Lecture III : Lists and Comprehensions.

```
nats :: [Int]
nats = [1..]

[5..10] :: [Int]
```

in the Prelude.

```
take :: Int → [a] → [a]
take 0 xs     = []
take n []     = []
take n (x:xs) = x : take (n-1) xs
```

< take :: Int → [a] → [a].

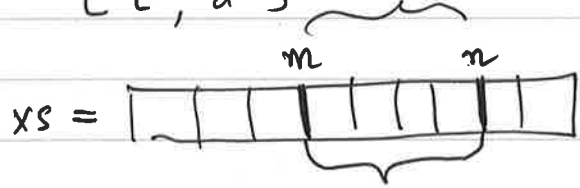
```
[]
x:xs
```

take n ~~x : xs~~ = wrong.

```
take n [x]
       [x, y]
       [(x:xs)]
~~(x0, x1, x2 ....~~
       x0 : x1 : x2 : xs
```

```
drop :: Int → [a] → [a]
drop 0 xs = xs
drop n [] = []
drop n (x:xs) = drop (n-1) xs
```

```
  drop 2 ['a', 'b', 'c', 'd']
=
  ['c', 'd']
```




```
  drop m (take n xs)
=
  take (n-m) (drop m xs)
```

```
alphabet :: Int → Char
alphabet 1 = 'a'
alphabet 2 = 'b'
    ⋮
```

map ~~24~~ ['a', 'b', 'c' ...]

map :: (a → b) → [a] → [b]

[1..26] = [1, 2, 3, 4 ··· 26] :: [Int]

['a', 'b', 'c', ··· 'z']

map alphabet [1..26]

between :: Int → Int → [Int]

~~between x y = x : between (x+1) y~~

~~between x (x+1) = []~~

between x y
  | x > y = []
  | otherwise = x : between (x+1) y

~~[x, x+1, .., y]~~

```
square :: Int → Int
square  x = x * x

map square nats
```

list comprehension style:

```
  [square n | n ← [1.. ]]
=
  map square nats

  even :: Int → Bool.

  map square (filter even nats)
=
  [ square n | n ← [1..], even n ]
```