The following code defines a type `Shape` with two constructors, `Circle` and `Rectangle`:

\[
\text{data Shape} = \text{Circle Float Float Float | Rectangle Float Float Float Float deriving (Show)}
\]

(It derives from `Show` so that we can print it on the console.)

Here is a function over this type that will calculate the area of a shape:

\[
\text{area} :: \text{Shape} \rightarrow \text{Float} \\
\text{area (Circle \_ \_ r)} = \pi \times r ^ 2 \\
\text{area (Rectangle x1 y1 x2 y2)} = (\text{abs} \times x2 - x1) \times (\text{abs} \times y2 - y1)
\]

Let’s see some examples of this in action:

> let circ = Circle 0 0 10
> let rect = Rectangle 0 0 20 40
> circ
Circle 0.0 0.0 10.0
> rect
Rectangle 0.0 0.0 20.0 40.0
> area circ
314.15927
> area rect
800.0
> let shapes = [circ, rect, circ]
> map area shapes
[314.15927, 800.0, 314.15927]

**Problem 1.** Write a function `shift` which takes a `Shape`, and x coord, and a y coord, and returns a new `Shape` with its coordinates moved accordingly. Also write a type signature for `shift`. 
The Maybe type is defined as:

\[
\text{data Maybe a = Nothing | Just a}
\]

**Problem 2.** Given your new knowledge of types, write a predicate `something_p` which accepts a Maybe and returns a Bool—True if Maybe is a Just, and False if it's Nothing.

Also write the type signature for `something_p`.

**Problem 3.** Write a predicate `member_p`, which accepts a predicate and a list, and returns True or False depending on whether the predicate finds an item in the list. Use `find` and `something_p`.

Remember the type signature for `find`:

\[
\text{find :: Foldable t => (a -> Bool) -> t a -> Maybe a}
\]

Also write the type signature for `member_p`.