Fill in the type of each of the following Haskell expressions. Remember that for numeric literals/functions except division, we use the `Num` typeclass. So we would write:

```
f :: Num a => a -> a
f x = x * 3
```

Concrete types, type variables, and currying.

```
x ::
x = if 3 > 4 then "Goodbye" else "Hello"
y ::
y = [4,5,6,7]
z ::
z = [[]], [null []], False, True, 3 > 4], [True]
f ::
f x = if x + 3 > 10 then "Hi" else "Bye"
g ::
g x = (head x) + 10
id ::
id x = x
h ::
h x = [x]
j ::
j x y = x > (y + 1)
j1 ::
j1 = j 2
k ::
k x y z = if x then y else z
k1 ::
k1 = k True "Hi"
```

These are higher-order functions. Use parentheses in your types to make it clear how to group the `->`.

```
applyToThree ::
applyToThree f = f 3
apply ::
apply f x = f x
makeAdder ::
makeAdder x = \y -> x + y
```

Classic list HOFs.

```
map ::
map f [] = []
map f (x:xs) = (f x):(map f xs)
filter ::
filter f [] = []
filter f (x:xs) =
  if f x
  then
    x:(filter f xs)
  else
    filter f xs
foldl ::
foldl f i [] = i
foldl f i (x:xs) =
  let z = f i x
  in  foldl f z xs
```

These are higher-order functions. Use parentheses in your types to make it clear how to group the `->`.