第 06 章：List Comprehension

主要知识点：

- Generator / Guard / String Comprehension

✧ List Comprehension

In mathematics, the **set comprehension** notation can be used to *construct new sets from old sets*.

$\{ x^2 \mid x \in \{1,2,3,4,5\} \}$

In Haskell, a similar **list comprehension** notation can be used to *construct new lists from old lists*.

```
[ x^2 | x <- [1..5] ]   ===   [1, 4, 9, 16, 25]
```

✧ Generator

The expression `x <- [1..5]` is called a **generator**, as it states how to generate values for x.

Comprehensions can have *multiple* generators, separated by commas. For example:

```
        [ (x,y) | x <- [1, 2, 3], y <- [4, 5] ]
   ===  [ (1,4), (1,5), (2,4), (2,5), (3,4), (3,5) ]
```

Changing the order of the generators changes the order of the elements in the final list:

```
        [ (x,y) | y <- [4, 5], x <- [1, 2, 3] ]
   ===  [ (1,4), (2,4), (3,4), (1,5), (2,5), (3,5) ]
```

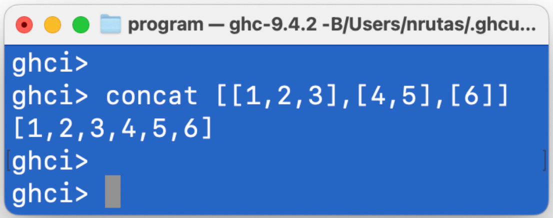
Dependant Generator: Later generators can depend on the variables that are introduced by earlier generators.

```
        [ (x,y) | x <- [1..3], y <- [x..3] ]
```

```
    ===  [ (1,1), (1,2), (1,3), (2,2), (2,3), (3,3) ]
```

Example: Using a dependant generator we can define the library function that *concatenates* a list of lists:

```
concat :: [[a]] -> [a]
concat xss = [ x | xs <- xss, x <- xs ]
```



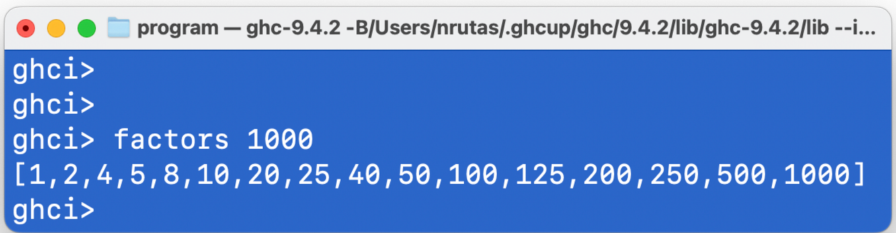
✧ Guards

List comprehensions can use **guards** to restrict the values produced by earlier generators.

```
          [ x | x <- [1..10], even x ]
    ===   [ 2, 4, 6, 8, 10 ]
```

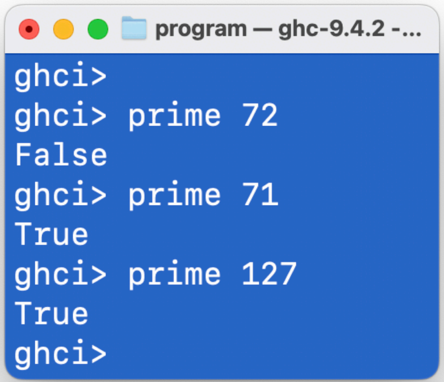
Example: Using a guard we can define a function that maps a positive integer to its list of factors:

```
factors :: Int -> [Int]
factors n  =  [ x | x <- [1..n], mod n x == 0 ]
```

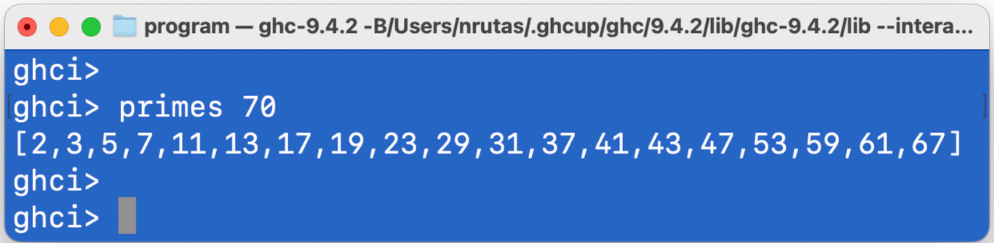


Example: A positive integer is prime if its only factors are 1 and itself. Hence, using factors we can define a function that decides if a number is prime:

```
prime :: Int -> Bool
prime n  =  factors n == [1,n]
```



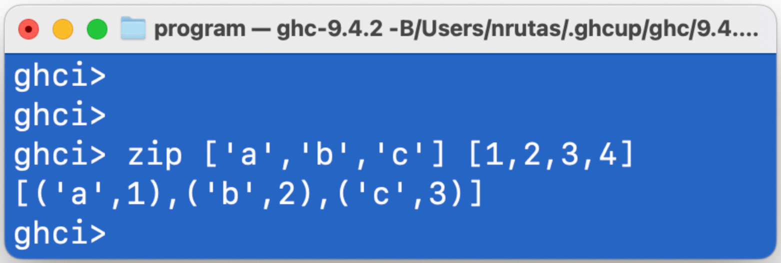

```
primes :: Int -> [Int]
primes n  =  [x | x <- [2..n], prime x]
```



✧ The **Zip** Function

A useful library function is zip, which maps two lists to a list of pairs of their corresponding elements.

```
zip :: [a] -> [b] -> [(a,b)]
zip [] _   =  []
zip _  []  =  []
zip (a:as) (b:bs)  =  (a,b) : zip as bs
```

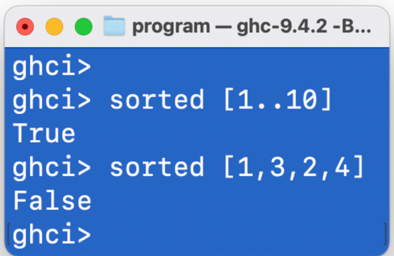


Example: Using zip, we can define a function that returns the list of all pairs of adjacent elements from a list:

```
pairs :: [a] -> [(a,a)]
pairs xs = zip xs (tail xs)
```

Example: Using zip, we can define a function that decides if the elements in a list are sorted:

```
sorted :: Ord a => [a] -> Bool
sorted xs  =  and [x <= y | (x,y) <- pairs xs]
```



Example: Using zip, we can define a function that returns the list of all positions of a value in a list:

```
positions :: Eq a => a -> [a] -> [Int]
positions x xs = [ i | (x',i) <- zip xs [0..], x == x' ]
```

```
program — ghc-9.4.2 -B/Users/nrutas/.ghcup/ghc/9....
ghci>
ghci>
ghci> positions 0 [1,0,0,1,0,1,1,0]
[1,2,4,7]
ghci>
```

✧ String Comprehension

A **string literal** is a sequence of characters enclosed in double quotes.

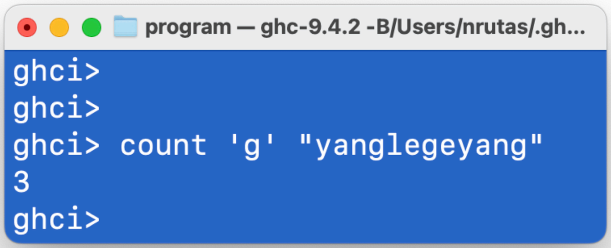
```
      "abcd" :: String
 === ['a', 'b', 'c', 'd'] :: [Char]
```

Because strings are just special kinds of lists, any polymorphic function that operates on lists can also be applied to strings.

```
program — ghc-9.4.2 -B/Users/nrutas/.ghcup/ghc/9.4....
ghci>
ghci> length "abcde"
5
ghci>
ghci> length "abcde"
5
ghci> take 3 "abcde"
"abc"
ghci> zip "abcd" [1,2,3,4]
[('a',1),('b',2),('c',3),('d',4)]
ghci>
```

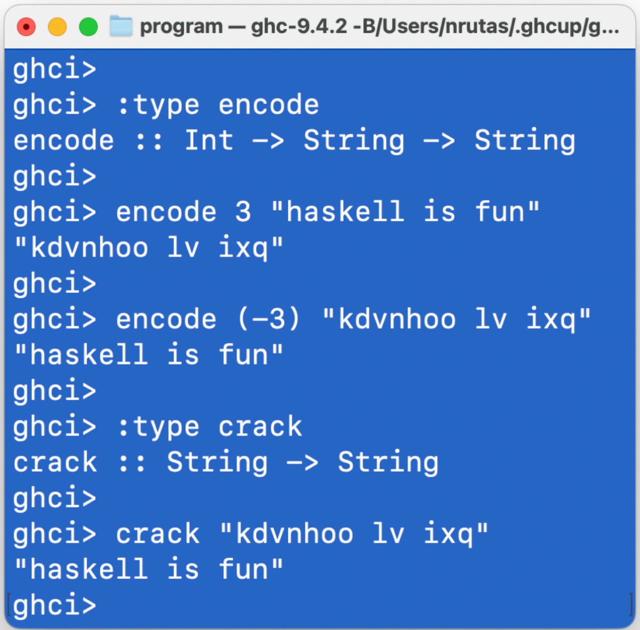
Similarly, list comprehensions can also be used to define functions on strings, such counting how many times a character occurs in a string.

```
count :: Char -> String -> Int
count x xs = length [ x' | x' <- xs, x == x' ]
```



✧ 凯撒加密问题

To encode a string, Caesar simply replaced each letter in the string by the letter three places further down in the alphabet, wrapping around at the end of the alphabet.



加密 / encode

```
import Data.Char(ord, chr, isLower)
-- ord :: Char -> Int          // 将字符转换为编码值
-- chr :: Int -> Char          // 将编码值转换为字符
-- isLower :: Char -> Bool   // 判断字符是否为小写字母

encode :: Int -> String -> String
encode n xs = [ shift n x | x <- xs ]
```

```
shift :: Int -> Char -> Char
shift n c | isLower c  =  int2let $ mod (let2int c + n) 26
          | otherwise  =  c

let2int :: Char -> Int
let2int c = ord c - ord 'a'

int2let :: Int -> Char
int2let n = chr $ ord 'a' + n
```

解密 / crack

The key to cracking the Caesar cipher is the observation that some letters are used more frequently than others in English text.

```
table :: [Float]
table = [ 8.1, 1.5, 2.8, 4.2, 12.7, 2.2, 2.0, 6.1, 7.0,
          0.2, 0.8, 4.0, 2.4,  6.7, 7.5, 1.9, 0.1, 6.0,
          6.3, 9.0, 2.8, 1.0,  2.4, 0.2, 2.0, 0.1 ]
-- table 中存放了 a,b,...,d 26 个英文字母在英文中出现的概率/频率
```

A standard method for comparing a list of observed frequencies ***os*** with a list of expected frequencies ***es*** is the *chi-square statistic*, defined by the following summation in which

- n denotes the length of the two lists, and
- xs_i denotes the ith element of a list xs counting from zero:

```
crack :: String -> String
crack xs  =  encode (-factor) xs
  where
    factor = position (minimum chitab) chitab
    -- minimum: a function exported by Prelude

    -- 计算每种加密偏移量下的chisqr
    chitab = [ chisqr (rotate n table') table | n <- [0..25] ]
```

```
    -- 计算密文中字母的出现频率
    table' = freqs xs

freqs :: String -> [Float]

chisqr :: [Float] -> [Float] -> Float
```

作业 01

请给出凯撒解密函数的完整定义:

```
crack :: String -> String
```

(仅考虑“明文中仅包含小写字母和空格”的情况)

作业 02

A triple (x,y,z) of positive integers is called pythagorean,

if $x^2 + y^2 = z^2$.

Using a list comprehension, define a function

```
pyths :: Int -> [(Int,Int,Int)]
```

that maps an integer n to all such triples with components in [1..n].

For example:

```
ghci> pyths 5
[ (3,4,5), (4,3,5) ]
```

作业 03

A positive integer is perfect if it equals the sum of all of its factors, excluding the number itself.

Using a list comprehension, define a function

```
perfects :: Int -> [Int]
```

that returns the list of all perfect numbers up to a given limit.

For example:

```
ghci> perfects 500
[ 6, 28, 496 ]
```

作业 04

The scalar product of two lists of integers xs and ys of length n is give by the sum of the products of the corresponding integers:

Using a list comprehension, define a function that returns the scalar product of two lists.