Haskell Summary Sheet
reminders about Haskell syntax & Prelude functions for CISC 260, winter 2017

Boolean Operations:

(&&) :: Bool->Bool->Bool ("and")
True && False = False

(||) :: Bool->Bool->Bool ("or")
True || False = True

not :: Bool->Bool
not True = False

Numeric Operators & Functions:

Int only:

arithmetic: +,-, *

div 17 3 = 5
mod 17 3 = 2
abs 17 = abs (-17) = 17

Float only:

/ (division)
** (exponentiation)

Relational: >,>=,==,=/=,<

floor :: Float -> Int
floor 7.8 = 7
ceiling :: Float -> Int
ceiling 7.8 = 8
fromIntegral :: Int -> Float (simplified)
fromIntegral 7 = floating point version of 7

Char Type: (functions in module Data.Char)

ord :: Char->Int
ord 'a' = 97
chr :: Int->Char
chr 97 = 'a'

toUpper :: Char -> Character
toUpper 'a' = 'A', toUpper 'B' = 'B'
also toLower...

isDigit :: Char->Bool
isDigit '3' = True, isDigit 'a' = False
also isAlpha, isUpper, isLower

Function Example With Guards & "where":

-- Ask if an integer equals 2 to a positive power
posPowOf2 :: Int -> Bool
posPowOf2 x
| x <= 1 = False
| x == 2 = True
| mod x 2 /= 0 = False -- odd number
| otherwise = posPowOf2 half
where
half = div x 2

Example of error message:

divide :: Float -> Float -> Float
divide a 0 = error "division by zero"
divide a b = a/b

List Functions & Operators:

(:) :: a->[a]->[a]
1:[2,3] = [1,2,3]

(++): [a]->[a]->[a]
"King"++"ston" = "Kingston"

concat :: [a]->[a]
concat [ [1,2],[3,4,5],[6] ] = [1,2,3,4,5,6]

length :: [a]->Int
length "abcd" = 4

head :: [a]->a
head [5,4,3] = 5
tail :: [a]->[a]
tail [5,4,3] = [4,3]

reverse :: [a]->[a]
reverse "computer" = "retupmoc"

take :: Int->[a] ->[a]
take 4 [5,2,7,3,9,8,1] = [5,2,7,3]

zip [a]->[b]->{(a,b)}
zip [1,2] [3,4,5] = [(1,3),(2,4)]

unzip [(a,b)]=>>([a],[b])
unzip [(2,5),(6,3)] = ([2,6],[5,3])

zipWith :: (a->b->c)->[a]->[b]->[c]
zipWith (*) [1,2,3] [5,6,7] = [5,12,21]

and :: [Bool]->Bool
and [True,False,True] = False
or :: [Bool]->Bool
or [True,False,True] = True

sum :: [Int]->Int or [Float]->Float
sum [3,1,5,7] = 16
product :: [Int] or [Float]->Float
product [2,4,3] = 24

Example using a tuple:

tupleSum :: (Int,Int) -> Int
tupleSum (a,b) = a+b

List Comprehension Example:

ums = [2,5,6,10,13,15]
lc = [z+1|z<-nums, mod z 5 == 0,
mod z 2 /= 0]
The value of lc is [6,16]
Higher-Order Functions:

map :: (a -> b) -> [a] -> [b]
map tail ["abc","de"] = ["bc","e"]

filter :: (a -> Bool) -> [a] -> [a]
filter (>0) [3,5,-2,6,-1] = [3,5,6]

foldr1 :: (a -> a -> a) -> [a] -> a
foldr1 (-) [3,5,4] = 2

foldl1 :: (a -> a -> a) -> [a] -> a
foldl1 (-) [3,5,4] = -6

foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f 4 [1,2,3] = f 1 (f 2 (f 3 4))

foldl :: (a -> a -> a) -> a -> [a] -> a
foldl f 4 [1,2,3] = f (f (f 4 1) 2) 3

flip :: (a -> b -> c) -> b -> a -> c
flip () [1,2,3] 7 = [7,1,2,3]

(.) :: (a -> b) -> (c -> a) -> c -> b
(f.g) x = f (g x)

Operator sections:
map (/10) [20.0,40.0] = [2.0,4.0]
map (10/) [2.0,5.0] = [5.0,2.0]

Partial function application:
map (\x\y -> x - 2 * y) [1,2,3] = [8,6,4]

Lambda notation:
map (\x\y = x + 2 * y) [1,2,3] = [3,5,7]
\x y -> x*y-1 2 3 = 5

curry :: ((a,b) -> c) -> a -> b -> c
(curry f) 1 2
  where f (x,y) = x+y
  = 3

uncurry :: (a -> b -> c) -> (a,b) -> c
(uncurry (+)) (4,3) = 7

Algebraic Types:

data Person = CreateProf String Int Bool
  | CreateStudent String Int [Float]
deriving (Show)

instance Eq Person where
  -- two people are equal if they have
  -- the same id number
  p1 == p2 = (getID p1) == (getID p2)

harry = CreateStudent "Sam" 1234 [82,75]
snape = CreateProf "Snape" 5678 False

ggetID :: Person -> Int
  getID (CreateProf _ _ _) = id
  getID (CreateStudent _ _ _) = id

isStudent :: Person -> Bool
  isStudent (CreateStudent _ _ _) = True
  isStudent (CreateProf _ _ _) = False

Type Classes: