HW4 Theoretical Questions

1. True, we can implement set! Evaluation as non-sepcial form, we re change line 12 so that we return the evaluation of the expression we are giving binding to. So instead of “return undefined” we will write “return val”

2. We do not really need the box datatype in TypeScript, since TypeScript supports mutation (it is not a pure functional language). But we introduce this datatype to mark explicitly the places where we use mutation - so that we can analyze and control precisely the impact of these mutations.

3. a. The statement is false because we don’t have any assumptions of g so it might not satisfy the type rules
   b. The statement is true, full type of the function to be f: [Number->T2] and for every consistent replacement of T2 we will know that the type of (f h) is T2
   c. The statement is false cause f is a function that receives only one argument and we give 2 arguments to f

4. Stage1: renaming
   ((lambda ([x1: Tx1]): Tb (if (> x1 7) #t #f)) 8)

   Stage2: Assign type variable
   ((lambda (x1) (if (> x1 7) #t #f)) 8)- T0
   (if (> x1 7) #t #f))- T1
   (> x1 7)- Tg
   X1- Tx1
   7- Tnum7
   #t- Tboolt
   #f- Tboolf
   8- Tnum8

   Stage3: Construct type equations
   Primitive equations:
   7 Tnum7= Number
   8 Tnum8 = NUmber
   Tboolt = Boolean
   Tboolf = Boolean

   Composite Equations:
   ((lambda (x1) (if (> x1 7) #t #f)) 8)- T0= [Tf -> T1]

(if (> x1 7) #t #f))- T1= [Tbool->T2]
(> x1 7)- Tbool= [Tx1*Number->Boolean]

Stage4: Solving Equations
T0= [Tf -> T1], Substitution {}
T1= [Tbool>T2] { T0= [Tf -> T1] }
Tbool= [Tx1*Number->Boolean] { T0= [Tf-> [Tbool->T2]] }
Tx1 { T0= [Tf-> [[Tx1*Number->Boolean]->T2]] }
Tboolt { T0= [Tf-> [[Tx1*Number->Boolean]->T2]], Tx1}
Tboolf { T0= [Tf-> [[Tx1*Number->Boolean]->T2]], Tx1, Tboolt }
Tnum8 { T0= [Tf-> [[Tx1*Number->Boolean]->T2]], Tx1, Tboolt, Tboolf }
We know that Tbool must be [Number*Number->Boolean] so we can infer
Tx1=Number and
So we get the following substitution:
{ T0= [Tf-> [[Number*Number->Boolean]->T2]], Tx1=Number, Tboolt=Boolean,
Tboolf=Boolean }

5.
```
const f = (x, callback) => {
        let err = 1/x
        if(err === undefined)
                callback(err, undefined)
        else
                callback(undefined, err)
}

const g = (x, callback) => {
        let err = x*x
        if(err === undefined)
                callback(err, undefined)
        else
                callback(undefined, err)
}

const h = (x, callback) => {
        return callback(x)
}
```

6.
```
(define even?$
        (lambda (n cont)
                (if (= n 0)
                (cont #t)
                (odd?$ ( - n 1)
                        (lambda (res) (cont (- n 1 )))))))
(define odd?$
        (lambda (n cont)
                (if (= n 0)
                (cont #f)
                (even?$ ( - n 1)
                        (lambda (res) (cont (- n 1 )))))))
(define f
        (lambda (x cont)
                If (even?$ x) cont x
                        (lambda (res) (odd?$ (- x 1) cont))))
```