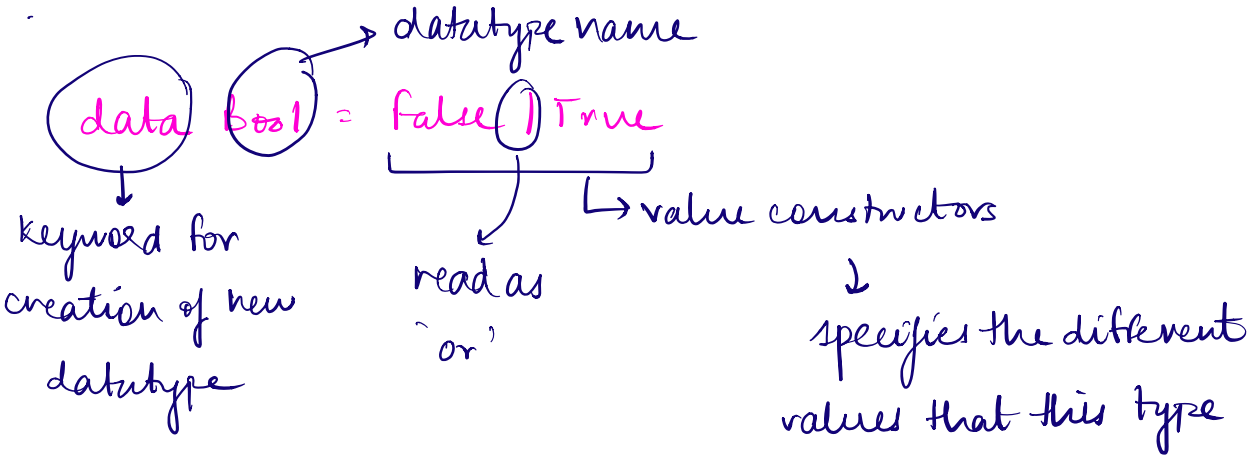
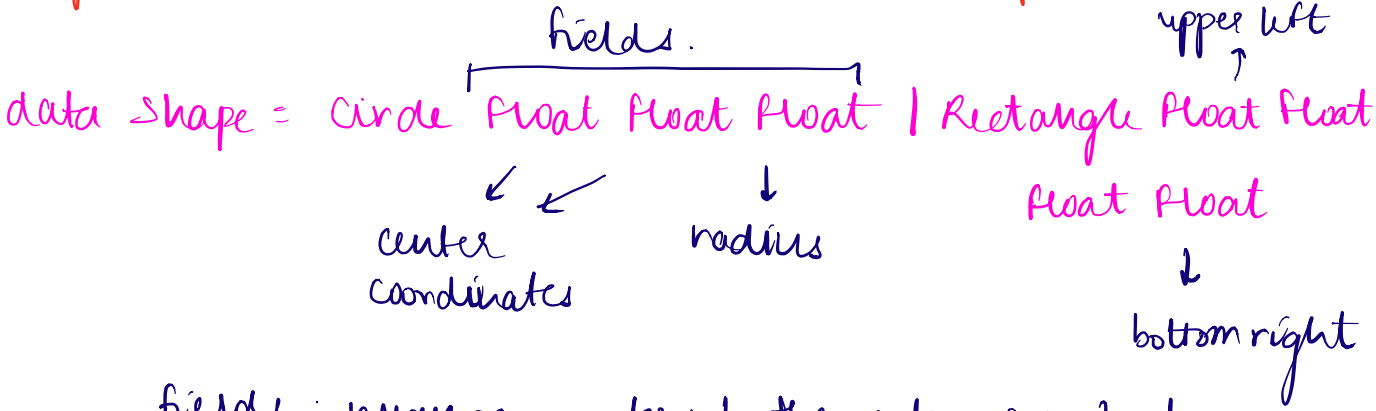
Making our own types and typeclasses.

1. One way to make our own data type is to use the 'data' keyword.



Type name and the value constructors → capital cased.



fields: mean parameters to the value constructors

↓

are actually functions that ultimately return a value of data type.

```
:t Circle
Circle :: Float -> Float -> Float -> Shape

:t Rectangle
Rectangle :: Float -> Float -> Float -> Float -> Shape
```

·) Takeaway: value constructors are functions like everything else.

```
surface :: Shape -> Float   [takes a shape and returns a float]
surface (Circle _ _ r) = pi * r ^ 2
surface (Rectangle x1 y1 x2 y2) =
    (abs $ x2 - x1) * (abs $ y2 - y1)
```

we can pattern match against value constructors

Circle can be treated like any other function

function application → meant to save us keystrokes, space is left associative, and $ is right associative

```
sqrt $ 3 + 4 + 9  ≅  sqrt (3 + 4 + 9)
```

or

```
sum (filter (> 10) (map (* 2) [2..10]))
    ≅  sum $ filter (> 10) $ map (* 2) [2..10]
```

```
surface $ Circle 10 20 10
surface $ Rectangle 0 0 100 100
```

↓

But print won't work, since Haskell doesn't know how to display our data type as a string. When we try to print a value to prompt, Haskell first runs the show function to get the string representation of our value.

↓

So, if we make our Shape part of the Show typeclass, we would be good to go.

data shape = Circle Float Float Float | Rectangle Float Float
Float Float

deriving (show)