CS 558 — 27 August 2024

Note Title 2024-08-27

```
-- n! = 1·2·3·...·n          1! = 1     0! = 1
fact :: Integer -> Integer
fact n = if n == 0 then 1 else n * fact (n-1)
```

↑ a single defining equation

```
fact' :: Integer -> Integer
fact' 0 = 1
fact' n = n * fact (n-1)
```

} 2 defining equations

```
fact'' :: Integer -> Integer
fact'' n
  | n == 0    = 1
  | otherwise = n * fact'' (n-1)
```

single defining equation (with guards)

```
fact''' :: Integer -> Integer
fact''' n = product [1..n]
```

Lists

```
nums  :: [Int]
nums  = [1, 3, 7, 9]
range = [1..10]
range = [100, 98 .. 0]
nats  = [0..]
names :: [String]
names = ["Alice", "Bob"]
```

`[]` empty list

```
list1 = 1 : []          this is the same as [1]
list2 = 2 : (3 : (4 : []))
list2' = 2 : 3 : 4 : []
list2'' = [2,3,4]
```

length [2,3,4] → 3

```
length :: [a] -> Int        a: type variable
length [] = 0
length (x:xs) = 1 + length xs
```

length [100,98..0] → 51

length [square, cube, fourthpower] → 3

length names → 2

length "Alice" → 5

```
head :: [a] -> a
head [] = error "head: empty list"
head (x:xs) = x
```

use with caution

```
sum :: (Num a) => [a] -> a
sum [] = 0
sum (n:ns) = n + sum ns
```

(Num a) => is a type constraint

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

take 2 [1,2,3,4] → [1,2]

take 5 [1,2,3,4] → [1,2,3,4]

take 5 nats → [0,1,2,3,4]

nats = [0..]

```
ones :: [Int]
ones = 1 : ones
```

take 3 ones → [1,1,1]

head ones → 1

sum ones — diverges

List comprehensions

```
[square x | x <- [1..10]]
```

→ [1,4,9,16,25,36,49,64,81,100]

```
[x * y | x <- [1..10], y <- [1..10]]
```

```
[square x | x <- [1..100], x `mod` 3 == 1]
```

```
[square x | x <- nats]
```

→ [0,1,4,9,16,...]