```
 [x ** 2 | x <- [1..10]]
```

The pipe here designates the separation between the output function and the input.

Funcy Lists

```
 Prelude> [1..10]
[1,2,3,4,5,6,7,8,9,10]
Prelude> enumFromTo 1 10
[1,2,3,4,5,6,7,8,9,10]
Prelude> [2,4..10]
[2,4,6,8,10]
Prelude> enumFromThenTo 2 4 10
[2,4,6,8,10]
```

```
take :: Int -> [a] -> [a]
drop :: Int -> [a] -> [a]
splitAt :: Int -> [a] -> ([a], [a])
takeWhile :: (a -> Bool) -> [a] -> [a]
dropWhile :: (a -> Bool) -> [a] -> [a]
```

List comprehensions : Generating new list from a list, with predicates

```
 [x^2 | x <- [1..10], odd x]
 [x^y | x <- [1..5], y <- [2, 3], x^y < 200] -- [1,1,4,8,9,27,16,64,25,125]
 [(x, y) | x <- [1..5], y <- [6, 7]] -- [(1,6),(1,7),(2,6),(2,7),(3,6),(3,7)]
```

1 Spines and nonstrict evaluation

```
   :
  / \
 1   :          -- The list 1 : (2 : (3 : [])) can be visualized like this
    / \          --
   2   :          --
      / \
     3   []
```

Evaluation proceeds **up** the list. And spines are evaluated **independently** of values. Some `func` just evaluate the spine, not the value (like `length, ...`), but if the spine itself is **bottom**.

```
Prelude> let x = [1] ++ undefined ++ [3]
Prelude> x
[1*** Exception: Prelude.undefined
Prelude> length x
*** Exception: Prelude.undefined
-- map is nonstrict
Prelude> take 2 $ map (+1) [1, 2, undefined]
[2,3]
In the final example, the undefined value wa
```

WHNF: Weak Head Normal Form

”Normal form”: the expression is fully evaluated. ”Weak head normal form”: the expression is only evaluated as far as is necessary to reach a **data constructor**.

An expression cannot be in normal form or weak head normal form if the outermost part of the expression isn't a *data constructor*.

```
 Prelude> zipWith (+) [1, 2, 3] [10, 11, 12]
[11,13,15]
```

Cons cell is a data constructor and a product of the types a and $[a]$.