن کوچکتر از ۱۰ باشند را بدست بیاورد

```
ghci> [ (a,b,c) | c <- [1..10] , b <- [1..c] , a <- [1..b] ,a^2 + b^2 == c^2 ]
[(3,4,5),(6,8,10)]
```

فصل بعدی در مورد Types and Typeclasses است. شما همچنین همین توضیحات را به صورت کامل تر می توانید از منابع معرفی شده بخوانید.

منابع

[1] http://learnyouahaskell.com

[2] http://haskell.org