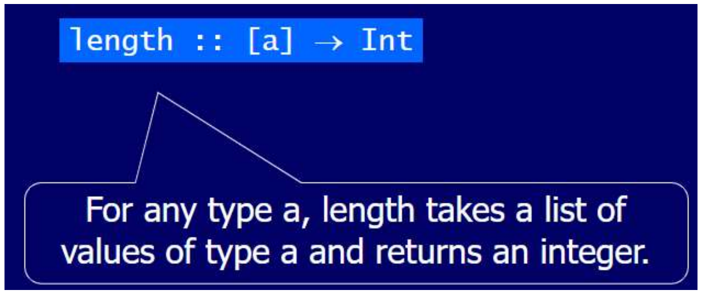
# Polymorphic functions

## Polymorphic functions

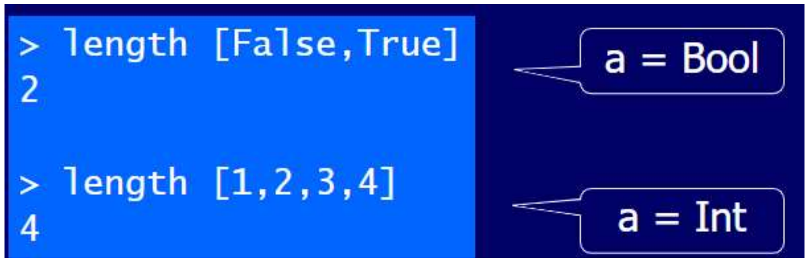
- A function is called polymorphic("of many forms") if its type contains one or more type variables.



```
length :: [a] → Int
```

For any type a, length takes a list of values of type a and returns an integer.

- Type variables can be instantiated to different types in different circumstances:

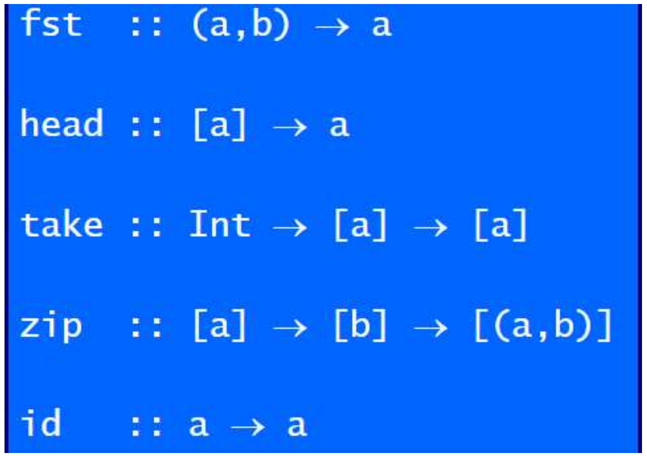


```
> length [False,True]
2

> length [1,2,3,4]
4
```

- Type variable must begin with lower case letter, usually named as a,b,c etc

## Polymorphic functions

- Many functions defined in the standard prelude are polymorphic.
- For example

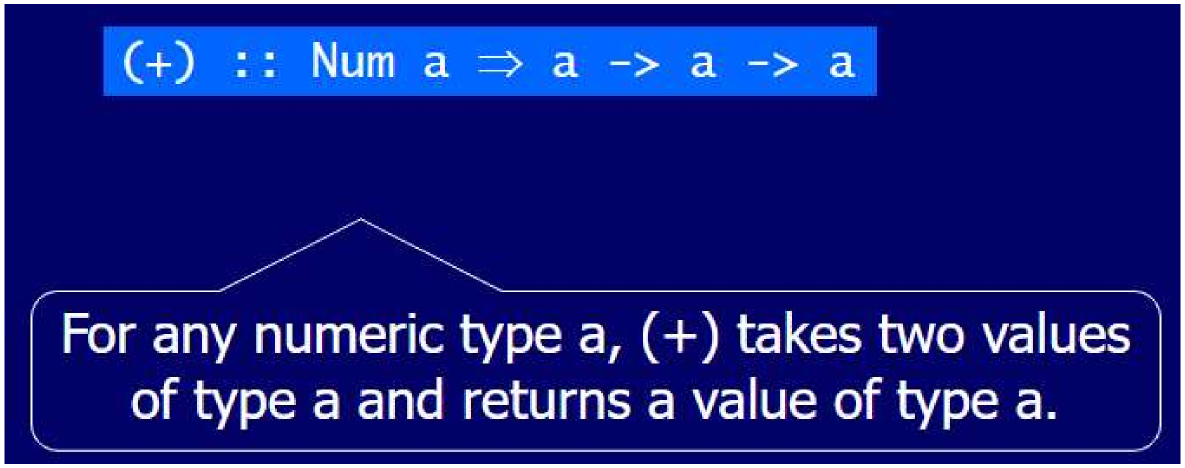


```
fst  :: (a,b) → a

head :: [a] → a

take :: Int → [a] → [a]

zip  :: [a] → [b] → [(a,b)]

id   :: a → a
```

## Overloaded functions

## Overloaded functions

- A polymorphic function is called overloaded if its type contains one or more class constraints.

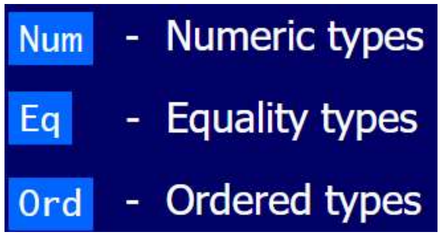


```
(+) :: Num a ⇒ a -> a -> a
```

For any numeric type a, (+) takes two values of type a and returns a value of type a.

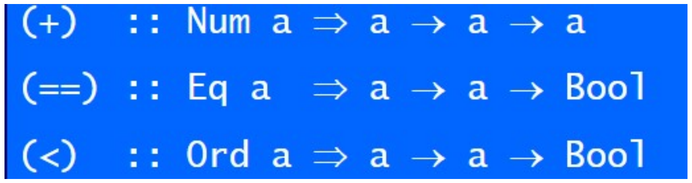
## Overloaded functions

- Haskell has number of type classed, including :



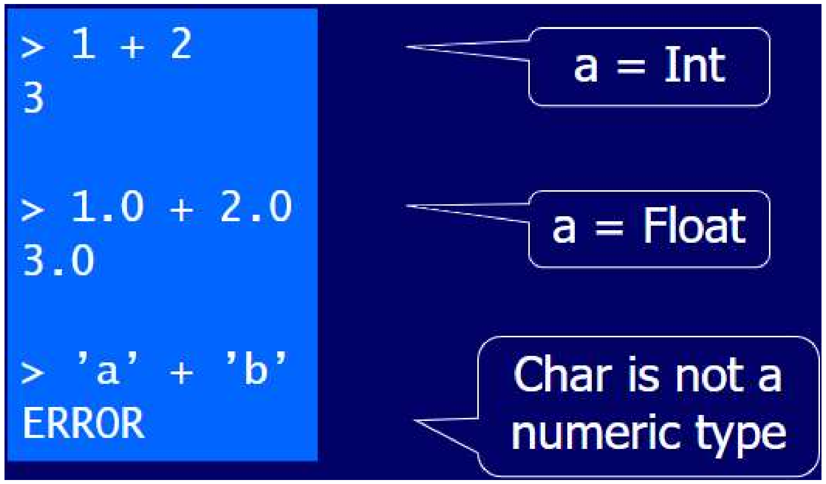
| | | |
|---|---|---|
| Num | - | Numeric types |
| Eq | - | Equality types |
| Ord | - | Ordered types |

- For example:



```
(+)  :: Num a ⇒ a → a → a
(==) :: Eq a  ⇒ a → a → Bool
(<)  :: Ord a ⇒ a → a → Bool
```

- Constraints type variable can be instantiated to any types that satisfy the constraints



```
> 1 + 2
3

> 1.0 + 2.0
3.0

> 'a' + 'b'
ERROR
```