## Week 1

NO STATE UPDATES, 字符单引号，字符串双引

Function:

Name, input (argument), output 可以没有argument

swap:: (Int,Int) -> (Int,Int)

swap (x,y) = (y,x)

&& || not == /=

:t functionname (可以查function的type)

if then else

List: 同种type, 可改长度，可以直接比，空列表最小

let lostNums =[4,78,5,900]

++，两个list合并

5 : [6,34,23]， 一个element往前加

！！ 取值

List of list

[20,19..1]

take 11 [13,26..]， take 5 (cycle [4,3,2])， take 6 (repeat 7)， replicate 5 ‘a’

Float 的 precision有点问题

[ x | x <- [50..100], x `mod` 7 == 5]

`elem`

sum

\*prefix, infix

Tuple: 长度不变，可多种类型

但如果放到list里，就必须同种类型，tuple的长度也要一样

zip [1,3,5,7,9] [‘a’,’b’,’c’,’j’,’k’]，长度跟短的走

[(1,’a’),(3,’b’),(5,’c’),(7,’j’),(9,’k’)]

[ (a,b,c) | c <- [1..10], a <- [1..10], b <- [1..10] ]

[ (a,b,c) | c <- [1..10], a <- [1..c], b <- [1..a], a^2 + b^2 == c^2 ]

多个input，可以直接input tuple

curried function (每次take一个input，return一个function）

length :: [a] -> Int

当不在乎take的是什么type时 （polymorphic）

Class constraint:

(+) :: Num a => a -> a -> a

Eq, Ord, Show, Read, Num Classes

## Week 2

Conditional:

if then else

Guard:

abs n | n ≥ 0 = n

| otherwise = -n

Pattern matching:

fib :: Int -> Int

fib 0 = 0

fib 1 = 1

fib x = fib (x-1) + fib (x-2)

两种顺序都很重要

\_代表不在乎这个变量是啥

Lambda function

\x -> x+1

(\x -> x+1) 3

(\x->\y->x+y) 3 4

(\(x:xs) -> xs) list

double :: Int -> Int

double x = ((\x->x\*2) x)\*2

map (\*3) [1,2,3]

null :: [a] → Bool can be used to test if a list is empty

## Week 3

higher-order if function takes a function as an argument or returns a function as a result

twice :: (a → a) → a → a

twice f x = f (f x)

map :: (a → b) → [a] → [b]

map (+1) [1,3,5,7]

[2,4,6,8]

Prefix a b

a `prefix` b

a infix b

(infix) a b

filter :: (a → Bool) → [a] → [a]

filter even [1..10]

[2,4,6,8,10]

foldr ::(a -> b -> b) -> b -> [a] -> b

foldr (+) 0 [1,2,3] （符号可以是：）也可以是max，min

（从后面、右面开始算的，空列表贴在右面） 从左面的，空列表贴在左边的是foldl

foldr (-) 0 [1,2,3]

1 - ( 2 - ( 3 - 0 ) ) = 2

Pattern match 和 guard 混用

f :: [Int] -> Int

f [] = 0

f (x:xs) | x>0 = (x\*x) + f xs

| otherwise = f xs

Where

f :: [Int] -> Int

f xs = foldr (+) 0 (map sqr (filter pos xs))

where

sqr x = x\*x

pos x = x > 0

## Week 4

type Pos = (Int,Int)

type Pair a = (a,a)

Type and constructor names must be always begin with an uppercase letter

data Shape = Circle Float | Rect Float Float deriving Show

square :: Float -> Shape

square n = Rect n n

area :: Shape -> Float

area (Circle r) = pi \* r^2

area (Rect x y) = x \* y

data Guess a = Naa | Is a deriving Show

safeDiv :: Int -> Int -> Guess Int

safeDiv \_ 0 = Naa

safeDiv m n = Is (m `div` n)

data Student = First | Second | Third | Fourth deriving (Show, Eq, Ord)

可以derive 多个，先定义的小

data Nat = Zero | Succ Nat

nat2int :: Nat -> Int

nat2int Zero = 0

nat2int (Succ n) = 1 + nat2int n

int2nat :: Int -> Nat

int2nat 0 = Zero

int2nat n = Succ (int2nat (n-1))

add Zero n = n

add (Succ m) n = Succ (add m n)

data Tree a = Leaf a | Node (Tree a) a (Tree a)

Efficiency

data Q a = Q [a] [a] deriving (Eq, Show)

## Week 5

words:: String -> [String] 直接用

words "L a n i c e" 以空格划分，相反为unwords

["L","a","n","i","c","e"]

read "34":: Int

type Matrix = [[Int]]

all :: (a -> Bool) -> [a] -> Bool

all p xs = foldr (&&) True (map p xs)

zipWith (+) [1,2,3] [4,5,6]

[5,7,9]

zipWith (-) [1,2,3] [4,5,6]

[-3,-3,-3]

zipWith (zipWith (+)) [[1,2,3],[4,5,6]] [[1,2,3],[4,5,6]]

[[2,4,6],[8,10,12]]

IO a

IO ( ) return no result value

getChar :: IO Char

putChar :: Char -> IO ()

return :: a -> IO a returns the value v, without performing any interaction

getTwo :: IO (Char,Char)

getTwo = do

x <- getChar

y <- getChar

return (x,y)

do : 一连串action

最后一个是function的返回值

getLine :: IO String

putStr :: String -> IO ()

putStrLn :: String -> IO ()

putStr (show (length xs))

reverse [1,2,3]

[3,2,1]

return()