GHCi: Getting started (1A)

Young Won Lim
6/5/17

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

Interpreter GHCi

```
young@MNTSys-BB1 ~ $ ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude> "hello, world!"
"hello, world!"
Prelude> putStrLn "hello, world!"
hello, world!
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Function

```
Prelude> let fac n = if n == 0 then 1 else n * fac (n-1)
Prelude> fac 5
120
Prelude> fac 2
2
Prelude> fac 3
6
Prelude> fac 4
24
Prelude>
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Compiler GHC

```
young@MNTSys-BB1 ~ $ ghc -o hello hello.hs
[1 of 1] Compiling Main             ( hello.hs, hello.o )
Linking hello ...
young@MNTSys-BB1 ~ $ ./hello
hello, world!
```

```
young@MNTSys-BB1 ~ $ cat hello.hs
main = putStrLn "hello, world!"
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Layout

t.hs

```
main = do putStrLn "Type an integer : ?"
          x <- readLn
          if even x
           then putStrLn "even number"
           else putStrLn "odd number"
```

the first non-space character after **do.**

every line that starts in the same column
as that p is in the d**o** block

If you indent more, it is the <u>nested</u> block in **do**

If you indent less, it is an <u>end</u> of the **do** block.

```
ghc t.hs
./t
```

```
ghc –o  run t.hs
./t
```

Multi-line in GHCi

ghci multi-line

```
Prelude> :{
Prelude| main = do { putStrLn "Type an integer: "; x<-readLn;
Prelude| if even x then putStrLn "even" else putStrLn "odd"; }
Prelude| :}
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Types

Int	an integer with at least 30 bits of precision.
Integer	an integer with unlimited precision.
Float	a single precision floating point number.
Double	a double precision floating point number.
Rational	a fraction type, with no rounding error.

Types and Class Types start with capital letters

Variables start with lower case letters

Declaring a type	:: type
Asking which type	:t something

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Type Classes

class constraint

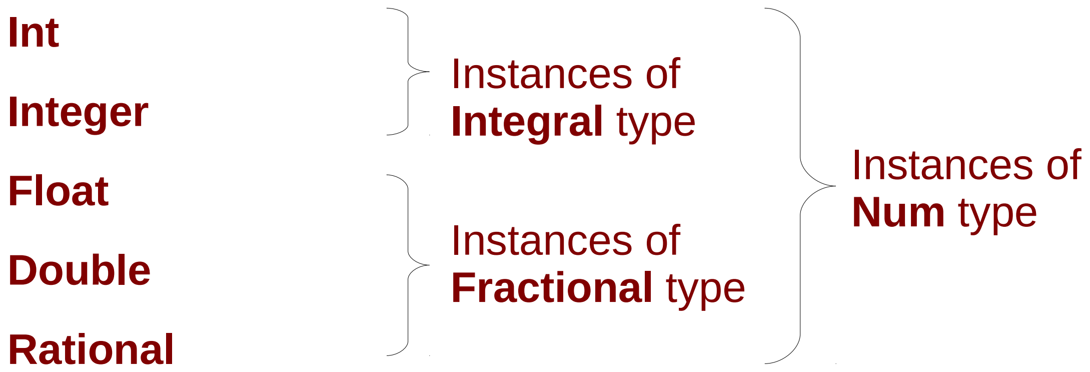
the <u>type</u> t is *constrained* by the <u>context</u>
(Num t), (Fractional t), (Integral t)

(Num t) =>	the **types** of t must be **Num type class**
(Fractional t) =>	the **types** of t must be **Fractional type class**
(Integral t) =>	the **types** of t must be **Integral type class**

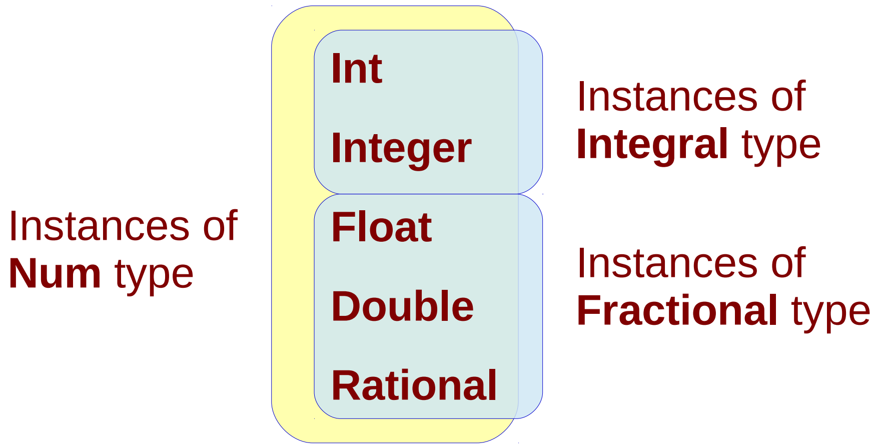
```
Prelude> 3 :: Int
3
Prelude> 3 :: Float
3.0
Prelude> 4 :: Double
4.0
Prelude> 2 :: Integer
2
Prelude> :t 3
3 :: Num a => a                 3 can be used as any numeric type
Prelude> :t 2.0
2.0 :: Fractional a => a        2.0 can be used as any fractional type
Prelude> :t gcd 15 20
gcd 15 20 :: Integral a => a    gcd 15 20 can be used as any integral type
Prelude> :t True
True :: Bool
Prelude> :t 'A'
'A' :: Char
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Type Classes


Int
Integer
Float
Double
Rational
Instances of **Integral** type
Instances of **Fractional** type
Instances of **Num** type

Type Class : a set of type (instances)


Int
Integer
Float
Double
Rational
Instances of **Integral** type
Instances of **Fractional** type
Instances of **Num** type

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Lists and Tuples

Lists	multiple values of the same type
Strings	lists of characters.
Tuples	a fixed number of values, which can have different types.

The **:** operator appends an item to the beginning of a list

Zip : two lists into a list of tuples.

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Functions

[1 .. 10]	[1,2,3,4,5,6,7,8,9,10]
map (+ 2) [1 .. 10]	[3,4,5,6,7,8,9,10,11,12]
filter (> 2) [1 .. 10]	[3,4,5,6,7,8,9,10]
fst (1, 2)	1
snd (1, 2)	2
map fst [(1, 2), (3, 4), (5, 6)]	[1,3,5]

~~fst (1, 2, 3)~~

~~snd (1, 2, 3)~~

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Functions

```
my_sum m n = m+n

main = do  putStrLn "Give two numbers: "
           x <- readLn
           y <- readLn
           print (my_sum x y)
```

```
Give two numbers:
10
20
30
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

Convenient Syntax

```
secsToWeeks secs =let perMinute = 60
                      perHour   = 60 * perMinute
                      perDay    = 24 * perHour
                      perWeek   =  7 * perDay
                  in secs / perWeek
```

```
classify age = case age of  0 -> "newborn"
                            1 -> "infant"
                            2 -> "toddler"
                            _ -> "senior citizen"
```

https://wiki.haskell.org/Learn_Haskell_in_10_minutes

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf