module Lambda where

-- Character testing functions come from here:
import Data.Char

-- Returns the input, but with only the upper-case characters
uppersOnly :: String -> String
uppersOnly str = filter isUpper str

-- Same, but Î©-reduced
uppersOnly' :: String -> String
uppersOnly' = filter isUpper

-- Capitalize every letter in the string
toUpperStr :: String -> String
toUpperStr str = map toUpper str

-- increment every number by 1
incByOne :: [Integer] -> [Integer]
incByOne nums = map inc nums
  where
    inc x = x + 1

-- same, but using a Î›-expression
incByOne' :: [Integer] -> [Integer]
incByOne' nums = map (\x -> x + 1) nums

-- same, but using an operator section
incByOne'' :: [Integer] -> [Integer]
incByOne'' nums = map (1+) nums

-- same, but Î©-reduced. (This is how I would write it.)
incByOne''' :: [Integer] -> [Integer]
incByOne''' = map (1+)

-- Filter the input list so that only numbers divisible by 2 or 3
-- are in the list
filterNums :: [Integer] -> [Integer]
filterNums = filter (\x -> even x || (x 'mod' 3 == 0))

-- check whether a number is prime
isPrime :: Integer -> Bool
isPrime n = n > 1 && null (filter (\d -> n 'mod' d == 0) [2 .. n-1])

-- all primes up to a limit
allPrimesUpTo :: Integer -> [Integer]
allPrimesUpTo n = filter isPrime [2..n]

-- all primes.
-- Don't print this out. Instead, print out, e.g., 'take 20 allPrimes'.
allPrimes :: [Integer]
allPrimes = filter isPrime [2..]

-- split off any leading zeroes
leadingZeroes :: String -> (String, String)
leadingZeroes numStr = span (== '0') numStr

-- split off the first sentence, ended by a '.'
firstSentence :: String -> (String, String)
firstSentence str
  -- this next line is called a "pattern guard". The guard is accepted
  -- if and only if the pattern matches. Any variables brought into scope
  -- here remain in scope.
  | (sentence_no_dot, ',': rest) <- break (== '.') str
  | = (sentence_no_dot ++ ",", rest)
  | otherwise
firstSentenceWithCapital :: String -> (String, String)
firstSentenceWithCapital str
    | (sentence_no_dot@(first:_), '.' : rest) <- break (== '.') str
    , isUpper first   -- we can have multiple guards, separated by commas
    = (sentence_no_dot ++ ".", rest)
    | otherwise
    = (str, "")