G6021: Comparative Programming

Exercise Sheet 7

1 Types

What are the types of the following? In each case, try to work out the type first, then check with Haskell. (Note: at least one of these does not have a type!)

```
i x = x
k x y = x
zero f x = x
one f x = f x
two f x = f(f x)
three f x = f(f(f x))
s x y z = x z (y z)
w x y = x y y
d x y = x x y
newi = s k k
fib n x y = if n==1 then x else fib (n-1) (x+y) x
fib2 (n,x,y) = if n==1 then x else fib2 (n-1, x+y, x)
```

2 Fixpoints

1. Write the function `fix` from the notes in Haskell.
2. Write factorial as a functional `fact`, without recursion (as given in the notes). Test that

   ```
   fix fact 4
   ```

   computes factorial of 4 as expected.
3. Experiment with fix and other functions.

3 Comparison with Java

1. Haskell (and functional languages in general) are very good as sophisticated calculators. To demonstrate this, compute the following:

   ```
   67^457
   ```

   How would you go about doing this is Java? (You don't have to write any Java, but think about what you would need to do to be able to write it.)
2. In Haskell we can create a new data-type together with functions that work over the new data-type such as this:

```
data Shape = Square Float | Circle Float
             deriving (Show)
area (Square x) = x*x
area (Circle r) = pi*r*r
```

We can test with:

```
area (Square 2)
area (Circle 4)
```

In Java we can represent this by letting `Square` and `Circle` be subtypes of an abstract type `Shape`. Write Java code to do the above example, and compare the code with Haskell.

3. Suppose in you wanted to write a function/method to return two values (say a pair of integers). Outline how you would do this in both Haskell and Java.