

Functional programming

Basic Haskell syntax and function calls

Installing Haskell



Windows



Linux



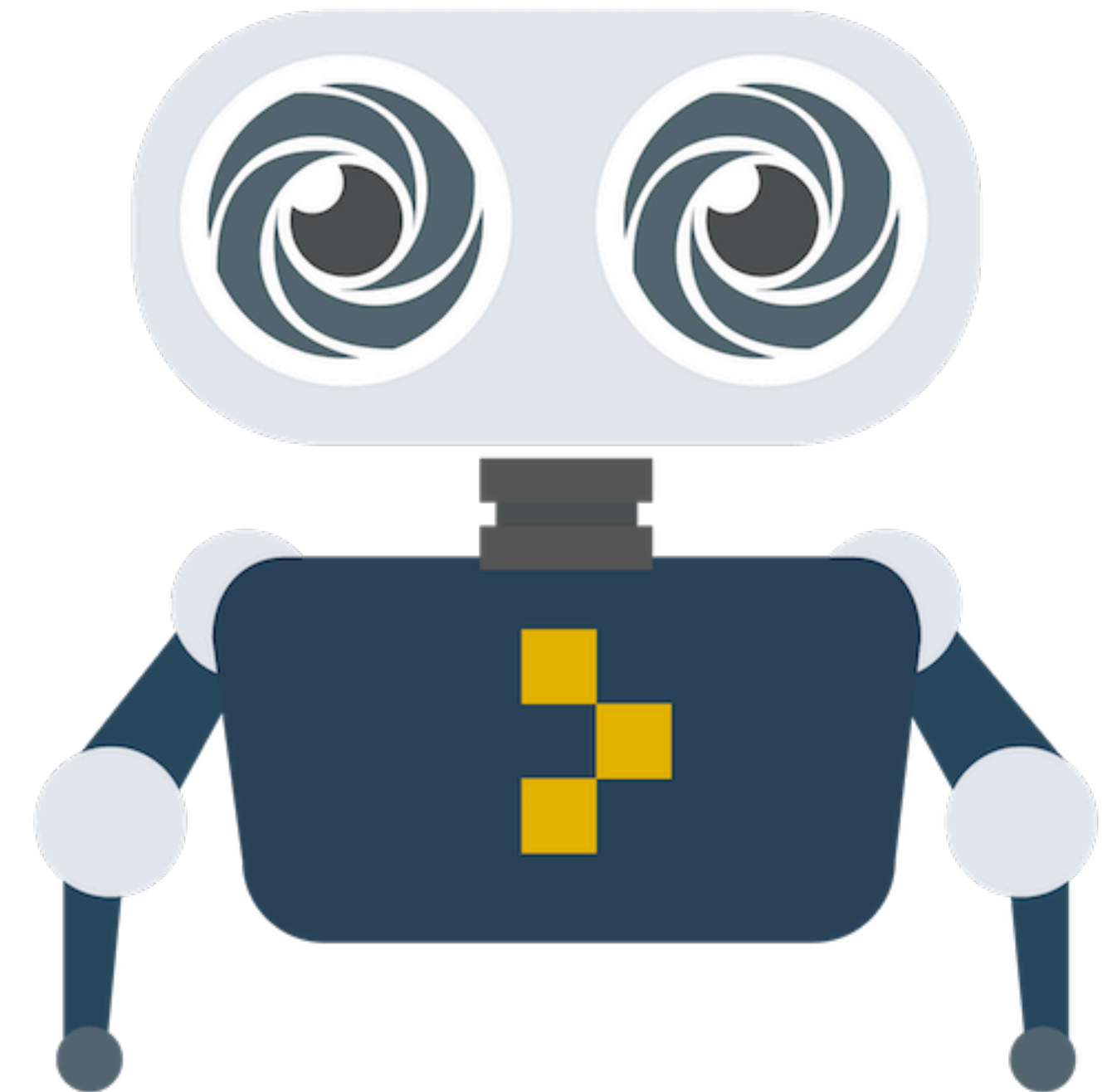
Mac OS X

<https://www.haskell.org/platform/>

REPL



- **Read-eval-print-loop**
- <https://repl.it/languages/Haskell>



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 '*'
[infix1 7] and prefix '-'
[infix1 6] in the same infix
expression
```

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

Boolean algebra

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

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

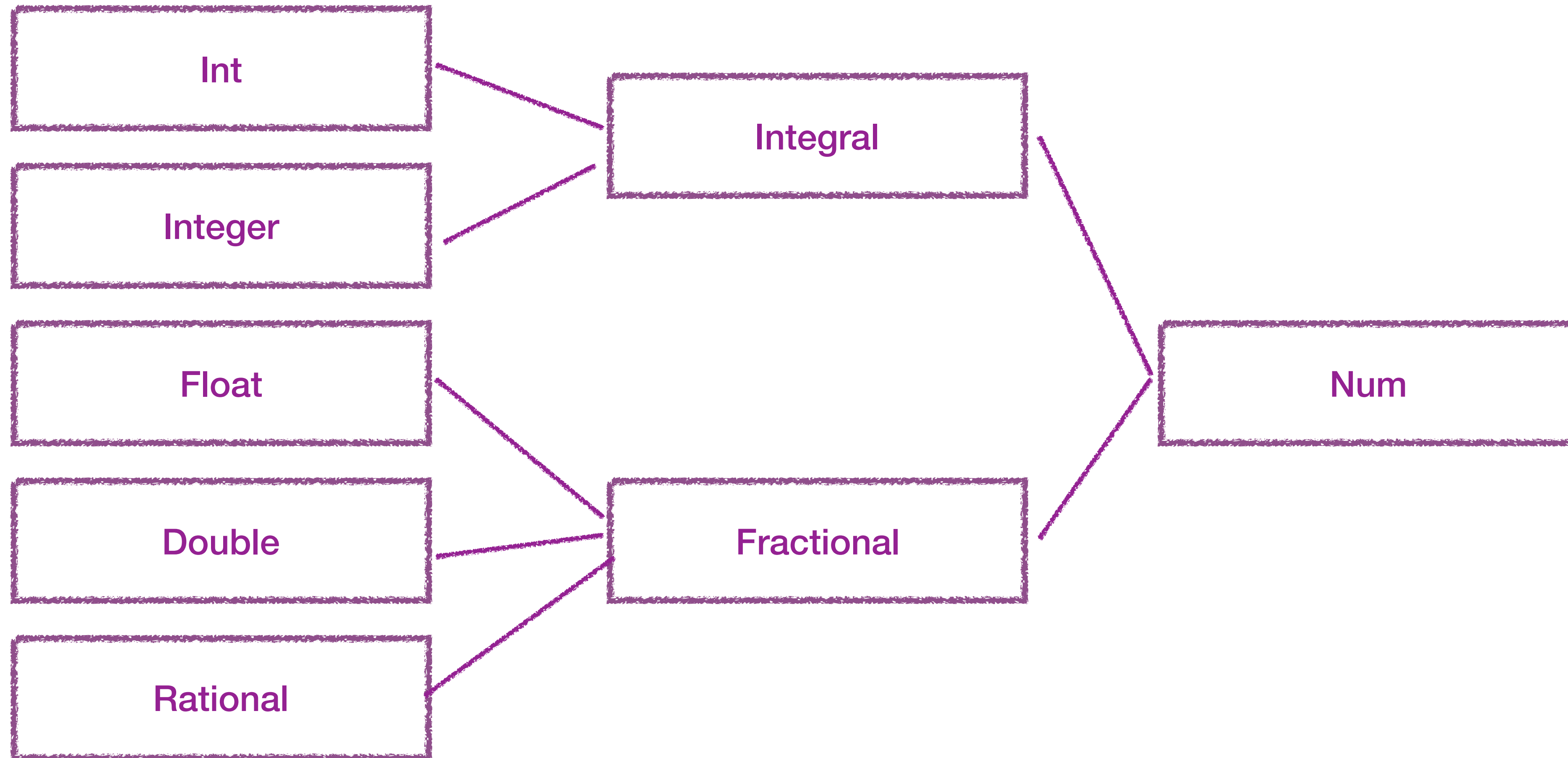
Type system

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
- Function declaration follows Polish Notation
- How many arguments does a function take?:
 - pred, succ (predecessor, successor)
 - min, max

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

```
λ min 10 5 + max 5 15
=> 20
λ 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!

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



What the results will be?

```
λ succ 9 * 10
=> 100
λ 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 `

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


Let's write a function



arguments

`doubleMe x = x + x`

function name

function body

```
> :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

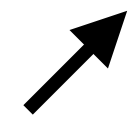
Another function

arguments

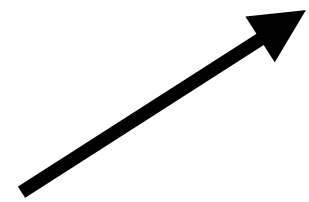


```
doubleUs x y = doubleMe x + doubleMe y
```

function name



function body



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


More useful function



```
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

```
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
- **Nums** are divided into **Integral** and **Fractional** numbers
- Even **if** is a function in Haskell

More Resources



- <http://learnyouahaskell.com/introduction>