- birst character in capital case

## TOIPES & TOIPEKLASHES

haskell has type inference. it can inter type of an expression on its own.

λ:t 'a'

λ:t True

True :: Bool

2: t "HELLO!"

"HELLO!" :: [Char]

λ:t (True, 191)

(True, 191) :: (Bool, Char)

- has type of

λ:t 4==5

4==5 :: Book

when writing functions, were can choose to give them an

explicit type declaration. this is considered good practice.
except when writing very short functions.

remove Non Uppercase :: [char] -> [char]

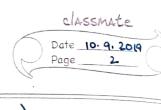
remove Non uppercase st = [c, 1 c+st, c'elem' ['a'.. 'z']]

remove Non Uppercase :: String -> String /

add Three:: Int -> Int -> Int -> Int

add Three x y Z = x + y + Z

it add Three

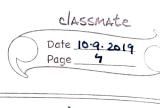


Int : int32 / int64 (int in c) Integer: BigIntege (not bounded) factorial :: Integer -> Integer factorial n = product [1...n] 2 factorial 50 304 140 932 017 133 780 436 126 081 660 647 688 443 776 415 689 605 120 000 000 000 00 Float single precision bloating point Double double precision bloating Point circumperence :: Float -> Float circumperence r = 2 \* pi \* r 2 circomference 4.0 25. 132742 circumperence :: Double -> Double circumference v = 2 \* pi \* v 25. 132741 228718 345 Book = True | False Char x' () - empty tuple - polymorphic function 2 t head type variable

head :: [a] -> a

2 :t fest fst :: (a, b) -> a — all except 10, burctions λ :t (==) (==) :: (Eq a) => a -> a → Bool class constraint == + \* - , / = bunctions only special characters => infix by default to examine type pass it to another function surround with parentheses. call it as a prefex function λ:t elem elem: (Eq a) => (a -> [a] -> Bool types that support equality testing. bunctions its members implement - == , /= λ 5==5 True λ 5/= 5 False λ 'a' == 'a' True χ "Ho Ho" == "Ho Ho" True 1 3.432 == 3.432

True



(Ord) types that have ordering. (except lins.)

(>) :: (ord a) => a → a → Bool

compare (ord a) => a -> a -> Ordering

2 "Abrakadabra" < "Zebra"

True

 $\lambda$  5 >= 2

λ 5 'compare' 3

2 show 3

2 Show 5.334

2 show True

"5-334"

"True"

True

LT

GT

11311

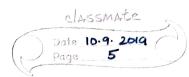
Ordering = GT | LT | EQ

2 "Abrakadabra" 'compare "Zebra"

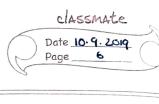
Show) can be presented as string

(Show) > String

 $\lambda : t ()$ 



(Read) parses string to type.
 (read) -> - String -> Type (Read)
λ read "True" Il False
True
λ read "8.2" + 3.8
12.0
2 read "5" - 2
3
 2 read "[1,2,3,4]" ++ [3]
 [1, 2, 3, 4, 3]
λ read "4" X requires type signature
2 wood: t read
read :: (Read a) => String -> a
needs explicit type annotation
 2 read "5" :: Int
5
2 read "5" :: Float
5.0
λ (read "5":: Float) * 4
20.0
λ read "[1,2,3,4]" :: [Int]
[1, 2, 3, 4]
 2 read "(3, 'a')" :: (Int, Char)
 (3, 191)



Enum members are sequentially ordered types

they can be enumerated - can be used in list ranges
have defined successors & predecessors
(succ) (pred)

(), Bool, Char, Ordering, Int, Integer, Float, Pouble

2 ['a'..'e']

"abcde"

2 [LT. GT]

[LT, EQ, GT]

LLT, EQ, GT]
λ [3. 5]

[3, 4, 5]

<sup>1</sup>21

λ succ 'B'

2 max Bound : Bool

2: t maxBound

False

(Bounded) members have an upper and lower bound.

2 min Bound :: Int

- 21 4748 3648

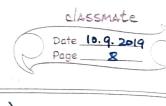
2 maxBound :: Char

2 minBound :: Bool

poly morphic constant

max Bound :: (Bounded a) => a

	3
2 max Bound :: ( Book, Int, Ch.	ar)
(True, 2147848 3647, 11114111	')
(Num) numeric type class	(needs Show, Eq.)
λ: t 20 Poly mar	Phic constant
20 :: (Num t) >> t	
λ 20 :: Int	
20	
λ 20:: Integer	
26	
λ 20 :: Float	
20.0	
20 :: Double	
20.0	
λ: + (*)	
(*) :: (Numa) => a -> a -> a	
(5:: Int) * (6:: Integer) X	
[5 * (6:: Integer) ~	
Integral only eintegral (whol	e) himbors
(Int, Integer)	C)
(In the state of	
(Floating) only floating point n	um her
(Floating) only floating point n	IVIII OCIS



rfrom Integral :: (Num b, Integral a) => a -> b

length :: [a] -> Int

length [1,2,3,4] + 3.2 ×

from Integral (length [1,2,3,4]) + 3.2 ×