Names for Compound Types

type XY = (Double, Double)

Not a new type, just shorthand
Write types to represent:

**Circle**: $x$-coord, $y$-coord, radius

type Circle = (Double, Double, Double)

**Square**: $x$-coord, $y$-coord, side

type Square = (Double, Double, Double)
Type Synonyms

Bug Alarm!
Call areaSquare on circle, get back junk

def type Circle = (Double, Double, Double)
def areaCircle (_,_,r) = pi * r * r

def type Square = (Double, Double, Double)
def areaSquare (_,_,d) = d * d
Solution: New Data Type

```haskell
data CircleT = Circle (Double,Double,Double)
data SquareT = Square (Double,Double,Double)
```

Creates New Types

CircleT
SquareT
Solution: New Data Type

\[
\text{data } \text{CircleT} = \text{Circle} \ (\text{Double},\text{Double},\text{Double}) \\
\text{data } \text{SquareT} = \text{Square} \ (\text{Double},\text{Double},\text{Double})
\]

Creates New Constructors

\[
\text{Circle} :: (\text{Double},\text{Double},\text{Double}) \rightarrow \text{CircleT} \\
\text{Square} :: (\text{Double},\text{Double},\text{Double}) \rightarrow \text{SquareT}
\]

Only way to create values of new type
Solution: New Data Type

data CircleT = Circle (Double,Double,Double)
data SquareT = Square (Double,Double,Double)

Creates New Constructors

Circle :: (Double,Double,Double) -> CircleT
Square :: (Double,Double,Double) -> SquareT

How to access/deconstruct values?
Deconstructing Data

areaCircle :: CircleT -> Double
areaCircle (Circle(_,_,r)) = pi * r * r

areaSquare :: SquareT -> Double
areaSquare (Square(_,_,d)) = d * d

How to access/deconstruct values?
Pattern Match...!
Deconstructing Data

areaSquare :: CircleT -> Double
areaCircle (Circle(_,_,r)) = pi * r * r

areaSquare :: SquareT -> Double
areaSquare (Square(_,_,d)) = d * d

Call areaSquare on CircleT?
Different Types: GHC catches bug!
How to build a list with squares & circles?
Restriction: List elements have same type!
How to build a list with squares & circles?
Solution: Create a type to represent both!
Variant (aka Union) Types

Create a type to represent both!

data CorS =
    Circle (Double,Double,Double) |
    Square (Double,Double,Double)

Circle(1,1,1) :: CorS
Square(2,3,4) :: CorS
[Circle(1,1,1), Square(2,3,4)] :: [CorS]
Variant (aka Union) Types

Access/Deconstruct by Pattern Match

data CorS =
  | Circle (Double,Double,Double)
  | Square (Double,Double,Double)

area :: CorS -> Double
area (Circle(_,_,r)) = pi*r*r
area (Square(_,_,d)) = d*d
data Shape =
  | Rectangle (Double, Double)
  | Ellipse   (Double, Double)
  | RtTriangle(Double, Double)
  | Polygon   [(Double, Double)]

Let's drop the parens...
A Richer Shape

data Shape =
  Rectangle Double Double
  Ellipse Double Double
  RtTriangle Double Double
  Polygon [(Double, Double)]

Let’s drop the parens...
data Shape =
  | Rectangle  Double  Double
  | Ellipse    Double  Double
  | RtTriangle Double  Double
  | Polygon    [(Double, Double)]

Why can’t we drop last case’s parens?
data Shape =
  | Rectangle  Side Side
  | Ellipse     Radius Radius
  | RtTriangle  Side Side
  | Polygon     [Vertex]

type Side    = Double

type Radius  = Double

type Vertex  = (Double, Double)
Calculating The Area

```haskell
area :: Shape -> Double
area (Rectangle l b) = l*b
area (RtTriangle b h) = b*h/2
area (Ellipse r1 r2) = pi*r1*r2
```

GHC warns about missing case!
Calculating Area of Polygon

\[ \text{area (Polygon (v1:v2:v3:vs))} = \text{triArea v1 v2 v3} + \text{area (Polygon (v1:v3:vs))} \]
\[ \text{area (Polygon _ )} = 0 \]