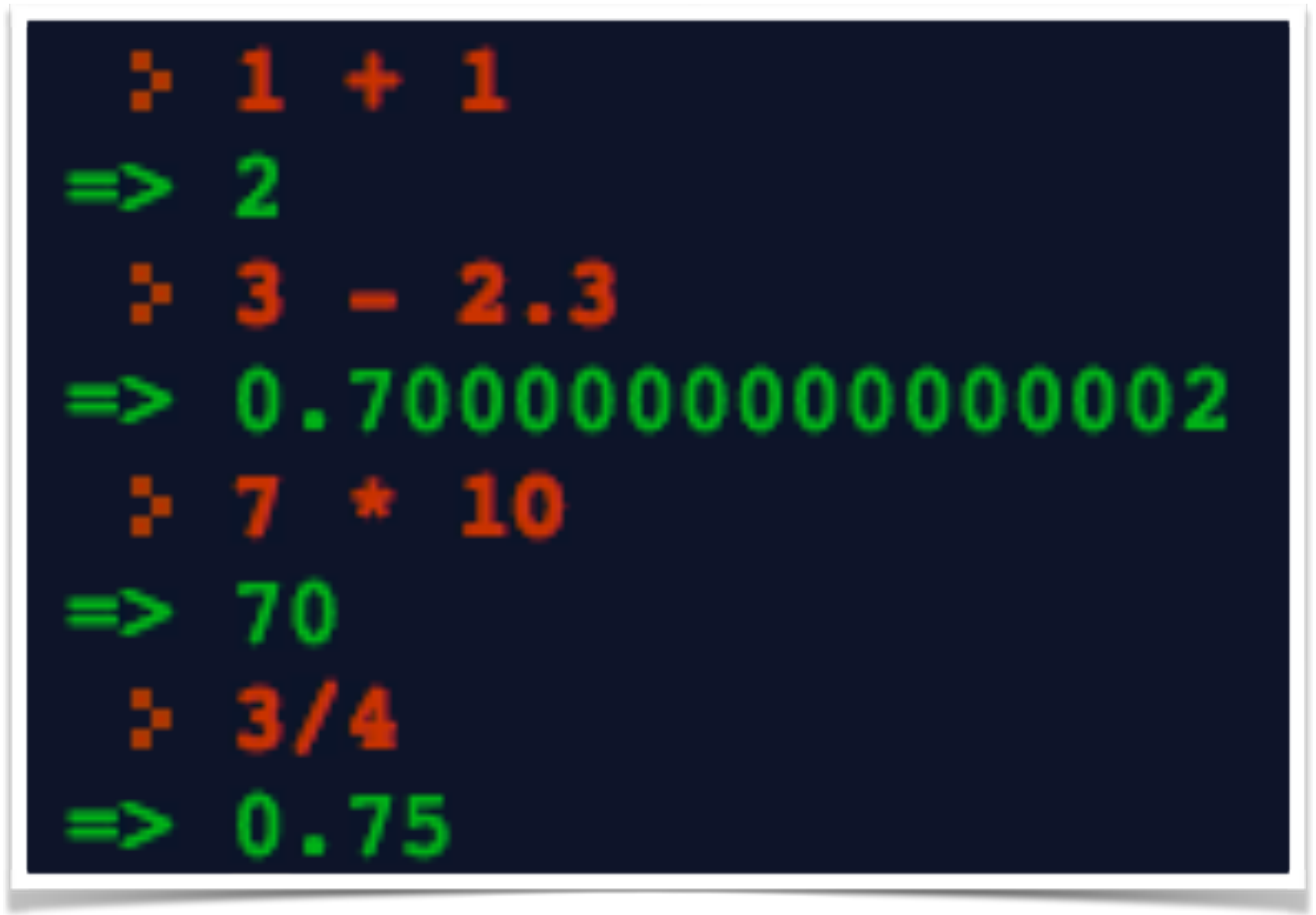
Functional programming

Basic Haskell syntax and function calls

Arithmetic operators

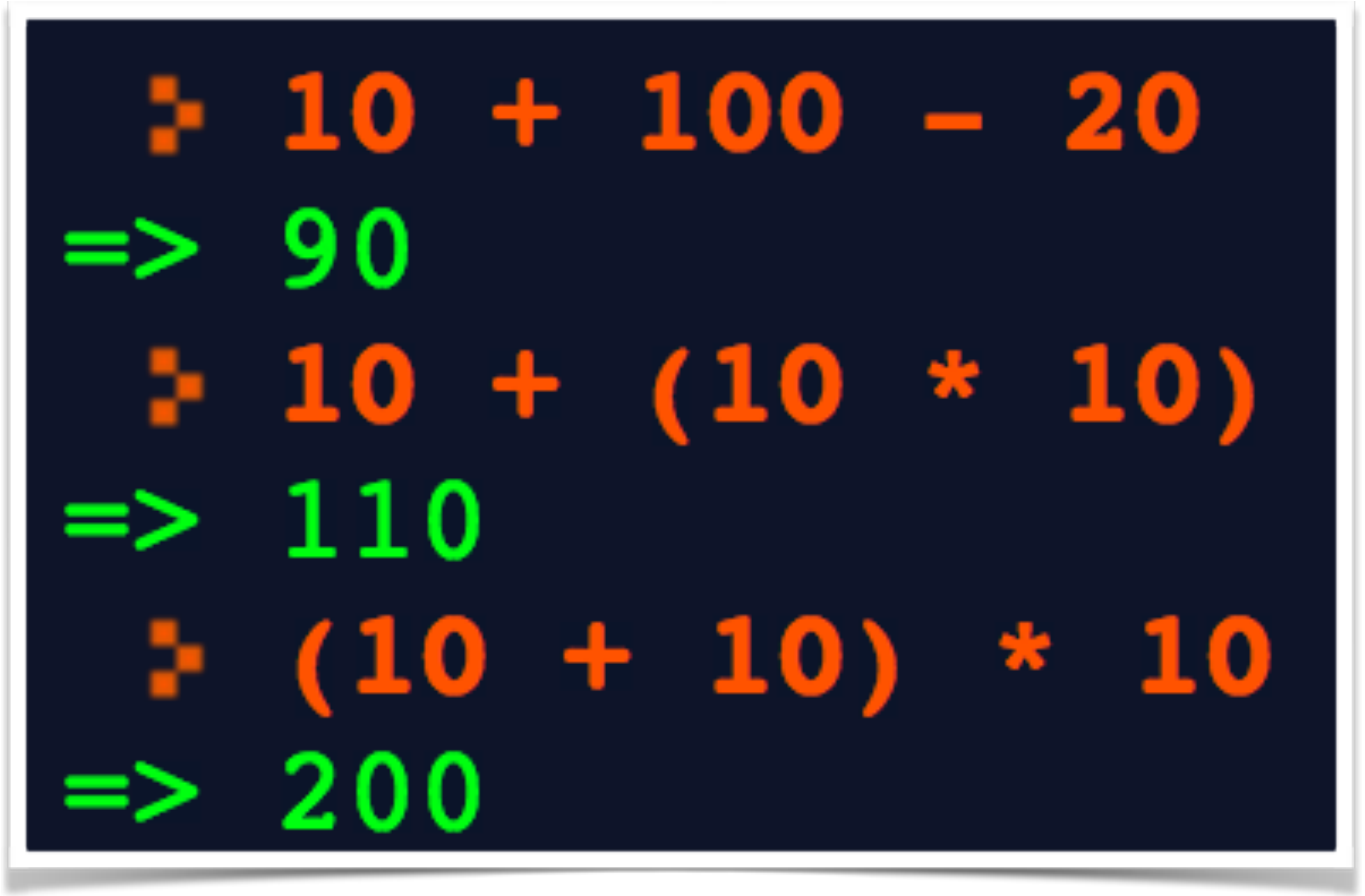
- Simple arithmetics
  - + - * /
- Why the second answer looks like this?
- What result will be give (3 / 4) in C/C++?



```
 > 1 + 1
=> 2
 > 3 - 2.3
=> 0.7000000000000002
 > 7 * 10
=> 70
 > 3/4
=> 0.75
```

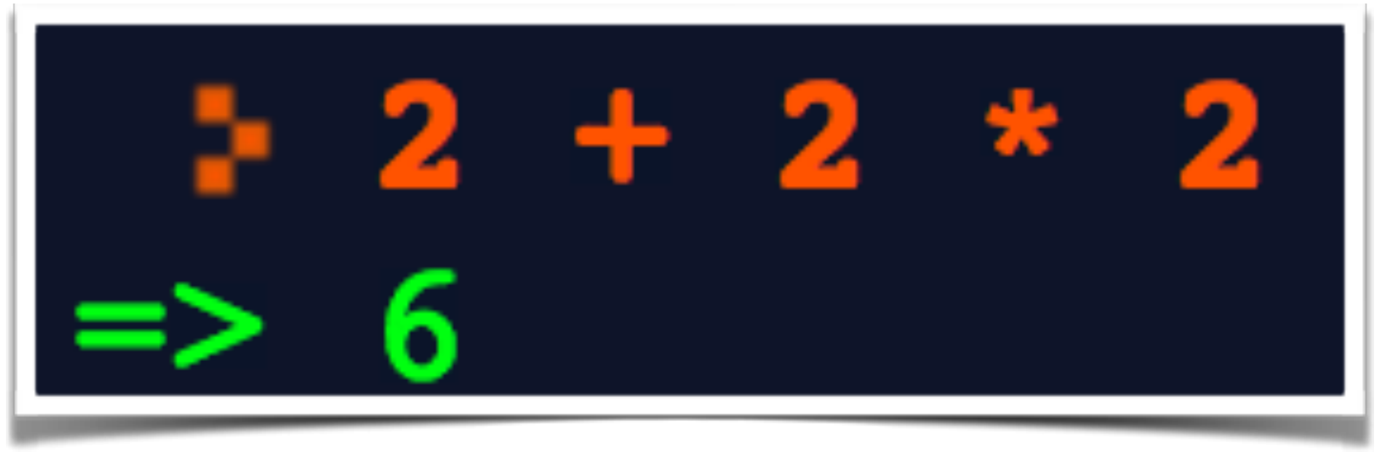
Arithmetic operators

- Several operators and precedence rules
  - + - and * / - which will be calculated first?
  - Use brackets to change calculation order



```
 ⋗ 10 + 100 - 20
=> 90
 ⋗ 10 + (10 * 10)
=> 110
 ⋗ (10 + 10) * 10
=> 200
```

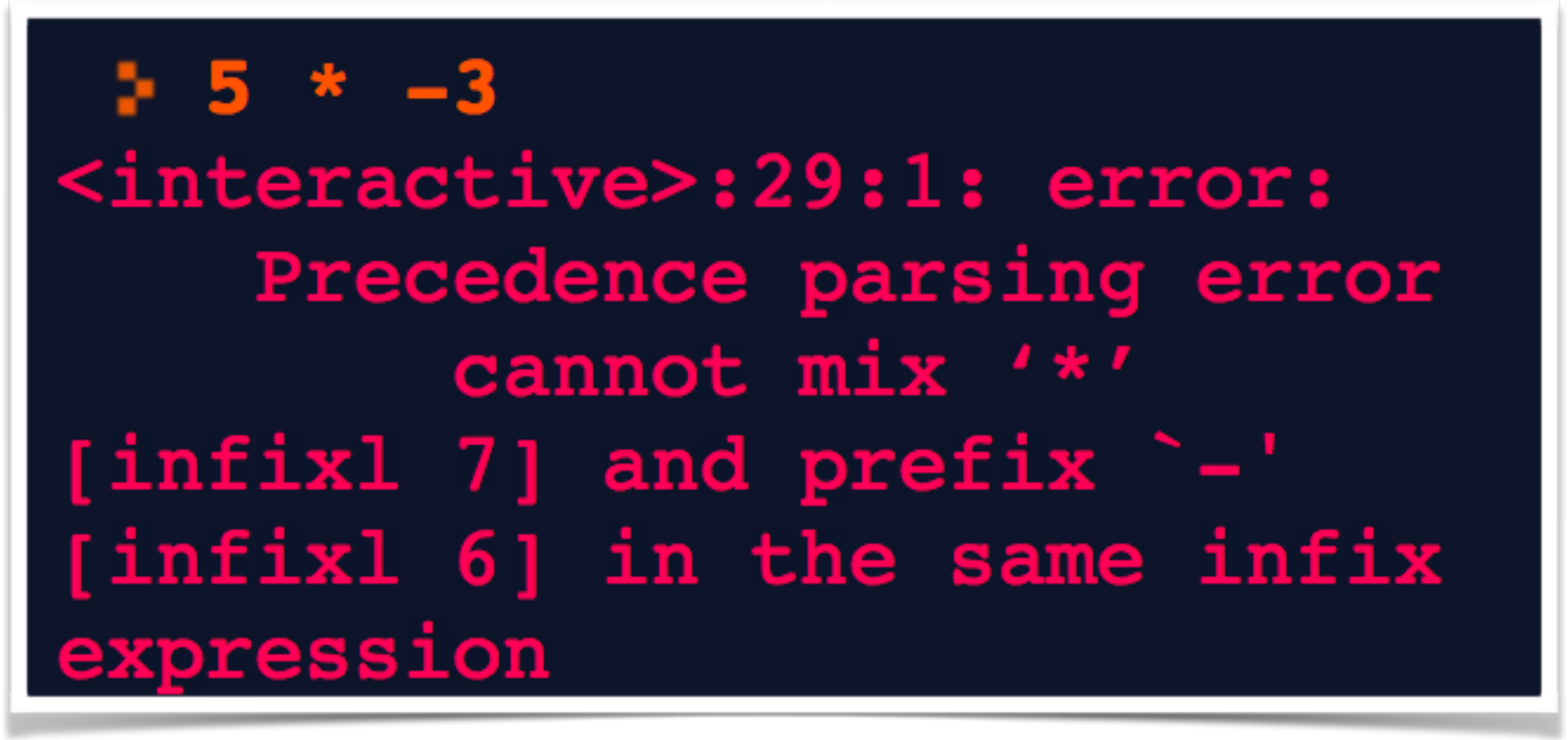
What the results will be?



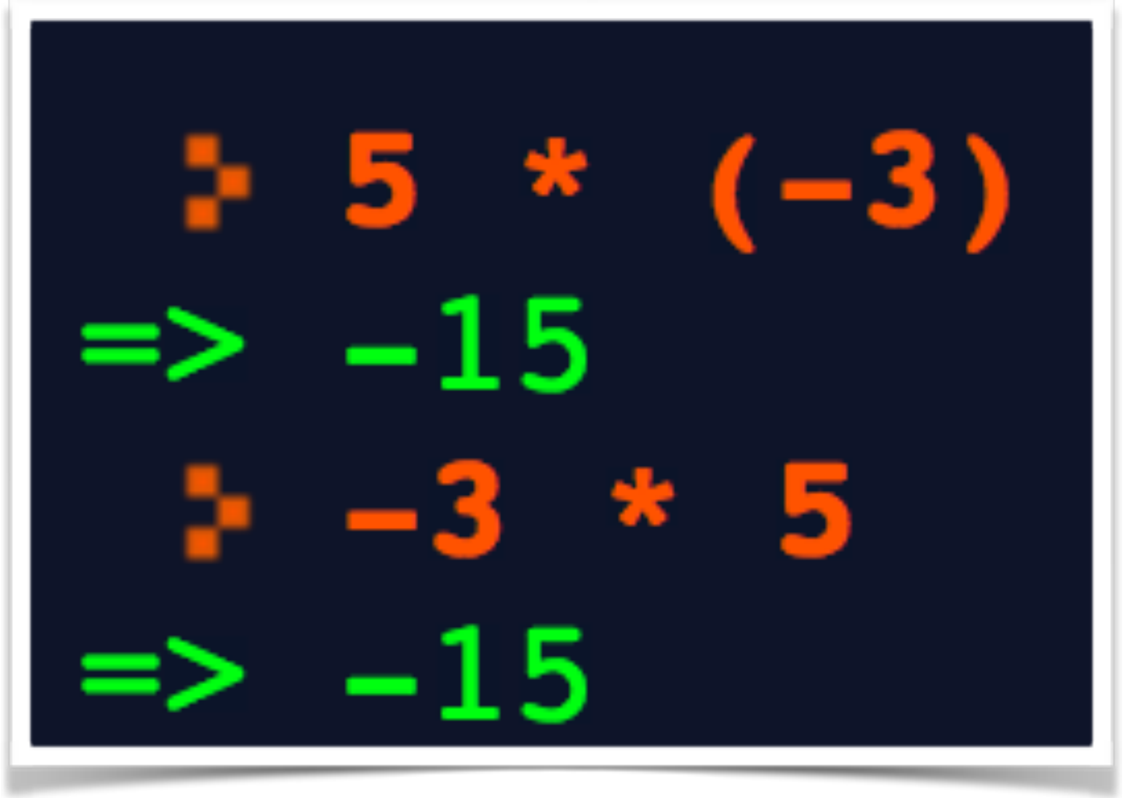
```
 ⋗ 2 + 2 * 2
=> 6
```

Arithmetic operators

- Minus sign can be treated as:
  - infix function with two arguments
  - prefix function with one argument



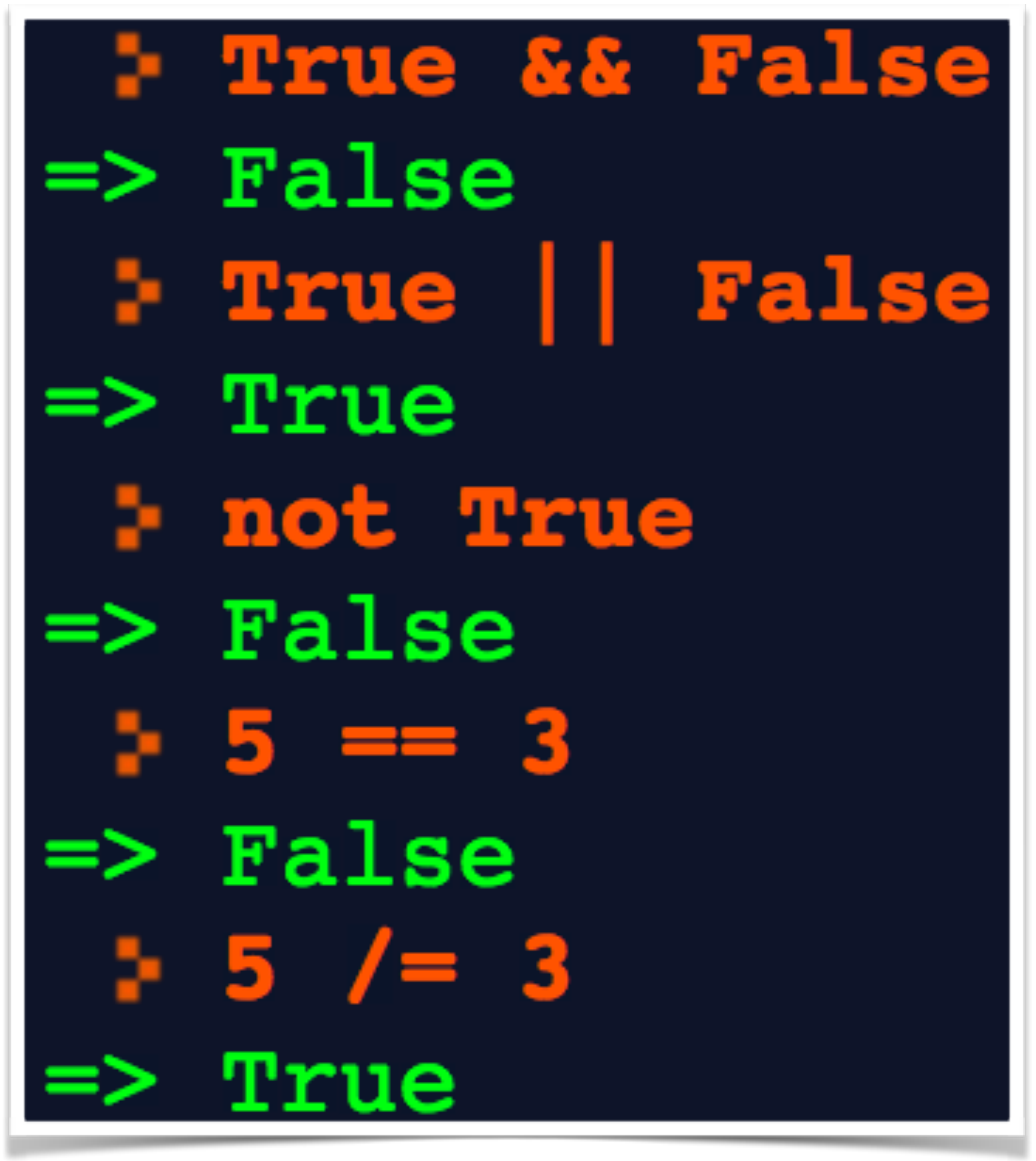
```
 λ 5 * -3
<interactive>:29:1: error:
    Precedence parsing error
        cannot mix '*'
[infixl 7] and prefix `-'
[infixl 6] in the same infix
expression
```



```
  λ 5 * (-3)
=> -15
  λ -3 * 5
=> -15
```

Boolean algebra

- Constants
  - True, False
- Operators
  - &&, ||, not
- Checking equality
  - ==, /=

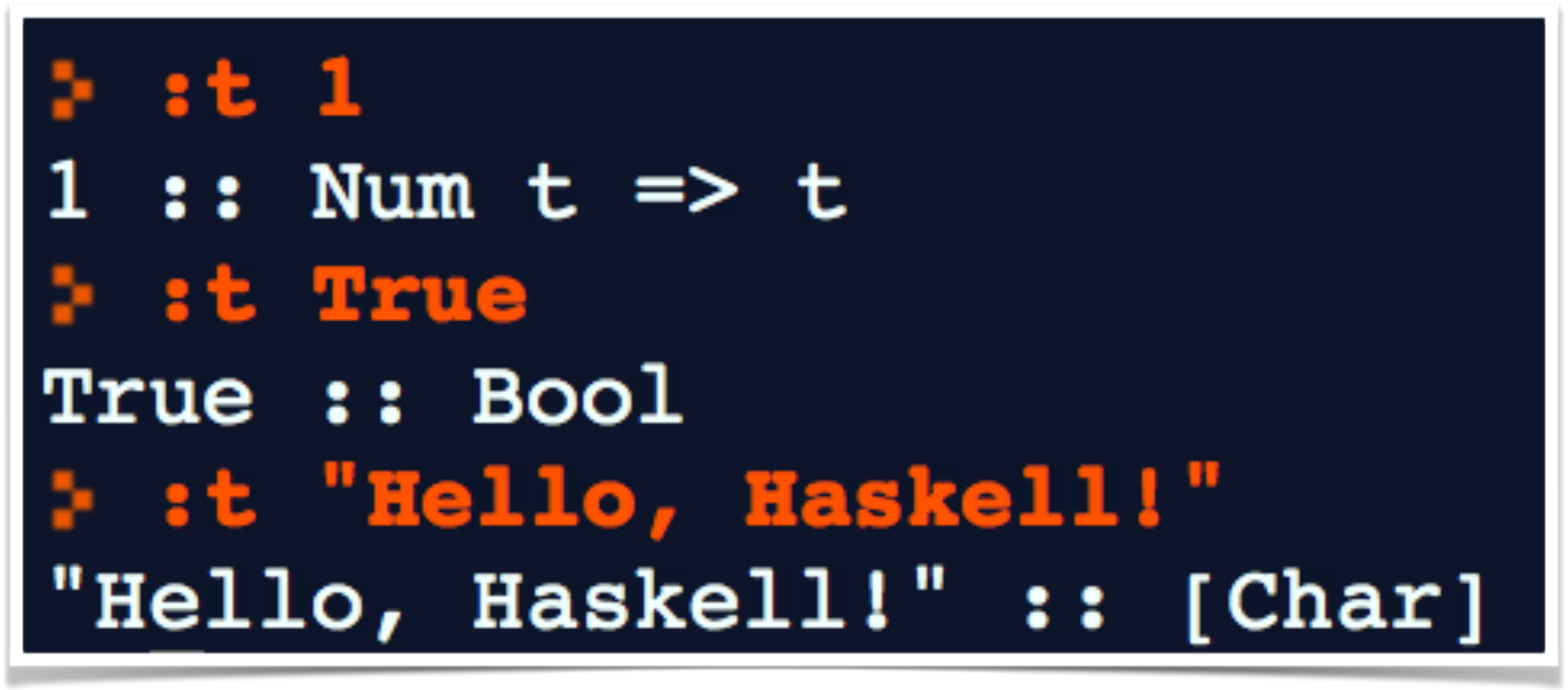


```
 > True && False
=> False
 > True || False
=> True
 > not True
=> False
 > 5 == 3
=> False
 > 5 /= 3
=> True
```

Type system

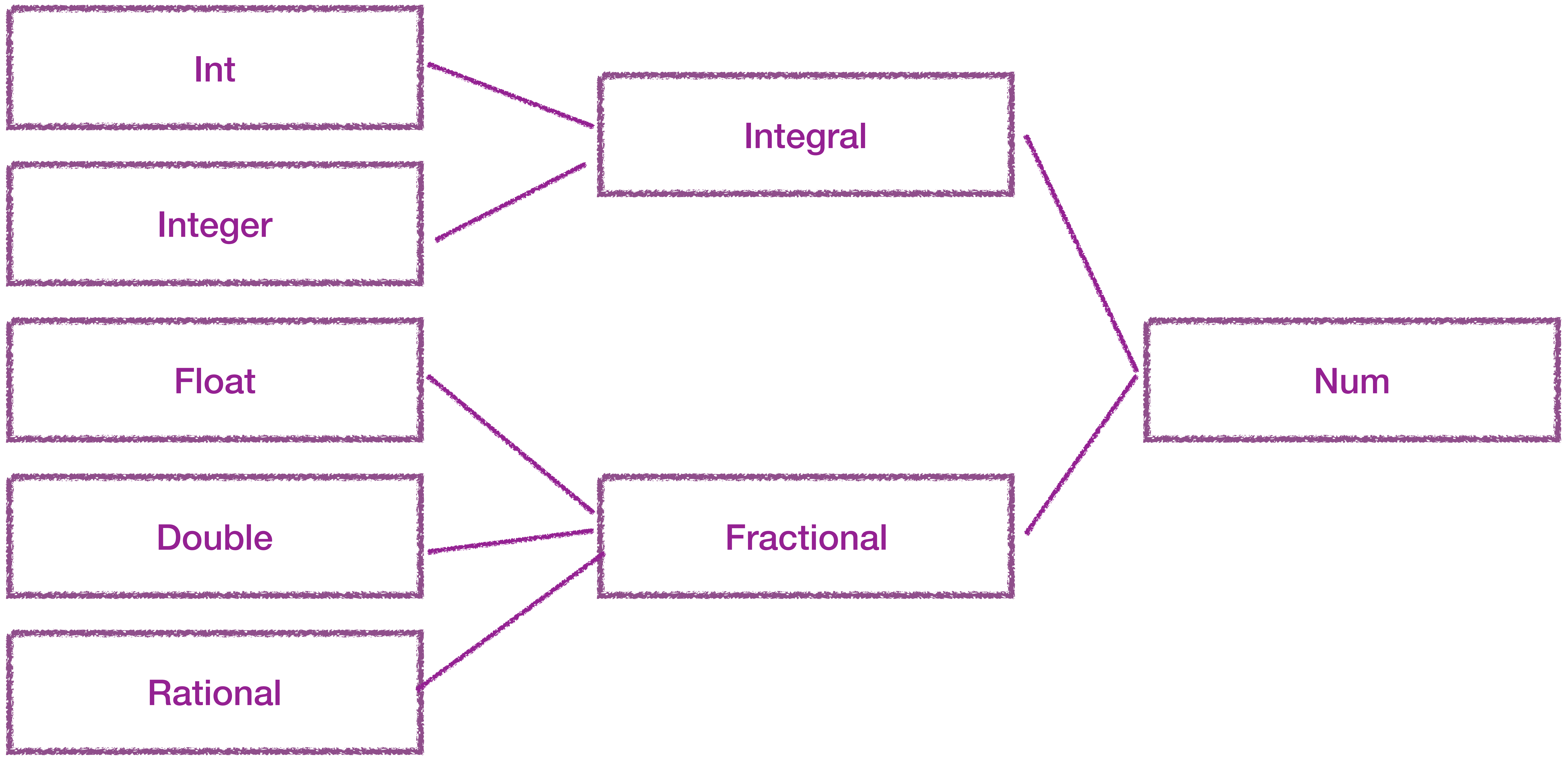
Checking types

- To check type of something just type **:t <something>**



```
> :t 1
1 :: Num t => t
> :t True
True :: Bool
> :t "Hello, Haskell!"
"Hello, Haskell!" :: [Char]
```

Numbers in Haskell

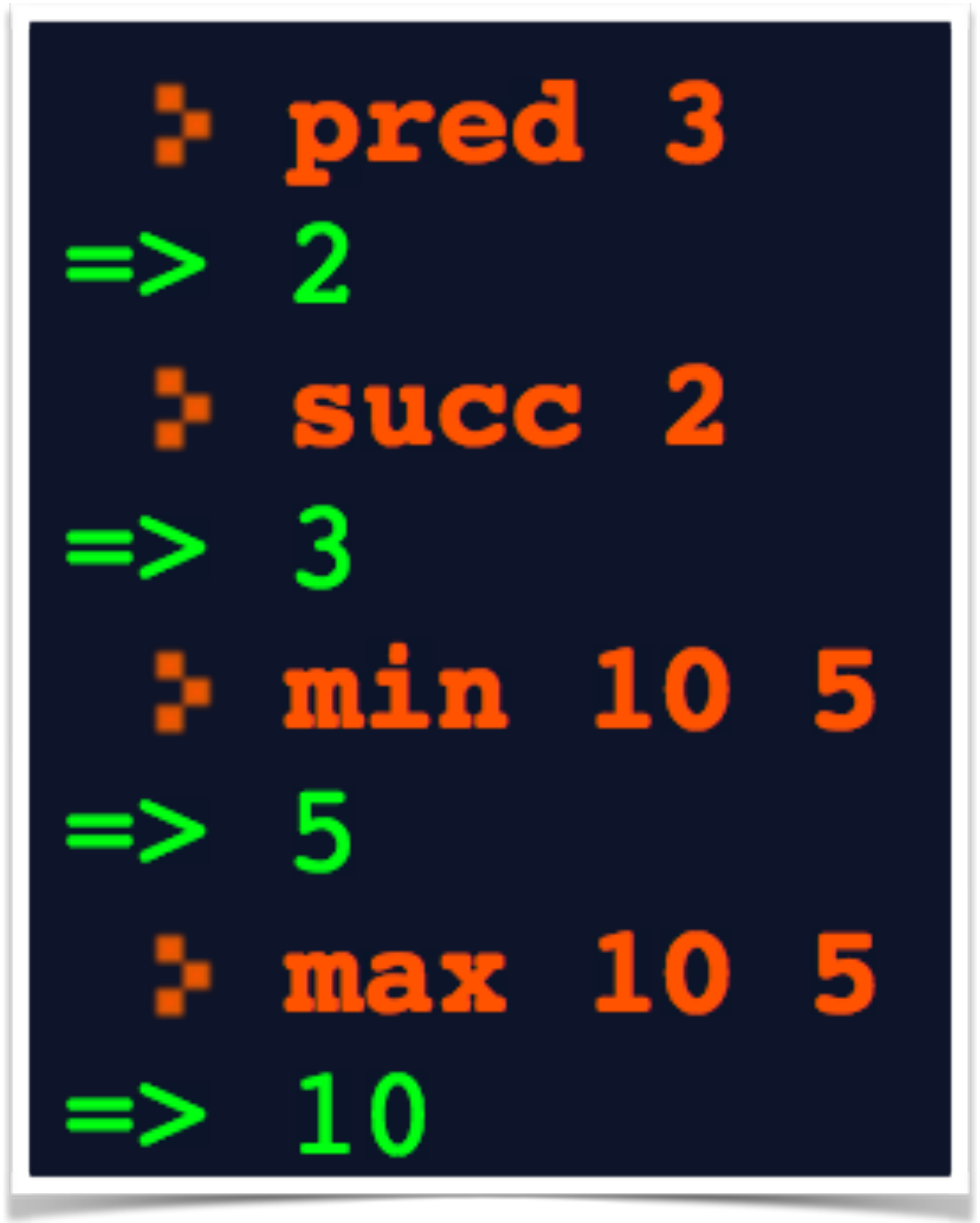


Numbers in Haskell

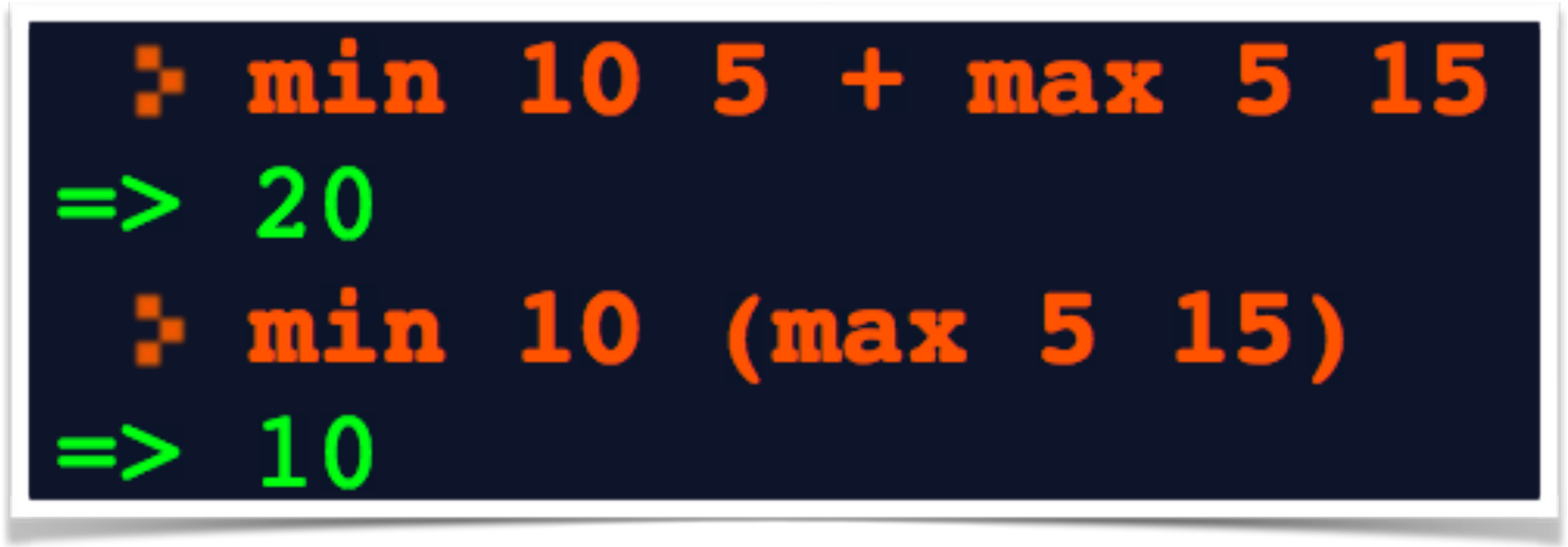
- Numeric types
    - **Int** is an integer with at least 30 bits of precision
    - **Integer** is an integer with unlimited precision
    - **Float** is a single precision floating point number
    - **Double** is a double precision floating point number
    - **Rational** is a fraction type, with no rounding error

Calling functions

- Function name followed by arguments
- How many arguments does a function take?:
  - pred, succ (predecessor, successor)
  - min, max



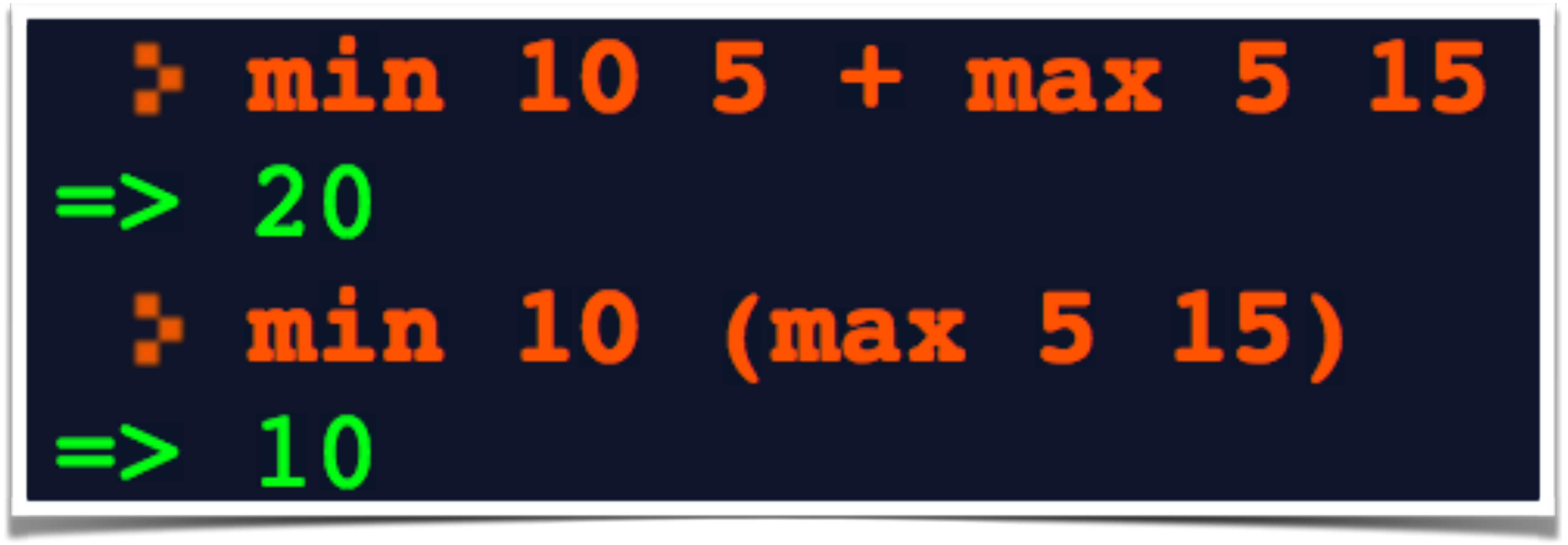
```
 > pred 3
=> 2
 > succ 2
=> 3
 > min 10 5
=> 5
 > max 10 5
=> 10
```



```
 > min 10 5 + max 5 15
=> 20
 > min 10 (max 5 15)
=> 10
```

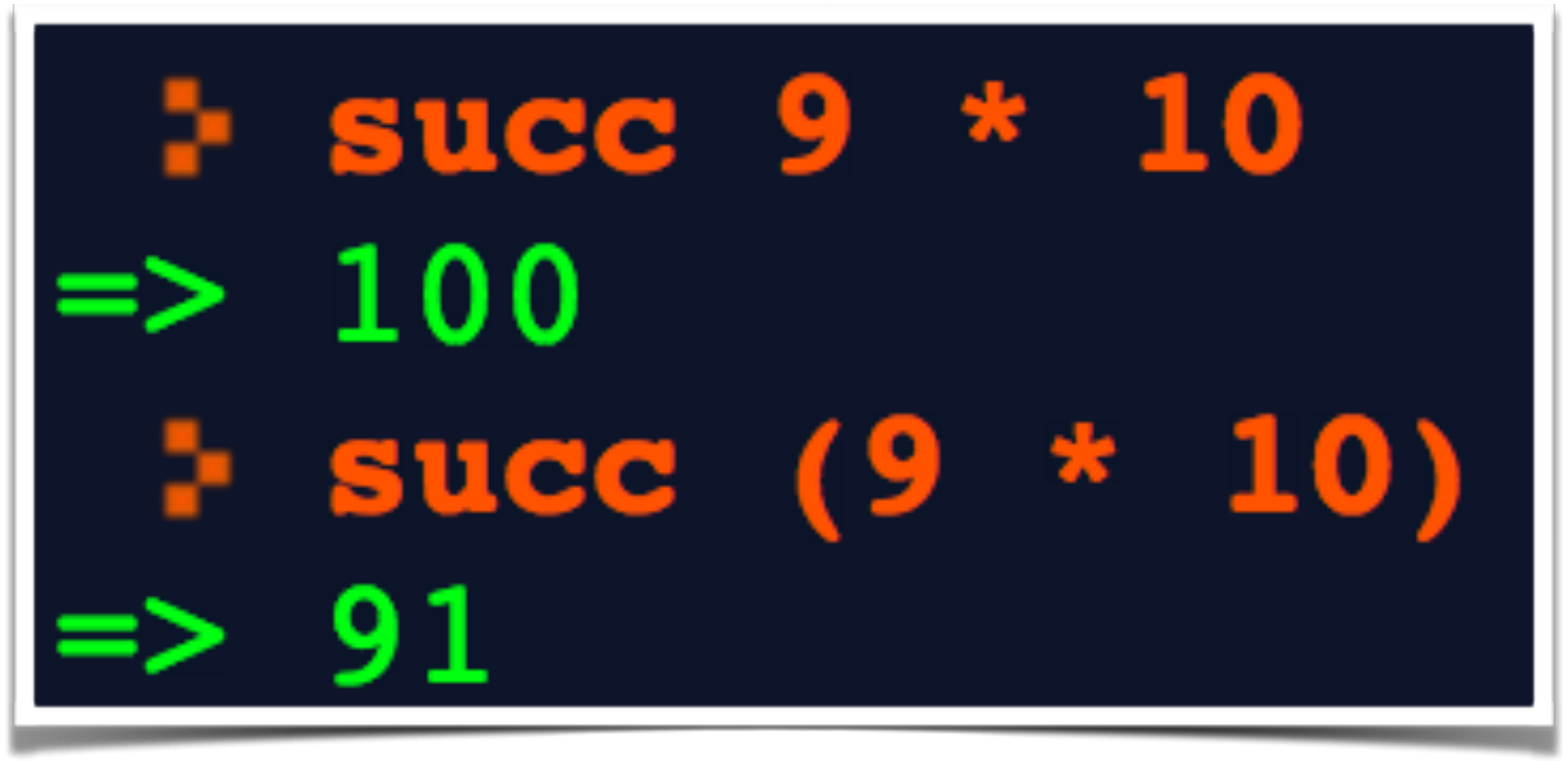
Calling functions

- A function can be called as an argument of another function
- Function calls have higher precedence than arithmetic or boolean operators!



```
 > min 10 5 + max 5 15
=> 20
 > min 10 (max 5 15)
=> 10
```

What the results will be?

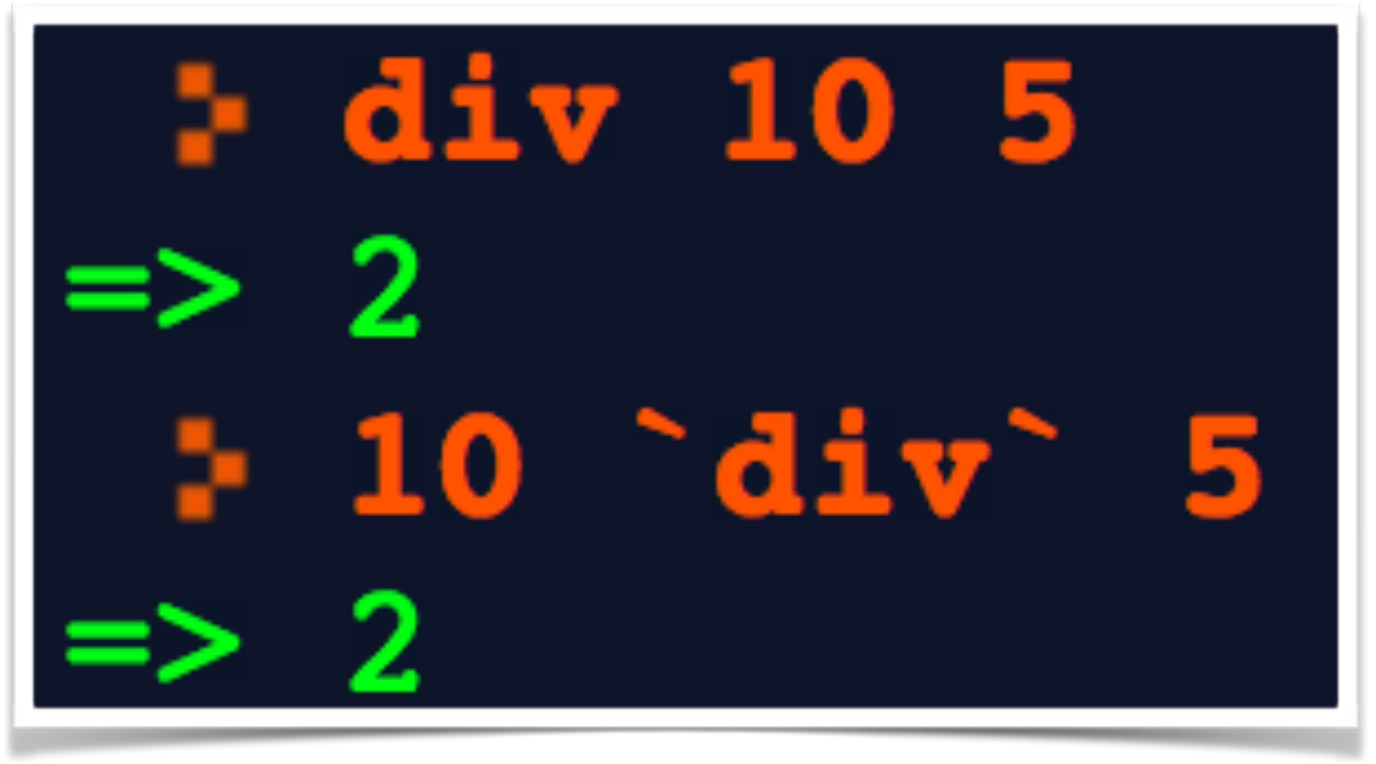


```
 > succ 9 * 10
=> 100
 > succ (9 * 10)
=> 91
```

Infix functions

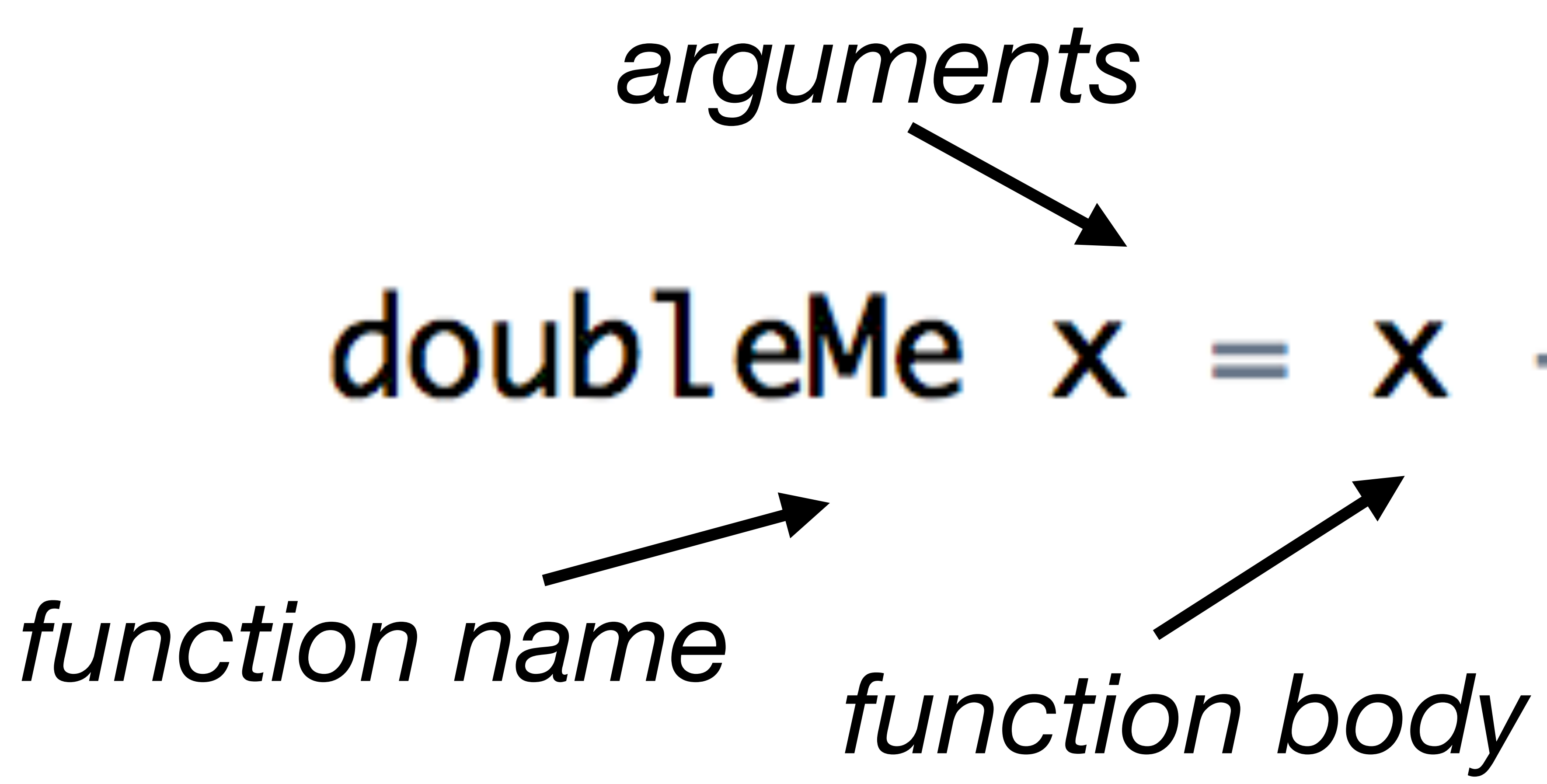
- A function that takes two arguments can be called as an infix function

- An infix function is surrounded by backticks `

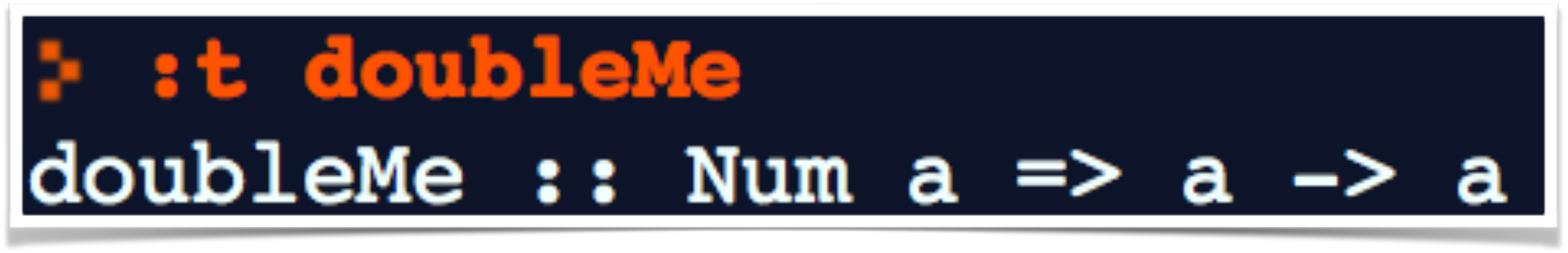


```
 > div 10 5
=> 2
 > 10 `div` 5
=> 2
```

Let’s write a function



```
doubleMe x = x + x
```

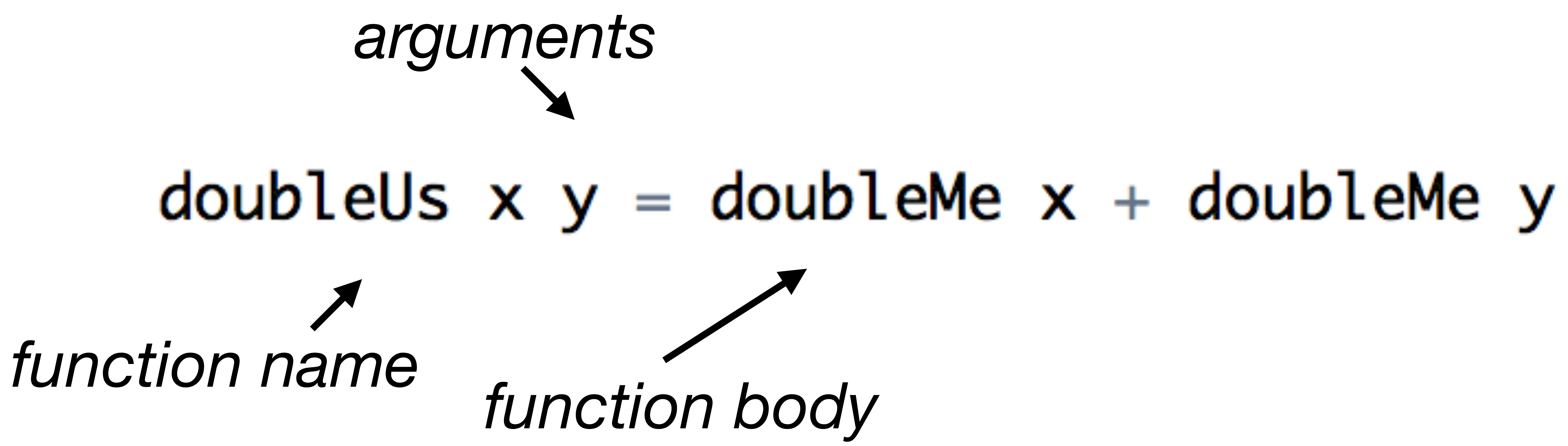


```
> :t doubleMe
doubleMe :: Num a => a -> a
```

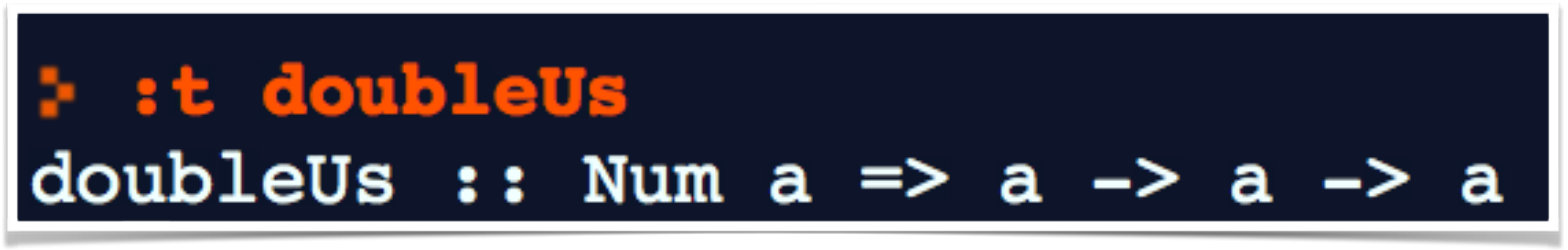
:t displays the type of the function chosen by Haskell

doubleMe has one argument of type Num and returns the result of the same type

Another function

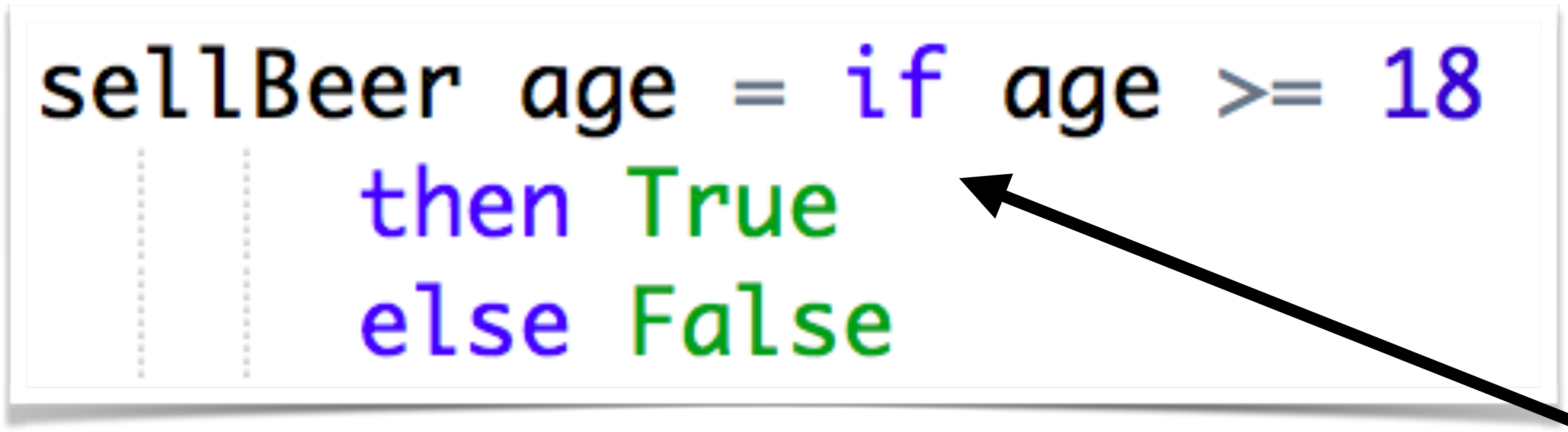


```haskell
doubleUs x y = doubleMe x + doubleMe y
```



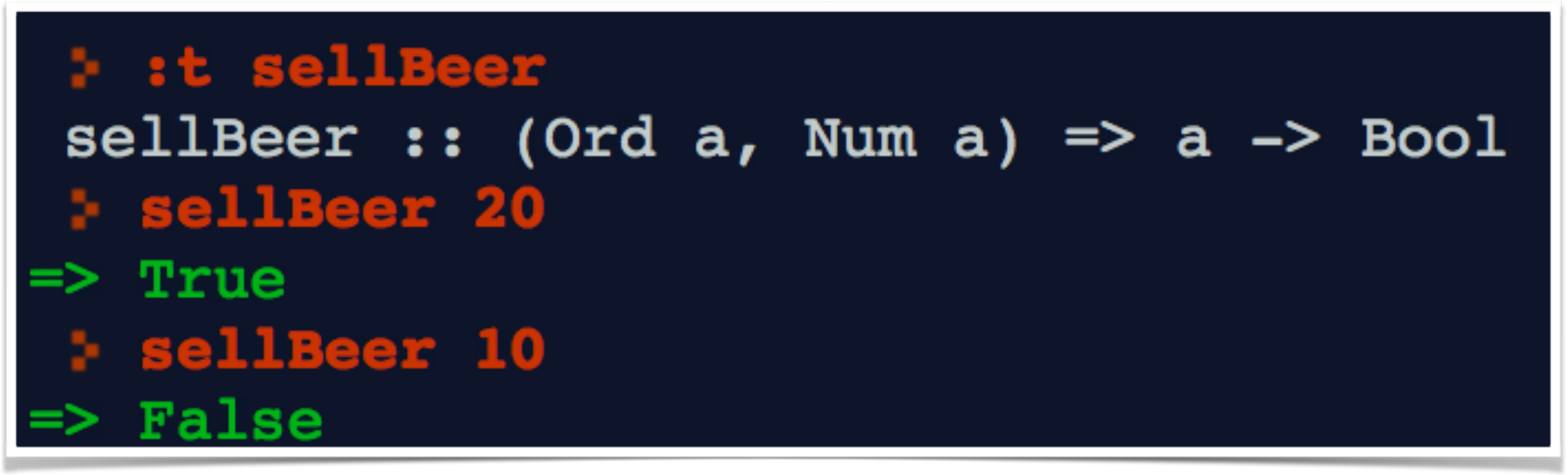
```
> :t doubleUs
doubleUs :: Num a => a -> a -> a
```

More useful function



```haskell
sellBeer age = if age >= 18
                 then True
                 else False
```

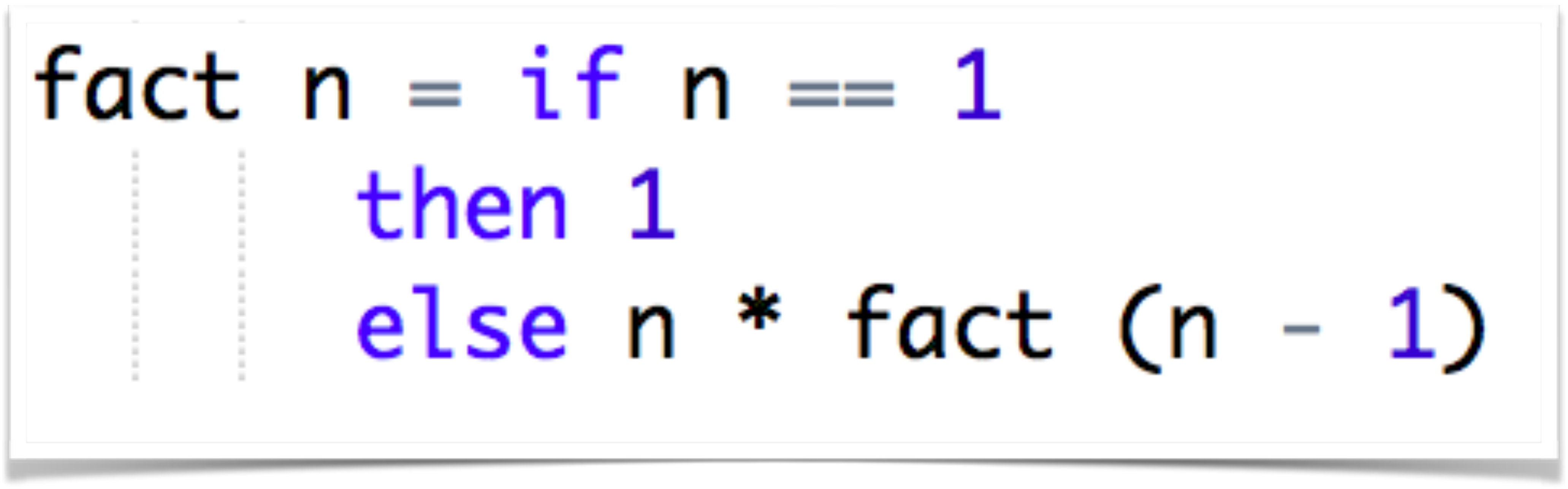
if *is a function that returns a value*



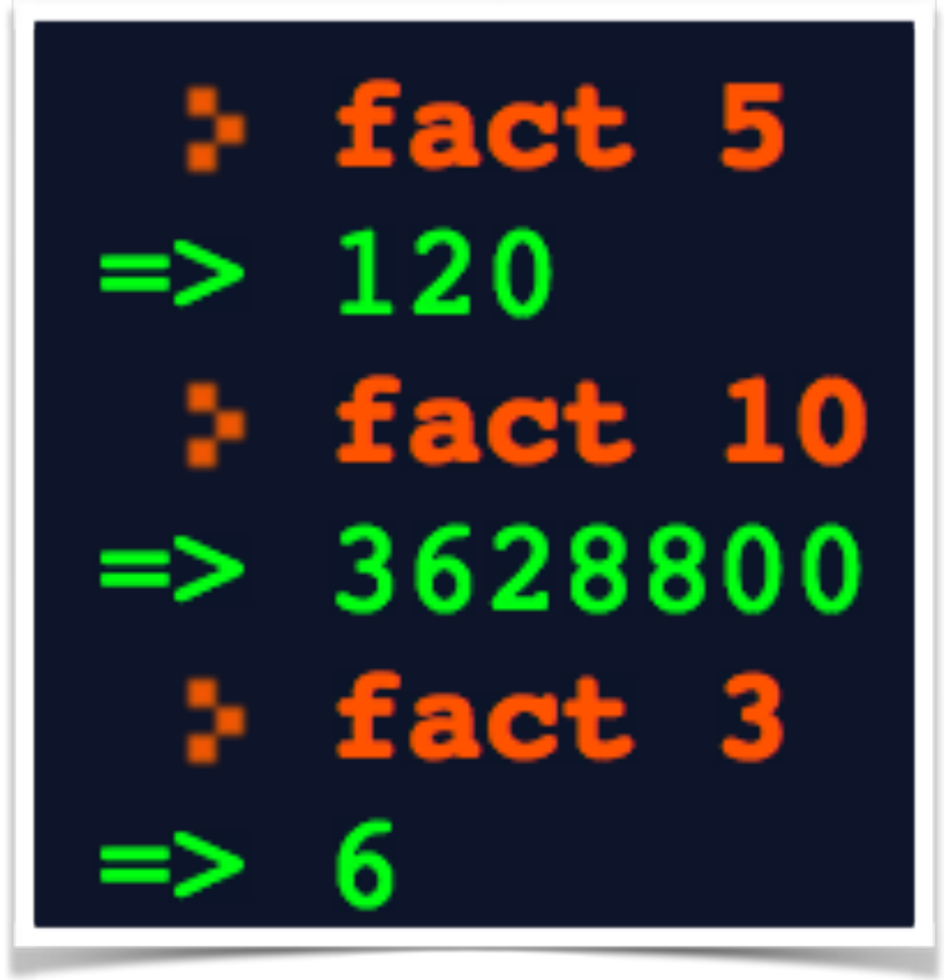
```
 > :t sellBeer
 sellBeer :: (Ord a, Num a) => a -> Bool
 > sellBeer 20
=> True
 > sellBeer 10
=> False
```

Really useful function

- Challenge:
  - write a function to calculate factorial of a number



```haskell
fact n = if n == 1
           then 1
           else n * fact (n - 1)
```



```
 > fact 5
=> 120
 > fact 10
=> 3628800
 > fact 3
=> 6
```

Summary

- Arithmetic operations are similar to ones used in other languages
- Use **not** for negation and **/=** for not equals check
- **Num**s are divided into **Integral** and **Fractional** numbers
- Even **if** is a function in Haskell